

Kapitel 3: Virtualisierung



Vorlesung
**CLOUD
COMPUTING**

Anforderungen an Cloud Computing

- Organisation der Ressourcen in einem Rechenzentrum
 - Effiziente Nutzung der gegebenen Ressourcen zur Minimierung der Kosten
 - Isolation der Ressourcen. Kunden sollen andere nicht sehen und auch nicht von ihnen beeinflusst werden. Seiteneffekte vermeiden. Security
 - Entkopplung von der Hardware für mehr Flexibilität im Betrieb und Robustheit bei Ausfällen
 - Ressourcen sollen flexibel vergeben werden. Steuerung mittels Software defined resources
- Lösung für diese Anforderungen ist Virtualisierung und macht Cloud Computing erst möglich.

Virtualisierungsarten

Virtualisierung ist stellvertretend für mehrere grundsätzlich verschiedene Konzepte und Technologien:

- Virtualisierung von Hardware-Infrastruktur
 - Emulation
 - Voll-Virtualisierung
 - Para-Virtualisierung
- Virtualisierung von Software-Infrastruktur
 - Betriebssystem-Virtualisierung (*Containerization*)
 - Anwendungs-Virtualisierung (*Runtime*)



Virtualisierungsarten: Hardwarevirtualisierung

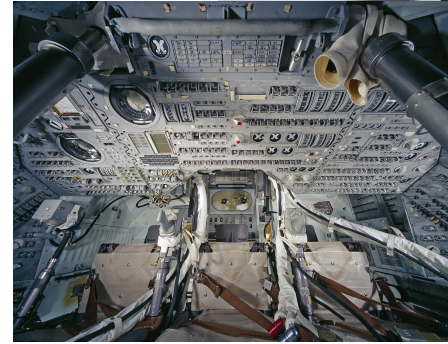
Was wird virtualisiert?

Hardwarevirtualisierung arbeitet auf Ebene der Rechnerarchitektur.

- **Prozessor**
 - Der State des Prozessors. Im wesentlichen Prozessorregister.
 - Maschinencode
 - Memory Management Unit
- **Hauptspeicher**
 - Linear adressierter physikalischer Speicher
- **Netzwerk**
 - z. B. Input-Output Stream von Ethernet Frames
- **Storage**
 - Blockspeicher (linear adressiert. Lesen und Speichern von Blöcken)
- **Grafikkarte**
 - z. B. Framebuffer (2D-Array mit Pixeldaten)
 - 3D Funktionalität (DirectX, OpenGL), siehe https://en.wikipedia.org/wiki/GPU_virtualization
 - Computing (KI, Simulationen) (Zunehmend wichtig für die Cloud)
- Evtl. Peripherie wie USB, Maus, Keyboard
- Timer, Interrupt Controller

Was ist Emulation?

- Emulation: Bildet die Hardware eines nicht vorhandenen oder nicht kompatiblen Rechnersystems oder Teile eines entsprechenden Rechnersystems nach
- Emulationen sind der Regel sehr langsam und nicht parallelisierbar
- Anwendungen
 - Alte Software konservieren. Zum Beispiel alte Spielekonsolen oder Apollo Guidance Computer: <https://svtsim.com/moonjs/agc.html>
 - Embedded Entwicklung ohne echte Hardware
 - Reine CPU Emulation
 - Rosetta von Apple. CPU Instruktionsübersetzung von Power-PC zu x86 mittels dynamic binary translation, also Just in Time Übersetzung. (2004). Gerade wird Rosetta 2 entwickelt für die Intel -> ARM Transition
 - <https://www.heise.de/news/Windows-on-ARM-Testphase-fuer-x64-Emulation-startet-im-November-2020-4918453.html>
 - QEMU User Mode Emulation
 - Voll-Virtualisierer (Hardware und spezielle Instruktionen der CPU)
 - QEMU und Bochs können Windows und Linux praktisch überall starten.

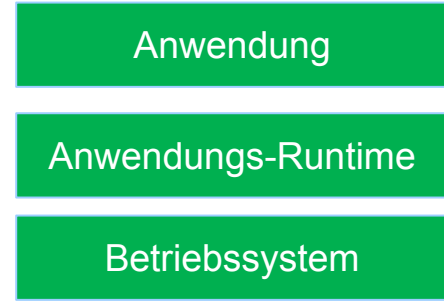


Windows 95 auf der Apple Watch

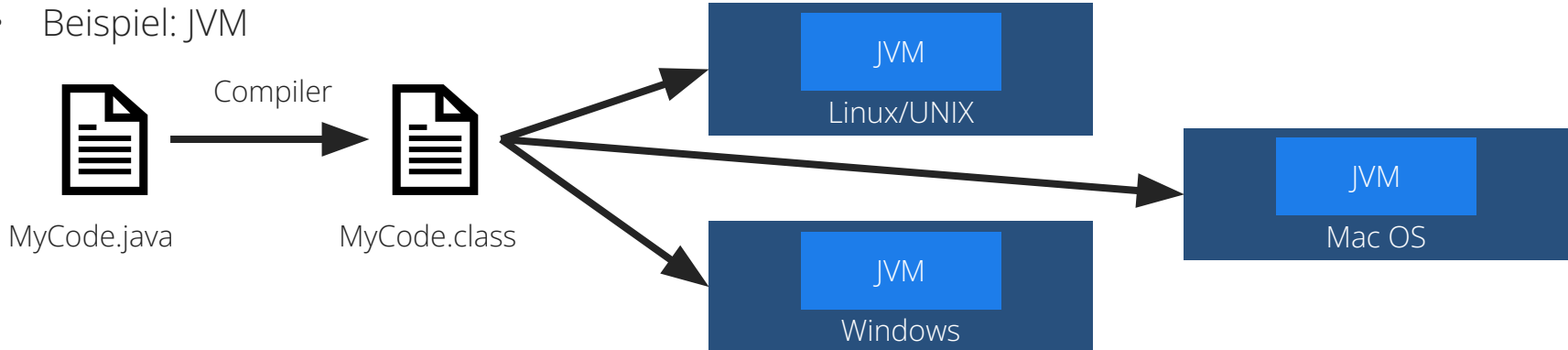


Zur Vollständigkeit: Was ist Anwendungs-Virtualisierung?

- Anwendungs-Virtualisierung: Stellt Anwendungen eine Programmierschnittstelle und eine Laufzeitumgebung (Runtime) zur Verfügung, die komplett vom darunter-liegenden Betriebssystem entkoppelt.
Zweck u.A.: Portable Anwendungen.



- Beispiel: JVM



Virtualisierung, aber performant

- Emulationen erfüllen zwar viele Voraussetzungen für das Cloud Computing wie Isolation und Entkopplung, sind aber bei der Nachbildung einer ganzen Rechnerarchitektur sehr langsam und daher ungeeignet für den massenhaften produktiven Einsatz.
 - Hauptverantwortlicher dabei ist die CPU.
- Kann man die selben Ziele mit minimalem zusätzlichem Ressourcenaufwand erreichen?
 - Antwort Ja, aber nur wenn die Gast-Rechnerarchitektur des virtualisierten Systems die gleiche ist wie die Host-Rechnerarchitektur.
 - x86 Host → x86 Guest

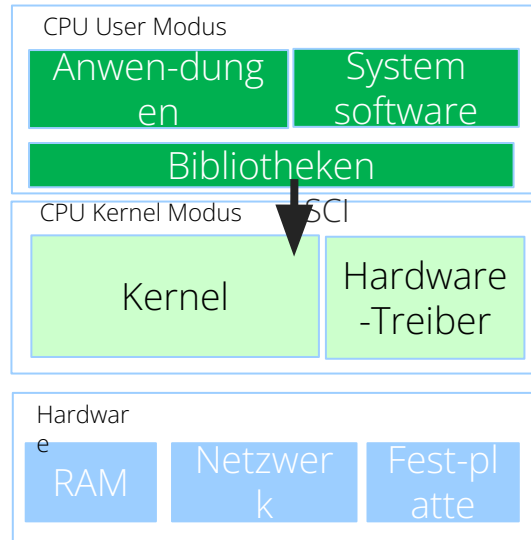
Klassischer Aufbau eines Betriebssystems mit Unterstützung der Rechnerarchitektur

CPU User Mode

- Niedrigste Berechtigungsstufe
 - Keine direkten Hardwarezugriffe
 - Speicherschutz über die Memory Management Unit

CPU Kernel Mode

- User Mode ruft Kernel über das System Call Interface (SCI) auf. Aktuell besteht das SCI bei Linux aus ca. 380 System Calls.
- Höchste Berechtigungsstufe
 - Privileged CPU Instruktionen
 - Zugriff auf Hardware mittels Treiber
- Übernimmt z. B. Dateisystemverwaltung und Scheduling der Anwendungen

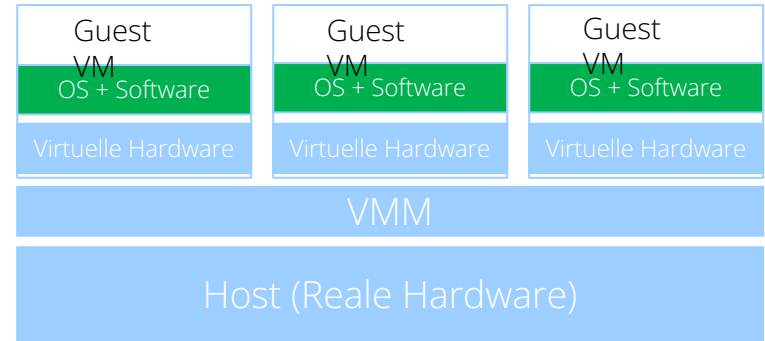


Unterstützung durch die Hardware

- Software based virtualization
 - In dem klassischen Betriebsmodell hat nur der Usermodus die notwendigen Isolationseigenschaften. Der Kernelmodus kann hier emuliert werden. z. B. mittels "trap-and-emulate". mit wenigstens 10% Performanceverlust.
- Hardware assisted virtualization
 - Oftmals wurden daher CPU Extensions entwickelt wie Intel-VT und AMD-V. Diese fügen einen neuen Prozessormodus (z. B. virtual execution mode) hinzu, bei dem sich das Gastbetriebssystem als mit vollen Privilegien arbeitend wahrnimmt, das Hostbetriebssystem jedoch geschützt bleibt
 - Virtuelle Hauptspeicher-Partition im echten physikalischen Speicher. (Die Null verschiebt sich). Management der realen Repräsentation mittels der Management Memory Unit (MMU).
 - Für die Durchreichung (Pass-Through) der Schnittstellen von echten Hardwaregeräten muss die Verschiebung der Null durch eine IOMMU (I/O Memory Management Unit) ausgeglichen werden.

Hardware-Virtualisierung: Begrifflichkeiten

- Durch Hardware-Virtualisierung werden die Ressourcen eines Rechnersystems aufgeteilt und von mehreren unabhängigen Betriebssystem-Instanzen genutzt.
- Anforderungen der Gastinstanzen werden von der Virtualisierungs-software (Virtual Machine Monitor, VMM) abgefangen und auf die real vorhandene Hardware umgesetzt.
- Der VMM (oder Hypervisor) verteilt die Hardwareressourcen des Rechners an die VMs
- Es werden aber 2 Virtualisierungsmodi und 2 Arten von Hypervisor



Host

- Der Rechner der eine oder mehrere virtuelle Maschinen ausführt und die dafür notwendigen Hardware-Ressourcen zur Verfügung stellt.

Guest

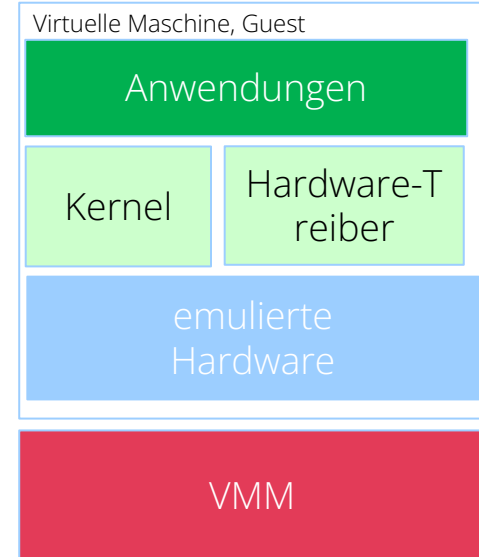
- Eine lauffähige / laufende virtuelle Maschine

VMM (Virtual Machine Monitor, auch Hypervisor genannt)

- Die Steuerungssoftware zur Verwaltung der Guests und der Host-Ressourcen

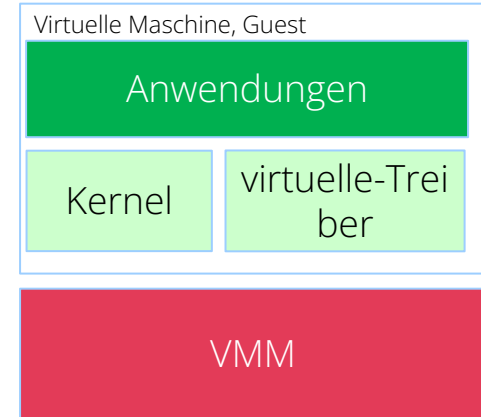
Voll-Virtualisierung

- Jedem Gastbetriebssystem steht ein eigener virtueller Rechner mit virtuellen Ressourcen wie CPU, Hauptspeicher, Laufwerken, Netzwerkkarten, usw. zur Verfügung.
- Das Gastbetriebssystem muss also nicht angepasst werden. Zum Zeitpunkt des Starts muss das Gastbetriebssystem nicht bekannt sein.
- Die VMM emuliert auch weiterhin echte Hardware wie Storage (SATA) und Netzwerk (Ethernet).
- Die VMM kann aber zur Beschleunigung oder zur besseren Nutzung (Grafik, Mouse) spezielle virtuelle Hardware zur Verfügung stellen.
 - z. B. Einfacher Pass-Through von USB
 - Fließender Übergang zur Paravirtualisierung
- Leistungsverlust: 1 - 5%



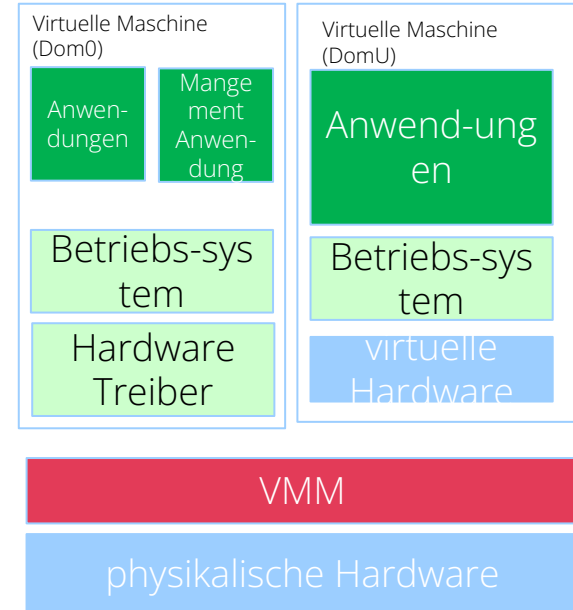
Para-Virtualisierung

- Dem Gast-Betriebssystem stehen keine direkt low-level virtualisierten Hardware-Ressourcen zur Verfügung sondern eine API.
 - Vereinfacht den Aufbau der VM
- Das Gast-Betriebssystem muss portiert werden.
 - Low Level Prozessorinstruktionen werden erst gar nicht ausgeführt oder durch API Aufrufe abgebildet
 - Virtuelle Treiber (z. B. virtio).
 - Vermeidung von Umformungen und Kopieraktionen durch Verwendung spezieller Treiber
 - Übertragung von IP Paketen und nicht von Ethernet Frames.
- Unterstützte Betriebssysteme und Hardware-Varianten aus Sicht des Gastes eingeschränkt pro Hypervisor-Implementierung.
- Leistungsverlust: 0 - 2%



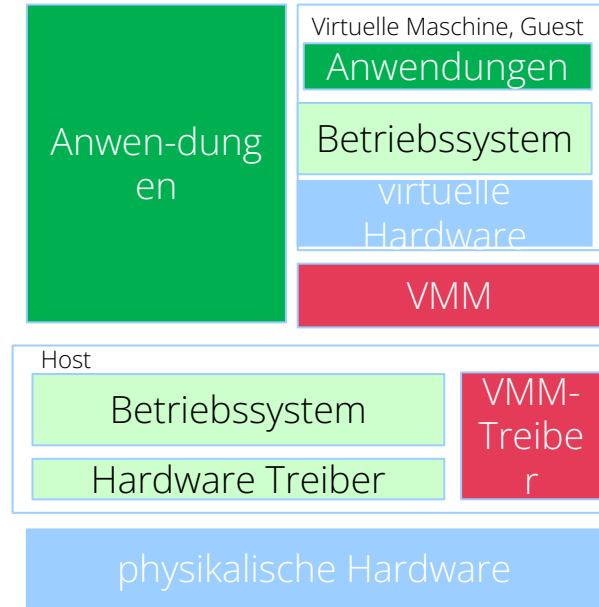
Typ 1: Bare-Metal Virtualisierung

- Der Hypervisor läuft direkt auf der verfügbaren Hardware. Er entspricht somit einem Betriebssystem, das ausschließlich auf Virtualisierung ausgerichtet ist.
- Der Hypervisor nutzt üblicherweise die Treiber eines Host-Betriebssystems, um auf die reale Hardware zuzugreifen. Damit brauchen im Hypervisor nicht aufwändig eigene Treiber implementiert werden.
- Vornehmlich mit Paravirtualisierung, da hier ebenfalls die Emulation der Hardware entfällt
- Ermöglicht einfacheren Pass-Through von echter Hardware. z. B. GPUs an einen Gast
- Beispiele: VMWare ESXi, Microsoft Hyper-V, XEN



Typ 2: Host Virtualisierung

- Der VMM läuft hosted als Anwendung unter dem Host-Betriebssystem
- Vornehmlich bei Voll-Virtualisierung verwendet
- Geringere Skalierbarkeit wegen Abhängigkeit zum Host System
- Mehr Overhead als Typ 1
- Beispiele:
 - Virtualbox
 - VMWare Workstation Player
 - Parallels
 - Achtung: Die Unterscheidung zwischen Typ 1 und Typ 2 ist in vielen Fällen nicht immer klar.



Virtualisierung im Enterprise Umfeld

Neben den bisher genannten Vorteilen bieten heutige VM Lösungen noch viele weitere Features

- Ressourcenverwaltung im laufenden Betrieb
 - Memory Ballooning – Hauptspeicher dynamisch vergrößern
 - Änderung der Anzahl an virtuellen Rechenkernen
 - Änderung der Festplattengröße mit virtuellen SANs (Storage Area Networks)
- Live Migration
 - Verschieben der laufenden physikalischen Maschine auf eine andere Hardware innerhalb von Millisekunden
 - CPU State
 - Speicher
 - Storage
 - Netzwerk
- (Echten) Zufall zu erzeugen ist in einer VM noch schwieriger als mit realer Hardware (z. B. mittels Maus und Tastatureingaben). Hier bieten die Hypervisors Schnittstellen an um zusätzlichen Zufall zu erhalten.



Hardware-Virtualisierung mit Vagrant und VirtualBox

Hardware-Virtualisierung: Vagrant und VirtualBox



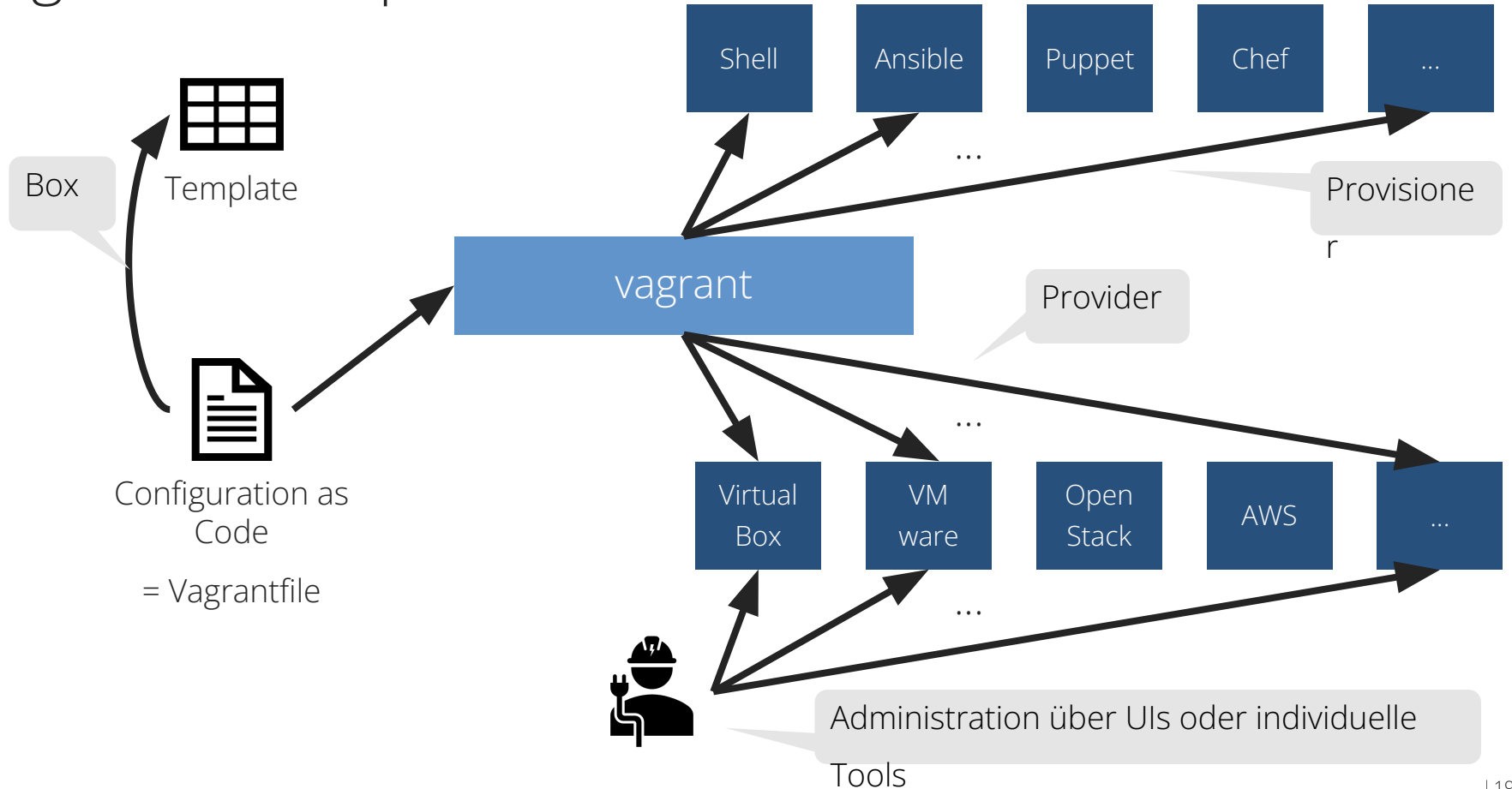
+



Open Source Typ 2 Virtualisierungs-Software (Voll-Virtualisierung) für Windows, Linux, MacOS und Solaris.

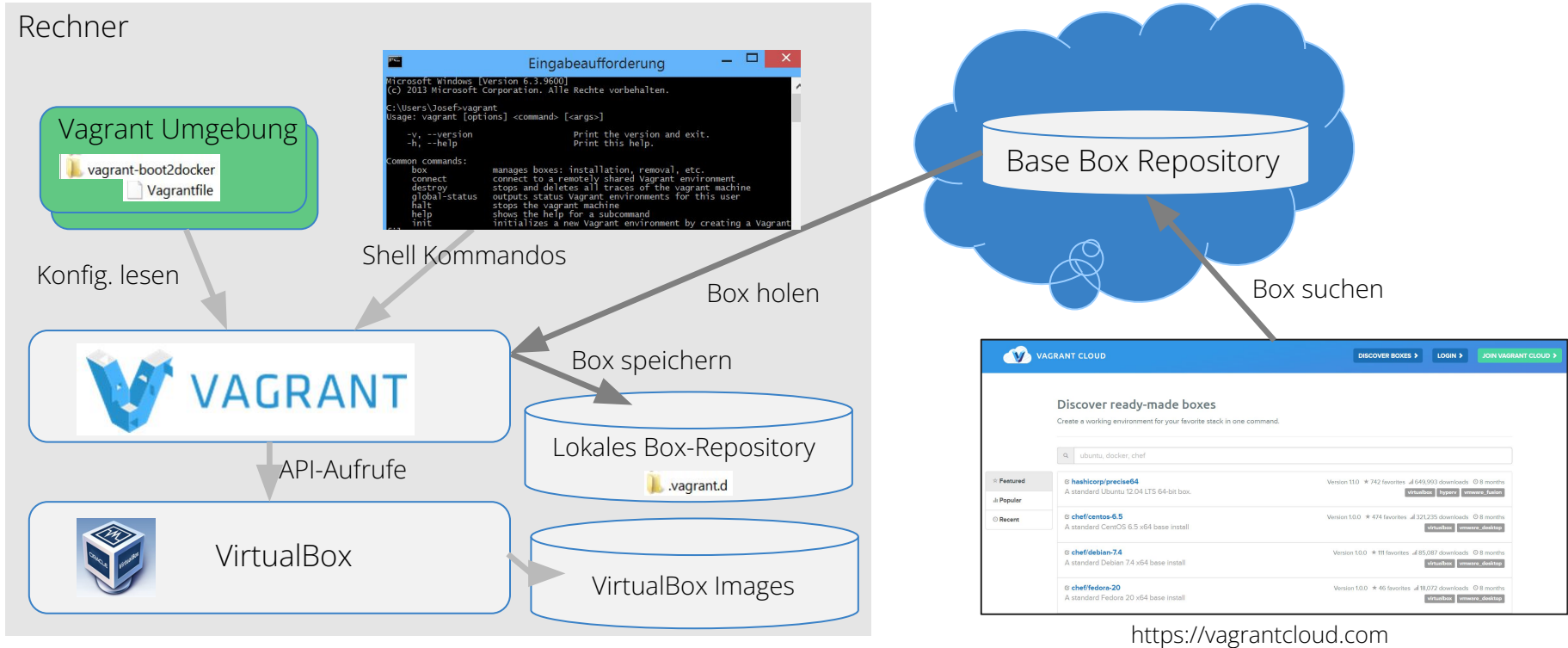
Automationssoftware für virtuelle Umgebungen auf einem Rechner. Virtuelle Maschinen per Kommandozeile erstellen und steuern.

Vagrant: Konzepte



Demo

Vagrant: Eine schematische Übersicht.



Das Vagrantfile beschreibt die zu erstellende virtuelle Maschine.

```
# -*- mode: ruby -*-  
# vi: set ft=ruby :
```

Vagrantfiles werden in Ruby geschrieben

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!  
VAGRANTFILE_API_VERSION = "2"
```

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
```

```
  # My base box  
  config.vm.box = "chef/ubuntu-14.04"
```

Definition der Basis-Box

```
  # Define shell provisioning  
  config.vm.provision :shell, path: "bootstrap.sh"
```

Konfiguration der Provisionierung

```
  # Define docker provisioning  
  config.vm.provision "docker" do |d|  
    d.run "nginx1", image: "dockerfile/nginx", args: "-p 8080:80", daemonize: true  
    d.run "nginx2", image: "dockerfile/nginx", args: "-p 9080:80", daemonize: true  
    d.run "haproxy", image: "dockerfile/haproxy", args: "-p 80:80 --link nginx1:nginx1 --link nginx2:nginx2 -v /vagrant:/haproxy-override"  
  end
```

```
  # Configure VirtualBox  
  config.vm.provider "virtualbox" do |v|  
    v.memory = 1024  
    v.cpus = 4  
  end
```

Konfiguration des Virtualisierungs-Providers

```
  # Forward ports  
  config.vm.network :forwarded_port, host: 80, guest: 80  
  config.vm.network :forwarded_port, host: 8080, guest: 8080  
  config.vm.network :forwarded_port, host: 9080, guest: 9080
```

Konfiguration des Netzwerks

```
end
```

Ein typischer Arbeitsablauf mit Vagrant.

#	Befehle auf Kommandozeile	Bedeutung
1	<code>mkdir <box-dir></code> <code>cd <box-dir></code>	Verzeichnis für Vagrant Umgebung erstellen und dorthin wechseln
2	<code>vagrant init [<box-name>] [<box-url>]</code>	Eine Vagrant Umgebung initialisieren. Dabei wird zunächst nur eine Datei <i>Vagrantfile</i> erstellt und initial mit dem Namen und der URL der Box (falls angegeben) initialisiert.
3		Vagrantfile anpassen nach Bedarf (z.B. IP vergeben, Port-Mapping zwischen Host und Guest, Verzeichnis-Share zwischen Host und Guest, ...)
4	<code>vagrant up</code>	Startet die virtuelle Maschine (Box → virtuelle Maschine) und konfiguriert sie entsprechend dem Vagrantfile
5	<code>vagrant ssh</code>	Per SSH auf die virtuelle Maschine verbinden
6	<code>exit</code>	Die SSH Kommandozeile in der virtuellen Maschine verlassen
7	<code>vagrant halt</code>	Die virtuelle Maschine stoppen

Weitere nützliche Kommandos:

- `reload`: Startet eine VM neu und aktualisiert die Konfiguration entsprechend dem Vagrantfile
- `package`: Erstellt aus einer virtuellen Maschine wieder eine Box

Weitere Kommandos: <http://docs.vagrantup.com/v2/cli/index.html>

Vagrant Befehle auf Kommandozeile

- vagrant box add – allows you to install a box (or VM) to the local machine
- vagrant box remove – removes a box from the local machine
- vagrant box list – lists the locally installed Vagrant boxes
- vagrant init – initializes a project to use Vagrant
- vagrant up – starts up the vagrant VM
- vagrant suspend – saves the state of the current VM.
- vagrant resume – will load up the suspended VM.
- vagrant halt – will shut down the VM, saving configuration. (restart with 'up' command)
- vagrant destroy – will destroy the VM with all config changes.
- vagrant reload – apply Vagrant configuration changes (like port forwarding) without rebuilding the VM.
- vagrant status – tells you the current state of the Vagrant project's VM
- vagrant gem – install Vagrant plugins via RubyGems
- vagrant ssh – short cut to SSH into the running VM
- vagrant package – create a distribution of the VM you have running.
- vagrant <command> -help - Command that will provide man pages for a vagrant command.



Virtualisierungsarten: Betriebssystemvirtualisierung

Hardwarevirtualisierer sind Schwergewichte

- Jede VM inkludiert eine virtuelle Kopie eines kompletten Betriebssystems und benötigt signifikante RAM und CPU Ressourcen, die nur schwer dynamisch geändert werden können
- Softwareentwicklung mit VMs ist träger und komplexer
- Aufgrund der Größe der Images ist die Portabilität ein Problem.
- Kompatibilität mit anderen VM Lösungen nicht vorhanden. Wechsel zwischen Rechenzentren nicht einfach möglich.

Demo

Der Urgroßvater: chroot (Jahr 1982)

- chroot gilt als Urgroßvater der Betriebssystemvirtualisierung (Jahr 1982)
- chroot setzt aus Sicht der laufenden Applikation das root Filesystem neu
 - Ermöglicht die Isolation des Filesystems
- Kein Overhead. Implementiert in 2 dutzend Zeilen C-Code im Kernel
- Benötigt root-Rechte
- Keine Netzwerk-Isolation, keine Prozess-Isolation, keine Disk Quotas, keine CPU Quotas, keine I/O Limitierung
- chroot Prozess sieht weiterhin fast alles vom System

Linux Kernel Namespaces (Jahr 2002)

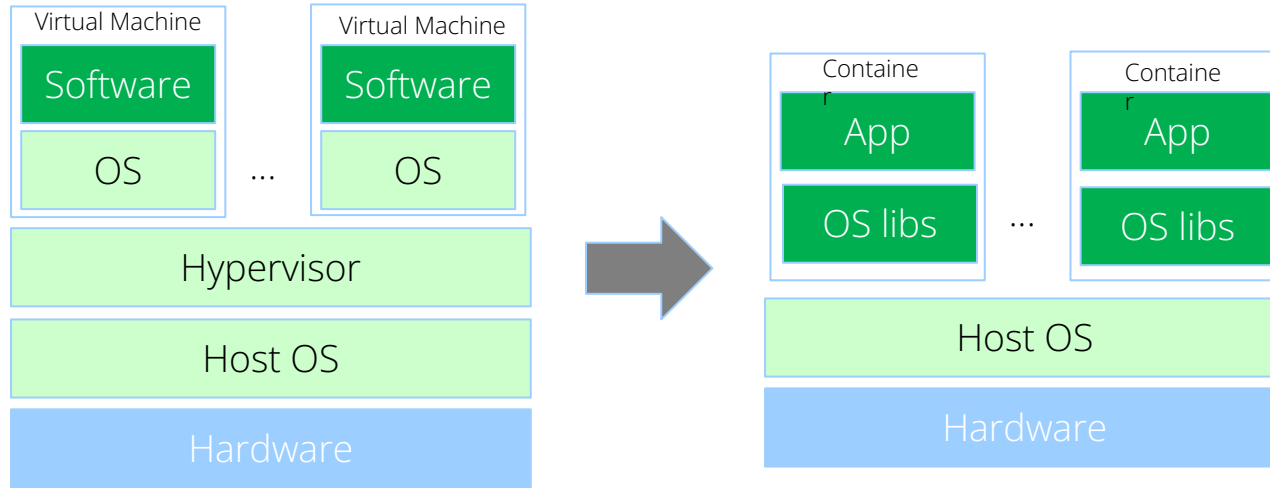
- Isolation durch Sichtbarkeit
- Ein Feature des Linux-Kernels, das die Sicht und den Zugriff auf das System einschränkt:
 - Prozessraum / Prozess-Ids
 - Netzwerk-Schnittstellen
 - Host-Name
 - Dateisystem-Mounts
 - IPC (Inter-Prozess-Kommunikation)
 - Benutzerkonten
 - Zeit
- Die Einschränkungen sind dabei für den isolierten Prozess transparent.
- Namespaces können geschachtelt sein.
- siehe [https://success.docker.com/KBase/Introduction to User Namespaces in Docker Engine](https://success.docker.com/KBase/Introduction%20to%20User%20Namespaces%20in%20Docker%20Engine)

Linux cgroups (Jahr 2007)

- Isolation durch Grenzen
- Ein Feature des Linux-Kernels, das maßgeblich durch Google entwickelt wurde
- Gruppiert Prozesse zu Gemeinschaften mit definiertem und beschränktem Ressourcen-Zugriff auf:
 - Prozessor
 - Hauptspeicher
 - I/O (insb. Netzwerk)
 - Disk
- Die Prozess-Gruppen können geschachtelt sein.
- cgroups stellen dabei für die Prozessgruppen sicher, dass
 - die Ressourcen limitiert sind und die definierten Grenzen nicht überschritten werden
 - die aktuell verbrauchten Ressourcen kontinuierlich gemessen und protokolliert werden
 - dass bei Überschreitung der definierten Grenzen die Prozess-Gruppen eingefroren und neu gestartet werden

• Siehe <https://docs.docker.com/engine/docker-overview/>

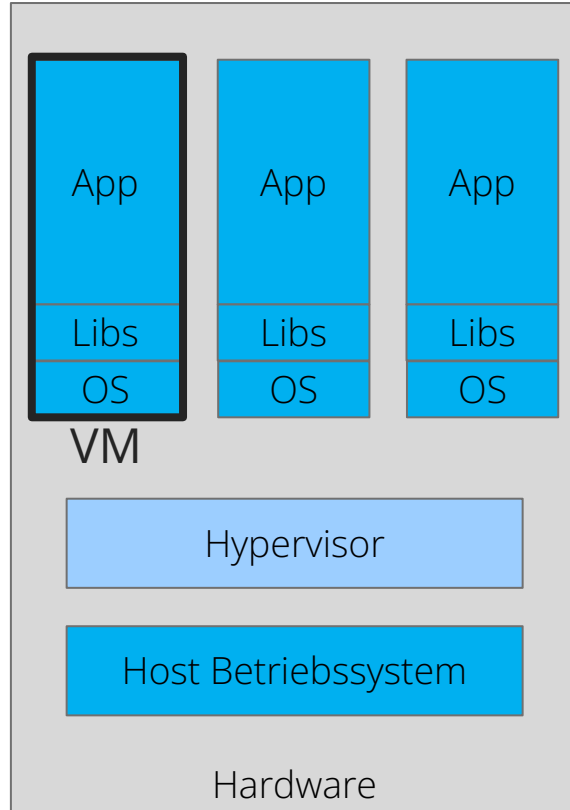
Betriebssystem-Virtualisierung



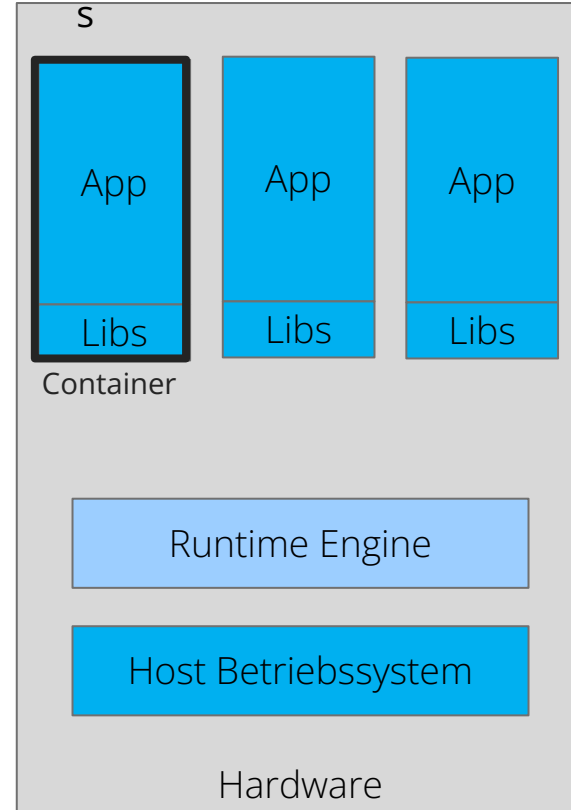
- Leichtgewichtiger Virtualisierungsansatz: Es gibt keinen Hypervisor. Jede App läuft direkt als Prozess im Host-Betriebssystem. Dieser ist jedoch maximal durch entsprechende OS-Mechanismen isoliert (z.B. Linux LXC).
 - Isolation des Prozesses durch Kernel Namespaces (bzgl. CPU, RAM und Disk I/O) und Containments
 - Isoliertes Dateisystem
 - Eigene Netzwerk-Schnittstelle
- CPU- / RAM-Overhead in der Regel nicht messbar (~ 0%)
- Startup-Zeit = Startdauer für den ersten Prozess

Virtual Machines vs. Containers

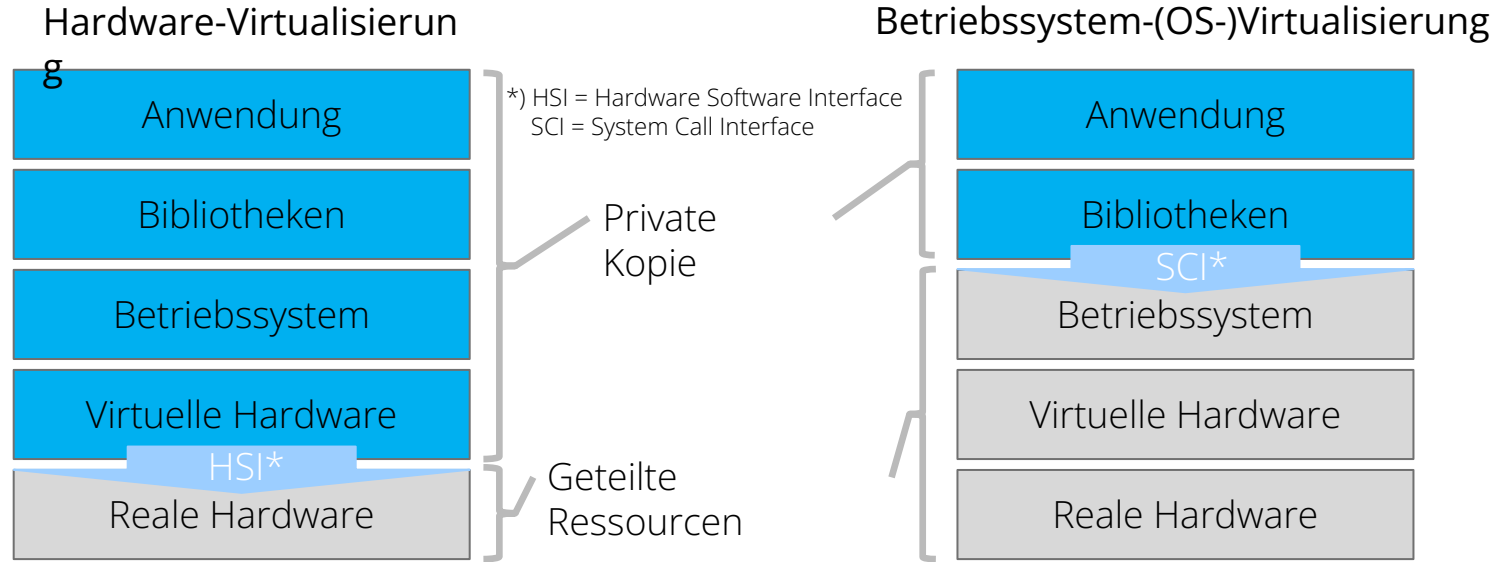
Virtual Machines



Container



Hardware- vs. Betriebssystem-Virtualisierung



- Benötigt Hardwareunterstützung
- Höhere Sicherheit. Die HSIs sind einfach.
- Stärkere Isolation.
- Hohes Volumen, Hohe Startzeit
- Unterschiedliche Betriebssysteme

- Ist eine reine Softwarelösung
- Geringere Sicherheit: System Call Interface ist sehr mächtig und komplex
- Geringeres Volumen, Geringerer Overhead, Kürzere Startup-Zeit
- Betriebssystem fest

Kontainerisierung ist angekommen!

Google Runs All Software In Containers

May 28, 2014 by Timothy Prickett Morgan



The overhead of full-on server virtualization is too much for a lot of hyperscale datacenter operators as well as their peers (some might say rivals) in the supercomputing arena. But the ease of management and resource allocation control that comes from virtualization are hard to resist and this has fomented a third option between bare metal and server virtualization. It is called containerization and Google recently gave a glimpse into how it is using containers at scale on its internal infrastructure as well as on its public cloud.

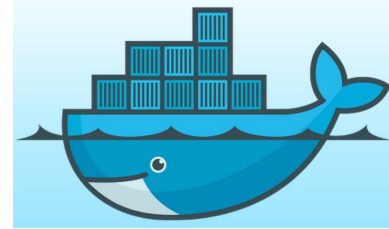
We are talking about billions of containers being fired up a week here, just so you get a sense of the scale.

Beispielhafte Technologien

Hardware-Virtualisierung:
Vagrant und VirtualBox



Betriebssystem-Virtualisierung:
Docker



Betriebssystem-Virtualisierung mit Docker (2013)

Containerization mit Docker: Standardisierung



<http://www.srf.ch/kultur/im-fokus/brasilien/favelas-im-wandel-die-armen-muessen-weichen>



Standard format for operations: start, stop, configure, wire, debug + software logistics.

Docker

- Docker ist eine Automationsumgebung für Betriebssystem-Virtualisierung.
- Aktuell unterstützt Docker Linux als Host-Betriebssystem.
 - Seit 2016 steht eine Windows-Variante zur Verfügung, die mit Hyper-V (Typ 1) virtualisierung läuft.
 - Seit 2020 steht mit WSL2 (Windows Subsystem for Linux) auch eine parallel laufende Linux-Kernel zur Verfügung auf dem Docker ausgeführt werden kann.
- Docker ist als Werkzeug eines Cloud-Anbieters entstanden und ist mittlerweile eines der sichtbarsten und aktivsten Open-Source-Ökosysteme.

In a Nutshell, docker...

... has had 228,212 commits made by 5,912 contributors
representing 10,917,944 lines of code

... is mostly written in Go
with an average number of source code comments

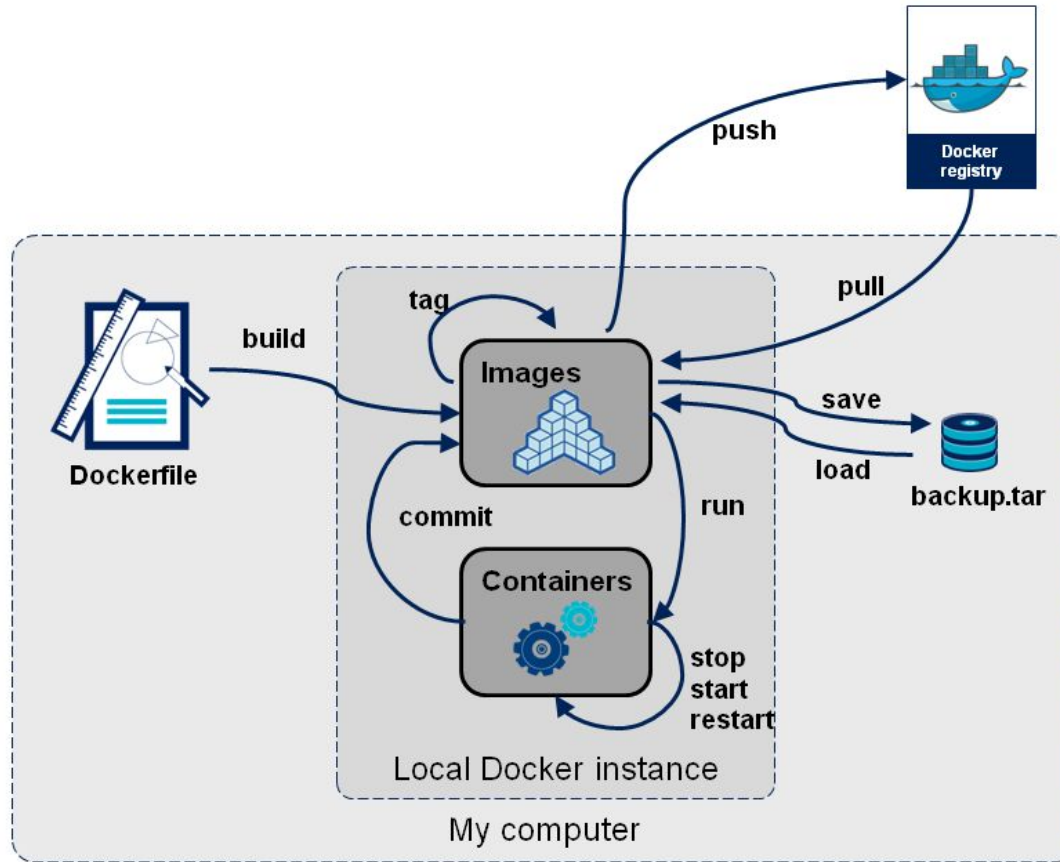
... has a well established, mature codebase
maintained by a very large development team
with decreasing Y-O-Y commits

... took an estimated 3,459 years of effort (COCOMO
model)
starting with its first commit in January, 2012
ending with its most recent commit 4 months ago

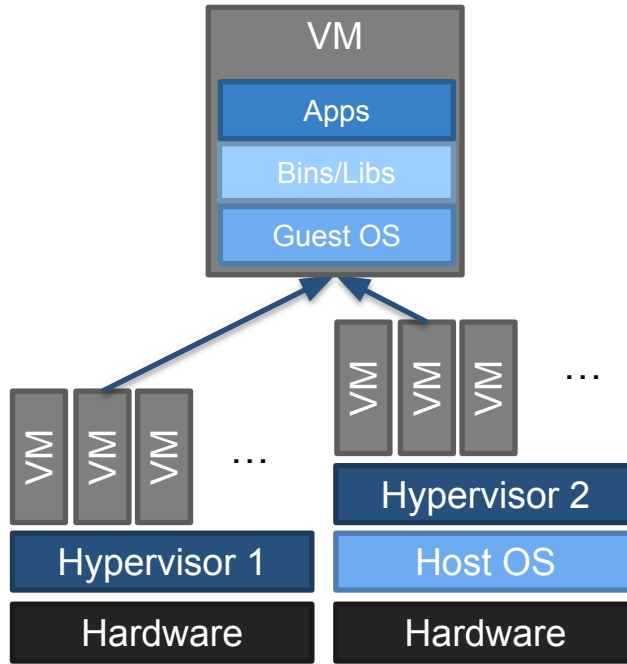
<https://www.openhub.net/p/docker>

Demo

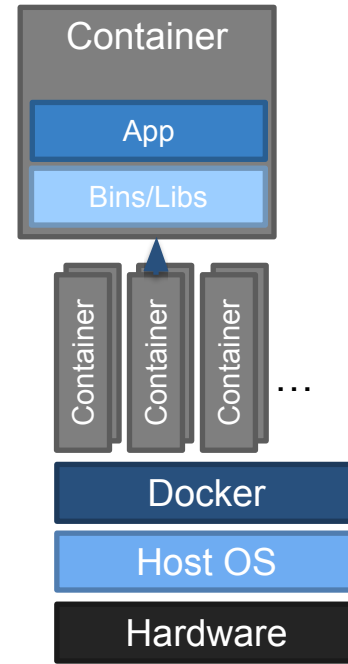
Der Docker Workflow



Containerization mit Docker

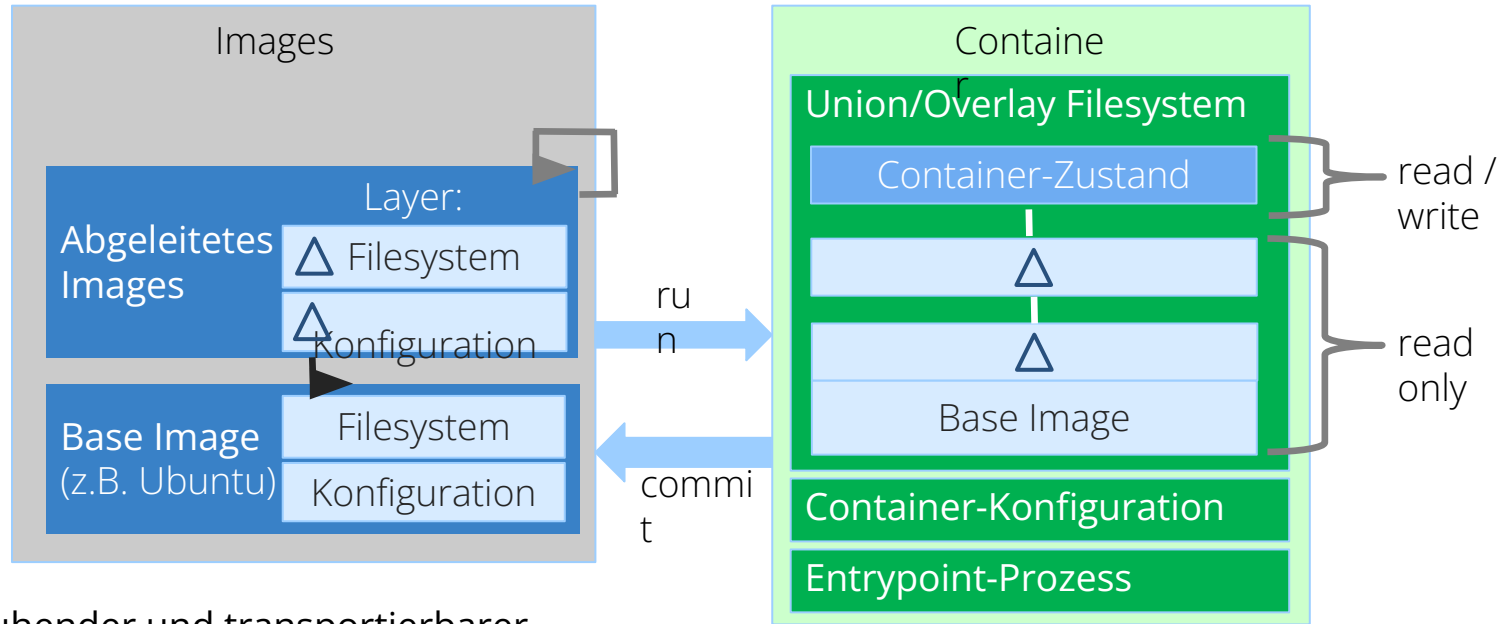


Type 1 / Type 2 Virtualization



Containerization

Im Zentrum von Docker stehen Images und Container.



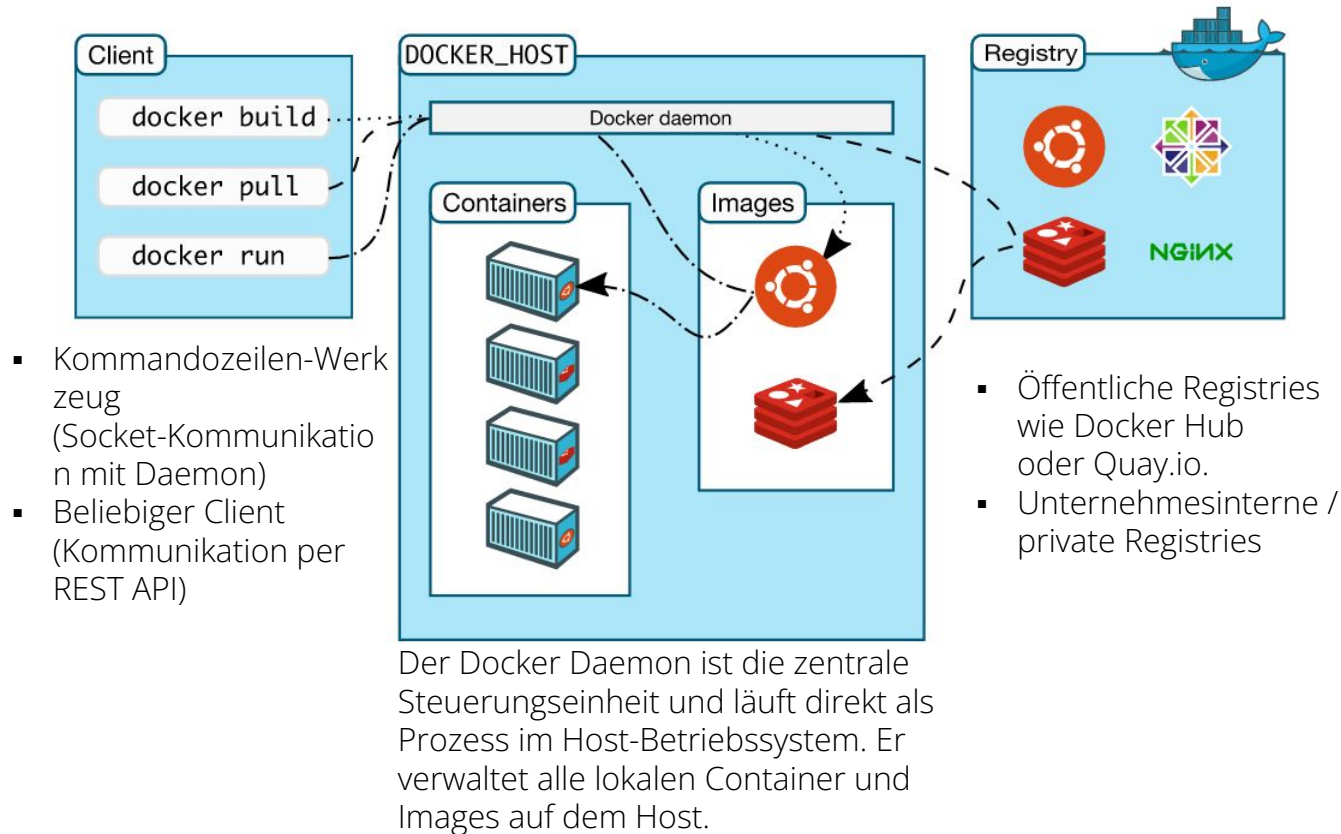
Ruhender und transportierbarer Zustand

- Ein Image basiert i.d.R. auf einem anderen Image und speichert nur das Delta Δ zu diesem Image.
- Ausnahme: Das Base-Image


Laufender Zustand

- Ein Container läuft so lange wie sein Entrypoint-Prozess im Vordergrund läuft. Docker merkt sich den Container-Zustand

Die Docker Architektur.









hub.docker.com ist die öffentliche Standard-Registry für Docker Images.

 Explore Help

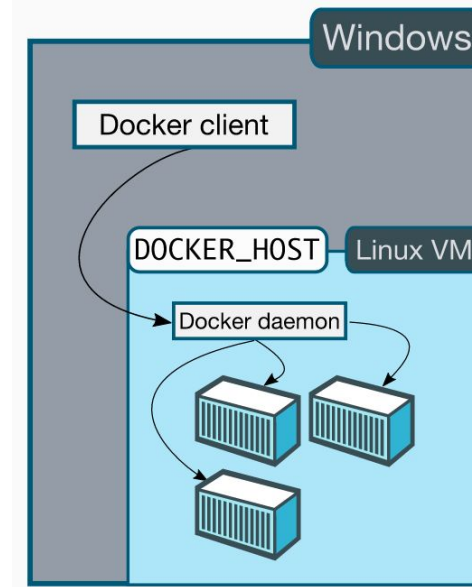
Q solr Sign up Log In

Repositories (555)

All

 solr official	280 STARS	100K+ PULLS	> DETAILS
 anapsix/solr public automated build	1 STARS	8.0K PULLS	> DETAILS
 apopelo/solr public automated build	1 STARS	1.6K PULLS	> DETAILS
 ckan/solr public automated build	1 STARS	896 PULLS	> DETAILS
 harisekhon/solr public automated build	0 STARS	6.6K PULLS	> DETAILS
 makuk66/docker-solr public automated build	81 STARS	50K+ PULLS	> DETAILS

Zusammenspiel zwischen Host und Docker Machine.



Provisionierung von Images mit dem Dockerfile

Ein Dockerfile erzeugt auf Basis eines anderen Images ein neues Images. Dabei werden die folgenden Aktionen automatisiert:

- Konfiguration des Images und der daraus resultierenden Container
- Ausführung von Provisionierungs-Aktionen

Ein Dockerfile ist somit eine Image-Repräsentation alternativ zu einem physischen Image (Bauanteilung vs. Bauteil).

- Wiederholbarkeit beim Bau von Containern
- Automatisierte Erzeugung von Images ohne diese verteilen zu müssen
- Flexibilität bei der Konfiguration und bei den benutzten Software-Versionen
- Einfache Syntax und damit einfach einsetzbar

Befehl: `docker build -t <ziel_image_name> <Dockerfile>`

Das Dockerfile definiert Aufbau und Inhalt des Image.

My Image

Layer 8

Layer 7

Layer 6

Layer 5

Layer 4

Layer 3

Layer 2

Layer 1

Niemals „latest“ verwenden. Antipattern

FROM qaware/alpine-k8s-ibmjava8:8.0-3.10

LABEL maintainer="QAware GmbH <qaware-oss@qaware.de>"

RUN mkdir -p /app

COPY build/libs/zwitscher-service-1.0.1.jar /app/zwitscher-service.jar

COPY src/main/docker/zwitscher-service.conf /app/

ENV JAVA_OPTS -Xmx256m

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app/zwitscher-service.jar"]

Dockerfile Kommandos

Element	Meaning
FROM <image-name>	Sets to base image (where the new image is derived from)
MAINTAINER <author>	Document author
RUN <command>	Execute a shell command and commit the result as a new image layer (!)
ADD <src> <dest>	Copy a file into the containers. <src> can also be an URL. If <src> refers to a TAR-file, then this file automatically gets un-tared.
VOLUME <container-dir> <host-dir>	Mounts a host directory into the container.
ENV <key> <value>	Sets an environment variable. This environment variable can be overwritten at container start with the <code>-e</code> command line parameter of docker run .
ENTRYPOINT <command>	The process to be started at container startup
CMD <command>	Parameters to the entrypoint process if no parameters are passed with docker run
WORKDIR <dir>	Sets the working dir for all following commands
EXPOSE <port>	Informs Docker that a container listens on a specific port and this port should be exposed to other containers
USER <name>	Sets the user for all container commands

Typische Kommandos eines Docker Workflows

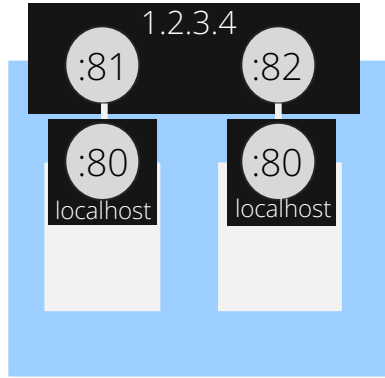
Command	Action
<code>docker build -t <image> .</code>	Build Docker image from „Dockerfile“ with given tag in current directory
<code>docker images</code>	Prints all local images
<code>docker run</code> <code> -d</code> <code> -v <volume mounts></code> <code> -p <host-port>:<container-port></code> <code> <image> <entrypoint process></code>	Run a Docker image: Creates and runs a container. <ul style="list-style-type: none">▪ in background▪ with host directory mounted into the container▪ with port forwarding from host to container▪ image name (and optional entrypoint process)
<code>docker run</code> <code> -ti</code> <code> <image> /bin/sh</code>	Run a Docker image and open a shell within the container <ul style="list-style-type: none">▪ ... with forwarding of local terminal▪ Image name and shell (or „/bin/bash“)
<code>docker ps -a</code>	Prints all containers (without -a = only running containers)
<code>docker commit <container> qaware/foo</code>	Store container as local image
<code>docker kill <container></code> <code>docker rm <container></code>	Terminate container (send SIGKILL to entrypoint process) Remove container
<code>docker rmi -f <image></code>	Remove local image

Hilfreiche Kommandos für Container Troubleshooting

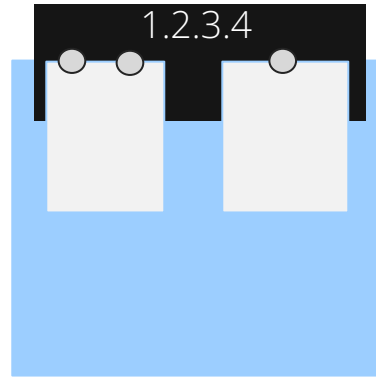
Command	Action
<code>docker inspect <container></code>	Shows container metadata (e.g. IP)
<code>docker logs <container></code>	Prints container syslog
<code>docker top <container></code>	Prints all running processes within a container (like <code>ps -a</code> within the container)
<code>docker exec -ti <container> /bin/sh</code>	Connect terminal to running container
<code>docker stats <container></code>	Shows container runtime statistics (e.g. CPU usage, IO intensity, ...)
<code>docker system prune</code>	Removes all stopped containers, all unused images and all unused volumes
<code>docker history <image></code>	Show the Dockerfile commands for each image layer

Die Docker Networking Modes.

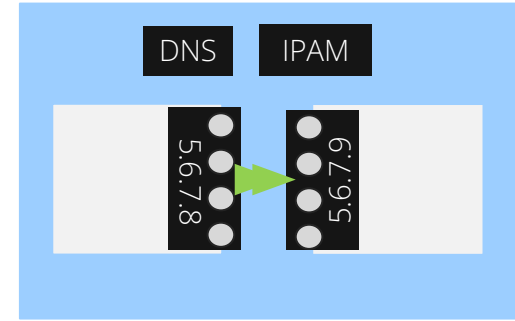
- Docker erlaubt ein getrenntes Netzwerk zwischen docker containern aufzubauen.



Bridge



Host



Overlay Network

`docker network ls`

`docker network inspect bridge`

`docker network create --driver overlay`

`multi-host-network`

`docker network connect multi-host-network container1`



Anhang

https://github.com/veggemonk/awesome-docker

This repository

Search

Pull requests

Issues

Gist

veggiemonk/awesome-docker

Watch

406

Star

4,796

Fork

536

Code

Issues

Pull requests

Projects

Wiki

Pulse

Graphs

A curated list of Docker resources and projects

http://veggemonk.github.io/awesome-docker/

801 commits

4 branches

1 release

138 contributors

Create new file · Upload files · Find file

Clone or download

Branch master · New pull request

gveggemonk committed on GitHub Merge pull Latest commit Bff5333 a day ago

travis-ci.com

CONTRIBUTING.md

LICENSE.md

README.md

Added 'Where to start - on Windows' section

1 month ago

28 days ago

6 months ago

a year ago

3 days ago

Awesome Docker

A curated list of Docker resources and projects inspired by @randresurfur awesome and improved by these amazing contributors.

It's now a GitHub project because it's considerably easier for other people to edit, fix and expand on Docker using GitHub. Just click README.md to submit a pull request. If this list is not complete, you can contribute to make it so.

Please, help organize these resources so that they are easy to find and understood for new comers. See how to Contribute for tips!

If you see a link here that is not (any longer) a good fit, you can fix it by submitting a pull request to improve this file. Thank you!

The creators and maintainers of this list do not receive and should not receive any form of payment to accept a change made by any contributor. The goal of this repo is to index articles, learning materials and projects, not to advertise for profit. All pull requests are merged by default and removed if inappropriate or unavailable, or fixed when necessary.

All the links are monitored and tested with awesome_bot made by @dhamaming

What is Docker ?

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

Source: What is Docker

Where to start ?

10-minute Interactive Tutorial

Docker Training

Read this complete article: Basics - Docker, Containers, Hypervisors, CephFS

Watch the video: Docker for Developers (54:26) by @lpetazzo

Docker Jumpstart: a quick introduction

Docker Curriculum: A comprehensive tutorial for getting started with Docker. Teaches how to use Docker and deploy dockerized apps on AWS with Elastic Beanstalk and Elastic Container Service.

Install Docker on your machine and play with a few Useful Images

Try Panamax: Docker Management for Humans It will install a CoreOS VM with VirtualBox and has nice front end

Install Docker Toolbox Docker Toolbox is an installer to quickly and easily

- Docker Containers on the desktop by @frazzle! The funniest way to learn about docker! (Tips: checkout her profiles and her dockerfiles)
- Container Images and Full Images by @frazzle @ DockerCon 2015 MUST WATCH VIDEO (38:50)
- Learn Docker Full environment set up, screenshots, step-by-step tutorial and more resources (video, articles, cheat sheets) by @dnyr!
- Docker Caves: What You Should Know About Running Docker in Production (written 17 APRIL 2016) MUST SEE
- How to Whole Learn Docker in your web browser, no setup or installation required.
- Docker for all - Developers, Testers, DevOps, Product Owners - Videos Docker Training Videos for all

Where to start - on Windows ?

- Windows Containers Quick Start Overview of Windows containers, drilling down to Quick Starts for Windows 10 and Windows Server 2016
- Build And Run Your First Docker Windows Server Container Walkthrough installing Docker on Windows 10, building a Docker image and running a Windows container
- Video: Windows Containers and Docker: The 101 A 20-minute overview, using Docker to run PowerShell, ASP.NET Core and ASP.NET apps
- A Comparative Study of Docker Engine on Windows Server vs Linux
- Comparing the feature sets and implementations of Docker on Windows and Linux
- Docker with Microsoft SQL 2016 + ASP.NET Demonstration running ASP.NET and SQL Server workflows in Docker
- Running a Legacy ASP.NET App in a Windows Container: Steps for Dockerizing a legacy ASP.NET app and running as a Windows container
- Exploring ASP.NET Core with Docker in both Linux and Windows Containers Running ASP.NET Core apps in Linux and Windows containers, using Docker for Windows

MENU

- What is Docker ?
- Where to start ?
- MENU
- Useful Articles
 - Main Resources
 - General Articles
 - Deep Dive
 - Networking
 - Metal
 - Multi-Server
 - Cloud Infrastructure
 - Good Tips
 - Newsletter
 - Continuous Integration
 - Optimizing Images
 - Service Discovery
 - Security
 - Performances
 - Raspberry Pi & ARM
- Books
 - In English
 - Chinese
 - German
 - Portuguese
- Tools
 - Terminal User Interface
 - Dev Tools
 - Continuous Integration / Continuous Delivery
 - Deployment
 - Hosting for repositories (registries)

- Reverse Proxy
- Web Interface
- Local Container Manager
- Volume management and plugins
- Useful Images
- Dockerfile
- Docker Compose file
- Storing Images and Registries
- Monitoring
- Networking
- Logging
- Deployment and Infrastructure
- PaDs
- Remote Container Manager / Orchestration
- Security
- Service Discovery
- Metadata
- Sides
- Videos
 - Main Account
 - Useful videos
- Interactive Learning Environments
- Interesting Twitter Accounts
- People
- Communities and Meetups

Useful Articles

Main Resources

- Docker Weekly Huge resource
- Docker Cheat Sheet by @swargent MUST SEE
- Docker Printable Refcard by @dmonmond
- CenturyLink Labs
- Valuable Docker Links Very complete
- Docker Ecosystem (Mind Map) MUST SEE
- Docker Ecosystem (PDF) MUST SEE find it on blog by Bryngolar Peter.
- Blog of @frazzlezell
- Blog of @lpetazzo
- Blog of @progrium
- Blog of @jwilder
- Blog of @robertymchael
- Blog of @delfraba
- Blog of @sebgas
- Blog of @codehelp
- Digital Ocean Community
- Container42
- Container solutions
- DockerOne Docker Community (in Chinese) by @LYingJie
- Project Web Dev : Article series: How to create your own website based on Docker
- Docker vs. VMlet Combining Both for Cloud Portability Nirvana
- Docker Containers on the desktop by @frazzle! The funniest way to learn about docker! (Tips: checkout her profiles and her dockerfiles!)
- Awesome Linux Container more general about container than this repo, by @Friz-zy.

General Articles

- Getting Started with Docker by @fideloper -- Servers For Hackers is valuable resource. At some point, every programmer finds themselves needing to know their way around a server.
- What is Docker and how do you monitor it?
- How to Use Docker on OS X: The Missing Guide
- Docker for Java Developers

- Deploying NGINX with Docker
- Eight Docker Development Patterns
- Rails Development Environment for OS X using Docker
- Logging on Docker: What You Need to Know - see the video (~50min)
- Comparing Five Monitoring Options for Docker
- Minimalistic data-only container for Docker Compose (Written Mar 1, 2015)
- Running Docker Containers with Systemd
- Dockerizing Flask With Compose and Machine - From Localhost to the Cloud -- GitHub Learn how to deploy an application using Docker Compose and Docker Machine (written 17 April 2015)
- Why and How to use Docker for Development (written 28 APR 2015)
- Automating Docker Logging: ElasticSearch, Logstash, Kibana, and Logspout (written 27 APR 2015)
- Docker Host Volume Synchronization (written 1 JUN 2015)
- From Local Deployment to Remote Deployment with Docker Machine and compose (written 2 JUL 2015)
- Docker Build, Ship and Run Any App, Anywhere by Martin Deans, Wiebe van Gestel, Ma Nijssen, and Rick Wieman from Delft University of Technology (written 2 JUL 2015)
- Joining the Docker Ship Learn how to contribute to docker (written 9 JUL 2015)
- Continuous Deployment with Gradle and Docker Describes a complete pipeline from source to production deploy (includes a complete Spring Boot example project) by @svenlo
- Containerization and the PaaS Cloud -- This article discusses the requirements that arise from having to facilitate applications through distributed multcloud platforms.
- Docker for Development: Common Problems and Solutions by @druichas
- Docker Adoption Data A study by Datadog on the real world Docker usage statistics and deployment patterns.
- Docker on Windows behind a Firewall by @kaltrobert
- Pulling Git into a Docker image without leaving SSH keys behind by @khaah
- 6 Million Ways To Log In Docker by @raychaser
- Dockerfile Generator (ruby script)
- Running Production Hadoop Clusters in Docker Containers
- 10 practical docker tips (Dec 2015) by @jrodsidem
- Kubernetes Cheatsheet - A great resource for managing your Kubernetes installation
- Container Best Practices - Red Hat's Project Atomic created a Container Best Practices guide which applies to everything and is updated regularly.
- Production Meteor and Node Using Docker, Part 1 by @projectreactor
- Resource Management in Docker by @managiedmann

Portuguese Articles

- Uma rápida introdução ao Docker e instalação no Ubuntu
- O que é uma imagem e o que é um container Docker?
- Criando uma imagem Docker personalizada
- Comandos mais utilizados no Docker

Deep Dive

- Creating containers - Part 1 This is part one of a series of blog posts detailing how docker creates containers. by @robertymchael
- Data-only container madness

Networking

- Using Docker Machine with Weave 0.10 (written 22 APR 2015)
- How to Route Traffic through a Tor Docker container by @frazzle (written 20 JUN 2015)
- Demystifying Docker overlay networking. By @nigelboulton

Metal

- How to use Docker on Full Metal
- Ganglia & a bare essential OS for running the Docker Engine on bare metal or Cloud.

Multi-Server

- A Docker based mini-PaaS by @progrium
- A multi-host scalable web services demo using Docker swarm, Docker compose, NGINX, and Blockbridge

Cloud infrastructure

- Cloud Infrastructure Automation for Docker Nodes

Good Tips

- 24 random docker tips by @csabapalfi
- GUI Apps with Docker by @frghm
- Automated Nginx Reverse Proxy for Docker by @jwilder
- Using NGinex with Docker/Docker
- A Simple Way to Dockerize Applications by @jwilder
- Building good docker images by @lbergknoff
- 10 Things Not To Forget Before Deploying Docker in Production
- Docker CIFS - How to Mount CIFS as a Docker Volume
- Nginx Proxy for Docker (written 9 JUL 2015)
- Dealing with linked containers dependency in docker-compose by @rochacbruno
- Docker Tips by @jwilder
- Docker on Windows behind a Firewall by @kaltrobert
- Pulling Git into a Docker image without leaving SSH keys behind by @khaah
- 6 Million Ways To Log In Docker by @raychaser
- Dockerfile Generator (ruby script)
- Running Production Hadoop Clusters in Docker Containers
- 10 practical docker tips (Dec 2015) by @jrodsidem
- Kubernetes Cheatsheet - A great resource for managing your Kubernetes installation
- Container Best Practices - Red Hat's Project Atomic created a Container Best Practices guide which applies to everything and is updated regularly.
- Production Meteor and Node Using Docker, Part 1 by @projectreactor
- Resource Management in Docker by @managiedmann

Newsletter

- Docker Team
- CenturyLink Labs
- Tatum
- Shipable
- WebOps weekly

Continuous Integration

- Docker and Phoenix: How to Make Your Continuous Integration Merge Awesome
- Jenkins 2.0 - Sonarcast Series by Virendra Bhalothia
- Pushing to ECR Using Jenkins Pipeline Plugin by @mkazis87

Optimizing Images

- Create the smallest possible Docker container
- Creating a Docker image from your code
- Optimizing Docker Images
- How to Optimize Your Dockerfile by @mtumcloud
- Building Docker Images for Static Go Binaries by @kaleishightower
- Squashing Docker Images by @jwilder
- Dockerfile Golf (or optimizing the Docker build process)
- DockerSlim shrinks fat Docker images creating the smallest possible images.
- Skinnywhale Skinnywhale helps you make smaller (as in megabytes) Docker containers.
- MicroBadger - Analyze the contents of images and add metadata labels

Service Discovery

- @progrium Service Discovery articles series:
 - Consul Service Discovery with Docker
 - Understanding Modern Service Discovery with Docker
 - Automatic Docker Service Announcement with Registrar

53

Abstraktionsebene der Hardwarevirtualisierung

- Hardwarevirtualisierung „simuliert“ auf Ebene der Hardwarespezifikation
 - CPU Maschinencode der Rechnerarchitektur
 - Programmierschnittstellen der Hardware
 - Gleiche Ebene, auf der auch die Gerätetreiber arbeiten

Beispielspezifikation und Umsetzung in C:

This CPU has three 8-Bit general purpose registers named A, X and Y.



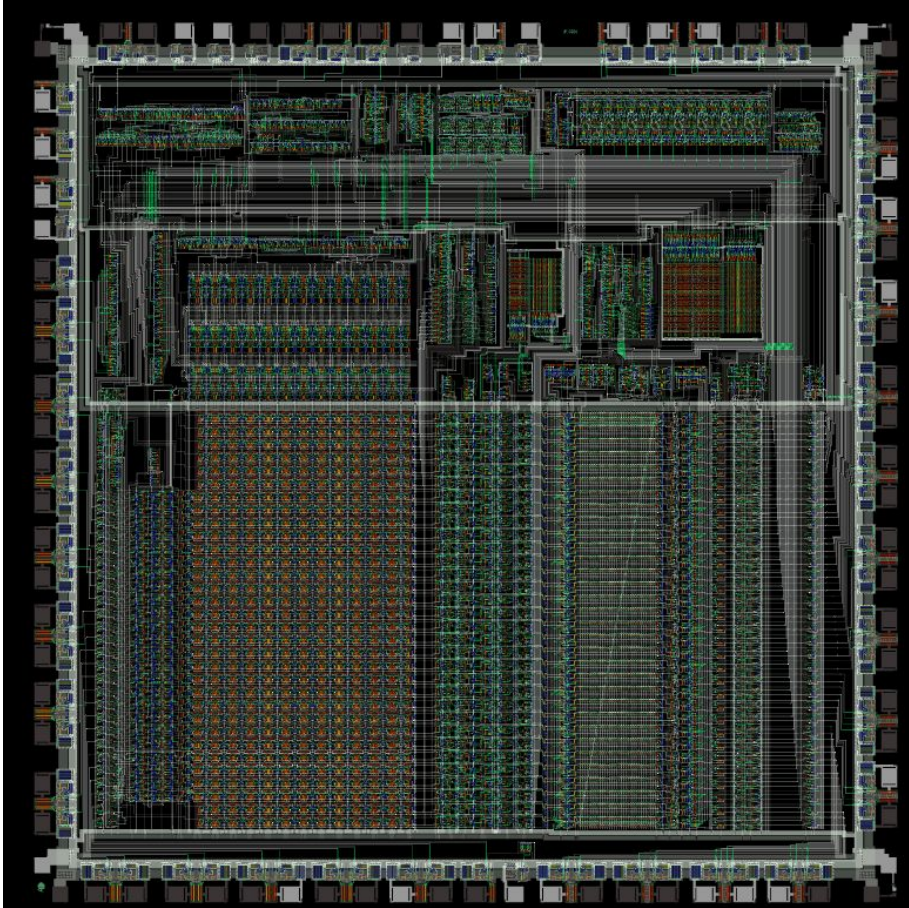
```
uint8_t reg_a, reg_x, reg_y;
```

Opcode E8 increments the X-register by one



```
uint8_t opcode = ram[instruction_address];  
switch(opcode) {  
    ...  
    case 0xe8:  
        reg_x++;  
        break;  
    ...  
}
```

Hardwarevirtualisierung ist keine Simulation



Aufgezeichnet von visual6502.org

Abstraktionsebene der Hardwarevirtualisierung

- Hardwarevirtualisierung „simuliert“ auf Ebene der Rechnerarchitektur Spezifikation
 - CPU Maschinencode der Rechnerarchitektur
 - Programmierschnittstellen zur Hardware
 - Gleiche Ebene, auf der auch die Gerätetreiber arbeiten
 - Siehe zum Beispiel
 - Intel® 64 and IA-32 Architectures Software Developer's Manuals
<https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html#three-volume>

Virtualisierung

Virtualisierung: die Erzeugung von virtuellen Realitäten und deren Abbildung auf die physikalische Realität.

Zweck:

- **Multiplizität** □ Erzeugung mehrerer virtueller Realitäten innerhalb einer physikalischen Realität
- **Entkopplung** □ Bindung und Abhängigkeit zur Realität auflösen
- **Isolation** □ Physikalische Seiteneffekte zwischen den virtuellen Realitäten vermeiden



Wie wird virtualisiert?

- **Prozessor**

- Virtuelle Rechenkerne. Der virtuellen Maschine werden CPU-Cores zugewiesen.
- Non-privileged operations werden durchgereicht, privileged werden emuliert oder die VM abgebrochen (trap and emulate). Genau hier helfen die CPU Extensions.

- **Hauptspeicher**

- Virtuelle Hauptspeicher-Partition im echten physikalischen Speicher. (Die Null verschiebt sich)
- Management der realen Repräsentation mittels der Management Memory Unit (MMU).

- **Netzwerk**

- Virtuelle Netzwerkschnittstellen und virtuelle Netzwerk-Infrastrukturen (VLAN)
- Brücken und Routing zwischen virtuellen und realen Netzwerken

- **Storage**

- Virtuelle Festplatten-Laufwerke. Abbildung auf Dateien im realen Dateisystem. Volumen entweder vor-allokiert oder dynamisch wachsend.
- Virtuelle SANs (Storage Area Networks) über Aufteilung der Daten eines virtuellen Laufwerks auf viele Storage-Einheiten.