Linux Container für den Alltagsgebrauch

Anian Ziegler

6.4.2019

Augsburger Linux-Infotag

Einstieg: Was sind Container?

Was sind Container?

- Basieren auf Features des Linux Kernels

Was sind Container?

- Basieren auf Features des Linux Kernels
- Isolieren und verpacken Anwendungen

Was sind Container?

- Basieren auf Features des Linux Kernels
- Isolieren und verpacken Anwendungen
- Werden meistens mit Docker verwaltet

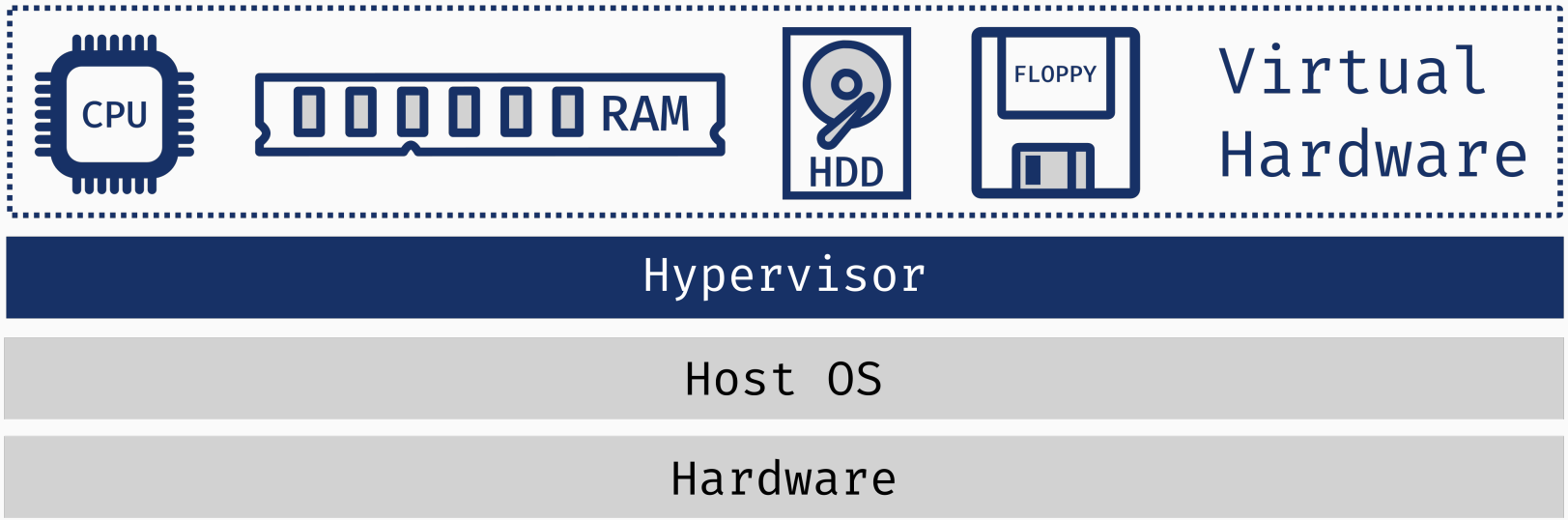
Was sind Container?

- Basieren auf Features des Linux Kernels
- Isolieren und verpacken Anwendungen
- Werden meistens mit Docker verwaltet
- Virtualisierung von Betriebssystem-Features

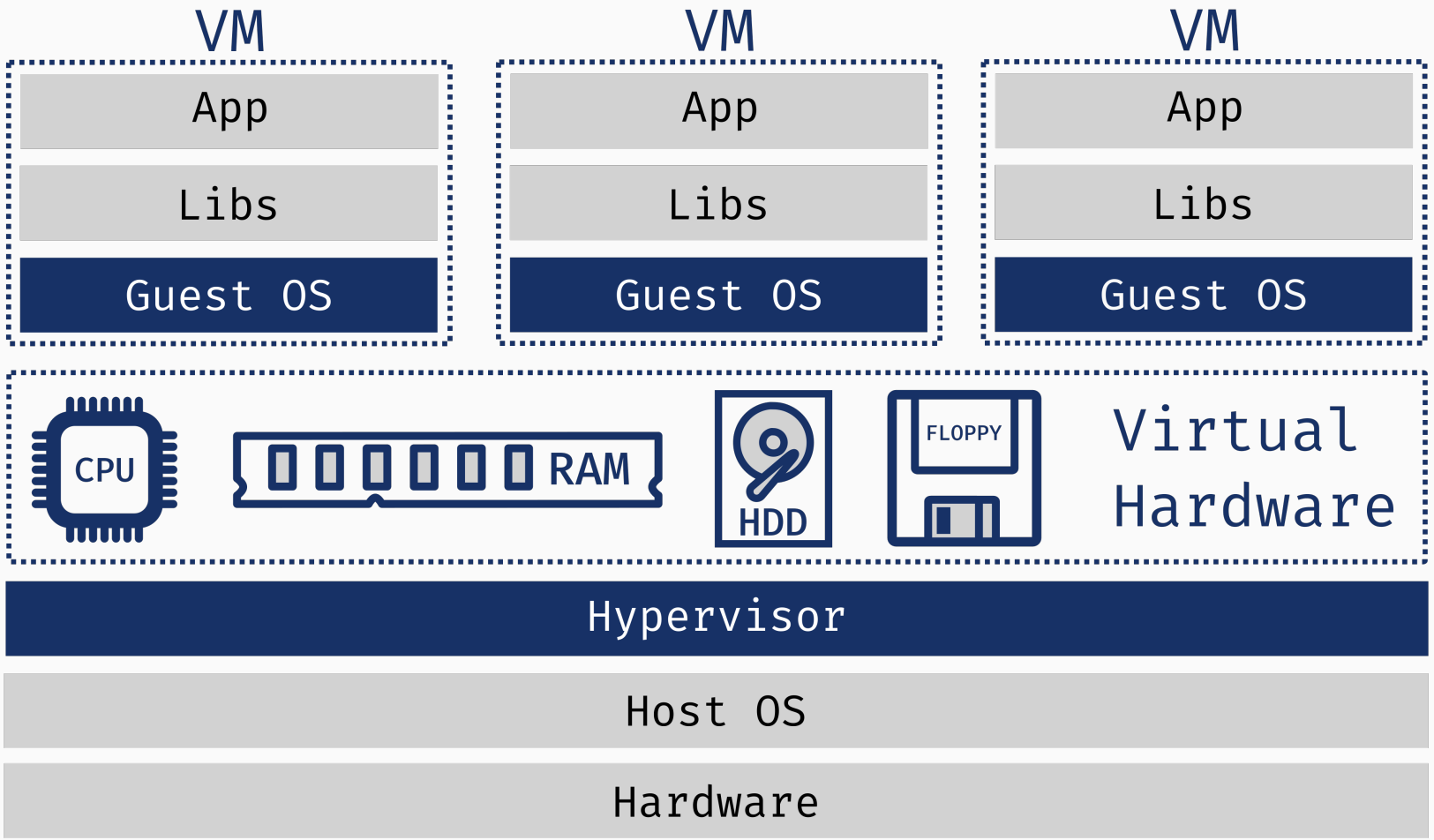
Was sind Container?

Container sind keine Hardware-Virtualisierung und kein Emulator. Alles passiert in einem einzelnen Kernel.

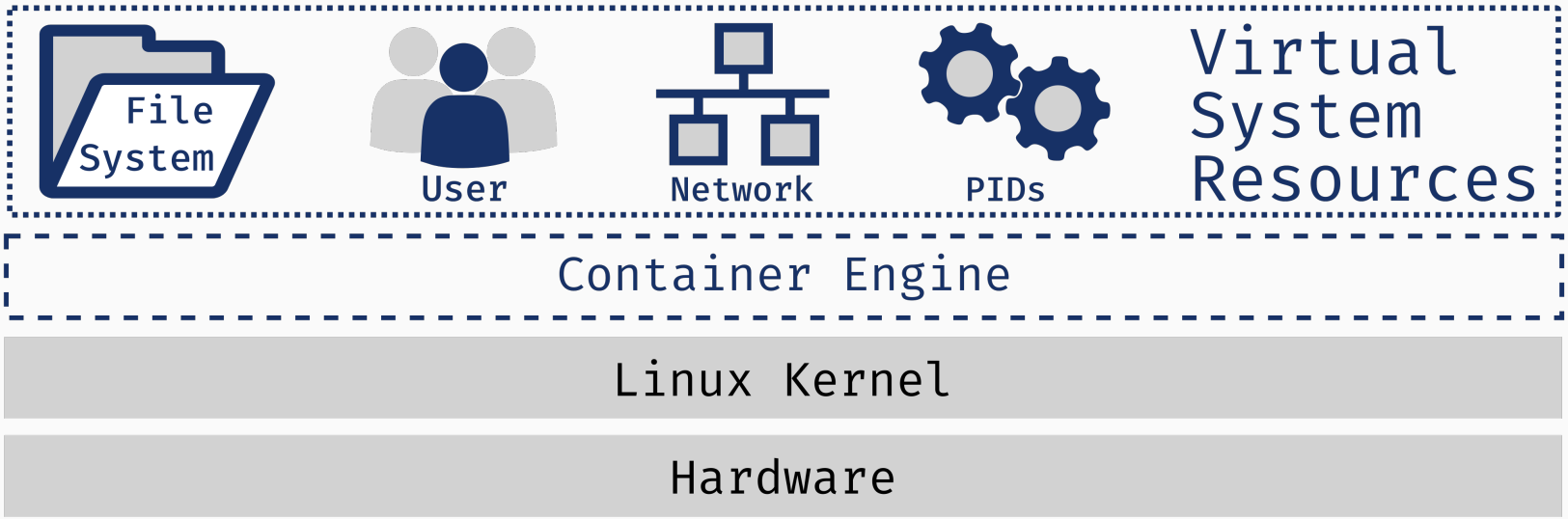
Exkurs: Hardware-Virtualisierung



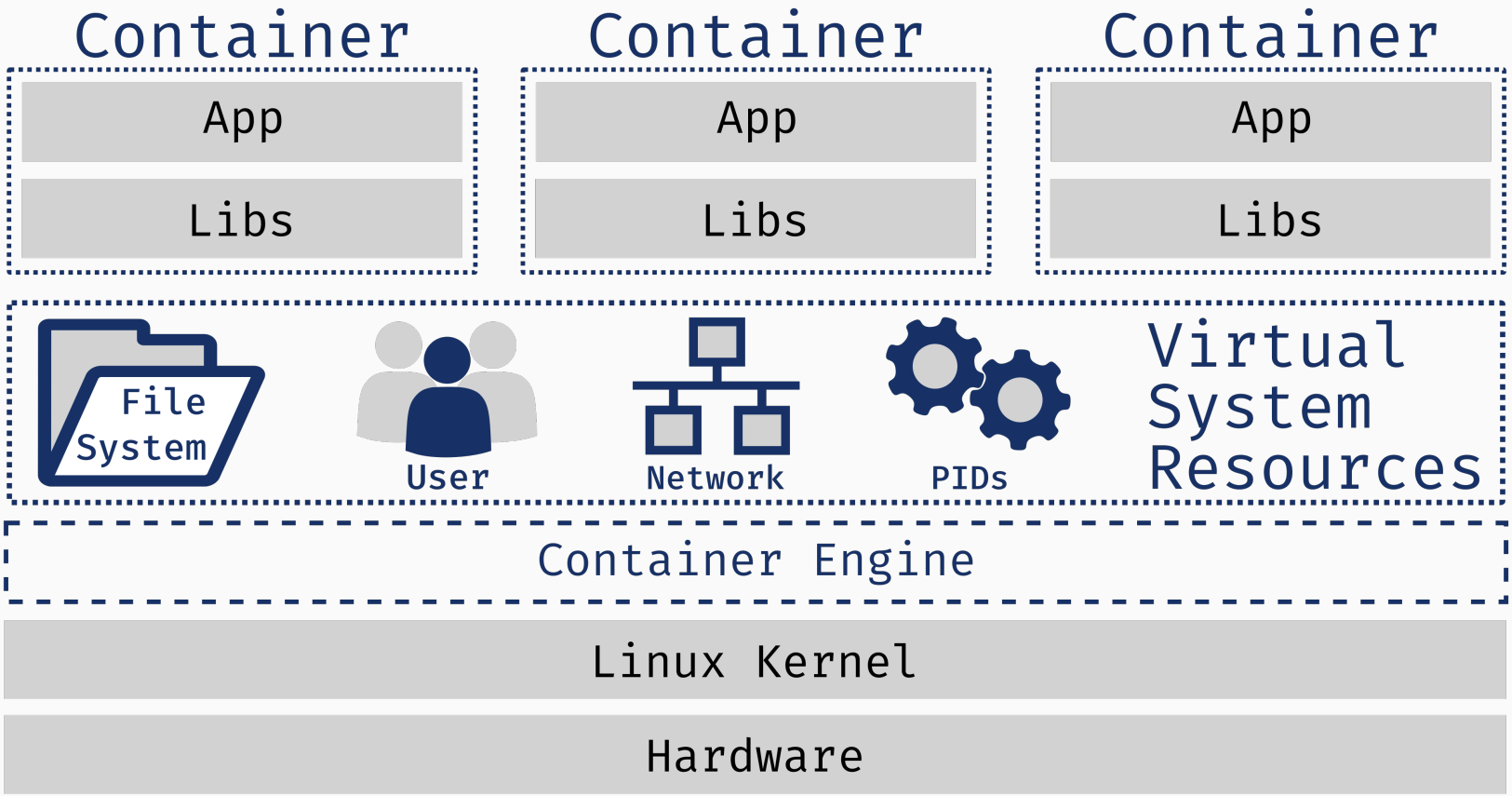
Exkurs: Hardware-Virtualisierung



Virtualisierung auf Betriebssystemebene

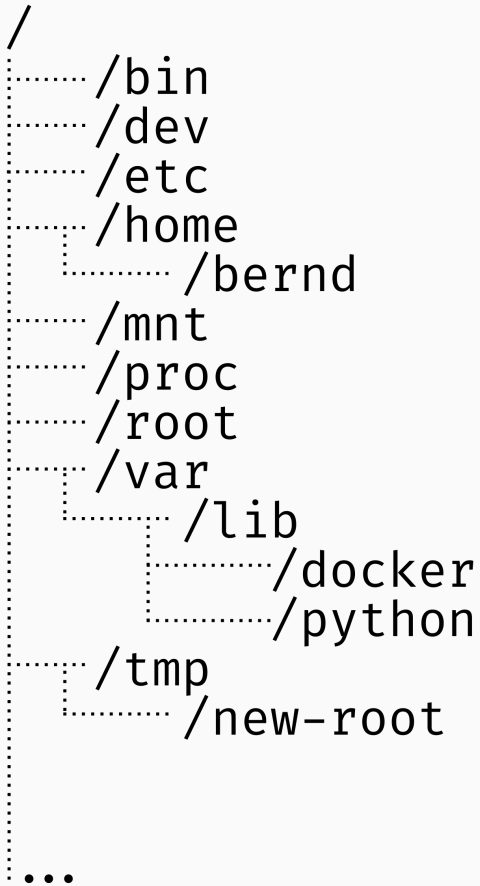


Virtualisierung auf Betriebssystemebene

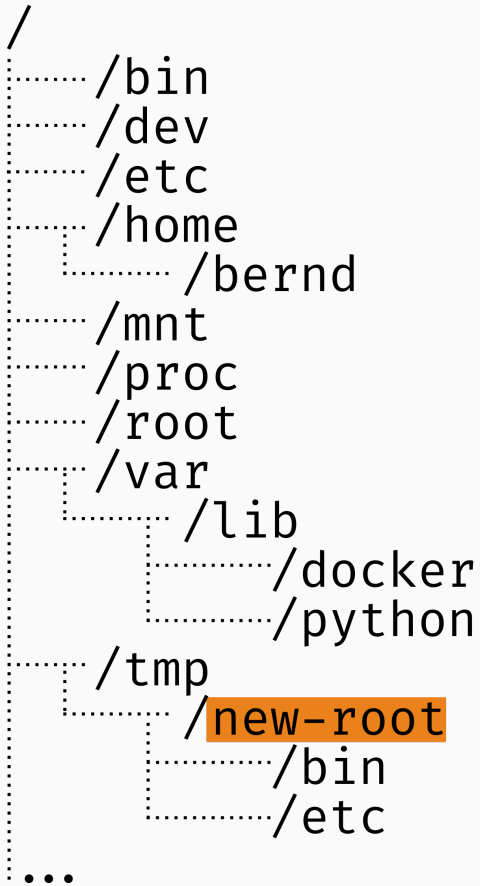


Implementierung in Linux

CHROOT


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

CHROOT


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

Demo

CHROOT

- Erlaubt es das root Verzeichnis / neu zu setzen
- Sehr Hilfreich zum Debugging, compiling oder bei der Installation von Linux
- An sich schon sehr nützlich
- **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. Beispiel: Auf seinem eigenen Rechner mit Network-Namespaces ein Netzwerk simulieren

Demo

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

Demo

Container

Container nennt man die Kombination all dieser Funktionen

Im Alltagsgebrauch?

Was kann man im Alltag damit machen?

- Wegwerf-Umgebung für Tests

Im Alltagsgebrauch?

Was kann man im Alltag damit machen?

- Wegwerf-Umgebung für Tests
- Isolierung einzelner Prozesse für mehr Sicherheit

Im Alltagsgebrauch?

Was kann man im Alltag damit machen?

- Wegwerf-Umgebung für Tests
- Isolierung einzelner Prozesse für mehr Sicherheit
- Netzwerktools auf eigenem PC testen

Im Alltagsgebrauch?

Was kann man im Alltag damit machen?

- Wegwerf-Umgebung für Tests
- Isolierung einzelner Prozesse für mehr Sicherheit
- Netzwerktools auf eigenem PC testen
- CPU oder RAM hungrige Prozesse auf dem Laptop in die Schranken weisen

Aber was ist jetzt Docker?

Docker?

Docker bündelt all diese Funktionen mit einfachen Werkzeugen für Entwickler und Sysadmins.

Docker Images

- Im Prinzip Tar-Archive von Dateisystemen + Metadaten

Docker Images

- Im Prinzip Tar-Archive von Dateisystemen + Metadaten
- Große Auswahl an fertigen Images im Docker Hub

Docker Images

- Im Prinzip Tar-Archive von Dateisystemen + Metadaten
- Große Auswahl an fertigen Images im Docker Hub
- Können einfach selbst mit sog. Dockerfiles gebaut werden

Docker Images

- Im Prinzip Tar-Archive von Dateisystemen + Metadaten
- Große Auswahl an fertigen Images im Docker Hub
- Können einfach selbst mit sog. Dockerfiles gebaut werden
- Sehr portabel und erzeugen reproduziebare Ergebnisse

Docker Daemon

- Service im Hintergrund

Docker Daemon

- Service im Hintergrund
- Started und managed container

Docker Daemon

- Service im Hintergrund
- Started und managed container
- Baut images

Docker Daemon

- Service im Hintergrund
- Started und managed container
- Baut images
- REST API via UNIX Socket

Docker CLI

- Interagiert mit dem Daemon

Docker CLI

- Interagiert mit dem Daemon
- Standart-Interface für Docker Entwickler und Sysadmins

Docker CLI

- Interagiert mit dem Daemon
- Standart-Interface für Docker Entwickler und Sysadmins
- Werkzeug zum bauen, starten und überwachen von containern

Demo

Docker Compose

- Zusätzliches Kommandozeilen Werkzeug zum Verwalten mehrer Container
- Konfiguration in leicht verständlichen yml Dateien
- Deklarativ: Beschreibung des Ergebnis, nicht der Schritte dahin

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

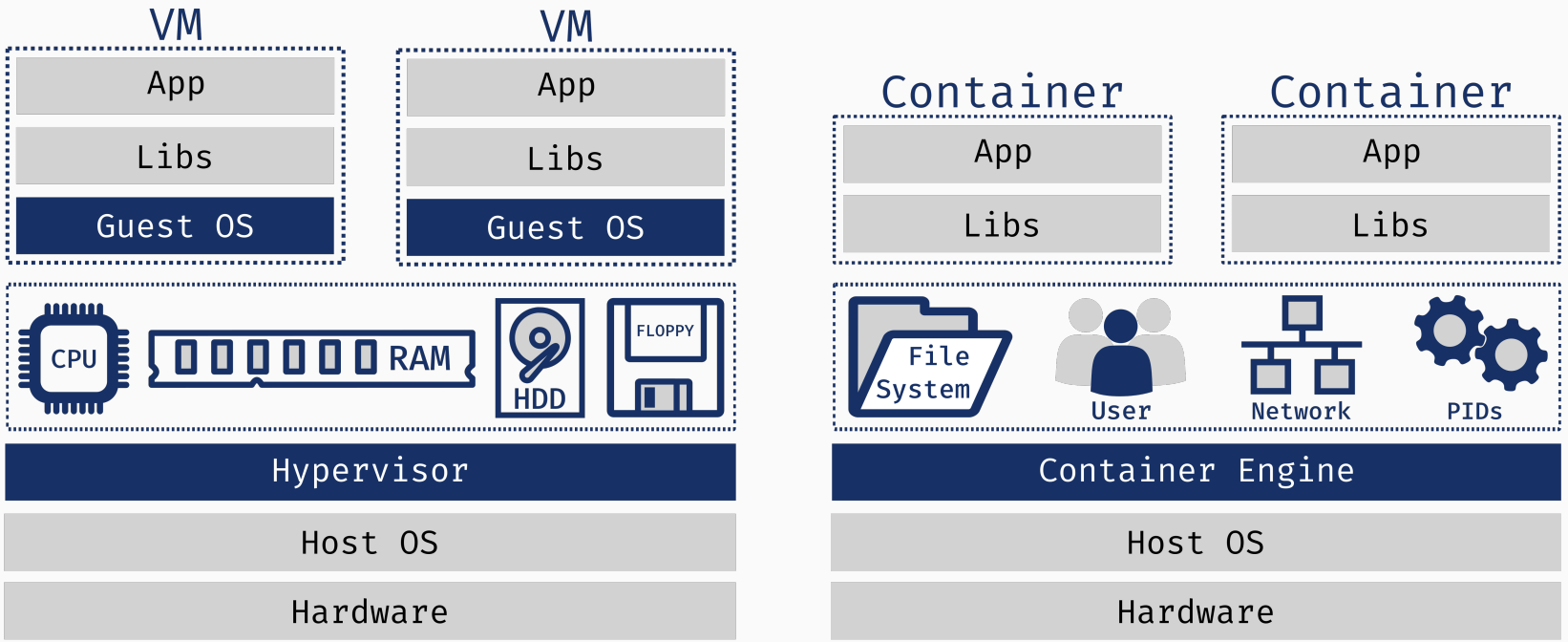
- **Konsolidierung** mehrerer Anwendungen ohne ineffiziente VMs
- Weg aus der **Dependency Hell**
- Portable **Development Environments**. Verhindert ”Works on My Machine”weil alle die gleiche Version haben
- **Isolierung** unsicherer Prozesse von einander

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

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
- Alles auf einem Linux Kernel

cioplenu.de

@AnianZ

Wir suchen Entwickler und Sysadmins!

Vielen Dank! Fragen?

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: erste OS-Virtualisierung als Kernel Patch.

Geschichte


1979
UNIX v7: chroot system call, später in BSD.
2000
FreeBSD Jails.
2001
Linux Vserver: erste OS-Virtualisierung als Kernel Patch.
2004
Solaris Zones.

Geschichte

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

2000 — FreeBSD Jails.

2001 — Linux Vserver: erste OS-Virtualisierung als Kernel Patch.

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: erste OS-Virtualisierung als Kernel Patch.

2004 Solaris Zones.

2007 Control Groups in den Linux Kernel integriert.

2008 LXC: Linux 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