Hypervisor vs. Container-based virtualization

Michael Eder

Chair for Network Architectures and Services
Department for Computer Science
Technische Universität München

October 6, 2015

Outline

- Motivation
- Container vs. Hypervisor-based virtualization
- Goals and use cases of both virtualization technologies
- From chroot over containers to Docker
  - Chroot
  - Linux Containers
  - Docker
- Security
- Questions / Discussion

Motivation

- „virtualization“ synonymous for hypervisor-based virtualization for a long time
- Container-based virtualization (especially Docker) gained a lot of attention in the last few years
- What are the goals of both technologies? How secure are they?

Motivation

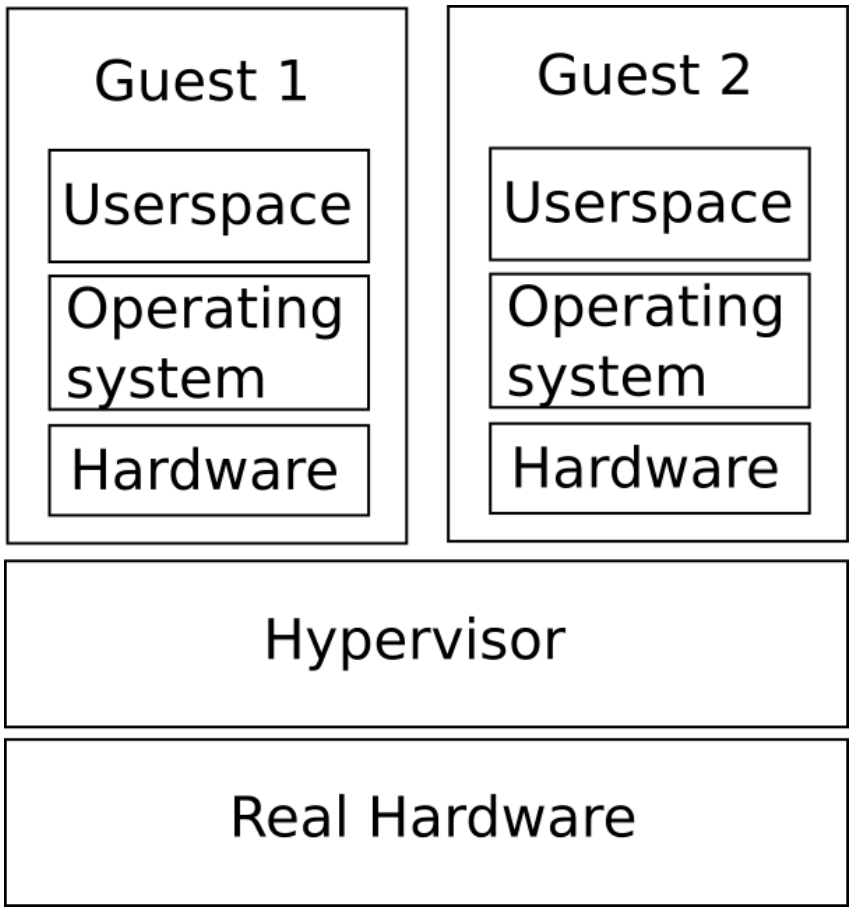
Goals of the talk

- Describe both types of virtualization and the underlying mechanisms
- Take a look at the novelties Docker introduced
- Compare the security of both approaches

Characteristics of hypervisor-based virtualization

- Abstraction from Hardware through Hypervisor $\rightarrow$ „virtual machine“ or „virtual computer“
- Emulated hardware running a complete operating system, the „guest operating system“ $\rightarrow$ running different operating systems on different emulated hardware at the same time is possible
- Type 1 hypervisors setting up directly on hardware and type 2 hypervisors requiring a so-called „host operating system“

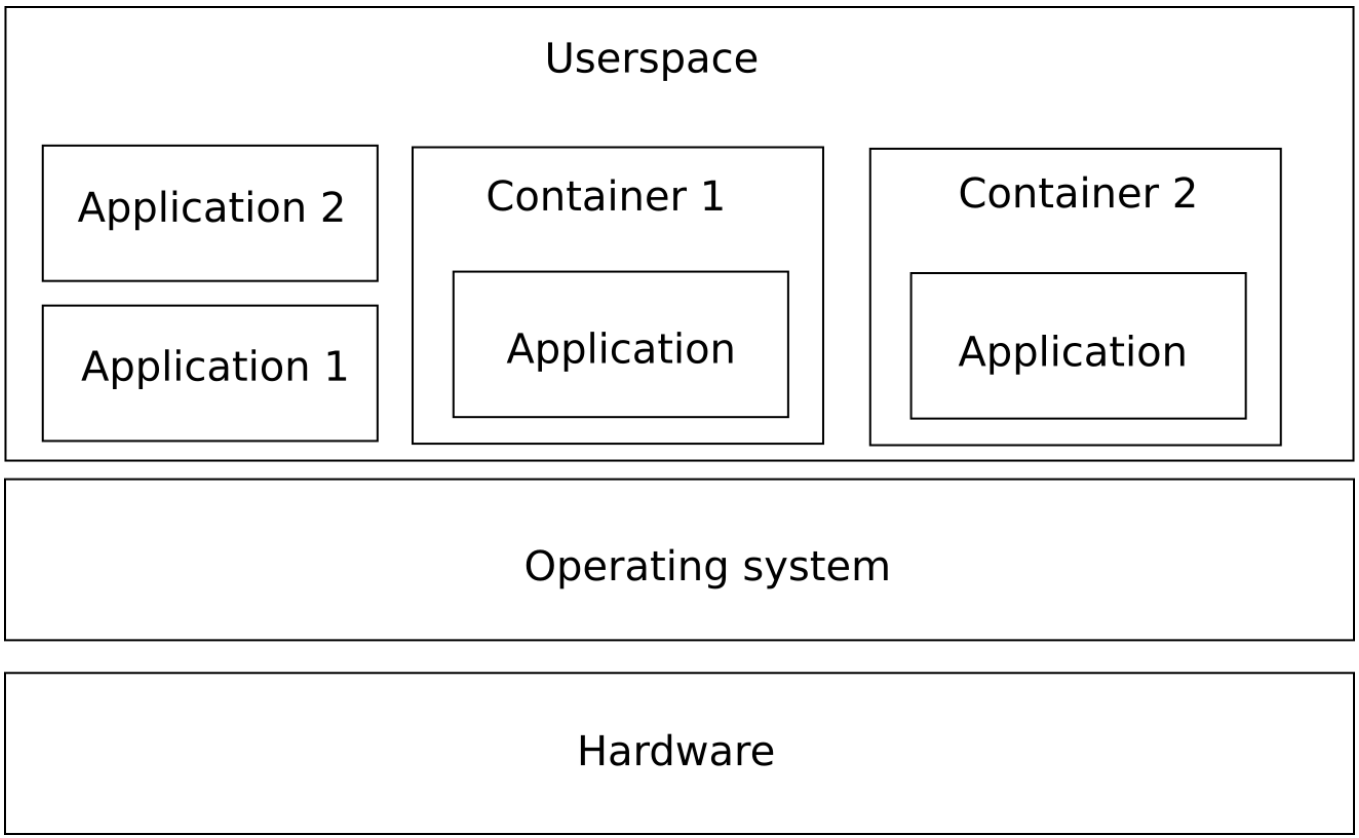
Scheme of hypervisor-based virtualization



Characteristics of container-based virtualization

- „Normal“ process running on the host operating system
  - abstraction from hardware through operating system
  - isolated from other processes and system resources
- Fast start and leightweight resource fingerprint

Scheme of container-based virtualization



Goals and use cases of hypervisor-based virtualization

- Emulate different or not even available hardware (e.g. smartphone)
- Run other operating systems in parallel to the host operating system
- Run virtual machines everywhere the hypervisor is able to run $\rightarrow$ eases i.e. migration

Goals and use cases of container-based virtualization

- Fast startup and small resource fingerprint compared to hypervisor-based virtualization $\rightarrow$ Possible to run a lot of containers on a machine in parallel
- Small image size because capabilities provided by the operating system do not need to be integrated $\rightarrow$ saving disk space, RAM and bandwith
- Containers run everywhere where a compatible operating system (usually Linux) is available $\rightarrow$ eases i.e. migration

Chroot → Containers → Docker

The evolution of containers:

- chroot
- Linux Containers
  - Kernel namespaces
  - cgroups
  - Mandatory Access Control (MAC)
- Docker
  - Single command line tool
  - Filesystem layers
  - Docker Hub

change **root** (chroot)

- Change the root directory of a process to another directory
  $\rightarrow$ process not able to access entire filesystem
- On Linux, not considered a security feature. Escaping chroot is possible and documented
- Nevertheless, adds a security layer that makes it harder for attackers or erroneous software to access the complete filesystem

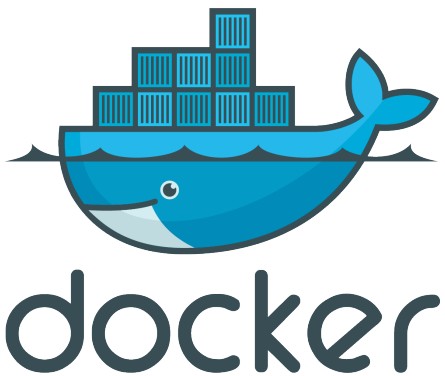
Linux Containers

Linux Containers aim to create isolated environments for processes using kernel capabilties. The most important are:

- Kernel namespaces, in order to separate process resources
- cgroups, in order to limit and control assigned resources
- Mandatory Access Control (MAC), in order to protect other processes from Linux containers

Docker

Docker gained a lot of attention over the past two years and made container-based virtualization popular. These properties are reasons for Dockers success:



- single command line tool: Working with Docker is simple
- Independent distributed filesystem layers: A Docker container consists of several images that can be combined like building blocks to create new containers
- Docker Hub, a web service allowing people to create, use, modify, share and collaborate on container images with each other

*image: Docker Media Kit, https://www.docker.com/brand-guidelines

Security of container vs. hypervisor-based virtualization

security hierarchy:

1. hypervisor-based virtualization (considered very secure because of high abstraction level and another operating system inside)
2. container-based virtualization (good isolation of processes)
3. standard security model of operating system processes

BUT: a good exploit (or some of them together) may even escape virtual machines and may result in a root shell on the host! There is no security guarantee!

Security of container vs. hypervisor-based virtualization

Various security problems:

- insufficient maintainence of virtual machines and containers; study shows that about 30% of **official** images on Docker Hub contain known **high priority** vulnerabilities!
- additional functionalities introduce new attack surfaces, i.e. Docker's image verification mechanism was not secure for a long time without anybody noticing
- potentially malicious containers may get distributed easily via the Internet

Security of container and hypervisor-based virtualization

But:

- virtual machines (and even containers) are a huge improvement making it harder for attackers and erroneous software to do harm
- It's free
- other nice side effects, e.g. hardening of the complete system by MAC or resource control preventing DoS attacks
- there's nothing speaking against combining both approaches, so no „vs." in this slide's title

Questions? Discussion!