QAWARE SOFTWARE ENGINEERING

HM Hochschule München University of Applied Sciences

Cloud Computing

Virtualisierung

Bernhard Schaidhammer

2025-04-03

Cloud Computing lebt von der Skalierung

- Erinnerung NIST-Definition: "**Resource Pooling**"
- Tendenz seit ca. 20 Jahren: Einzelne Server werden immer leistungsstärker.

Bildquelle: proshop.no

Server in **kube07:** HPE ProLiant DL385 Gen10
2x AMD EPYC 7402 à 24 Kerne, 48 Threads, 180 Watt, 2.8-3.3GHz
⇒ 48 Kerne, 96 Threads
512 GB RAM
2x10GbE Netzwerk
2x1TB SSD via SATA
Nettopreis 2020: 8670€
Stromverbrauch: 200W - 600W(?), 40-120€/Monat

Server auf AWS (via https://instances.vantage.sh/)
r6a.12xlarge
48 vCPUs, 384 GB RAM, 18 Gbit Networking
On-Demand: 2210€/Monat (oder: 5 ct/Minute)
Reserviert: 1461€/Monat
Spot Instance: 723€/Monat

https://10.28.229.7 130% preis kwh münchen

iLO 5
2.30 Aug 24 2020

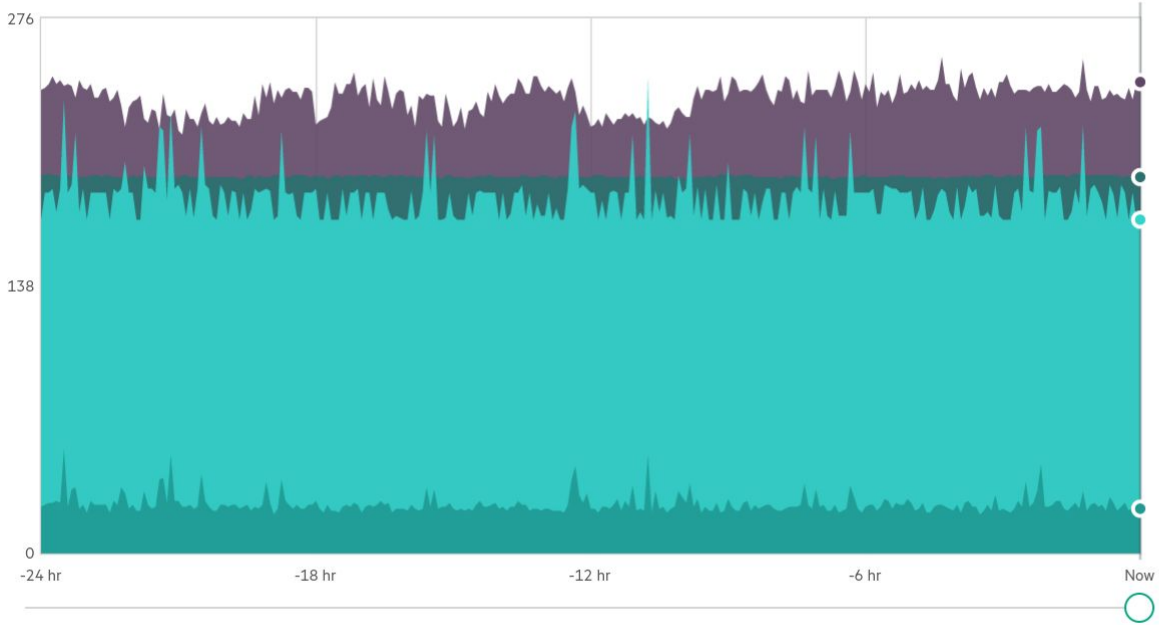
- Information
- System Information
- Firmware & OS Software
- iLO Federation
- Remote Console & Media
- Power & Thermal
- iLO Dedicated Network Port
- iLO Shared Network Port
- Remote Support
- Administration
- Security
- Management
- Lifecycle Management

Power & Thermal - Power Meter

Server Power | Power Meter | Power Settings | Power | Fans | Temperatures

24-Hour History Graph

Power consumption over the past 24 hours at five-minute intervals.



Maximum	243
Average	194
Total DIMM	172
Total CPU	23

2023-03-29 12:13:15

20 min | 24 hr

Watts | BTU/hr

Power Status Now Power History 5 min 30 min 24 hr

Cloud Computing lebt von der Skalierung

- Typischer Use-Case: Datenbank für Wordpress / kleine und mittlere Websites
  - Datenbankgröße: ~100-500MB
  - CPU-Nutzung: ~0.2 CPU
- Dazu Web-Server Apache oder Nginx
  - CPU-Nutzung: ~0.1 CPU
- Redundanz durch Einsatz mehrerer Server?

?

Cloud Computing lebt von der Skalierung

- Erinnerung: NIST-Definition - “Resource Pooling”

- 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 sollten vermieden werden, Security ist Ziel.
  - **Entkopplung von der Hardware** für mehr Flexibilität im Betrieb und Robustheit bei Ausfällen
  - Ressourcen sollen **flexibel** vergeben werden. Steuerung mittels “software defined resources”

- **Virtualisierung** löst diese Anforderungen und macht Cloud Computing erst möglich.

Effiziente Ressourcen Nutzung aus Kundensicht

- Überprovisionierung:
  - Bestellte Instanzen/Resourcen liegen ungenutzt herum. (28-32% ungenutzt[1])
  - Erinnerung: “Treat cloud infrastructure as cattles, not as pets”.
  - Erinnerung: Die New York Times Pipeline aus der ersten Vorlesung.
- Zuverlässigkeit vs. Kosten!
  - Ist ein eigenes riesiges Thema und wäre eine eigene Vorlesung Wert.
- Ein Cloud Ingenieur muss immer die richtige Balance aus Kosten, Sicherheit und Effizienz finden.
- Ein gutes Verständnis wie Virtualisierung funktioniert kann helfen die Cloud effizient zu nutzen.

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 addressierter physikalischer Speicher
- Netzwerk
  - z. B. Input-Output Stream von Ethernet Frames
- Storage
  - Blockspeicher (linear addressiert. 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

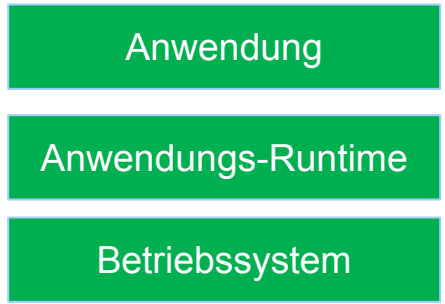
Was ist Emulation?

- Emulation: Bildet die Hardware eines nicht vorhandenen oder nicht kompatiblen Rechnersystems oder Teile eines entsprechenden Rechnersystems nach
- Emulationen sind 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
    - Rosetta2 von Apple. CPU Instruktionsübersetzung von x86 zu ARM. https://dougallj.wordpress.com/2022/11/09/why-is-rosetta-2-fast/
    - Windows on ARM: https://learn.microsoft.com/en-us/windows/arm/apps-on-arm-x86-emulation
    - QEMU User Mode Emulation
  - Voll-Virtualisierer (Hardware und spezielle Instruktionen der CPU)
  - QEMU und Bochs können Windows und Linux praktisch überall starten.

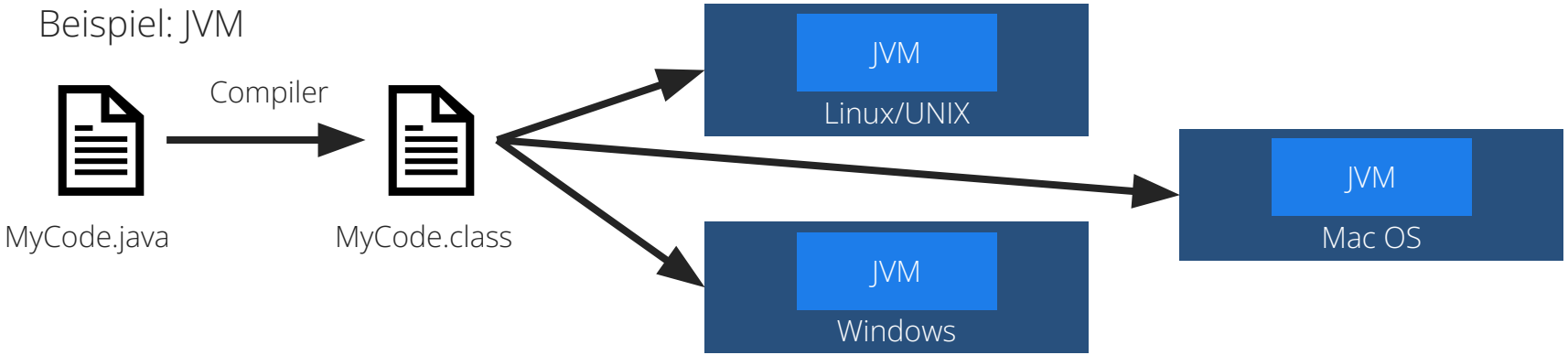
Windows 95 auf der Apple Watch

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



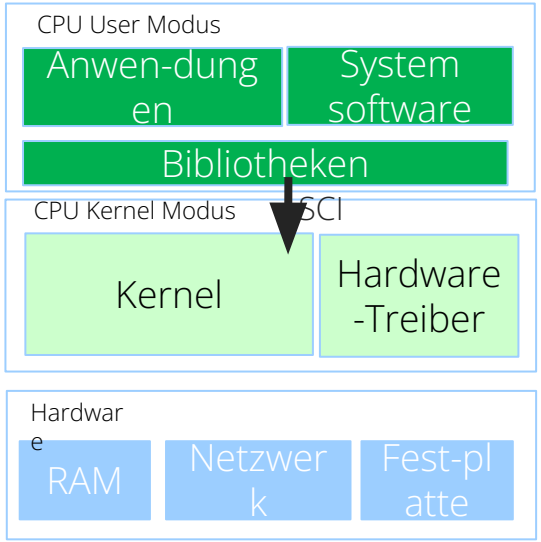
Virtualisierung, aber performant

- 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 zusätzlichen Ressourcenaufwand erreichen?
  - Anwort 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

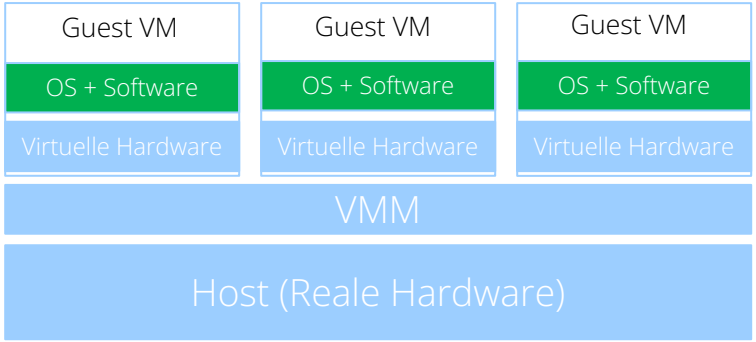
- User Mode ruft Kernel über das System Call Interface (SCI) auf. Aktuell besteht das SCI bei Linux aus ca. 380 System Calls.
- Höchste Berechtigungsstufe
  - Priviledged CPU Instruktionen
  - Zugriff auf Hardware mittels Treiber
- Übernimmt z. B. Dateisystemverwaltung und Scheduling der Anwendungen

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 unterschieden



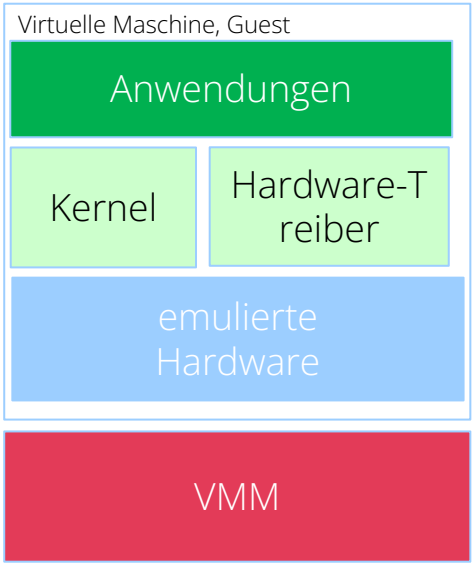
Host
- Der Rechner der eine oder mehrere virtuelle Maschinen ausführt und die dafür notwendigen Hardware-Ressourcen zur Verfügung stellt.

Guest
- Eine lauffähige / laufende virtuelle Maschine

VMM (Virtual Machine Monitor, auch Hypervisor genannt)
- Die Steuerungssoftware zur Verwaltung der Guests und der Host-Ressourcen

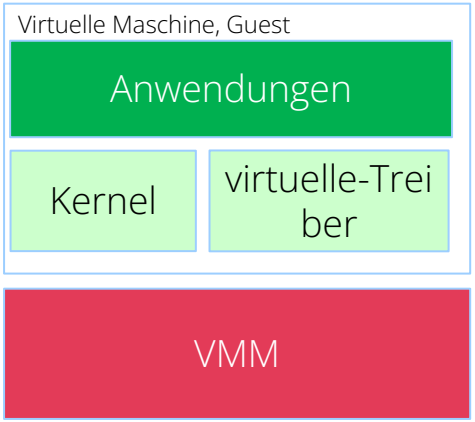
Voll-Virtualisierung

- 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%



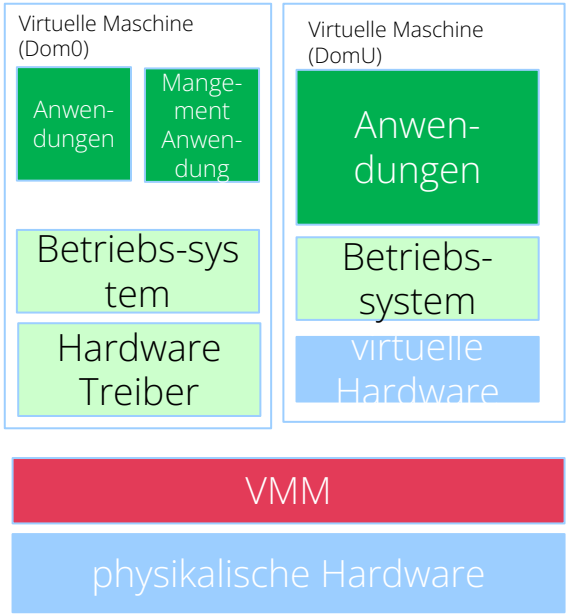
Para-Virtualisierung

- Dem Gast-Betriebssystem stehen keine direkten 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%



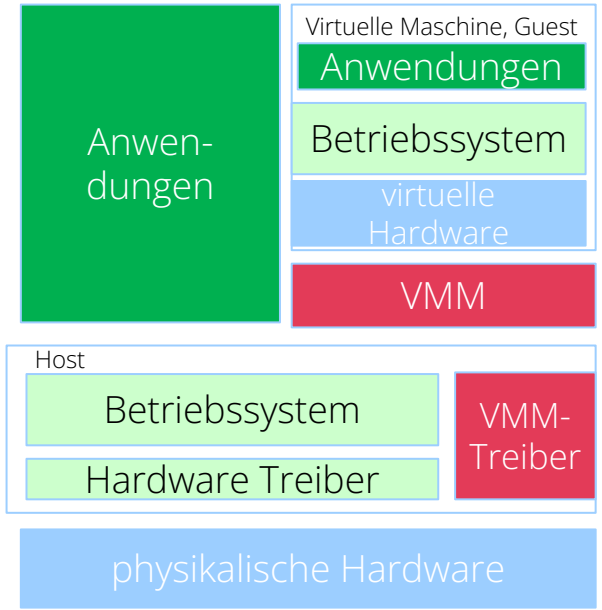
Typ 1: Bare-Metal Virtualisierung

- 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 – ungenutzten Hauptspeicher aus VMs dynamisch “wiedergewinnen”
  - 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

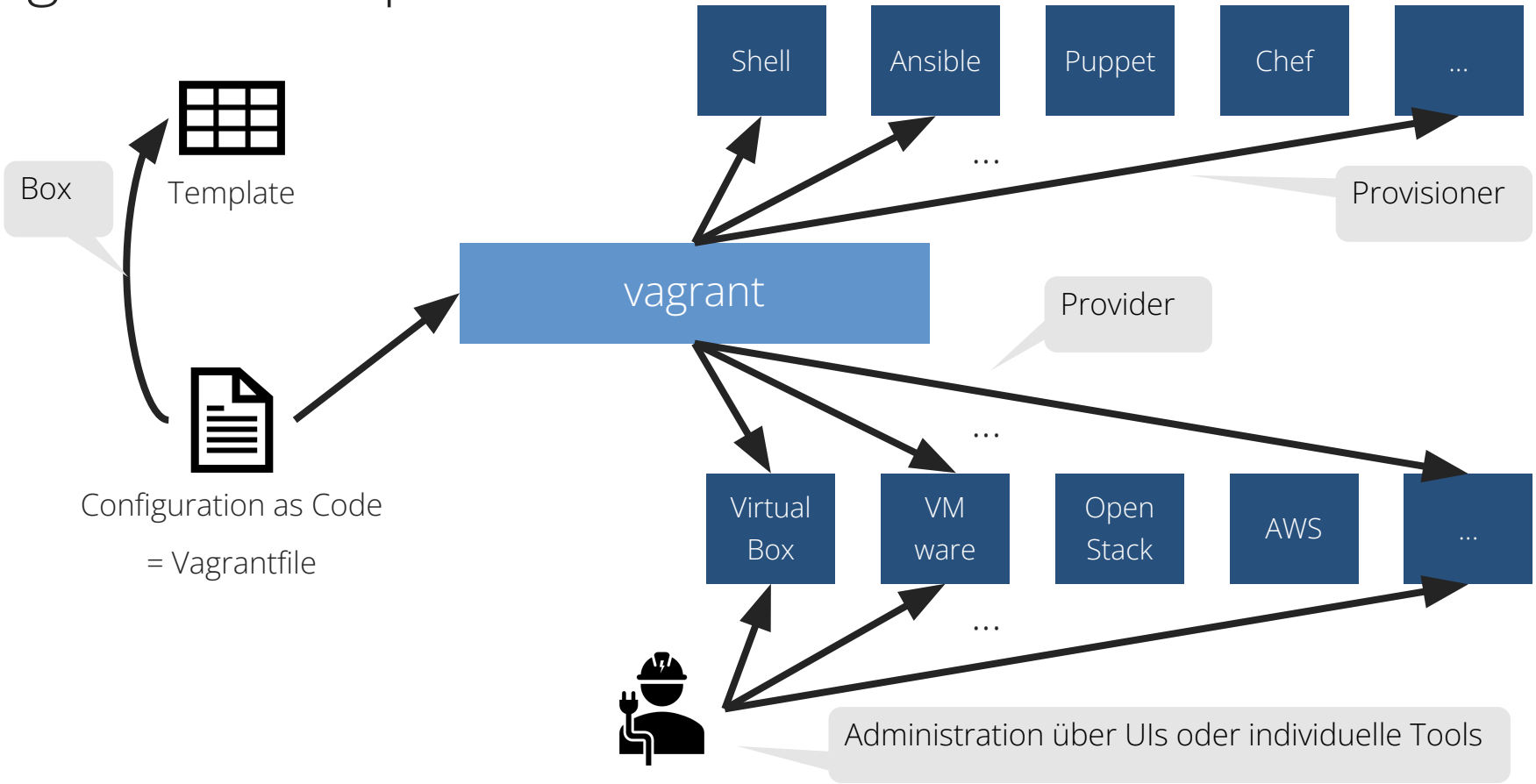
Hardware-Virtualisierung: Vagrant und VirtualBox

+

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

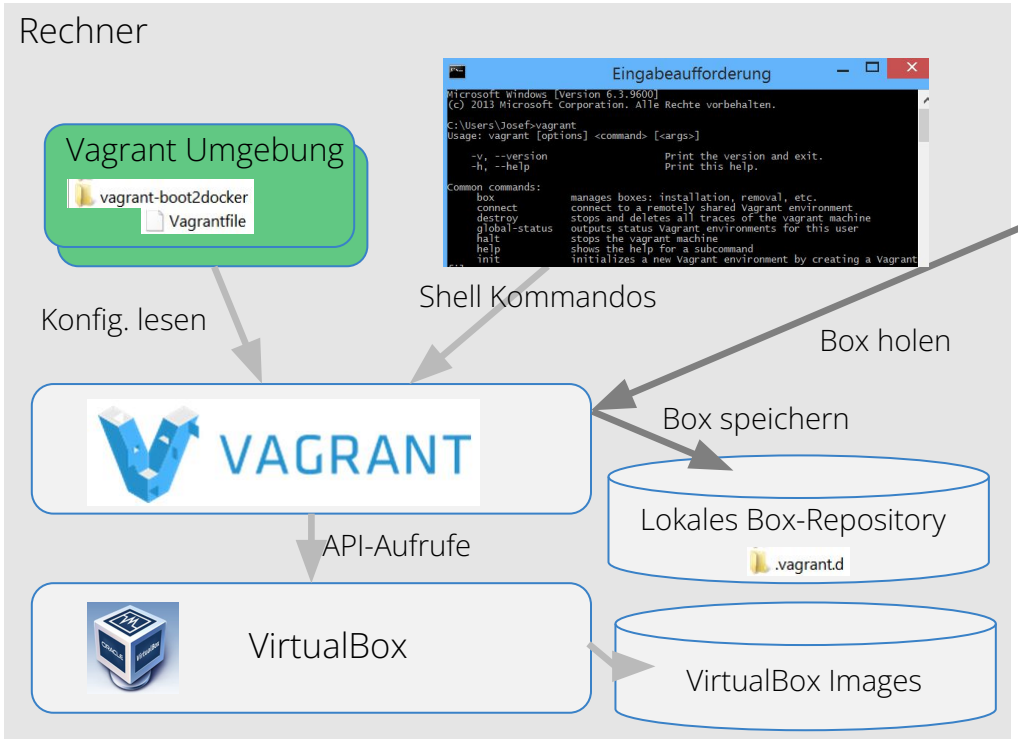
Automationssoftware für virtuelle Umgebungen auf einem Rechner. Virtuelle Maschinen per Kommandozeile erstellen und steuern.

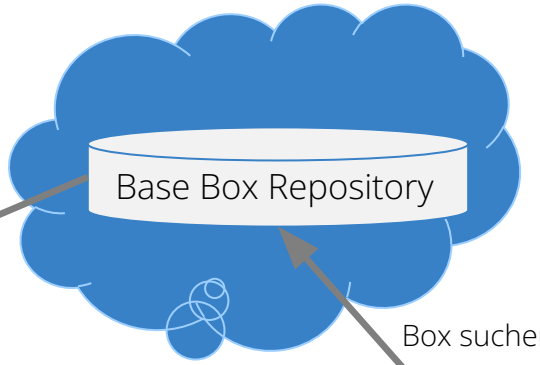
Vagrant: Konzepte

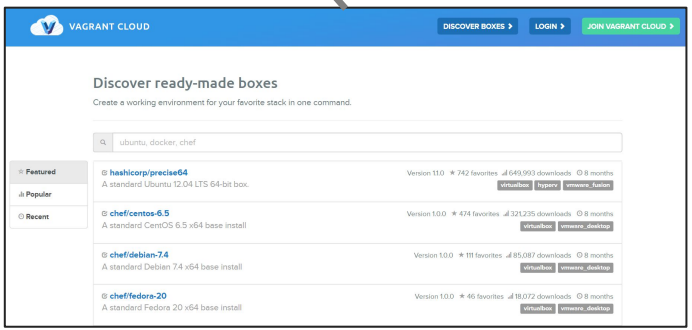


Demo

Vagrant: Eine schematische Übersicht.







https://vagrantcloud.com

Das Vagrantfile beschreibt die zu erstellende virtuelle Maschine.

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

# 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"

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

        # 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

        # 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
end
```

Vagrantfiles werden in Ruby geschrieben

Definition der Basis-Box

Konfiguration der Provisionierung

Konfiguration des Virtualisierungs-Providers

Konfiguration des Netzwerks

Virtualisierungsarten: Betriebssystemvirtualisierung

Hardwarevirtualisierer sind Schwergewichte

- Jede VM inkludiert eine virtuelle Kopie eines kompletten Betriebssystem 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

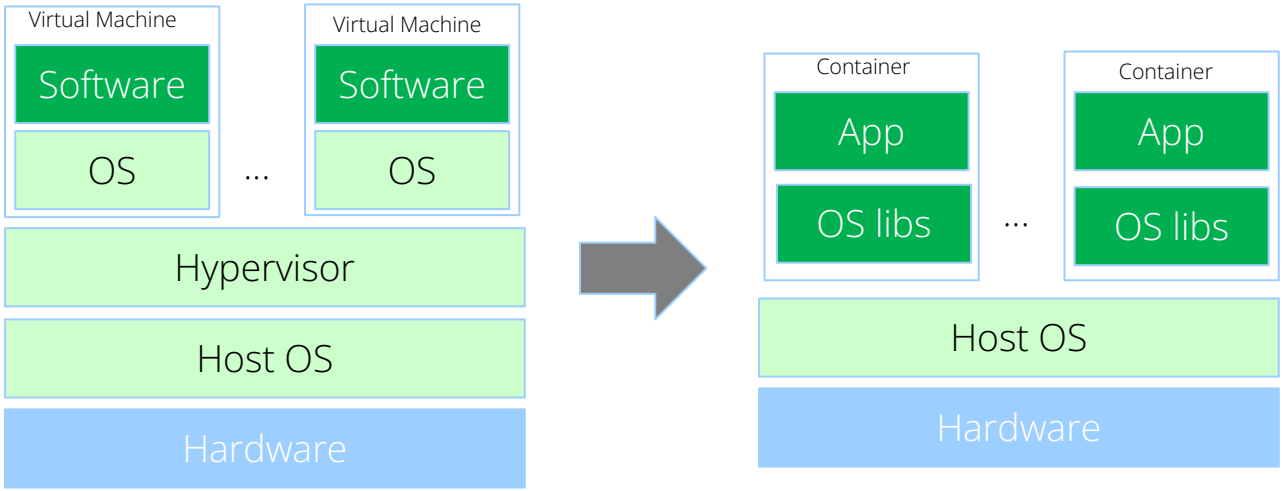
Linux Kernel Namespaces (Jahr 2002)

- 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

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/

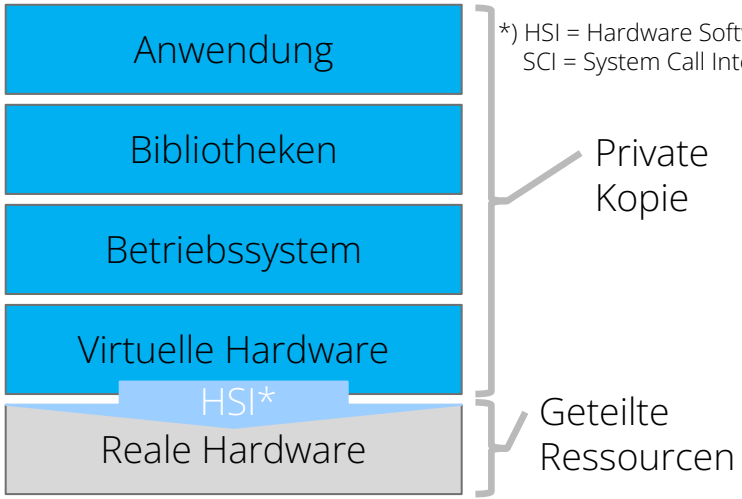
Betriebssystem-Virtualisierung



- 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

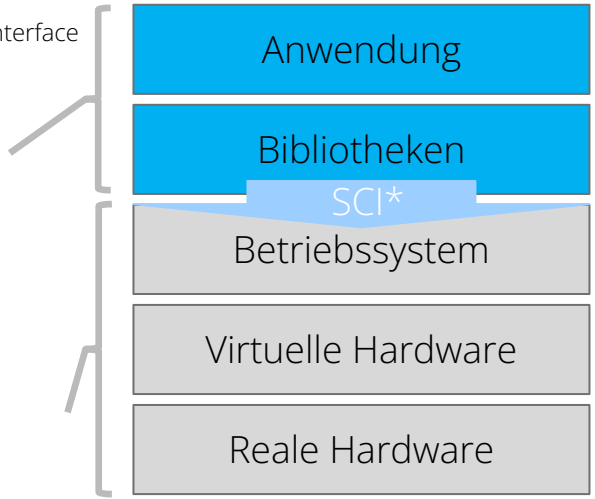
Hardware- vs. Betriebssystem-Virtualisierung

Hardware-Virtualisierung



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

Betriebssystem-(OS-)Virtualisierung



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

Containerisierung ist angekommen!

Google Runs All Software In Containers

May 28, 2014 by Timothy Prickett Morgan

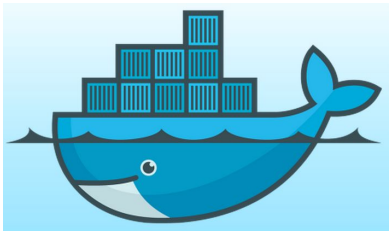
The overhead of full-on server virtualization is too much for a lot of hyperscale datacenter operators as well as their peers (some might say rivals) in the supercomputing arena. But the ease of management and resource allocation control that comes from virtualization are hard to resist and this has 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, Podman



podman

Betriebssystem-Virtualisierung mit Docker

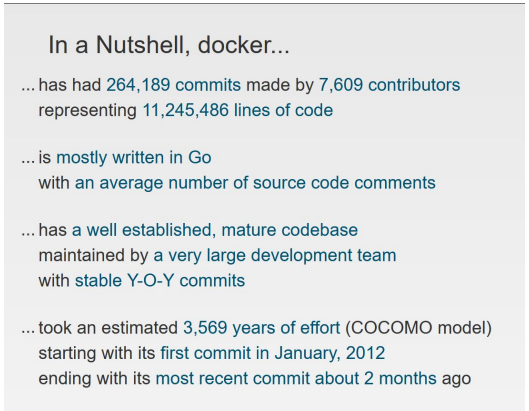
Containerization mit Docker: Standardisierung

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

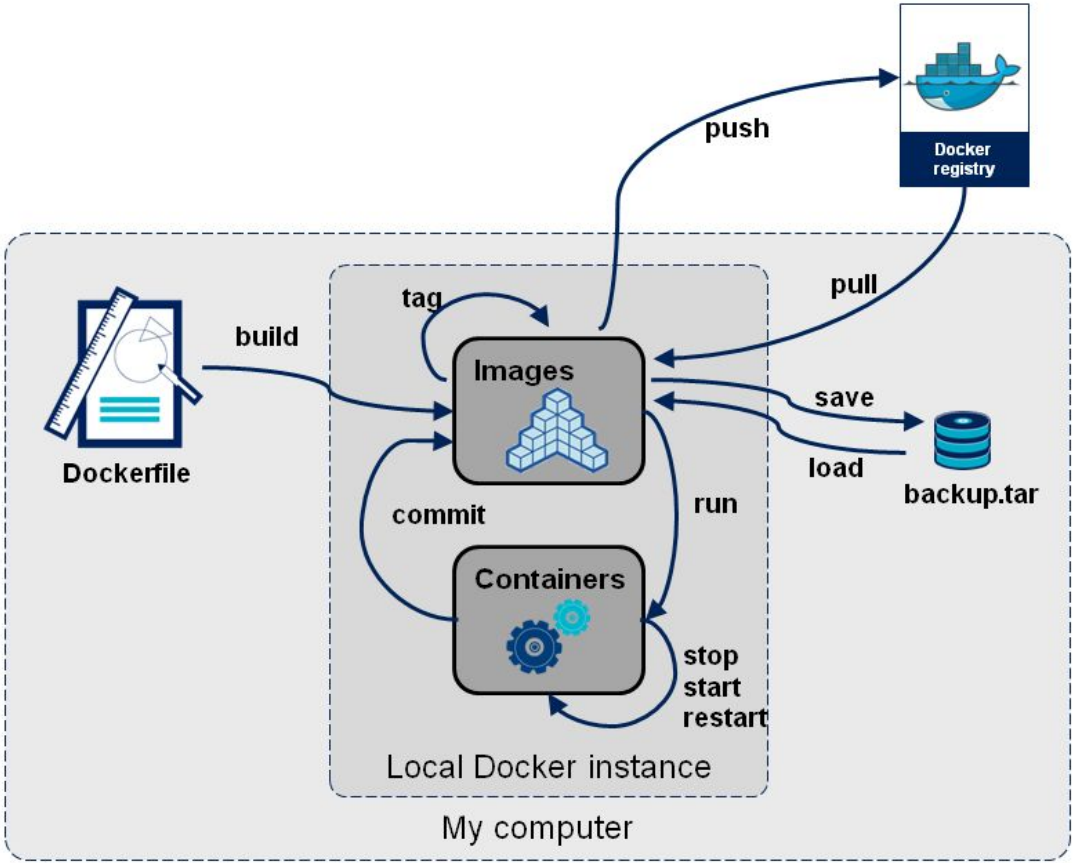
- Docker ist eine Automationsumgebung für Betriebssystem-Virtualisierung.
- Aktuell unterstützt Docker Linux als Host-Betriebssystem.
  - Seit 2016 steht eine Windows-Variante zur Verfügung, die mit Hyper-V (Typ 1) virtualisierung läuft.
  - Seit 2020 steht mit WSL2 (Windows Subsystem for Linux) auch 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. Leider ist Docker (die Firma) inzwischen dazu übergegangen, die freie Nutzung des Dockerhubs einzuschränken - daher kann es Sinn machen, eine eigene Registry zu verwenden.



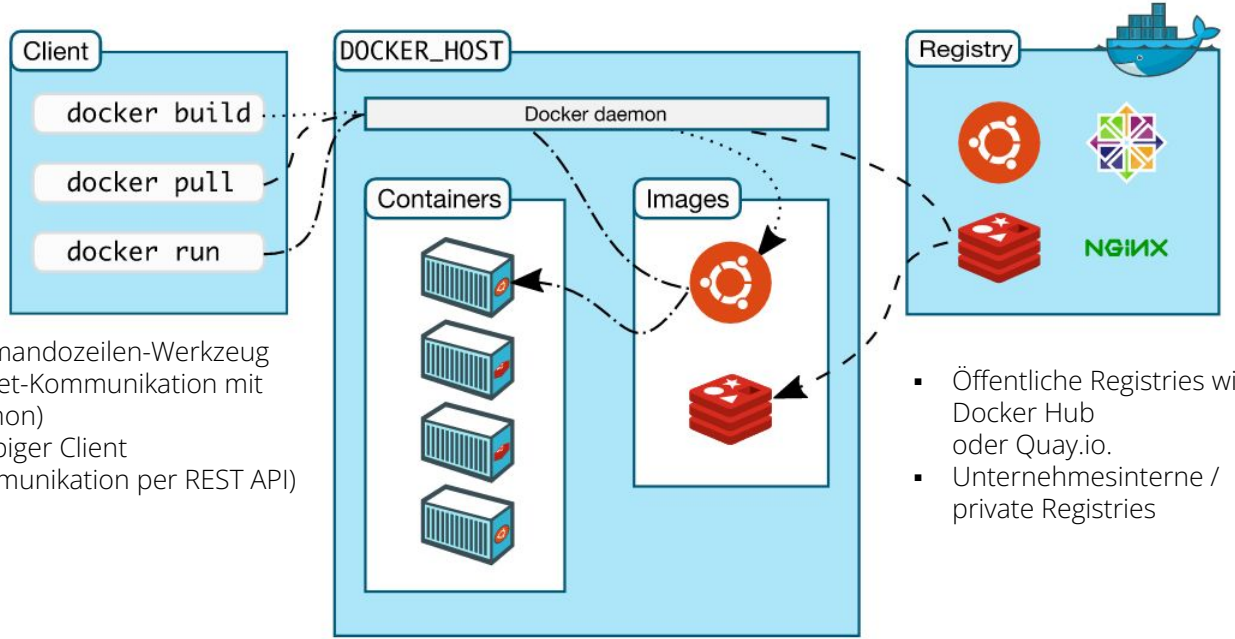
https://www.openhub.net/p/docker

Demo

Der Docker Workflow.



Die Docker Architektur.



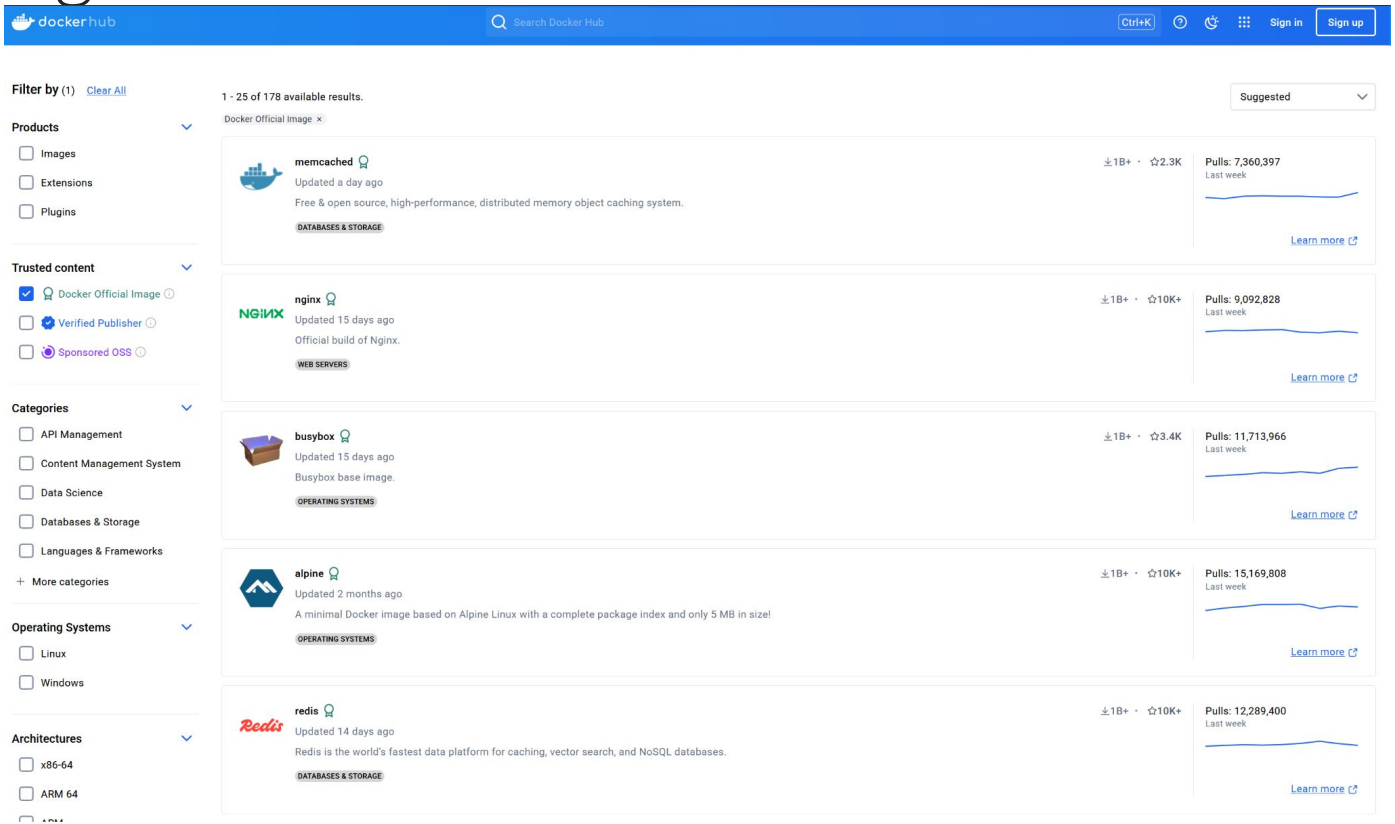
- Kommandozeilen-Werkzeug (Socket-Kommunikation mit Daemon)
- Beliebiger Client (Kommunikation per REST API)

- Öffentliche Registries wie Docker Hub oder Quay.io.
- Unternehmesinterne / private Registries

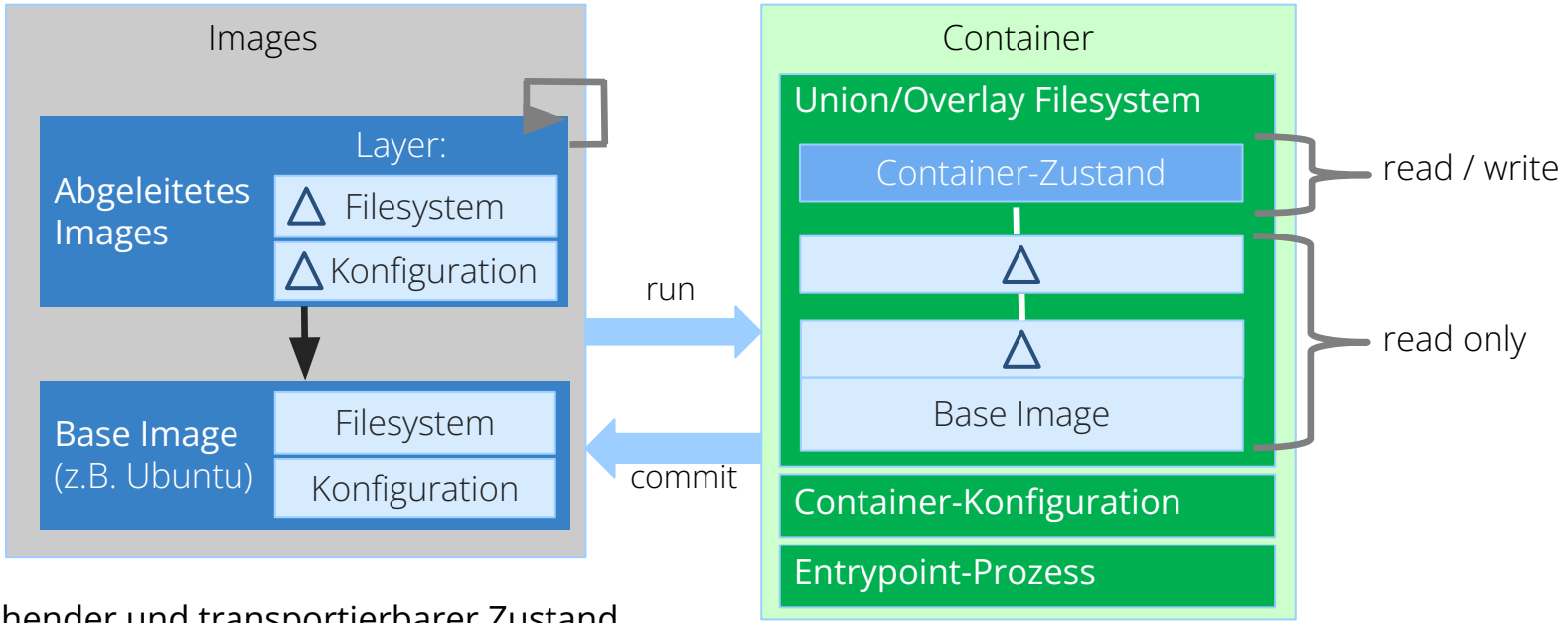
Der Docker Daemon ist die zentrale Steuerungseinheit und läuft direkt als Prozess im Host-Betriebssystem. Er verwaltet alle lokalen Container und Images auf dem Host.

Bildquelle: https://docs.docker.com/engine/docker-overview/

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



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 △ 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.

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

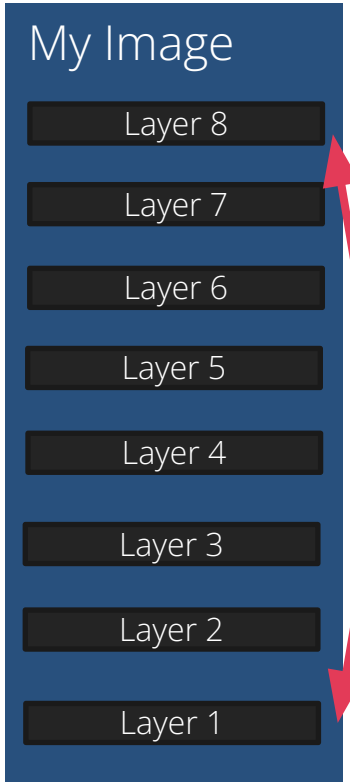
Ein Dockerfile ist somit eine Image-Repräsentation alternativ zu einem physischen Image

(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.



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 –e 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
`docker build -t <image> .`	Build Docker image from „Dockerfile“ with given tag in current directory
`docker images`	Prints all local images

`docker run`
`  -d`
`  -v <volume mounts>`
`  -p <host-port>:<container-port>`
`  <image> <entrypoint process>`
	Run a Docker image: Creates and runs a container.
	▪ in background
	▪ with host directory mounted into the container
	▪ with port forwarding from host to container
	▪ image name (and optional entrypoint process)

`docker run`
`  -ti`
`  <image> /bin/sh`
	Run a Docker image and open a shell within the container
	▪ … with forwarding of local terminal
	▪ Image name and shell (or „`/bin/bash`“)

`docker ps -a`	Prints all containers (without –a = only running containers)
`docker commit <container> qaware/foo`	Store container as local image
`docker kill <container>` `docker rm <container>`	Terminate container (send SIGKILL to entrypoint process) Remove container
`docker rmi -f <image>`	Remove local image

More commands: https://coderwall.com/p/2es5jw/docker-cheat-sheet-with-examples, https://docs.docker.com/reference

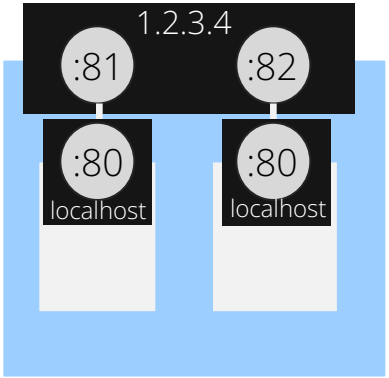
Hilfreiche Kommandos für Container Troubleshooting

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

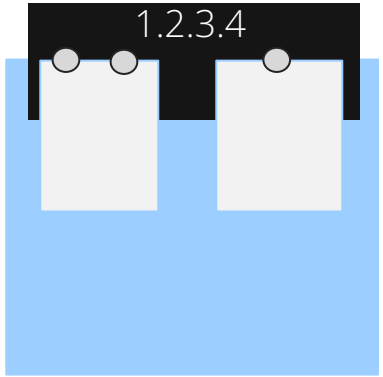
Weitere Kommandos: https://coderwall.com/p/2es5jw/docker-cheat-sheet-with-examples, https://docs.docker.com/reference

Die Docker Networking Modes.

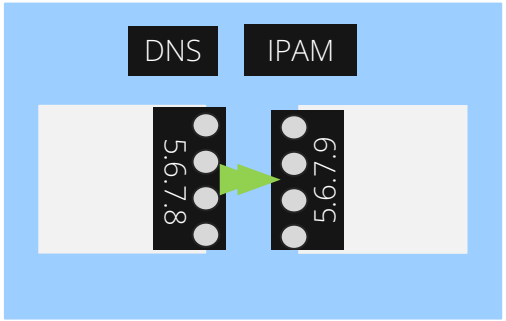
- Docker erlaubt ein getrenntes Netzwerk zwischen docker containern aufzubauen.



Bridge

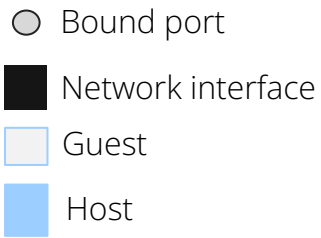


Host

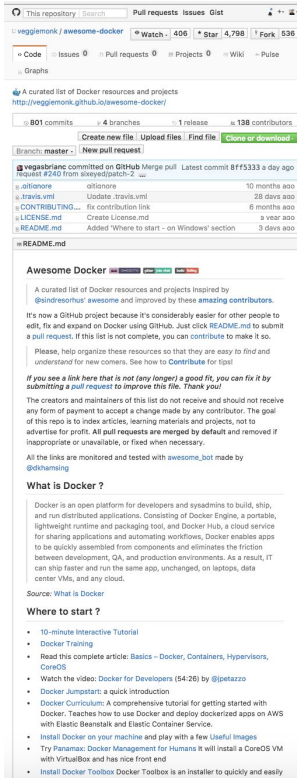


Overlay Network

```
docker network ls
docker network inspect bridge
docker network create --driver overlay multi-host-network
docker network connect multi-host-network container1
```



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



veggiemonk / awesome-docker

Watch 406 · Star 4,798 · Fork 536

Code · Issues 0 · Pull requests 0 · Projects 0 · Wiki · Pulse · Graphs

A curated list of Docker resources and projects
http://veggiemonk.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

vegasbrianc committed on GitHub Merge pull request #240 from sixeyed/patch-2 — Latest commit 8ff5333 a day ago

.altignore	.altignore	10 months ago
.travis.yml	Update .travis.yml	28 days ago
CONTRIBUTING....	fix contribution link	6 months ago
LICENSE.md	Create License.md	a year ago
README.md	Added 'Where to start - on Windows' section	3 days ago

README.md

Awesome Docker

A curated list of Docker resources and projects Inspired by @sindresorhus' 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 *understand* 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 @dkhamsing

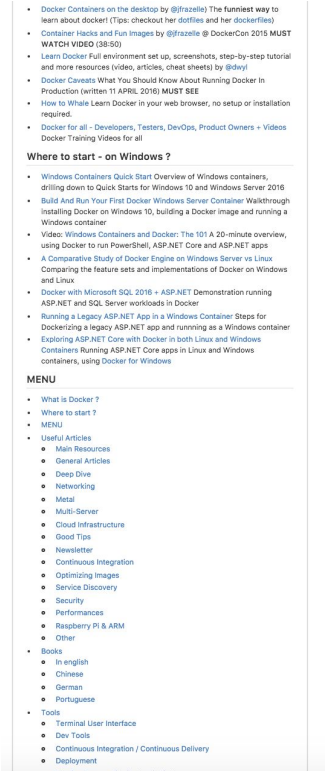
What is Docker ?

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, CoreOS
- Watch the video: Docker for Developers (54:26) by @petazzo
- 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



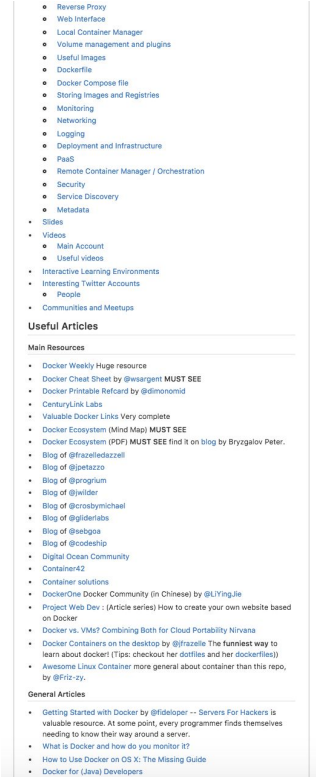
- Docker Containers on the desktop by @jfrazelle) The **funniest** way to learn about docker! (Tips: checkout her dotfiles and her dockerfiles)
- Container Hacks and Fun Images by @jfrazelle @ DockerCon 2015 **MUST WATCH VIDEO** (38:50)
- Learn Docker Full environment set up, screenshots, step-by-step tutorial and more resources (video, articles, cheat sheets) by @dwyl
- Docker Caveats What You Should Know About Running Docker In Production (written 11 APRIL 2016) **MUST SEE**
- How to Whale 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 workloads 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
  - Other
- 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
- PaaS
- Remote Container Manager / Orchestration
- Security
- Service Discovery
- Metadata
- Slides
- Videos
  - Main Account
  - Useful videos
- Interactive Learning Environments
- Interesting Twitter Accounts
  - People
- Communities and Meetups

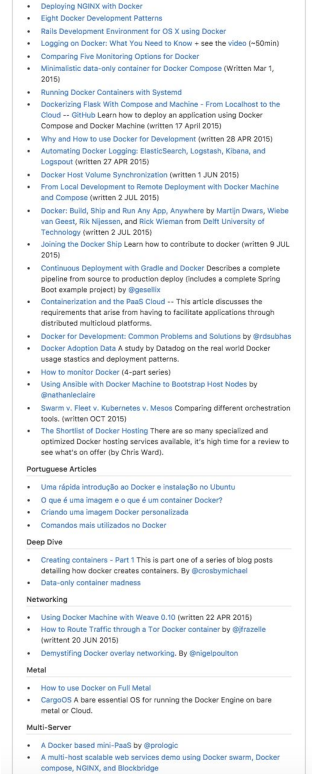
Useful Articles

Main Resources

- Docker Weekly Huge resource
- Docker Cheat Sheet by @wsargent **MUST SEE**
- Docker Printable Refcard by @dimonomid
- CenturyLink Labs
- Valuable Docker Links Very complete
- Docker Ecosystem (Mind Map) **MUST SEE**
- Docker Ecosystem (PDF) **MUST SEE** find it on blog by Bryzgalov Peter.
- Blog of @frazelledazzell
- Blog of @petazzo
- Blog of @progrium
- Blog of @jwilder
- Blog of @crosbymichael
- Blog of @gliderlabs
- Blog of @sebgoa
- Blog of @codeship
- Digital Ocean Community
- Container42
- Container solutions
- DockerOne Docker Community (in Chinese) by @LiYing_lie
- Project Web Dev : (Article series) How to create your own website based on Docker
- Docker vs. VMs? Combining Both for Cloud Portability Nirvana
- Docker Containers on the desktop by @jfrazelle **The funniest way** to learn about docker! (Tips: checkout her dotfiles 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 Development to Remote Deployment with Docker Machine and Compose (written 2 JUL 2015)
- Docker: Build, Ship and Run Any App, Anywhere by Martijn Dwars, Wiebe van Geest, Rik Nijessen, 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 @geseilix
- Containerization and the PaaS Cloud -- This article discusses the requirements that arise from having to facilitate applications through distributed multicloud platforms.
- Docker for Development: Common Problems and Solutions by @rdsubhas
- Docker Adoption Data A study by Datadog on the real world Docker usage stastics and deployment patterns.
- How to monitor Docker (4-part series)
- Using Ansible with Docker Machine to Bootstrap Host Nodes by @nathanleclaire
- Swarm v. Fleet v. Kubernetes v. Mesos Comparing different orchestration tools. (written OCT 2015)
- The Shortlist of Docker Hosting There are so many specialized and optimized Docker hosting services available, it's high time for a review to see what's on offer (by Chris Ward).

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 @crosbymichael
- Data-only container madness

Networking

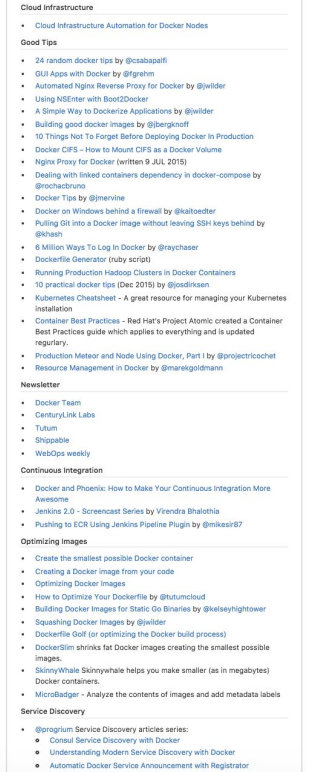
- Using Docker Machine with Weave 0.10 (written 22 APR 2015)
- How to Route Traffic through a Tor Docker container by @jfrazelle (written 20 JUN 2015)
- Demystifing Docker overlay networking. By @nigelpoulton

Metal

- How to use Docker on Full Metal
- CargoOS A bare essential OS for running the Docker Engine on bare metal or Cloud.

Multi-Server

- A Docker based mini-PaaS by @prologic
- 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 @fgrehm
- Automated Nginx Reverse Proxy for Docker by @jwilder
- Using NSEnter with Boot2Docker
- A Simple Way to Dockerize Applications by @jwilder
- Building good docker images by @bergknoff
- 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 @jmervine
- Docker on Windows behind a firewall by @kaitoedter
- Pulling Git into a Docker image without leaving SSH keys behind by @khash
- 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 @josdirksen
- 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 regurlary.
- Production Meteor and Node Using Docker, Part I by @projectricochet
- Resource Management in Docker by @marekgoldmann

Newsletter

- Docker Team
- CenturyLink Labs
- Tutum
- Shippable
- WebOps weekly

Continuous Integration

- Docker and Phoenix: How to Make Your Continuous Integration More Awesome
- Jenkins 2.0 - Screencast Series by Virendra Bhalothia
- Pushing to ECR Using Jenkins Pipeline Plugin by @mikesir87

Optimizing Images

- Create the smallest possible Docker container
- Creating a Docker image from your code
- Optimizing Docker Images
- How to Optimize Your Dockerfile by @tutumcloud
- Building Docker Images for Static Go Binaries by @kelseyhightower
- 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 Registrator

...