Was sind Linux Container und wie funktionieren sie?

Anian Ziegler

23. September 2018

cioplenu

Was ist ein Container?

Was sind Container?

- Virtualisierung auf Betriebssystemebene

Was sind Container?

- Virtualisierung auf Betriebssystemebene
- Basiert auf meheren Features des Linux Kernels

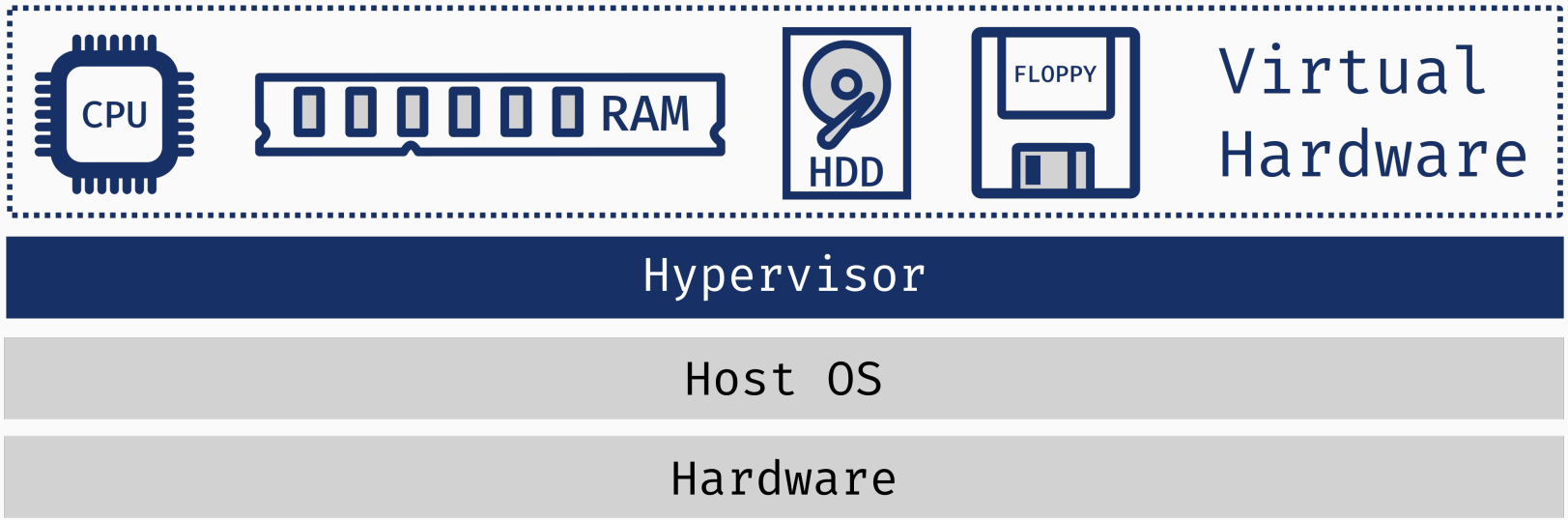
Was sind Container?

- Virtualisierung auf Betriebssystemebene
- Basiert auf meheren Features des Linux Kernels
- Isoliert und verpackt Anwendungen

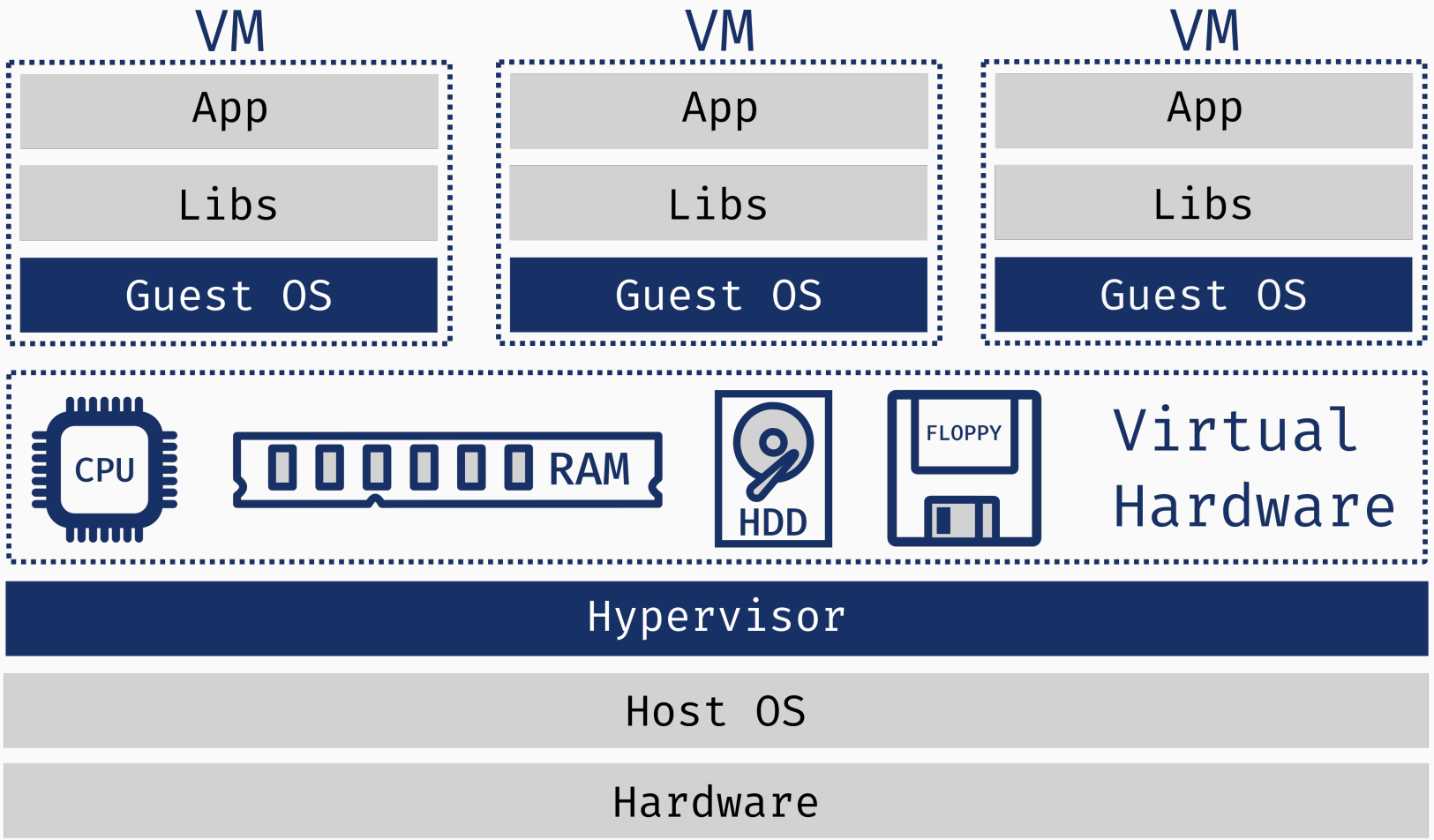
Was sind Container?

- Virtualisierung auf Betriebssystemebene
- Basiert auf meheren Features des Linux Kernels
- Isoliert und verpackt Anwendungen

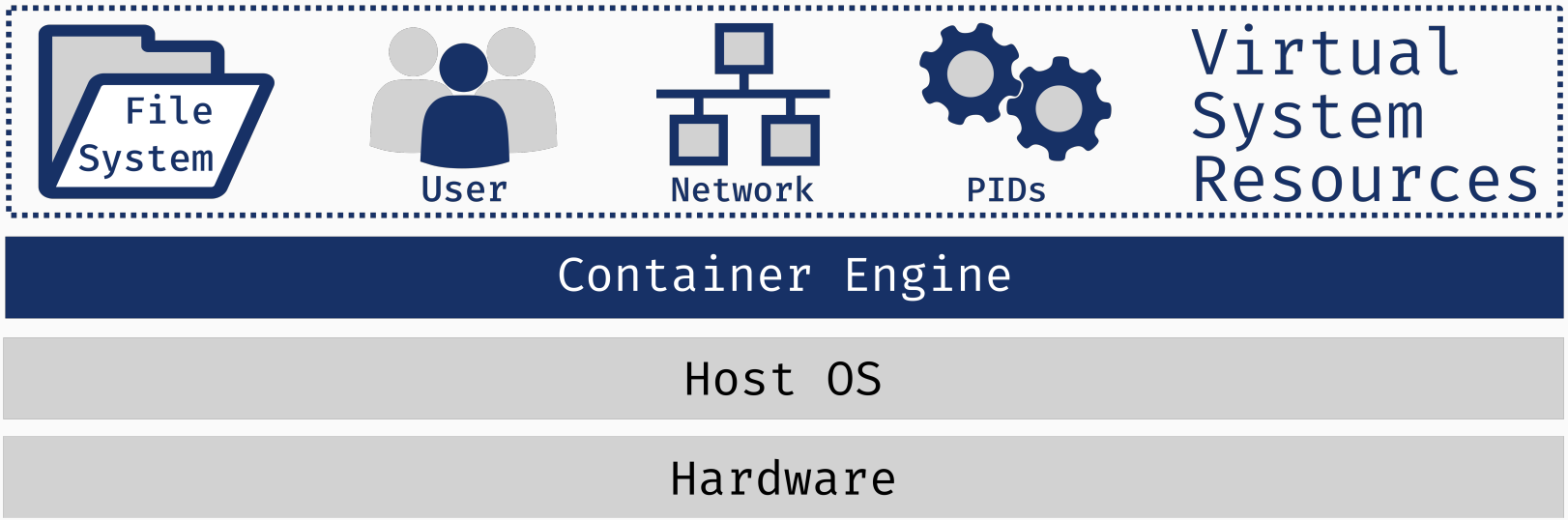
Exkurs: Hardware-Virtualisierung



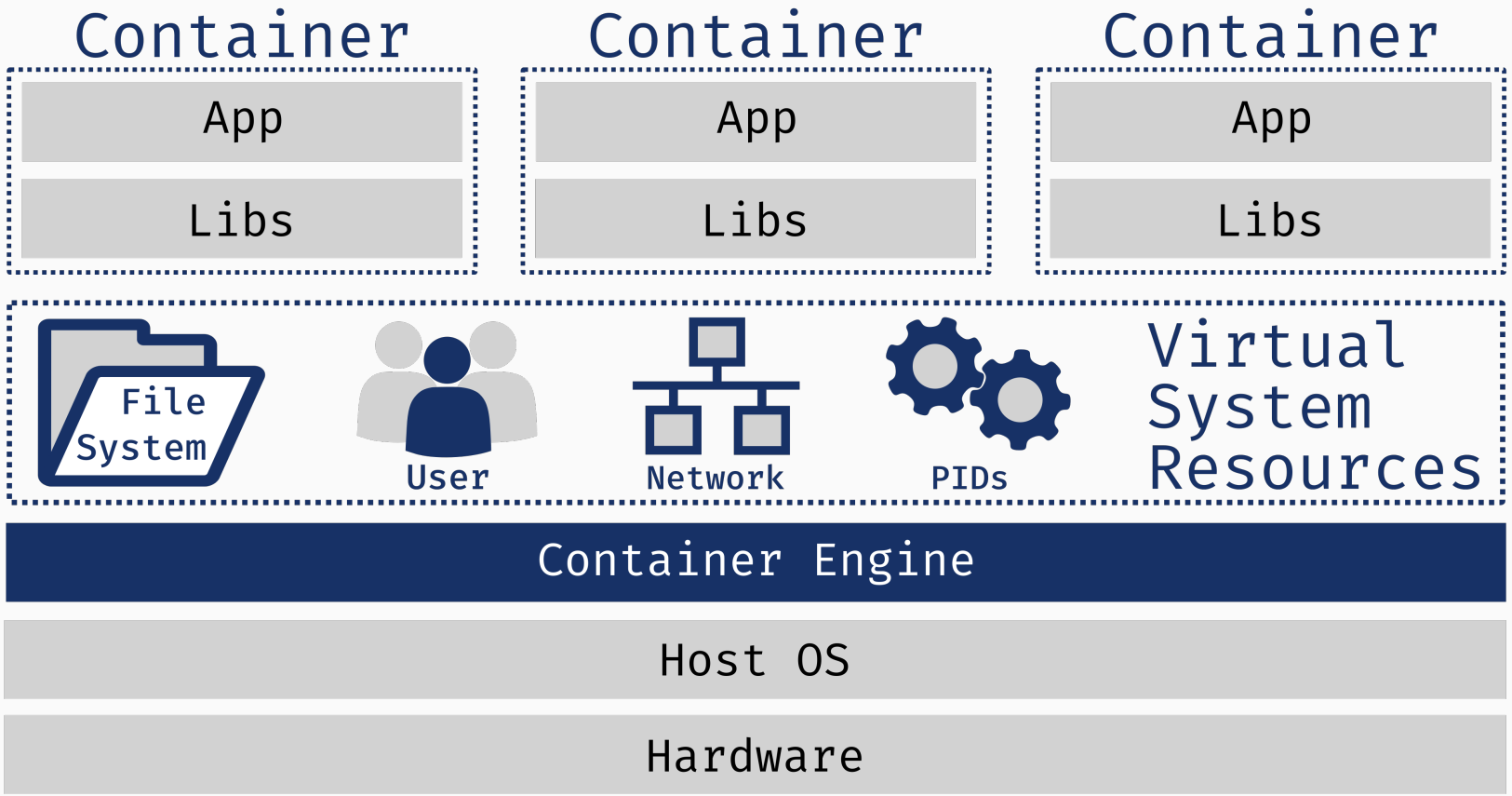
Exkurs: Hardware-Virtualisierung



Virtualisierung auf Betriebssystemebene



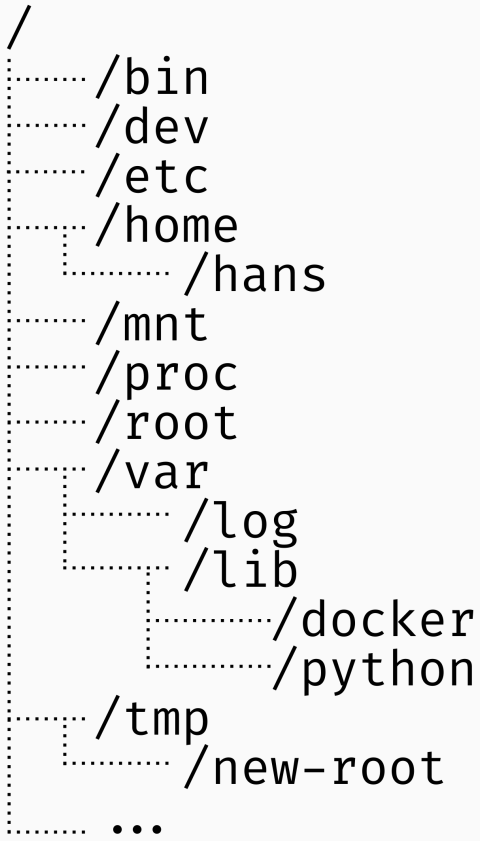
Virtualisierung auf Betriebssystemebene



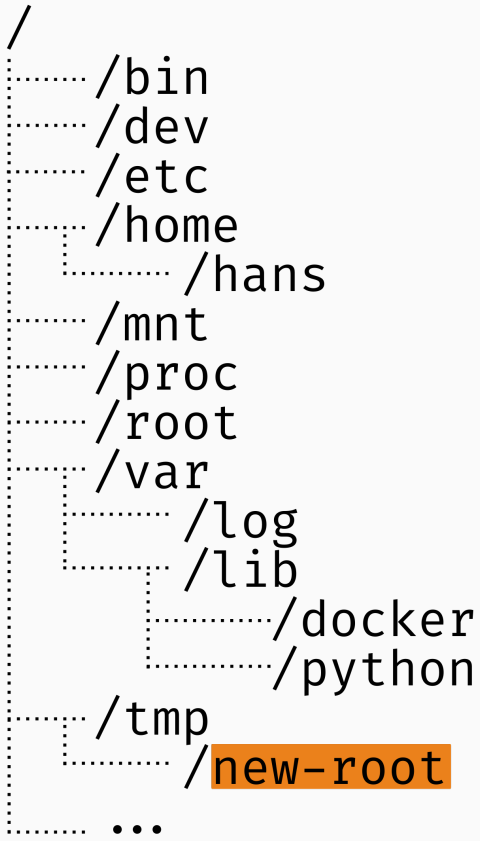
Demo

Implementierung in Linux

chroot


/
/bin
/dev
/etc
/home
/hans
/mnt
/proc
/root
/var
/log
/lib
/docker
/python
/tmp
/new-root
...

chroot


/
/bin
/dev
/etc
/home
/hans
/mnt
/proc
/root
/var
/log
/lib
/docker
/python
/tmp
/new-root
...

Demo

chroot

- Erlaubt es das root Verzeichnis / neu zu setzen
- Ermöglicht verschiedene Versionen eines Tools auf einem System zu installieren
- Sehr Hilfreich zum debugging oder bei der Installation von Linux
- **Aber:** Noch keine Isolierung der Prozesse, User etc. von einander

Namespaces

- API des Linux Kernel um **virtuelle System Ressourcen** wie Netzwerk Interfaces, Mount points, UserIDs und weitere System Ressourcen zu erstellen

Namespaces

- API des Linux Kernel um **virtuelle System Ressourcen** wie Netzwerk Interfaces, Mount points, UserIDs und weitere System Ressourcen zu erstellen
- Diese Ressourcen können einzelnen Prozessen zugewiesen werden

Namespaces

- API des Linux Kernel um **virtuelle System Ressourcen** wie Netzwerk Interfaces, Mount points, UserIDs und weitere System Ressourcen zu erstellen
- Diese Ressourcen können einzelnen Prozessen zugewiesen werden
- Können auch für sich genommen verwendet werden: Auf seinem eigenen Rechner mit Network-Namespaces ein Netzwerk zu simulieren

Control Groups

- Management von CPU Zyklen, Arbeitsspeicher oder Netzwerk Bandbreite für Gruppen von Prozessen

Control Groups

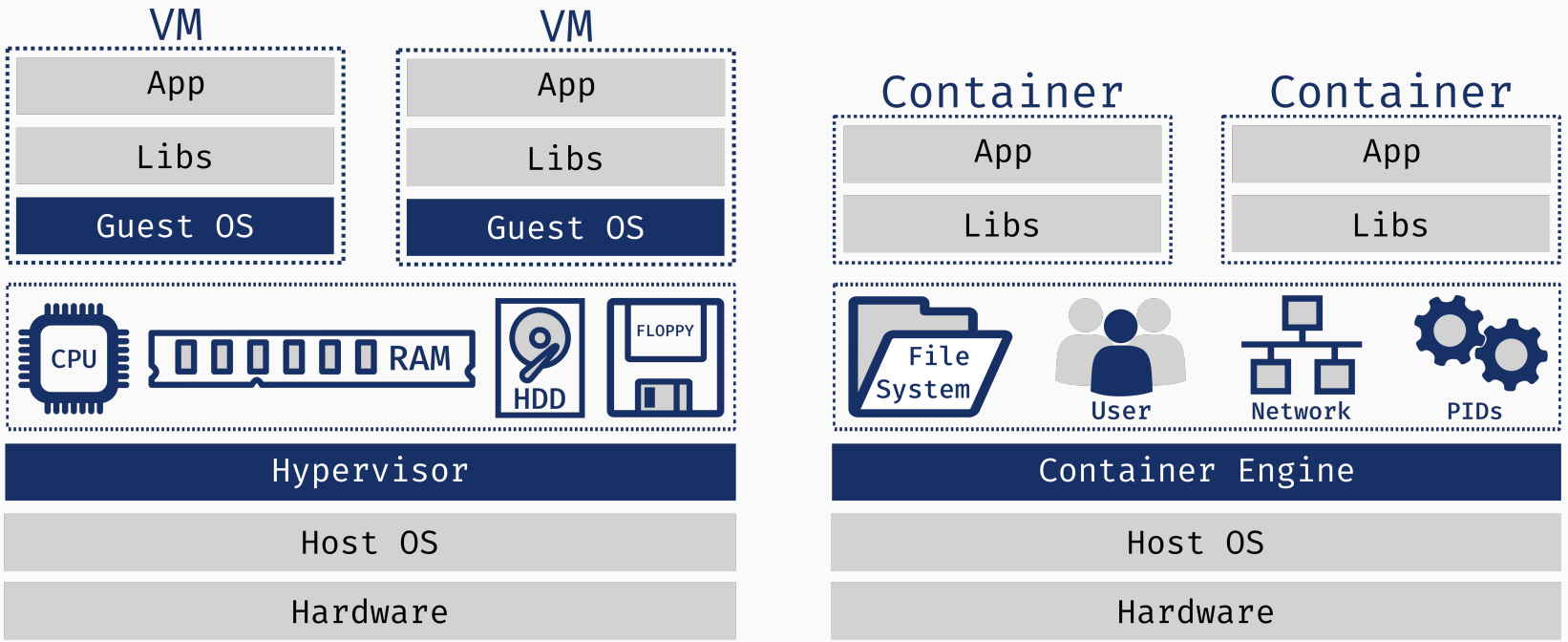
- Management von CPU Zyklen, Arbeitsspeicher oder Netzwerk Bandbreite für Gruppen von Prozessen
- Prozesse können in ihrem Ressourcenverbrauch eingeschränkt werden

Control Groups

- Management von CPU Zyklen, Arbeitsspeicher oder Netzwerk Bandbreite für Gruppen von Prozessen
- Prozesse können in ihrem Ressourcenverbrauch eingeschränkt werden
- Auch seperat Nutzbar

Vergleich mit VMs

Vergleich mit VMs



Unterschied: Bei VMs wird im Kernel/Hypervisor Hardware virtualisiert und darauf laufen andere Kernels. Container teilen sich einen Kernel, der OS-Ressourcen virtualisiert.

Vorteile VMs

- Voller virtueller Computer mit allen Features
- Egal welcher Kernel: Linux, BSD, Windows NT, x86, x64
- Starke Isolierung

Vorteile Container

- Effizienter: Weniger Ressourcenverbrauch und kein Boot-Vorgang
- Einfacher zu managen
- Auch einzelne Module verwendbar

Praktische Anwendung

Praktische Anwendung

- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs

Praktische Anwendung

- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs
- Weg aus der **Dependency Hell**

Praktische Anwendung

- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs
- Weg aus der **Dependency Hell**
- Portable **Development Environments**. Verhindert ”Works on My Machine”weil alle die gleiche Version haben

Praktische Anwendung

- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs
- Weg aus der **Dependency Hell**
- Portable **Development Environments**. Verhindert ”Works on My Machine”weil alle die gleiche Version haben
- **Isolierung** unsicherer Prozesse von einander

Praktische Anwendung

- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs
- Weg aus der **Dependency Hell**
- Portable **Development Environments**. Verhindert ”Works on My Machine”weil alle die gleiche Version haben
- **Isolierung** unsicherer Prozesse von einander
- Ermöglicht weitreichende **Orchestrierung** auf großen Rechner-Clustern mit Failover und großer Skalierung durch Lösungen wie Kubernetes

Container sind keine neue Erfindung

Geschichte

1979 — UNIX v7: chroot system call, später in BSD.

Geschichte


1979
UNIX v7: chroot system call, später in BSD.
2000
FreeBSD Jails.

Geschichte

1979 UNIX v7: chroot system call, später in BSD.

2000 FreeBSD Jails.

2001 Linux Vserver ermöglicht erste virtualisierung auf Betrtriebssystemebene durch Kernel Patching.

Geschichte

1979 — UNIX v7: chroot system call, später in BSD.

2000 — FreeBSD Jails.

2001 — Linux Vserver ermöglicht erste virtualisierung auf Betrtriebssystemebene durch Kernel Patching.

2004 — Solaris Zones.

Geschichte

1979 UNIX v7: chroot system call, später in BSD.

2000 FreeBSD Jails.

2001 Linux Vserver ermöglicht erste virtualisierung auf Betrtriebssystemebene durch Kernel Patching.

2004 Solaris Zones.

2007 Control Groups in den Linux Kernel integriert.

Geschichte

1979 UNIX v7: chroot system call, später in BSD.

2000 FreeBSD Jails.

2001 Linux Vserver ermöglicht erste virtualisierung auf Betrtriebssystemebene durch Kernel Patching.

2004 Solaris Zones.

2007 Control Groups in den Linux Kernel integriert.

2008 LXC: Linux Userspace tooling für cgroups und namespaces.

Geschichte

2013

Docker

- Entwickler Tooling
- Daemon für Container Management
- Standartisierung
- Packaging in Images
- Docker Hub

–> Container werden für viele zugänglich und interessant für Entwickler.

Alternativen zu Docker

Docker ist weit nicht die einzige Container-Software auf Linux:

- LXD
- Rocket
- systemd nspawn
- Flatpak
- Snappy

Vielen Dank! Fragen?