# Docker containers: from 1979 till today

Simone Bonato (bonato@kth.se)

May 13, 2022

## 1 Introduction

Nowadays it is getting more and more common in many Computer Science fields to hear about containers. But what are they exactly and where do they come from?

Containers [8][2] are units of software that include anything that the program requires in order to run, namely the code and all of its dependencies. They have become hugely widespread because they allow programmers to solve the so called "It works on my machine" problem. With this I mean the fact that sometimes a developer is able to produce a piece of code that runs smoothly on his machine (with a specific OS, environment variables and libraries), but when the code is sent to a colleague there can easily be some compatibility issues and errors to solve.

Containers help us alleviate this burden, since they contain all the necessary information for the program to run, we are completely avoiding to run into the issue just mentioned above.

Even though today containers are considered as a relatively new technology, they have been around for quite a while. However, there is still a lack of deep knowledge concerning containers amongst software developers. To this purpose, I think that software developers' knowledge regarding containers could deepen by investigating their history and how they became what they are today. In the next sections I will go through the major points in the history of their development [5].

### 1.1 The chroot command

The stepping stone for the creation of containers is the arrival of chroot in 1979, in version 7 of Unix [10]. Chroot is a command that can change the directory in which a program is being run, limiting its access to a directory (root) and the child directories. The main reason for this is security, in fact the code can't see the other directories outside of its new root and it is isolated from the file

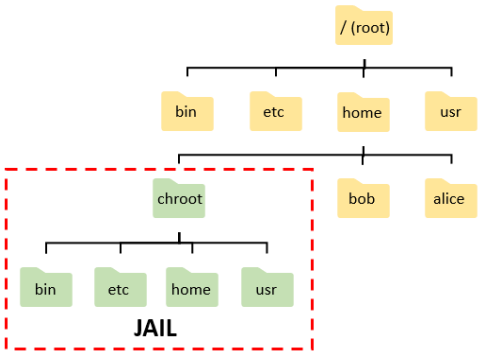



Figure 1: How the *chroot* command separates a program from the other directories

system. It also allowed programmers to build better dependencies for their programs.

We can already see the similarity between chroot and containers in the way we know them, since also containers are made to create a sort of isolated environment where we can run our code. However, containers had not been developed for over 2 decades after that.

## 1.2 FreeBSD Jails


The final push for the development of containers took place around the year 2000, when FreeBSD Jails [3] were introduced. A jail is basically a partition of a computer, it is possible to create different partitions on the same system sharing the same kernel. One of the major improvements was the ability to assign a specific IP address to each jail, adding another necessary feature for the containers ecosystem.

Jails were introduce by "R&D Associates" and the goal was to create an environment where the services of the company could be completely isolated from the ones of their clients". Poul-Henning Kamp, a Danish computer software

developer, realized that compartmentalizing the system so that only the right people could be given access was more convenient than adding another layer of complexity and options.

The main difference between chroot and jails [6] is that the former simply restricts processes to a specific view of the filesystem with respect to the rest of the system, while the latter assigns them specific IP addresses, allowing the possibility to sandbox processes.

With FreeBSD jails we can already make use of some of the features that make containers so appealing, such as the possibility to make different virtual machines, each made with its own set of dependencies and configurations. In this way the testing of software was made more flexible; for example one can run different versions of it.

### 1.3 Linux V-server


In the year 2001, the jail system was improved with the Linux V-Server [9]. The most relevant improvement was the fact that the jail mechanism called "virtual private server" that is now able to also partition resources on a computer (eg. file system, CPU, network addresses and memory). It was designed in a way that prevented processes to mount up to the denial-of-service (DoS) attack on what is outside of their partition scope.

This new type of virtualization of the OS was introduced as a patch for the Linux OS.

### 1.4 Solaris container


Not too many years later, in 2004, Oracle developed the so called "Solaris container" [7]. This is no other than another way to virtualize the OS, that puts together the partitioning of resources as well as the *zones*. A zone behaves as a fully isolated virtual server within a single instance of the Operating System.

A zone is provided with its own node name, access to virtual or physical network interfaces, and storage. What is surprising is that they have no limitation whatsoever in terms of minimum amount of needed hardware, other than the

minimum storage that can contain its own configuration. Oracle's Enterprise Manager Ops Center is used to set up zones and specify how the resources available to them.

But the most important feature is that each zone has its own security boundary, that prevents a process associated with one zone to interact or observe a process located in another one.

Solaris Containers may sometimes be a better virtualization solution than other tools. The main advantages include:

- Easy to configure: It is simply necessary to point and click through the Enterprise Manager Ops Center
- Easy to manage virtual resources: The host resources of a single system can be slit up rather than a cluster

However, there are also drawbacks to using Solaris Containers. The main disadvantages include:

- Container management does not scale well as it is only possible to manage zones to the limit of how many one person can handle manually
- Containers cannot be quickly created based on application images which limits the practicality of using Solaris Containers for continuous delivery (CD)
- Limited usage/flexibility as it is only possible to use Oracle solutions when to comes to container orchestration and monitoring tools

As such, even though Solaris Containers are not as flexible as Linux containers and are relatively old, they are easy to work with and configure and provide powerful features that can be utilized when dealing with Oracle data centers.

## 1.5 OpenVZ


A year later, in 2005, OpenVZ (Open Virtuzzo) [11] was initially released. This is an open source OS virtualization technology for Linux that uses Linux kernel for virtualization, isolation, resource management and checkpointing and is a virtualization approach that assists website operators in deploying OS instances

on web servers in a container oriented approach. OS virtualization ensures that certain modules reside while still on the system and are used by visitors identical to the same kernel. This makes sure that every container shares an upstream OS and also runs as a discrete Linux container. Every container performs like a stand-alone server, can be rebooted independently, has root access, users, IP addresses, memory, processes, files, applications, system libraries and configuration files. This approach ensures powerful OpenVZ tools and that use resources effectively.

Since OpenVZ is a Linux based virtualization, it can use Linux based OS such as Debian, Fedora, Centos and a lot more. It needs both the host and the user operating system to operate Linux. OpenVZ is very fast and effective, despite using a specific Linux kernel fix, as it does not use any overhead for the valid hypervisors. This is what makes OpenVZ standout and faster from its competition in the market.

The following summarizes the main advantages of using OpenVZ virtualization:

- Effective usage of resources saving a lot of memory and CPUs and helps in better output
- Ease of configuration, usage and management. It is the cheapest container per Virtual Private Server (VPS)
- Usually a good choice for anyone who does not have unusual requirements
- Configuration is convenient to get used to and has easier maintenance and lower overhead prices
- Gain the benefits of a dedicated server without any disadvantages
- Improvements to software supported by separate launches of containerized micro services making this versatility have minimal dark releases and rollback improvements at the micro service stage
- Easily scalable

However, there are also disadvantages to using OpenVZ. This includes the following:

- Possibility that if a single kernel crashes, then every VPS instance on the same host will be affected
- Limited to Linux OS and not possible to remotely integrate the OS
- Difficult to change the VPS setup
- Not possible to reach every bit module
- CPU and device port are shared among all the VPSs in the network leading to less privacy

- OpenVZ VPS at risk of over selling by dishonest hosts who over sell services to accounts

## 1.6 LinuX Containers LXC

One of the most important steps in the history of containers is the introduction of the *control groups*. Control groups are simply a way to isolate the usage of resources (CPU, memory, etc.) and these can be considered one of the first examples of what containers look nowadays. In fact in 2008 they were built upon LXC (LinuX Containers) which at the time were considered the most stable and reliable versions of the containers; they worked specifically on the Linux kernel.

The main advantage is the fact that it is no more necessary to instantiate a VM to allocate and prioritize the resources of the system. Moreover, there is also the *namespace isolation*, that allows complete isolation of the system's operating environment from the application's point of view. This makes it possible to create a sort of sandbox to provide isolation for the development of applications.

Given how good they were, other similar tools were developed not so many years later, Warden in 2011 and Docker in 2013.

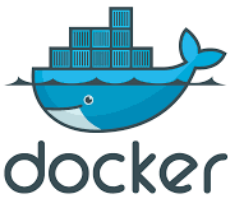
## 1.7 Warden


Coming into 2011, CloudFoundry began Warden [4]. Warden is a container implementation that runs as a daemon and separates environments from any operating systems as well as providing an API for container management. A client-server model was developed to manage containers across multiple hosts and Warden also provides a service to manage namespaces, cgroups and the life cycle process. It enables multiple local environments to run at the same time without port conflicts via the use of a few services for proxying requests. Warden utilizes *cgroups* to isolate usage of resources and *namespaces* to isolate applications running within containers from each other and host processes.

One of the main differences between Warden and Docker (2013) is how the container images are organized. Warden runs applications that obtain all their required dependencies from software called *buildpacks* and so Warden containers typically have two layers only. One of the layers is a read only layer with an operating system root file system and a second read-write layer that is for

the application to be run with all its required dependencies and temporary data.

## 1.8 Docker




In the year 2008 the company dotCloud was founded by Kamel Founadi, Solomon Hykes, and Sebastien Pahl in Paris. What is known to the world today as a Docker container, has its origin from this company, that in 2013 decided to release the code to the public as Open Source. In this way everyone could contribute to the development of the containers, and not only programmers at dotCloud. On October 29, 2013, dotCloud was renamed Docker.

Despite the fact containers were already considered to be an impressive technology, the arrival of Docker completely revolutionized the field and led the way for a Renaissance in the containers world. Initially it was built using LXC, but it did not take too long to be replaced by its own library called *libcontainer*.

What made Docker so much better than the competition was that it did not simply offer the tools to build containers, but the fact that it came in with a whole ecosystem to create and manage them.

In fact, here are some of the main benefits [1] of using Docker:

- Portability: once the application runs inside a container, it can simply be transferred to any other system where Docker is running and the user can be sure it is going to work there too.
- Performance: given the fact that containers do not include an OS, but use the one of the host machine, they are really fast (if compared to VMs) and much quicker to start.
- Agility:containers make the development process to be much more agile and responsive compared to how it was done before. It is now possible to take advantage of the CI/CD pipelines that make the whole making process, from creation to deployment, smooth and with less interruptions.
- Isolation: since Docker containers are completely independent from one another, it is possible to run applications that might require different versions of the same library or software.

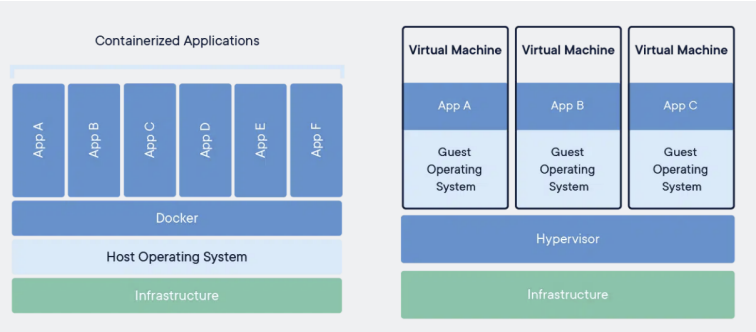



Figure 2: Containers (left) vs VM structure (right).

- Scalability: If the application needs to scale up, it can be simply done by creating other containers which can take advantage of many of the management options.

What makes Docker great is the ecosystem that it offers, and the fact that it can be run on Linux, Windows and Mac. With this I do not only mean the possibility to create containers, but also their management and integration with other software for CI/CD.

One thing that is fundamental to create a Docker container is the concept of Dockerfile. This is no other than a text-file (thus extremely lightweight!) that contains all the necessary configurations to create an Image. The fact that it is so portable makes it convenient to put the Dockerfile into a source code version control (eg. git).

Docker containers tend to be confused with Virtual Machines (VMs) when first introduced to developers and practitioners. As such, it is important to highlight that they are both virtualization tools, but while VMs include whole hard drives, dependencies, files, as well as the whole OS (order of GBs), containers are lightweight virtualization as they package up all the code and dependencies required to run a specific application while using the same kernel OS on the host (Mbs).

# 2 Conclusion

Containers are revolutionizing the way software developers interact with their code during the development and deployment phases, making sure that the process becomes as smooth and less prone to error as possible. Their most relevant features that make it possible are:

- they contain all the necessary dependencies to run the code, in just a few Mbs. Hence they solve the "It works on my machine" issue and allow a better development experience.

- they provide isolation from the system they are being run on; allowing, for example, to have different versions of the same program running simultaneously.
- they are much more flexible than VMs, both in terms of dimensions as well as of implementation.

# References


[1] *Benefits of Using Docker*. URL: https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ETS-help/GUID-F5BDACC7-6F0E-4EBB-9F62-E0046D8CCF1B.html.

[2] Carl Boettiger. "An introduction to Docker for reproducible research, with examples from the R environment". *ACM SIGOPS Oper. Syst. Rev.* 49 (Oct. 2014). DOI: 10.1145/2723872.2723882.

[3] *Chapter 15. Jails*. URL: https://docs.freebsd.org/en/books/handbook/jails/.

[4] *Cloud Foundry Containers: Warden, Docker, and Garden*. URL: https://www.altoros.com/blog/cloud-foundry-containers-warden-docker-and-garden/.

[5] *Infographic: A Brief History of Containerization*. URL: https://www.plesk.com/blog/business-industry/infographic-brief-history-linux-containerization/.

[6] Syahrul Sazli Shaharir. "The FreeBSD Jail: When Chroot is not Enough" ().

[7] *The Role of Oracle Solaris Zones and Linux Containers in a Virtualization Strategy*. URL: https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-zones-containers-virtualization.html.

[8] *Use containers to Build, Share and Run your applications*. URL: https://www.docker.com/resources/what-container/.

[9] *Welcome to Linux-VServer.org*. URL: http://linux-vserver.org/Welcome_to_Linux-VServer.org.

[10] *What Is chroot on Linux and How Do You Use It?* URL: https://www.howtogeek.com/devops/what-is-chroot-on-linux-and-how-do-you-use-it/.

[11] *What is OpenVZ Virtualization?* URL: https://www.hostnamaste.com/blog/what-is-openvz-virtualization/.