

Studiengang Informatik

CLOUD SOLUTIONS – LEKTION 11: DOCKER: CONTAINER ALS CLOUD ENABLING TECHNOLOGY

Vorlesung Frühjahrssemester 2017

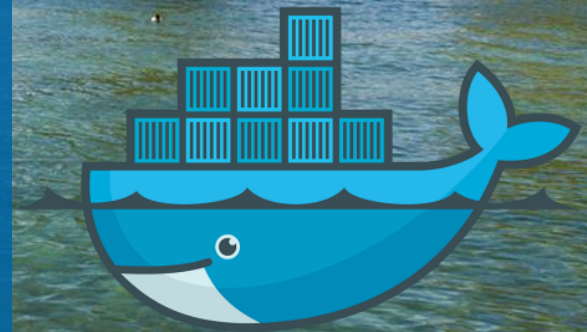
Mirko Stocker / @m_st
Institute for Software
Rapperswil, 8. Mai 2017



HSR

HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz



Agenda (Lektion 11)

■ Einführung in Container und Docker

- Was sind Container
- Wie ist Docker aufgebaut
- Wichtigste Befehle und Beispiele
- Anwendungsszenarien

■ Docker im Betrieb

- Orchestrieren von Docker Containern
- Wahl des Betriebssystems

■ Docker und Cloud Computing

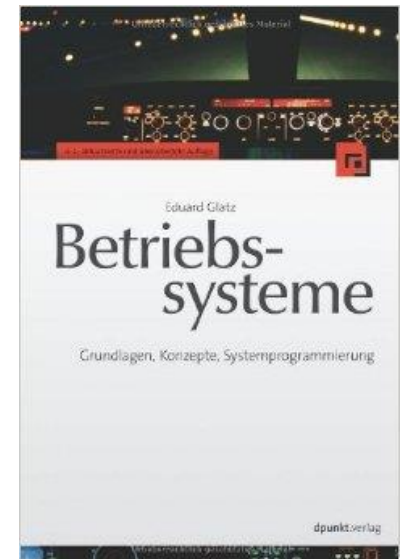
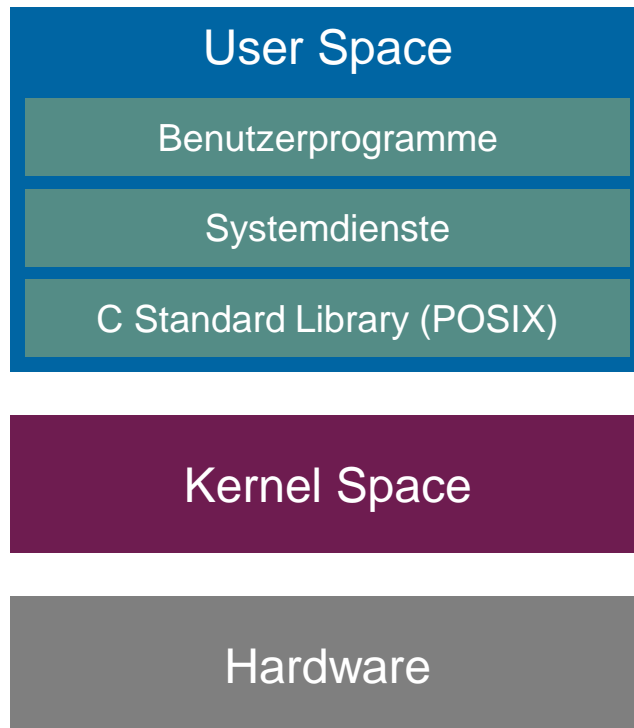
- Cloud-Computing Patterns
- Docker vs. PaaS

- Verstehen was Container aus Betriebssystemssicht sind
- Die wichtigsten Docker-Befehle anwenden können
- Eigene Anwendungen in einem Docker-Container starten
- Vor- und Nachteile von Containern kennen
- Den Unterschied von Docker zu PaaS und VMs erklären können
- Wissen, für welche Aufgaben Docker eingesetzt werden kann

Repetition Betriebssysteme 1

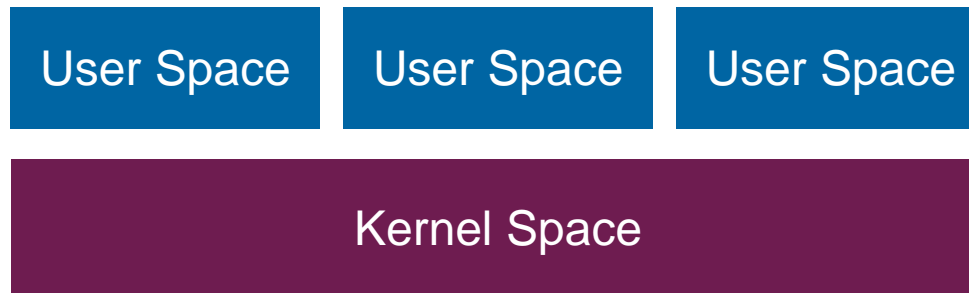
■ Betriebssystem unterteilt virtuellen Adressraum in zwei Bereiche:

- Kernel Space (hier läuft der Kernel, Treiber)
- User Space (Prozesse laufen in virtuellem Adressraum)



Operating-System-Level Virtualization

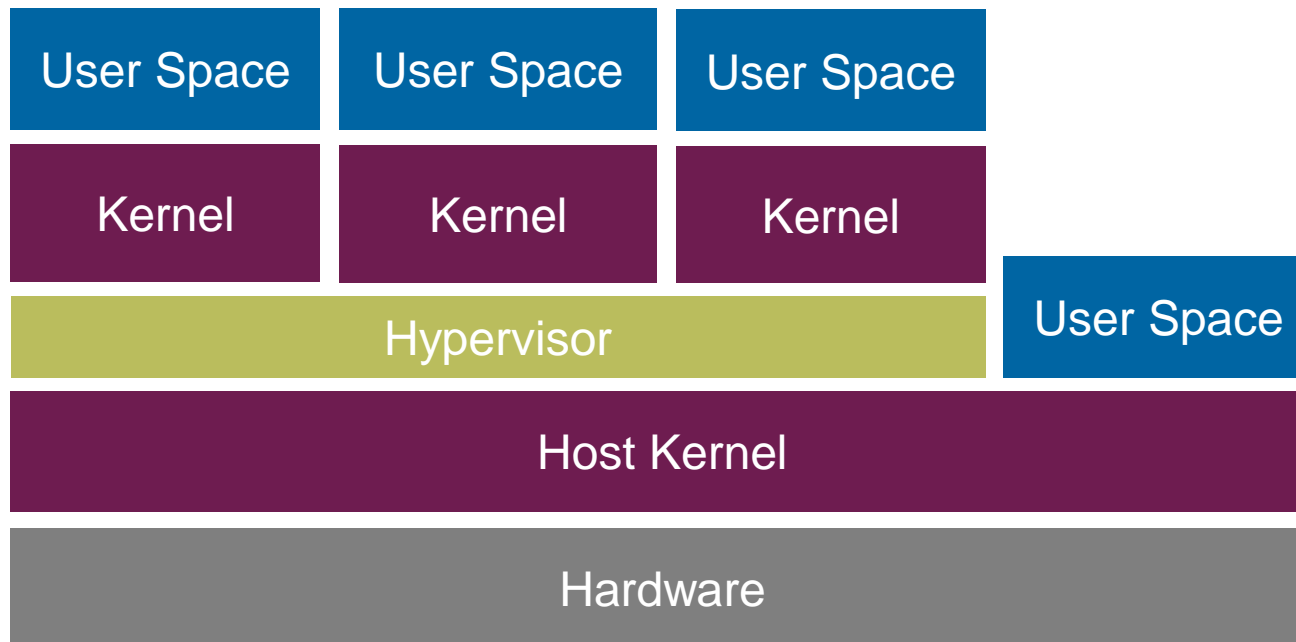
- Mehrere User Spaces
- Gemeinsamer Kernel, aber Isolation zwischen User Spaces



- Operating-System-Level Virtualization auch bekannt als:
 - Container (generisch), Jail (BSD), Zone (Solaris), Virtual Private Server (oft verwendet für Internet Hosting)
- Nicht zu verwechseln mit der besser bekannten Hardware Virtualization

Abgrenzung: Virtual Machine

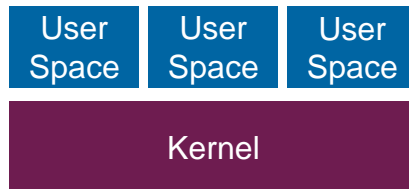
- Container sind *keine* virtuellen Maschinen
- Virtuelle Maschinen basieren auf Hardware-Virtualisierung
- Teilweise Emulation, Zwischenschicht zur Hardware:



Container vs. VM

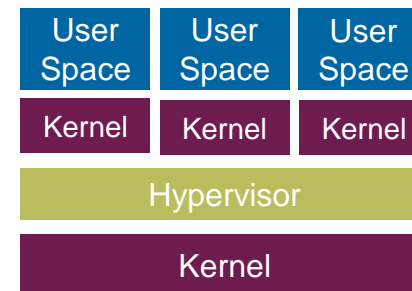
■ Container

- Derselbe Kernel ist für alle Container zuständig
- Kein Overhead, da nur Prozess
- Speichereffizient, Ressourcen können geteilt werden



■ Virtuelle Maschine

- Auch fremde Betriebssysteme können in einer VM laufen
- Overhead durch Virtualisierung
- Hoher Speicherverbrauch (RAM und Disk)



■ Keine Konkurrenz, da unterschiedliche Anwendungsszenarien

Vorteile von Containern gegenüber nacktem System

■ Isolation

- Applikationen können nicht aus Container ausbrechen (Sandboxing)
- Inkompatible Applikationen (z.B. unterschiedliche Ruby- oder Python Versionen) nebeneinander auf demselben System

■ Skalierbarkeit

- Mehrere Instanzen oder Versionen einer Applikation starten
- Regressionssicherheit: Container beinhalten ihre Dependencies

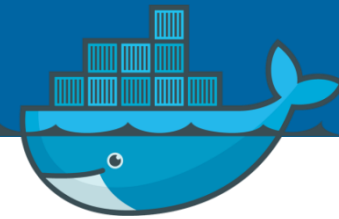
■ Deployment

- Portierbar: Container läuft auf dem System des Entwicklers A gleich wie bei Entwickler B und auf dem Produktivsystem
- Container kann einfach entsorgt werden
- Migration von Applikationen zwischen Servern vereinfacht

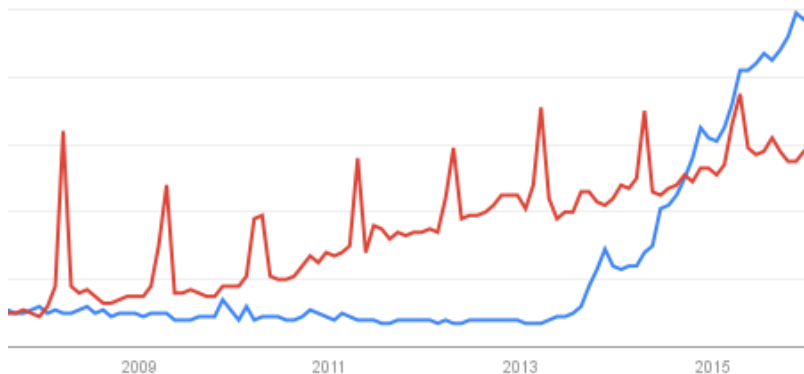
Google: 2 Milliarden Containerstarts pro Woche

<http://googlecloudplatform.blogspot.ch/2014/06/an-update-on-container-support-on-google-cloud-platform.html>

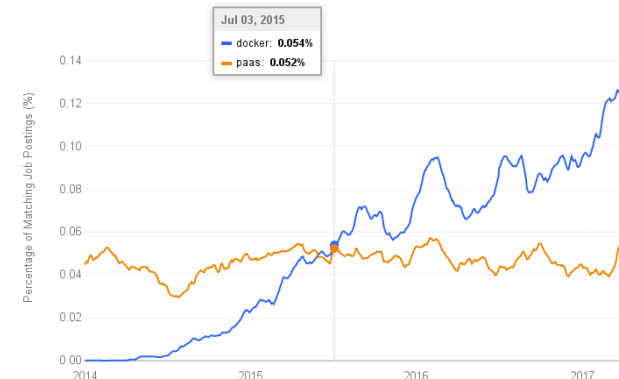
google.com/datacenters



- «*Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications.*» <https://www.docker.com>
- Open-Source Containerlösung
- Erster Release März 2013 von dotCloud (PaaS Anbieter)
- Abspaltung von dotCloud als Docker Inc. (65 Mio Venture Capital)
- Mehrheitlich in Go geschrieben



Google Trends: Docker (blau) vs PaaS (rot)

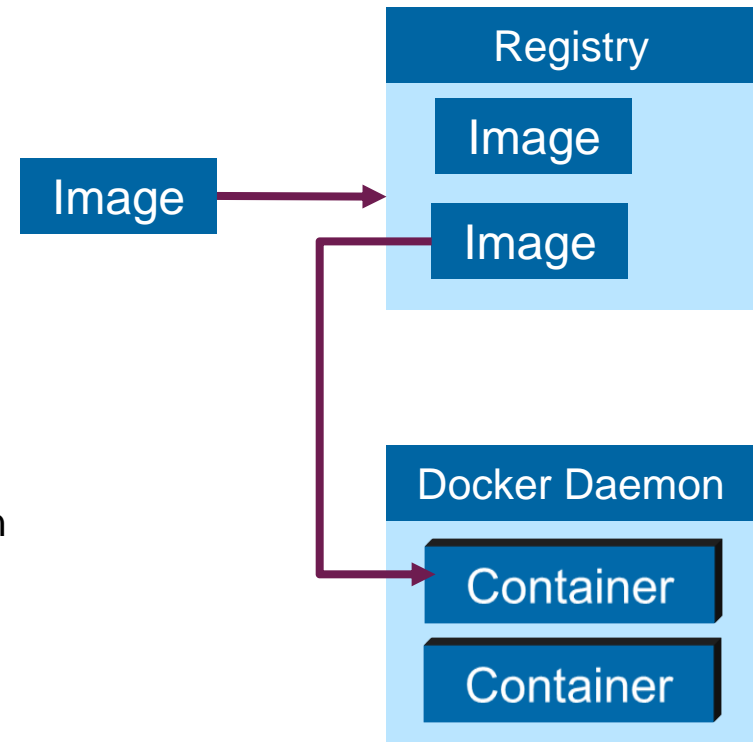


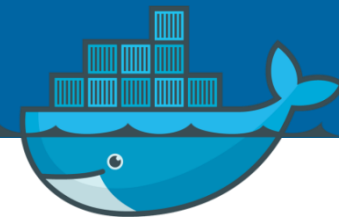


- **Docker ist mehr als eine Containerimplementierung**

- **Umfasst verschiedene Komponenten und Dienste:**

- Docker Engine
 - Client: docker Commandline Tool
 - Server: Daemon-Prozess
- Docker Images
 - Templates für Container
- Docker Container
 - Instanzierte, laufende Images
- Docker Registry/Hub
 - SaaS Registry um Images zu verwalten



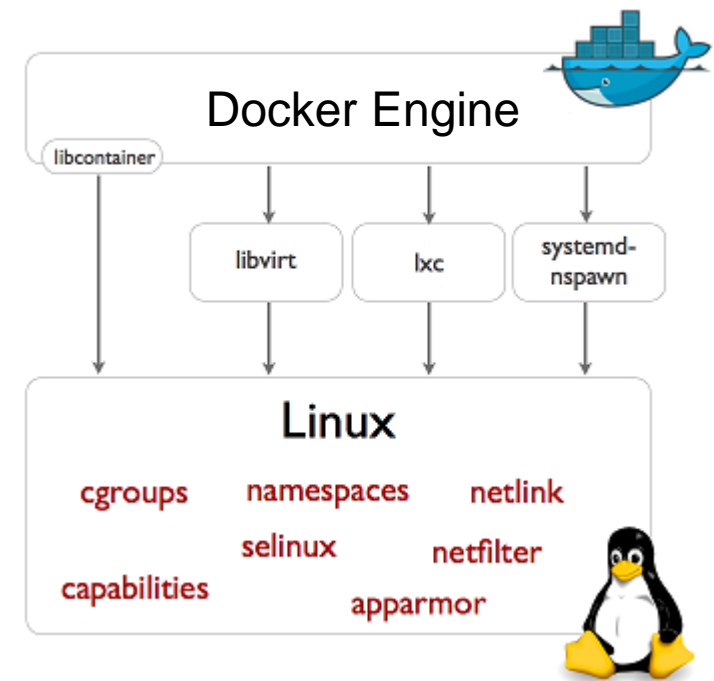
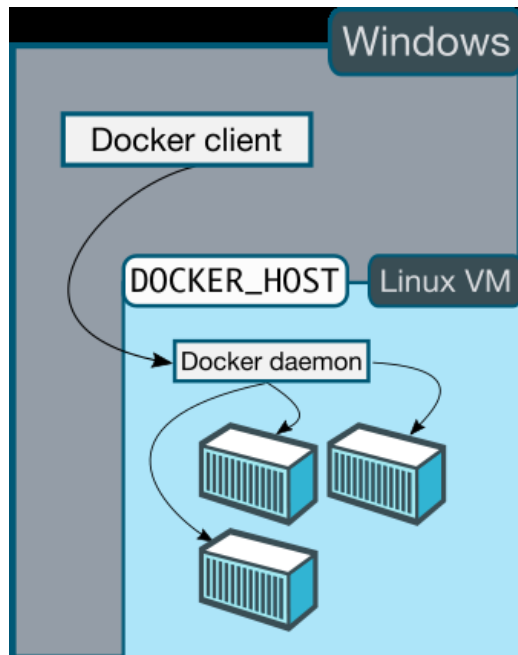


Engine

■ Docker Engine

- Front-end für verschiedene Container-Implementierungen

■ Linux basiert

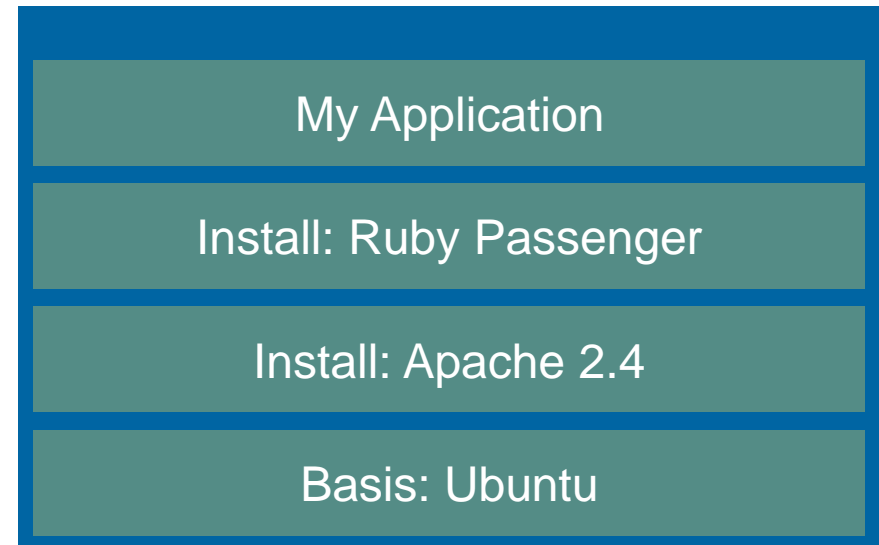


http://www.infoq.com/news/2014/03/docker_0_9



Image

- Docker Images sind Read-Only Templates
- Ein Image besteht aus mehreren Layern
- Ein Layer entspricht sozusagen einem «Diff des Dateisystems» (wie genau ist abhängig vom gewählten Storage-Driver)
- Beim Instanzieren eines Containers werden die Layer durch ein Union-Filesystem verschmolzen
- Bei Änderungen müssen nur die geänderten Layer neu erstellt werden, gleiche Layer werden wiederverwendet
- Änderungen am laufenden System können *committed* werden.
- «*Docker is Git for Deployment*»





■ Beim Ausführen eines Images wird ein Container instanziiert

```
% docker run -i -t ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
f3c84ac3a053: Downloading [=====] 82.09 MB/197.2 MB 44s
a1a958a24818: Verifying Checksum
9fec74352904: Verifying Checksum
d0955f21bf24: Download complete
511136ea3c5a: Already exists
```

Fehlende Layer werden vom Dockerhub heruntergeladen

```
% docker run -i -t ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
f3c84ac3a053: Pull complete
f3c84ac3a053: Download complete
a1a958a24818: Download complete
9fec74352904: Download complete
d0955f21bf24: Download complete
511136ea3c5a: Download complete
Status: Downloaded newer image for ubuntu:latest
root@e5f6103f238a:/#
```

■ run -i -t startet eine interaktive Shell im Container



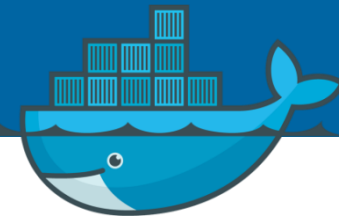
■ Was ist genau passiert?

1. Docker sucht das Image lokal, und lädt es andernfalls herunter
2. Ein Container wird aus den Read-Only-Layern erstellt
3. Zuerst kommt ein Read-Write Layer
4. Netzwerk-Interface/Bridge wird erstellt
5. Prozess im Container wird gestartet (/bin/bash)
6. Terminal Input-Output wird mit dem Container verbunden

■ Beim Beenden des Prozesses (schliessen der Shell) wird der Container beendet

■ Beim nächsten `docker run` wird ein frischer Container gestartet

- Gestoppte Container sind nicht verloren, aber um Änderungen dauerhaft im Image zu speichern sollten wir diese committen.



■ Im Read-Write Layer können wir ganz normal Software installieren:

Host
Shell

```
% docker run -t -i ubuntu /bin/bash
root@7f34e3f169ed:/# apt-get update
...
Fetched 20.7 MB in 18s (1113 kB/s)
Reading package lists... Done
root@7f34e3f169ed:/# apt-get install ruby
...
root@7f34e3f169ed:/# ruby -e 'puts "Hello HSR"'
Hello HSR
```

Container Shell

■ Und committen, wenn wir damit fertig sind:

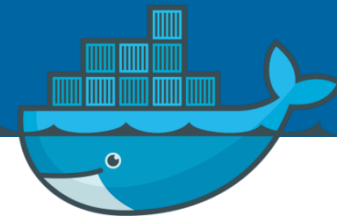
```
% docker commit 7f34e3f169ed mirko/ruby
cd60eaab395655b76a0e7caa017f41dfd8036bb5aefcbbc23a
```

Image Name

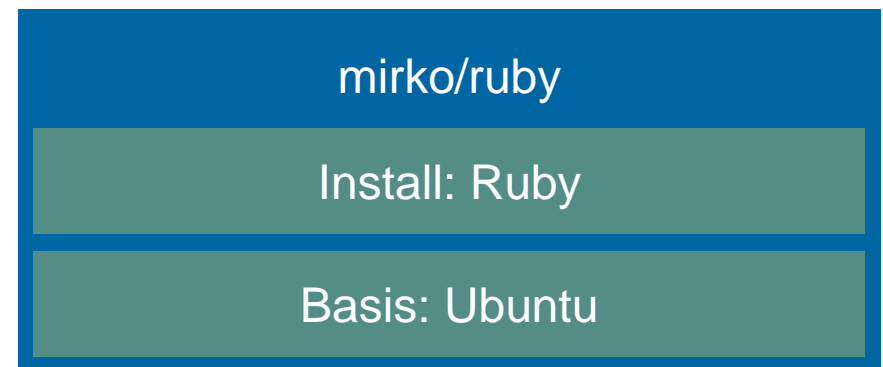
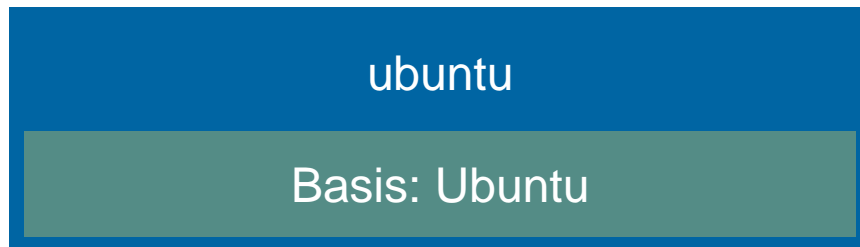
■ Dadurch entsteht ein neues Image:

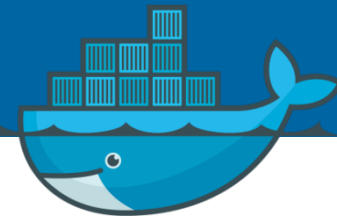
```
% docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
mirko/ruby	latest	cd60eaab3956	3 seconds ago	226.3 MB



Durch das Committed wurde der Read-Write Layer festgeschrieben:





■ Container können interaktiv oder als Daemon erstellt werden:

```
% docker run -i -t mirko/ruby /bin/bash
root@1d4ad600cd36:/# ...
```

Interaktiv

■ Daemon-Option -d gibt ID des Containers zurück:

```
% docker run -d apache /bin/bash -c "apache2ctl -D FOREGROUND"
Bcbb557c1f953cfd24e0b7af106b122fbcef08023e3e5b0b29b57ea058615ff9
```

Daemon

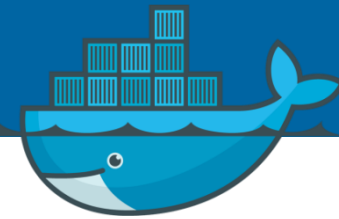
```
% docker ps
CONTAINER ID      IMAGE          COMMAND                  ...      NAMES
bcb557c1f95      apache:latest  "/bin/bash -c 'apach     ...      jovial_goldstine
```

■ Kann mit docker start/stop gestartet/gestoppt werden

```
% docker start jovial_goldstine
jovial_goldstine
```

Mit Container-Name oder Container-Id

```
% docker stop jovial_goldstine
jovial_goldstine
```



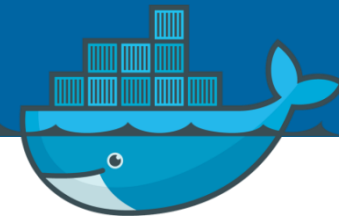
- **Container sind in einem virtuellen Subnetz und nicht von ausserhalb des Hosts erreichbar, müssen aber häufig Dienste im Netzwerk anbieten**
- **Lösung: Port-Mapping**
 - Port des Hosts wird an Port des Containers weitergeleitet («published»)
 - Verschiedene Varianten, z.B. an bestimmtes Interface des Hosts binden

```
% docker run --help
...
-p, --publish=[]          Publish a container's port to the host
                           format: ip:hostPort:containerPort | ip::containerPort |
                           hostPort:containerPort | containerPort
                           (use 'docker port' to see the actual mapping)
```

```
% docker port --help

Usage: docker port [OPTIONS] CONTAINER [PRIVATE_PORT[/PROTO]]

List port mappings for the CONTAINER, or lookup the public-facing port that is NAT-ed to
the PRIVATE_PORT
```



■ Wie können Container andere Container finden?

■ Lösung: Container-Networking

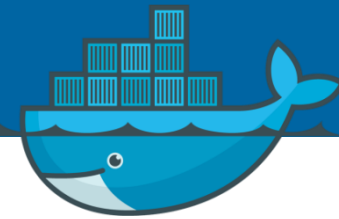
- Name des Containers wird in Hosts-File eingetragen

```
% docker run --help
...
--network string          Connect a container to a network (default "default")
```

```
% docker network create mynet
```

```
% docker run --name linked --network mynet -p 80 -t -i ubuntu /bin/bash
```

```
% docker run --network mynet -t -i ubuntu /bin/bash
root@d1c8cac33ee0:/# ping linked
PING linked (172.22.0.2) 56(84) bytes of data.
64 bytes from linked.mynet (172.22.0.2): icmp_seq=1 ttl=64 time=0.069 ms
```



- **Containers sollen unveränderbar (stateless) sein**
- **Volumes enthalten persistente Daten**
 - als separates Filesystem angelegt
 - vom Host gemountet
 - von anderem Container übernommen

```
% docker run --help
...
-v, --volume=[]          Bind mount a volume (e.g., from the host: -v /host:/container,
                          from Docker: -v /container)
--volumes-from=[]       Mount volumes from the specified container(s)
```

- **In diesen Verzeichnissen abgelegt Dateien gehen nicht verloren**
- **Beispiele: Datenbankcontainer**



- Ein Container, der Daten eines Volumes bereitstellt
- Wird in Container gemountet

```
docker create -v /data --name data library/ubuntu
```

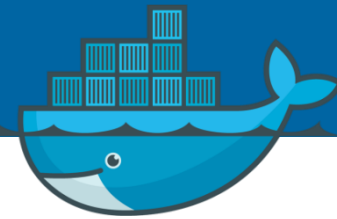
↑
create, der Container muss nicht gestartet werden

```
docker run -d --volumes-from data --name www1 apache
```

```
docker run -d --volumes-from data --name www2 apache
```

```
docker run -d --volumes-from data --name www3 apache
```

- Vor- oder Nachteil: Unabhängig von Host, alle Daten liegen in Containern



- Docker ist ein umfangreiches Tool und bietet viele Möglichkeiten
- Sehr gute Dokumentation auf <https://docs.docker.com>

```
Usage: docker COMMAND

A self-sufficient runtime for containers

Options:
  -D, --debug            Enable debug mode
  --help                Print usage
  -H, --host list        Daemon socket(s) to connect to
  -l, --log-level string  Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
  --tl                    Use TLS; implied by --tlverify
  --tlcacert string      Trust certs signed only by this CA (default "/home/nisto/.docker/ca.pem")
  --tlcert string        Path to TLS certificate file (default "/home/nisto/.docker/cert.pem")
  --tlskey string        Path to TLS key file (default "/home/nisto/.docker/key.pem")
  --tlverify             Use TLS and verify the remote
  -v, --version          Print version information and quit

Management Commands:
  container             Manage containers
  image                 Manage images
  network              Manage networks
  node                 Manage Swarm nodes
  plugin               Manage plugins
  secret               Manage Docker secrets
  service              Manage services
  stack                Manage Docker stacks
  swarm                Manage Swarm
  system               Manage Docker
  volume               Manage volumes

Commands:
  attach               Attach local standard input, output, and error streams to a running container
  build                Build an image from a Dockerfile
  commit              Create a new image from a container's changes
  cp                  Copy files/folders between a container and the local filesystem
  create              Create a new container
  diff                Inspect changes to files or directories on a container's filesystem
  events              Get real time events from the server
  exec                Run a command in a running container
  export              Export a container's filesystem as a tar archive
  history             Show the history of an image
  images              List images
  import              Import the contents from a tarball to create a filesystem image
  info                Display system-wide information
  inspect             Return low-level information on Docker objects
  kill                Kill one or more running containers
  load                Load an image from a tar archive or STDIN
  login               Log in to a Docker registry
  logout              Log out from a Docker registry
  logs                Fetch the logs of a container
  pause              Pause all processes within one or more containers
  port                List port mappings or a specific mapping for the container
  ps                  List containers
  pull                Pull an image or a repository from a registry
  push                Push an image or a repository to a registry
  rename              Rename a container
  restart             Restart one or more containers
  rm                  Remove one or more containers
  rmi                 Remove one or more images
  run                 Run a command in a new container
  save                Save one or more images to a tar archive (streamed to STDOUT by default)
  search              Search the Docker Hub for images
  start               Start one or more stopped containers
  stats               Display a live stream of container(s) resource usage statistics
  stop                Stop one or more running containers
  tag                 Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top                 Display the running processes of a container
  unpause             Unpause all processes within one or more containers
  update              Update configuration of one or more containers
  version             Show the Docker version information
  wait                Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
```



■ Unsere Möglichkeiten bisher:

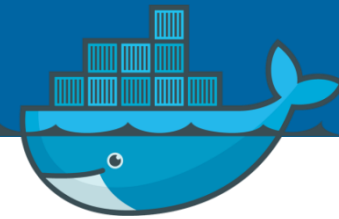
- Container starten, Änderungen in Shell machen
- Container committen

■ Probleme dieses Ansatzes:

- Nicht nachvollziehbar für Dritte (wurde noch ein Trojaner installiert?)
- Bei Änderung mühsam anzupassen (nochmals durchspielen?)

■ Lösung: Dockerfile

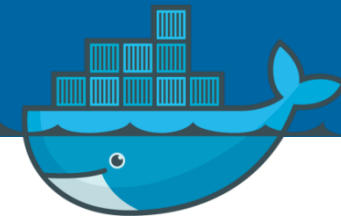
- Script / Beschreibung eines Docker-Images
- Herunterladen, editieren, Image aus Dockerfile erstellen
- Alle Schritte in Plain-Text nachvollziehbar
- Beinhaltet auch Informationen zu Maintainer, Ports, Volumes, etc.



■ Dockerfile für unser Ruby Image

```
% cat Dockerfile
FROM ubuntu:latest ← Basisimage im Format name:tag
MAINTAINER Mirko Stocker
RUN apt-get update && apt-get install -y ruby ← Ausführen
EXPOSE 80 ← Ports freigeben
CMD ["ruby", "-run", "-ehttpd", ".", "-p80"] ← Anwendung starten
```

■ Auch Volumes, etc. können angegeben werden.



■ Dockerfile für unser Ruby Image:

```
% docker build -t mirko/ruby .
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:latest
---> d0955f21bf24
Step 1 : MAINTAINER Mirko Stocker
---> Running in 286df2f0b973
---> 80796d2e5445
Removing intermediate container 286df2f0b973
Step 2 : RUN apt-get update && apt-get install -y ruby
---> Running in c61a5379db60
Ign http://archive.ubuntu.com trusty InRelease
Ign http://archive.ubuntu.com trusty-updates InRelease
...
Successfully built 7601e6e0ce8f
```


■ Unser Image kann danach wie gewohnt benutzt werden.

```
% docker images
```








REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
mirko/ruby	latest	7601e6e0ce8f	3 minutes ago	226.3 MB



- Docker Hub ist ein öffentliches Repository von Images
- <https://hub.docker.com>
- 100'000+ Images verfügbar
- Kostenpflichtige private Registry
- Continuous-Integration für Images
- Web-Hooks

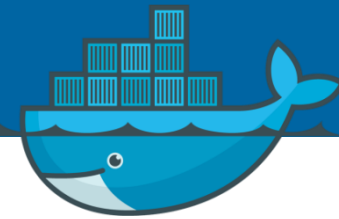


Explore Official Repositories

 busybox official	573 STARS	10M+ PULLS	> DETAILS
 nginx official	2.4K STARS	10M+ PULLS	> DETAILS
 ubuntu official	3.4K STARS	10M+ PULLS	> DETAILS
 swarm official	232 STARS	10M+ PULLS	> DETAILS
 registry official	675 STARS	10M+ PULLS	> DETAILS
 redis official	1.8K STARS	10M+ PULLS	> DETAILS
 mysql official	2.0K STARS	10M+ PULLS	> DETAILS

Anwendungsszenarien für Docker

- **Eigenes PaaS aufbauen (Baidu, Yandex)**
- **Entwicklungsumgebung lokal erstellen (z.B. LAMP mit WordPress)**
- **Applikationen mit exotischen/veralteten Dependencies installieren**
- **Continuous Integration: Tests in einem Container starten, z.B. um Installationsscripts zu testen (Atlassian, Yelp)**
- **Continuous Delivery: Erfolgreicher Build wird in einem Image festgehalten/archiviert und kann einfach benutzt werden (Spotify, Ebay)**
- **Mehr Docker Use Cases:**
<https://www.docker.com/resources/usecases/>
- **Mehr Ideen für Docker Development Patterns:**
<http://www.hokstad.com/docker/patterns>



- Container isolieren Applikationen (Filesystem, Prozess, Netzwerk)
- Container können **verlinkt** werden um miteinander zu kommunizieren
- **Port-Mapping** verbindet Container mit der Aussenwelt
- Container werden aus **Images** erzeugt
- Images können interaktiv oder durch ein **Dockerfile** erstellt werden
- **Registries** wie der **DockerHub** hosten die Images

Docker erlaubt es uns, Software auf einer isolierten, flexiblen und portbierbaren Infrastruktur zu deployen ohne dabei unnötig Ressourcen zu verbrauchen.

Docker im Betrieb

■ Traditionelles Vorgehen (leicht überspitzt):

- Software wird von Entwicklern geschrieben
- Wenn dieser fertig sind, «werfen sie diese über den Zaun» zu den «Betriebsleuten»
- Software wird installiert, produktiv geschaltet, überwacht, getuned

■ Probleme

- Entwickler geben die Verantwortung ab
- Änderungen sind langsam und teuer
- Was macht die Software genau? Welche Abhängigkeiten hat sie?
- Aufwändiges Change-Management

«Never change a running system», aber unser System ändert sich stetig!

Deployment von Software

■ Wir wissen es besser:

- Agile Development
- Continuous Integration

■ DevOps schliesst den Kreis von der Entwicklung zur Produktion



■ Auslegungen von DevOps:

- Dev und Ops arbeiten in einem Team zusammen
- Dev macht selbst Ops
- Ops verwendet dieselben Techniken wie Dev (CI, SCM, Agile, etc.)

■ Docker erleichtert DevOps massgeblich

- Container läuft auf Entwicklerlaptop wie auch auf dem Produktivsystem
- Isolation der Applikationen; der Betrieb mehrerer Applikationen ist also viel einfacher
- Installation durch Dockerfile sozusagen automatisiert und dokumentiert



■ Ablauf im Betrieb mit Docker:

- Container aus Repository holen
- Container starten

■ Neue Probleme treten auf:

- Welche Container müssen ins selbe Netzwerk?
- Welche Ports werden wie gemappt?
- Anstelle eines Servers habe ich plötzlich dutzende, mit eigenen Filesystemen und Volumes!

Vom Regen in die Traufe?

Docker Container Management

- **Tools zur Verwaltung von Docker-Containern entstehen im Moment am Laufmeter:**
 - [Apache Mesos](#)
 - [Atomic](#)
 - [Deis](#)
 - [Fleet](#)
 - [Panamax](#)
 - [Shipper](#)
 - [VNS3](#)
- **Von Docker selbst:**
 - Docher [Machine](#), [Compose](#) und [Swarm](#)
- **Auch schon erste Alternativen zu Docker: [Rocket](#)**
- **Und Pläne zur Standardisierung: [Open Container Initiative](#)**

■ Ziel: möglichst schnell eine Docker-fähige Maschine zu erstellen

- Amazon Web Services, Google Compute Engine, Microsoft Azure, Digital Ocean und weitere sind unterstützt

■ Beispiel Digital Ocean:

```
$ docker-machine create \  
  --driver digitalocean \  
  --digitalocean-access-token 0ab77166d407f479c6701652cee3a46830fef88b8199722b87 \  
  staging  
INFO[0000] Creating SSH key...  
INFO[0000] Creating Digital Ocean droplet...  
INFO[0002] Waiting for SSH...  
INFO[0085] "staging" has been created and is now the active machine  
INFO[0085] To point your Docker client at it, run this in your shell: eval "$(docker-  
machine env staging)"
```

■ In wenigen Minuten ist eine neue Docker-fähige Maschine konfiguriert

Docker Compose

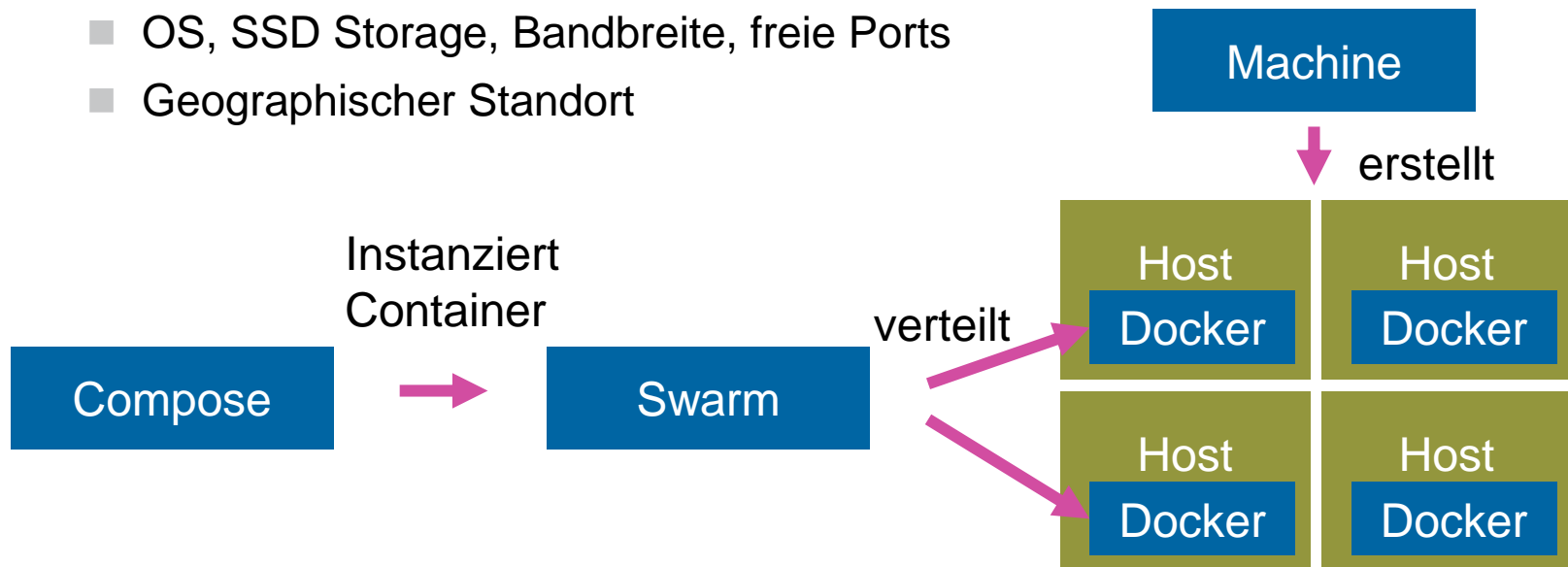
- Löst das Problem, dass eine Anwendung häufig aus mehreren Containern besteht (Trend zu Microservices-Architekturen)
- Konfigurationsfile, um Container zu definieren

```
wordpress:
  image: wordpress
  links:
    - db:mysql
  ports:
    - 8080:80
db:
  image: mariadb
  environment:
    MYSQL_ROOT_PASSWORD: mysecret
```

```
$ docker-compose up
Creating app_db_1...
Creating app_wordpress_1...
Attaching to app_db_1, app_wordpress_1
...
wordpress_1 | [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
```

Docker Swarm

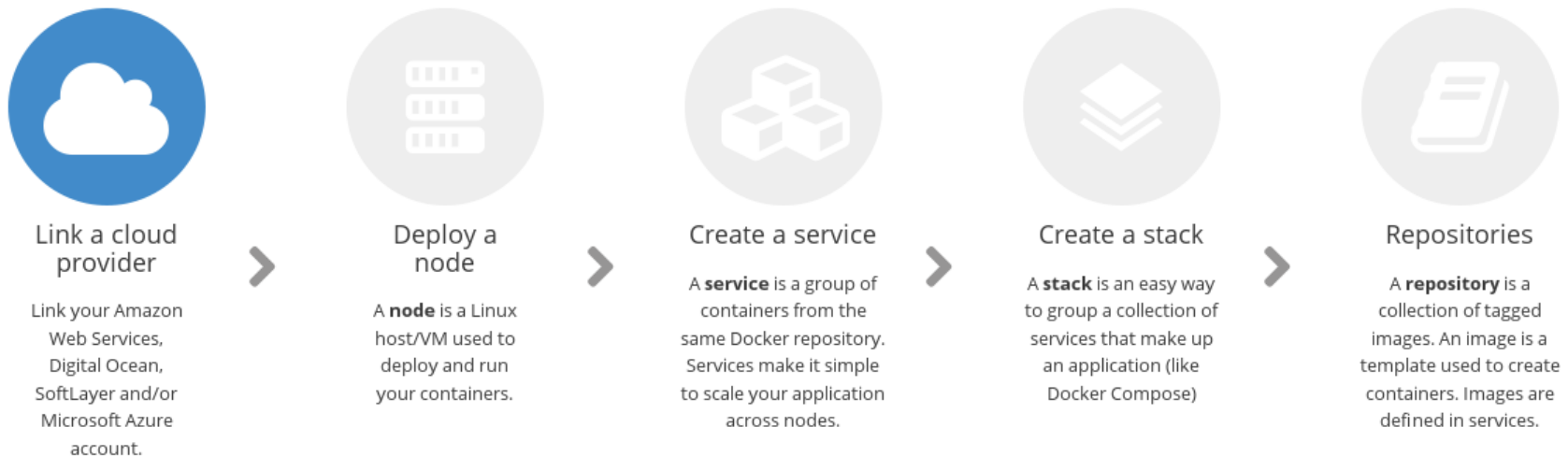
- **Scheduler um Container auf Hosts zu verteilen**
- **Verschiedene Strategien:**
 - Random
 - BinPacking (möglichst dicht packen, nach dem [bin packing problem](#))
- **Filter: Key-Value Paare die mit einem Host assoziiert werden**
 - OS, SSD Storage, Bandbreite, freie Ports
 - Geographischer Standort



- Web-basiertes, bei Docker gehostetes, UI für Docker-Management
- Bis vor kurzem bekannt unter Tutum, von Docker übernommen:

“Docker hosting for your containers. Run, monitor and scale your apps. AWS-like control, Heroku-like ease.”

- Lässt sich mit eigener Infrastruktur oder IaaS-Anbietern einsetzen («self-hosted node»)



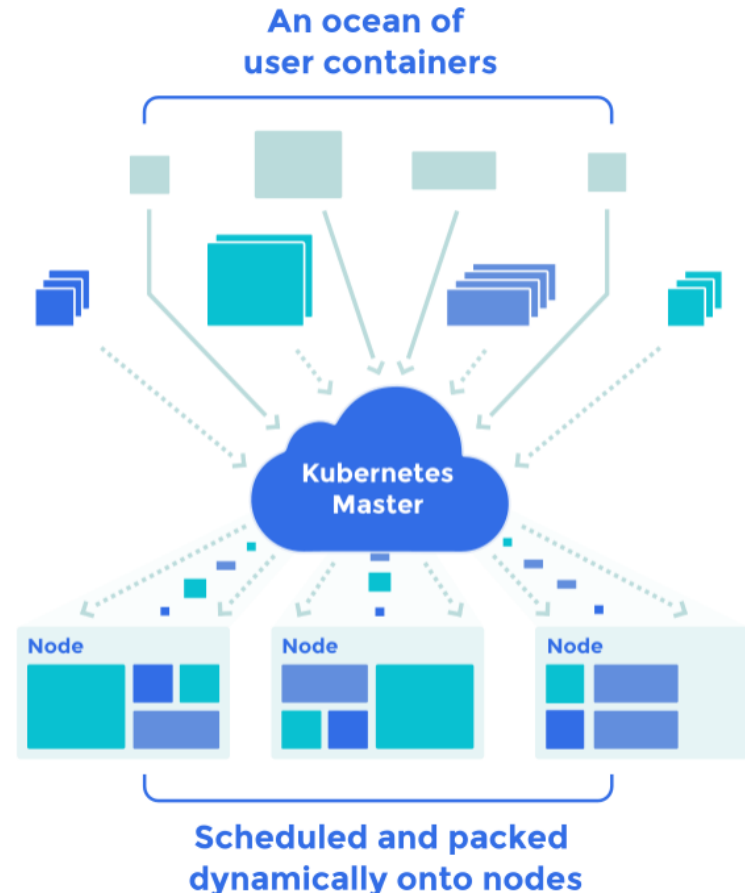
Welches OS für Docker?

- Bei klassischen Hosts oder VMs kümmert sich ein Administrator um Updates und Systempflege
- In Containern unnötig, stattdessen wird der Container einfach auf einem aktualisierten Basissystem neu erstellt und der alte Container entsorgt
- Man spricht in diesem Zusammenhang häufig auch von «Pets versus Cattle»
 - Zu Pets trägt man Sorge, Cattle sind Nutzvieh und werden entsorgt sobald sie ihren Zweck erfüllt haben¹
- Welche Linux Distribution eignet sich um Container zu betreiben?
 - Bekannte Distributionen enthalten häufig zu viel Ballast
 - Alternativen: [CoreOS](#) (Google und VC funded) und [Atomic](#) (RedHat)
 - Enthalten nur die für Container-Betrieb nötigen Tools und häufig noch spezielle Management- und Orchestration-Tools.

¹ <http://it20.info/2012/12/vcloud-openstack-pets-and-cattle/>

Google Kubernetes

- Framework zur Container-Orchestrierung von Containern
- Open Source von Google
- Replikation, Service-Discovery, automatisches Scheduling und Skalierung
- Als Google Container Engine von Google als Service angeboten
 - Neben Google App Engine (PaaS) und Google Compute Engine (IaaS)
- Nächste Woche im Detail!



Docker im Cloud Computing



■ Welchem Processing Offering entsprechen Container?

- Execution Environment
- Hypervisor
- ~~Map-Reduce~~

■ Kein Hypervisor, aber eine Art Execution Environment:

- Common functionality is summarized in an Execution Environment **providing functionality in platform libraries** to be used in custom application implementations and in the form of the **middleware**. The environment, thus, executes custom application components and provides common functionality for data storages, communication etc.

■ Eventuell ein neues Pattern?

■ PaaS bietet eine Plattform (Abstraktion)

- Grössere Abhängigkeit zu Provider
- Viele Aspekte vordefiniert (Scaling, Monitoring)
- PaaS kann im Hintergrund Docker verwenden

■ Docker entstand aus einer PaaS (dotCloud)

- Mehr Freiheit und Flexibilität
- Höherer Aufwand für Installation und Wartung
- Auch für Stateful-Services (non-[12-Factor Apps](#)) geeignet

■ Abwägung zwischen Komfort (PaaS) und Kontrolle (Docker)

- Kombination möglich
- Kann eigenes PaaS mit Docker bauen: [Dokku](#), [Deis](#)

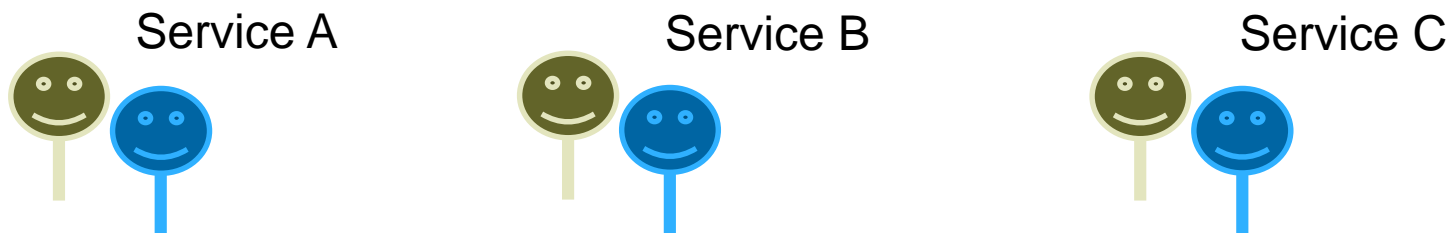
Trend: Container im Microservices-Architekturstil

■ Martin Fowler schreibt über Microservices:

[..] developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. [..] These services are built around business capabilities and independently deployable by fully automated deployment machinery.

■ Unterstützt durch DevOps und Docker

- Agile Teams erstellen und deployen einzelne Services
- Services sind isoliert (inkl. Backend) und bringen ihre Dependencies mit
- Hohe Iterationsgeschwindigkeit dank kurzer Deploymentzeit



■ Mehr zu Microservices in Woche 13

■ Immer mehr Cloud Anbieter unterstützen Docker

- IBM BlueMix
- Microsoft Azure
- Google Container Engine
- RedHat
- CloudFoundry

■ Security und Multitenancy-Fähigkeit lässt noch zu wünschen übrig:

- Von Docker verwendete Kernel-Features bringen gewisse Sicherheit
- Docker-Daemon muss als Root laufen → nur vertrauenswürdigen Usern Zugriff geben
- Verbesserungen in Arbeit ([Docker Security Dokumentation](#), [Docker Secure Deployment Guidelines](#))

Container als Cloud Enabling Technology

- **Operating-System-Level Virtualization ist nichts Neues ...**
- **... aber in einer Welt von DevOps und Microservices ein nützlicher Baustein.**
- **Docker ist so erfolgreich, weil es viele nützliche Tools mitbringt und ein ganzes Ökosystem geschaffen hat.**
- **Container und Docker sind klare Cloud-Enabling Technologies und können auch ein Cloud-Offering sein (siehe Google).**

- Was unterscheidet ein Image von einem Container?
- Warum wird ein Port Mapping benötigt?
- Wie heisst das Docker Konzept, mit dem Container einander finden/aufrufen können?
- Wozu dient das Dockerfile?
- Welches ist der Hauptunterschied zwischen Containern und VMs?
- Was ist die Grundidee von DevOps?
- Worauf muss ich achten, wenn ich ein PaaS auf Basis von Docker aufbauen möchte?