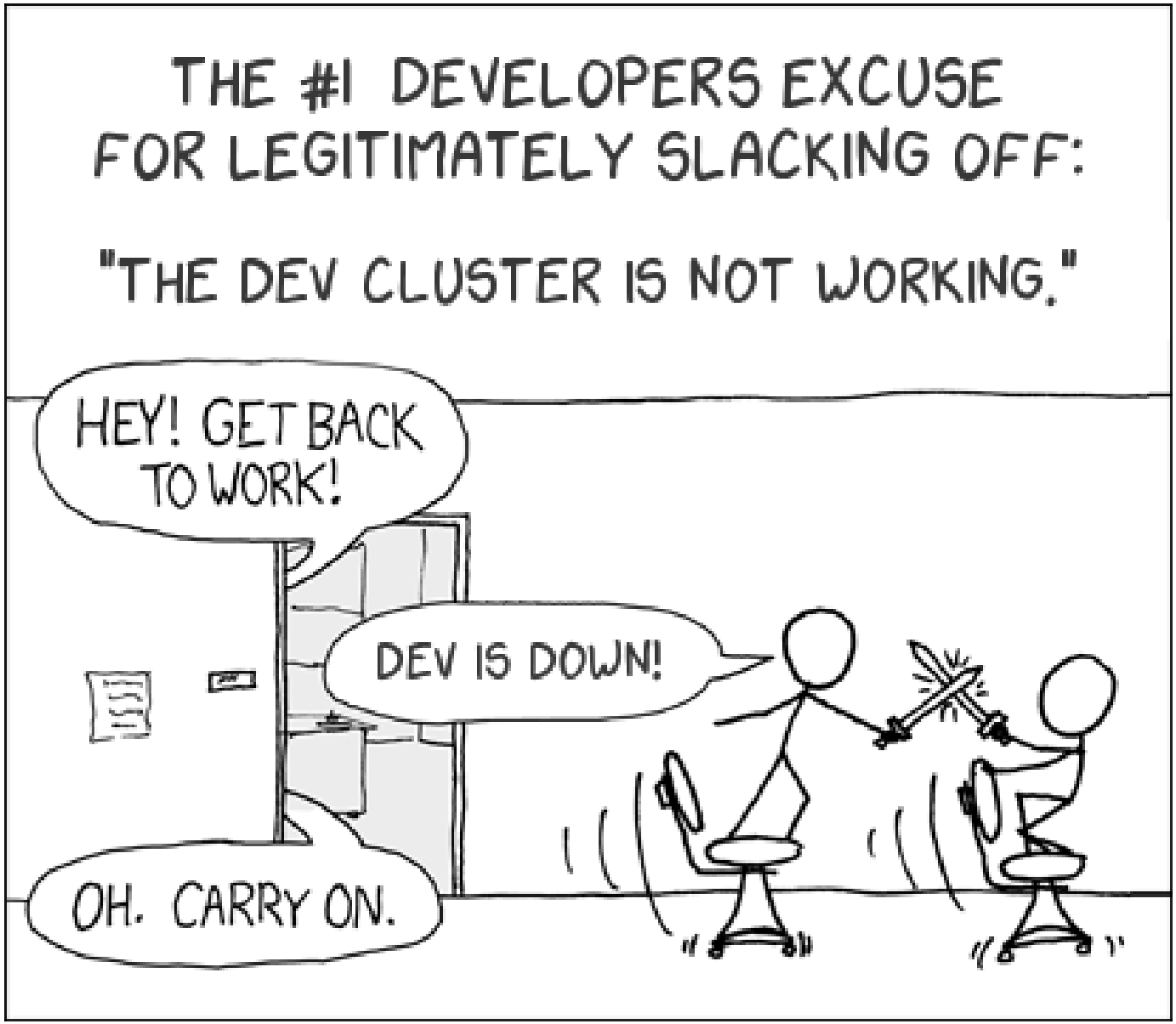


Wir sind Teil der teccle group.

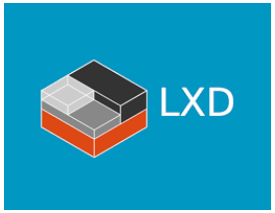
Hands On - 1.

Docker - Grundlagen.

Anmerkung.

- Container != VM

- Alternativen zu Docker
  - Podman
  - Buildah
  - LXD
  - Buildkit
  - rkt



Ein paar Daten.

Docker Inc:

Plattformen:

Release:

2017.

Mirantis, Infrastruktur und Service Provider für Open Source & Container, übernimmt mit sofortiger Wirkung die Enterprise Sparte der Containertechnologie Docker zu einem nicht genannten Preis.

Fakt ist, dass die Übernahme nicht nur das Produkt und die Rechte an Docker Enterprise, bestehend aus der Docker Enterprise Engine, der Docker Trusted Registry, der Docker Unified Control Plane und dem Docker CLI beinhaltet.

Nicht Teil des Deals sind Docker Hub und Desktop, mit denen Docker, Inc. zukünftig weiter arbeiten kann.

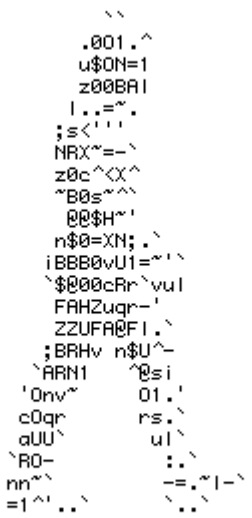
Daher wird es sich nicht lohnen, Docker Swarm langfristig weiter zu unterstützen.

~~scheduling~~

- ~~KITEMATIC~~
  - ~~Desktop User Interface for Docker~~

Wir sind Teil der teccle group.

Etwas Hintergrund.



Hintergrund, Standards.

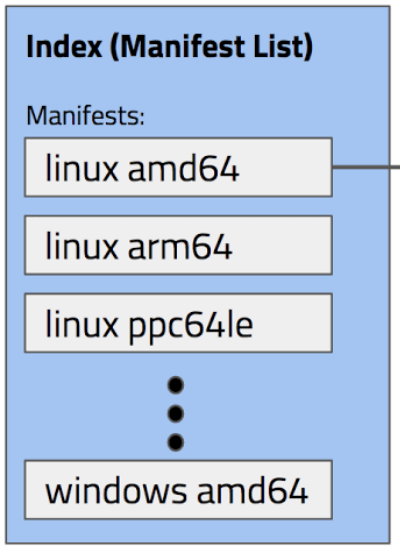
Standard - Open Container Initiative - OCI.

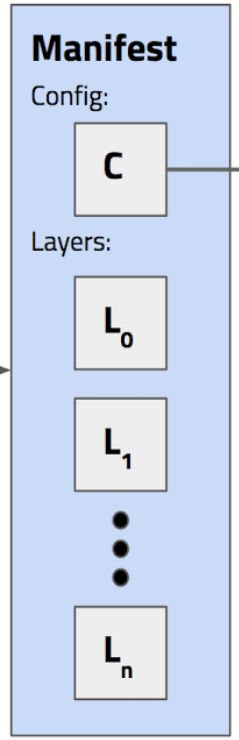
- Seit Juni 2015 Linux Foundation Projekt
  - Gegründet von container Inc., CoreOS, Google, Huawei
- Ziele
  - Entwicklung von offenen Standards für Container(-Image) Technologien
  - Verhinderung einer Fragmentierung

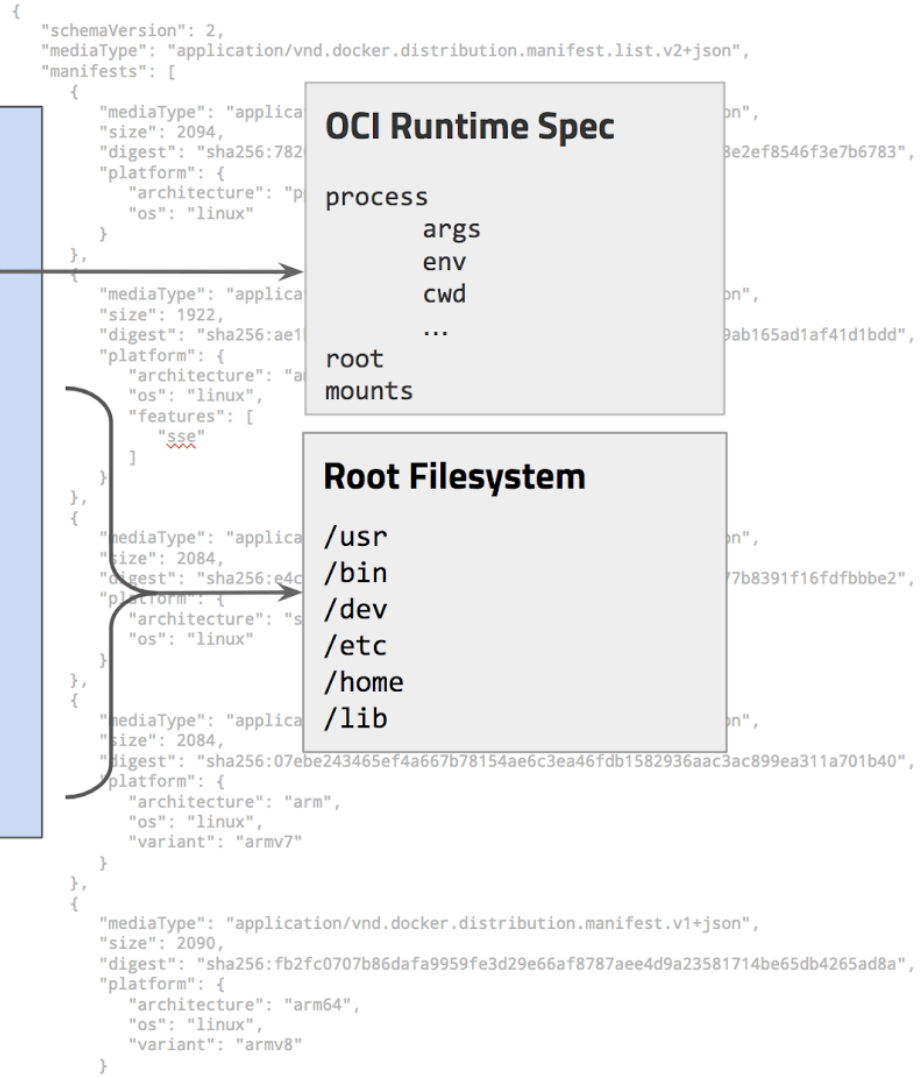
- v1.0.2 runtime-spec (3/27/20) / v1.0.1 distribution-spec (11/17/21)
  - https://opencontainers.org/release-notices/overview/
  - Runtime- und Image-Spezifikationen
    - Implementierung OCI-kompatibler Bundles und Container-Konfigurierungen
    - Werkzeuge für das Entwickeln von Container Images und für deren Austausch und Betrieb in Container-Laufzeitumgebungen
  - Richtlinien
    - Composable, Portable, Secure, Decentralized, Backward Compatible, ...

Standard - Open Container Initiative – OCI.

Image Manifests

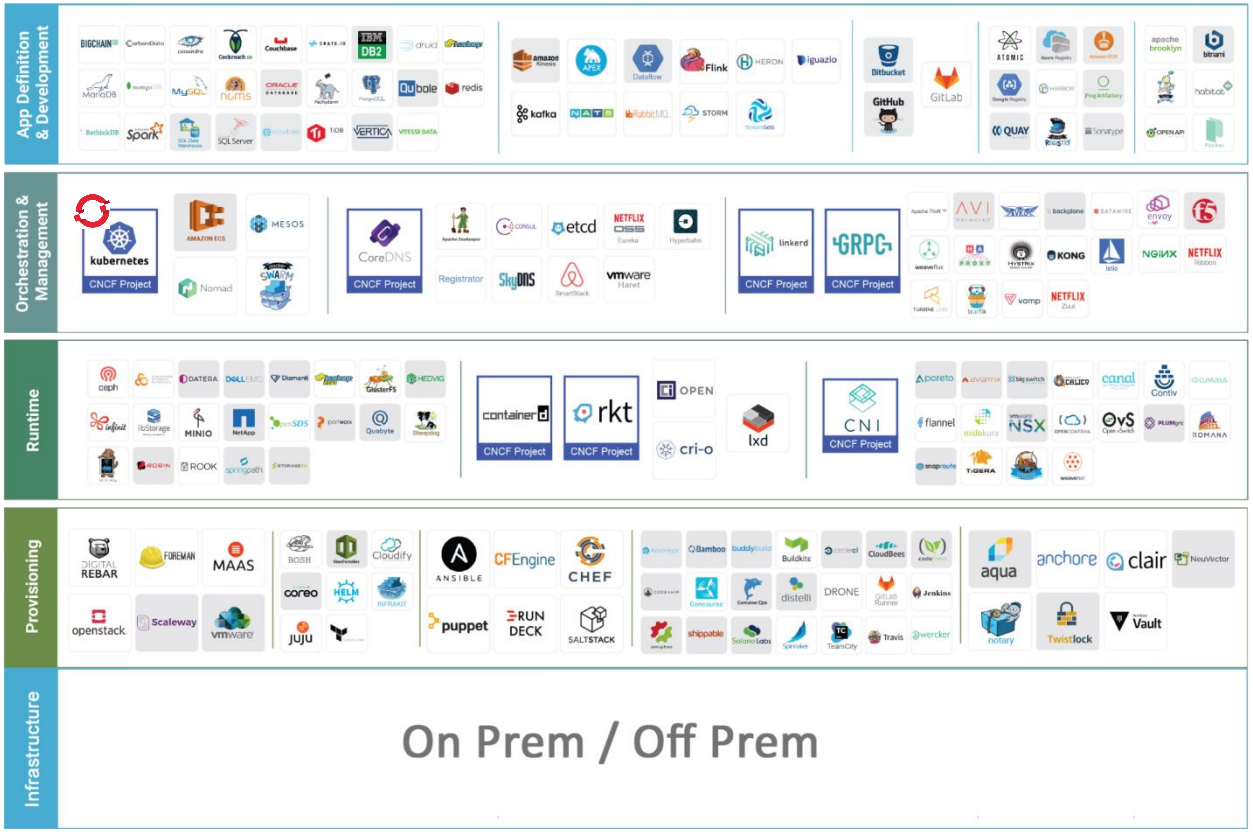






https://github.com/opencontainers/image-spec/blob/main/spec.md

Standard - Cloud Native Computing Foundation (CNCF).

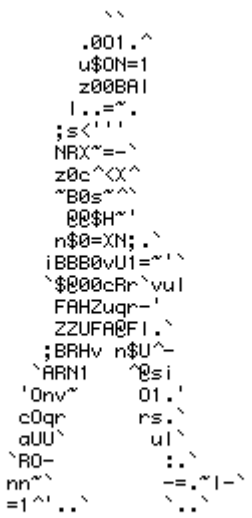


CLOUD NATIVE COMPUTING FOUNDATION

Non-Profit-Organisation (NPO) der Linux Foundation

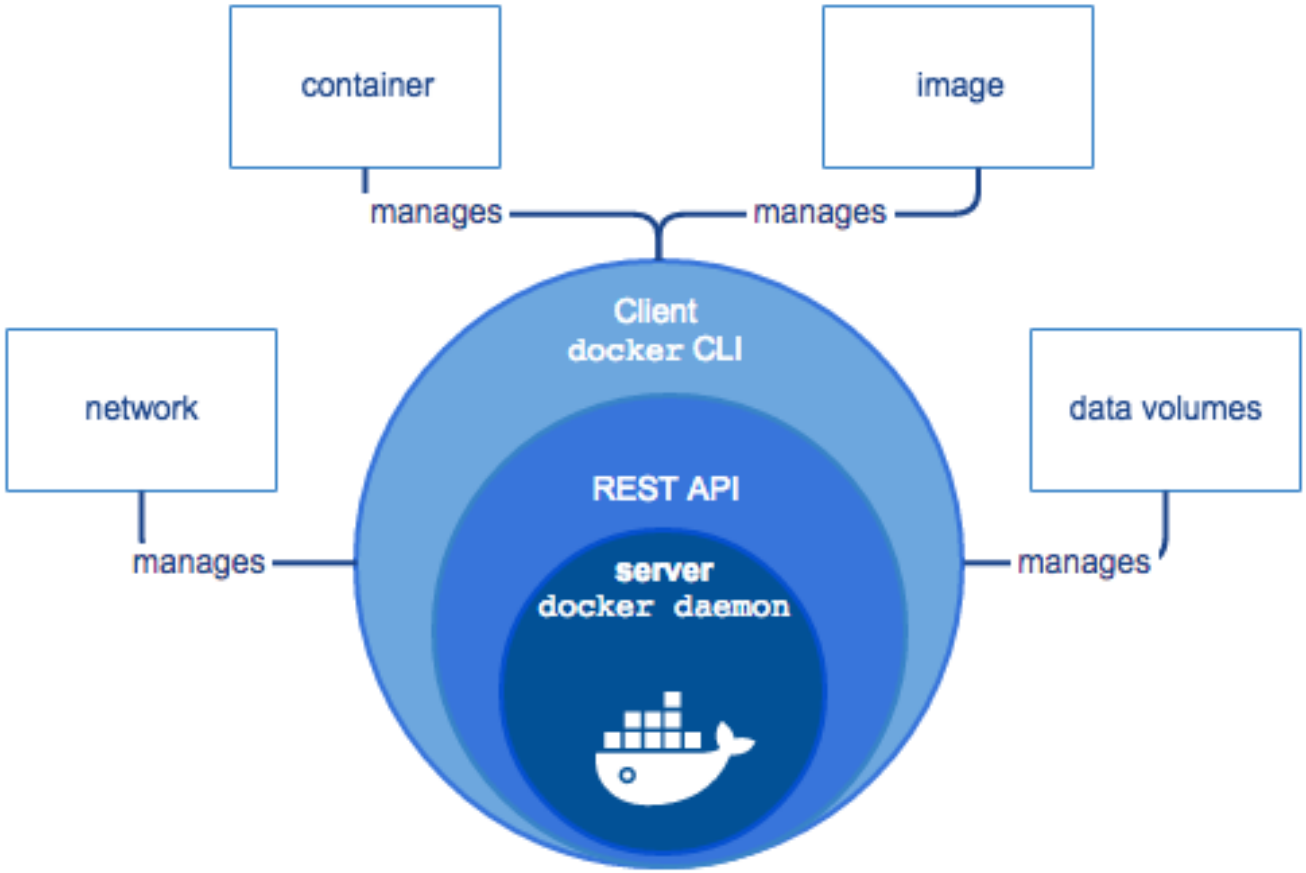
- Aufgaben
  - Standardisierung der Entwicklung von Cloud nativen Technologien und Diensten
  - Erstellung von Referenzen für einen Technologiestack

Docker Grundlagen.



Architektur + Grundlagen

Die Docker-Architektur - Flughöhe.

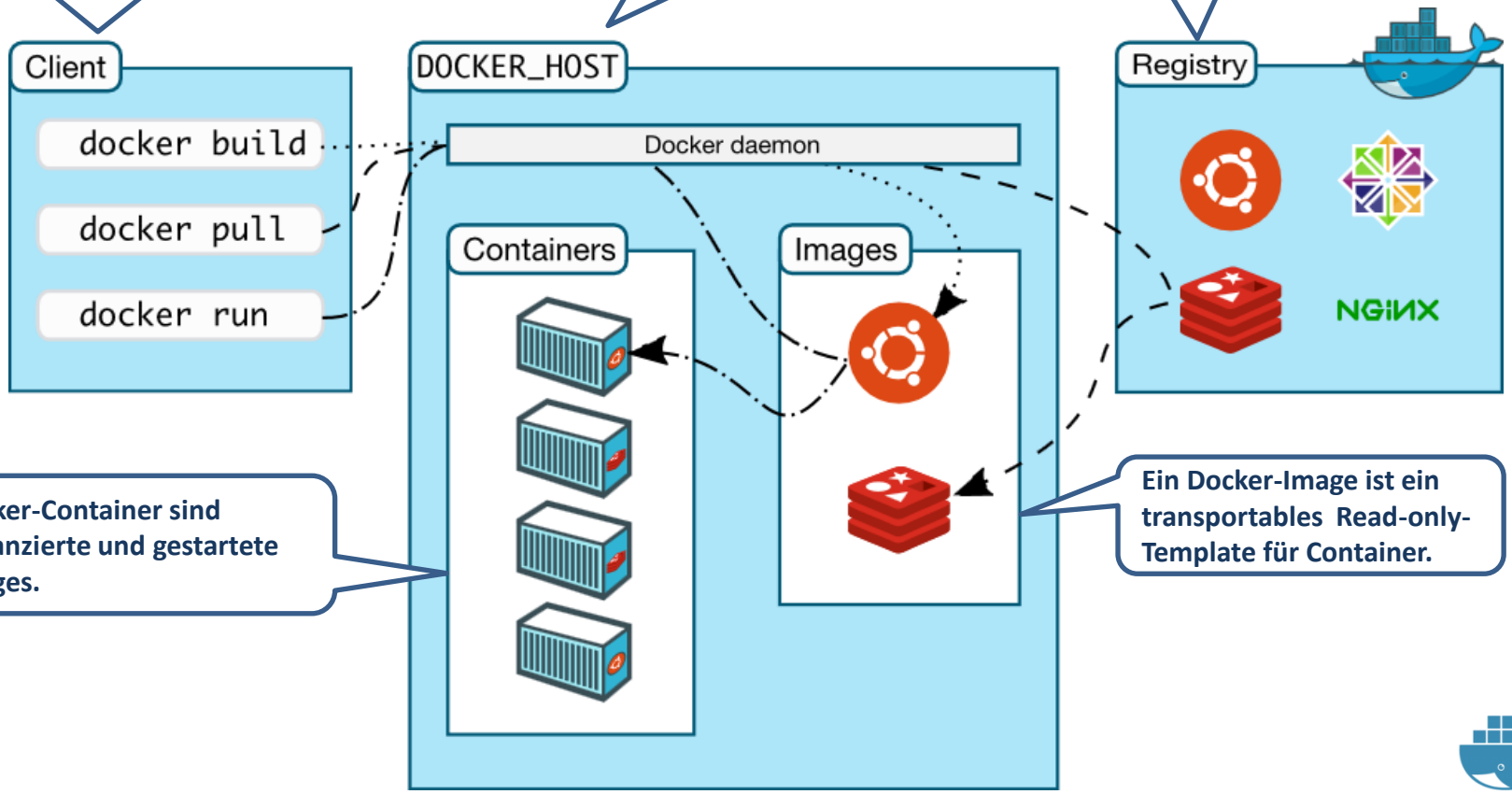


Die Docker-Architektur.

Der Docker-Client CLI interagiert mit dem Docker-Daemon über Unix-Domain-Socket(non-networked) und remote via REST-API.

Der Docker-Daemon stellt den docker-service auf dem Docker Host zur Verwaltung der Instanzen zur Verfügung.

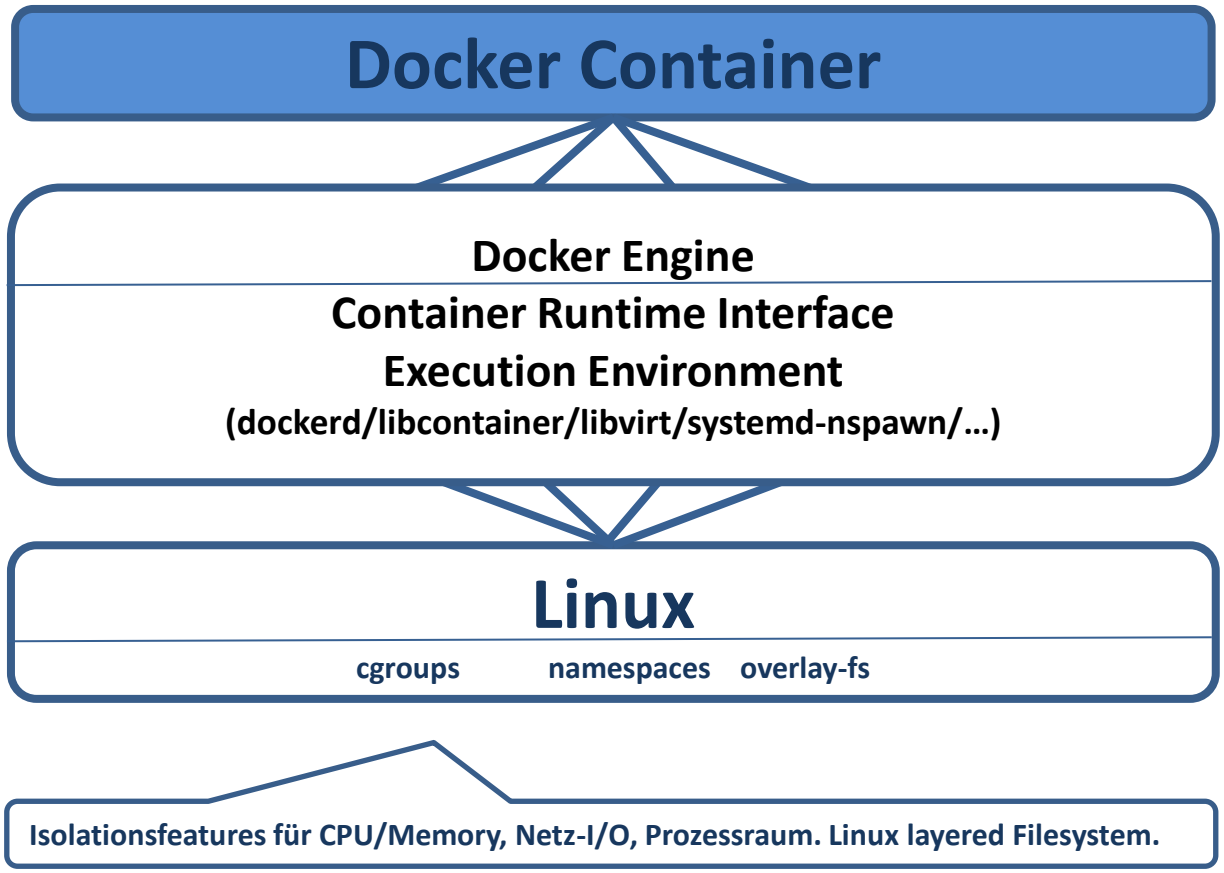
Docker Registries sind der Distributionsservice für Docker Images.



Docker-Container sind instanzierte und gestartete Images.

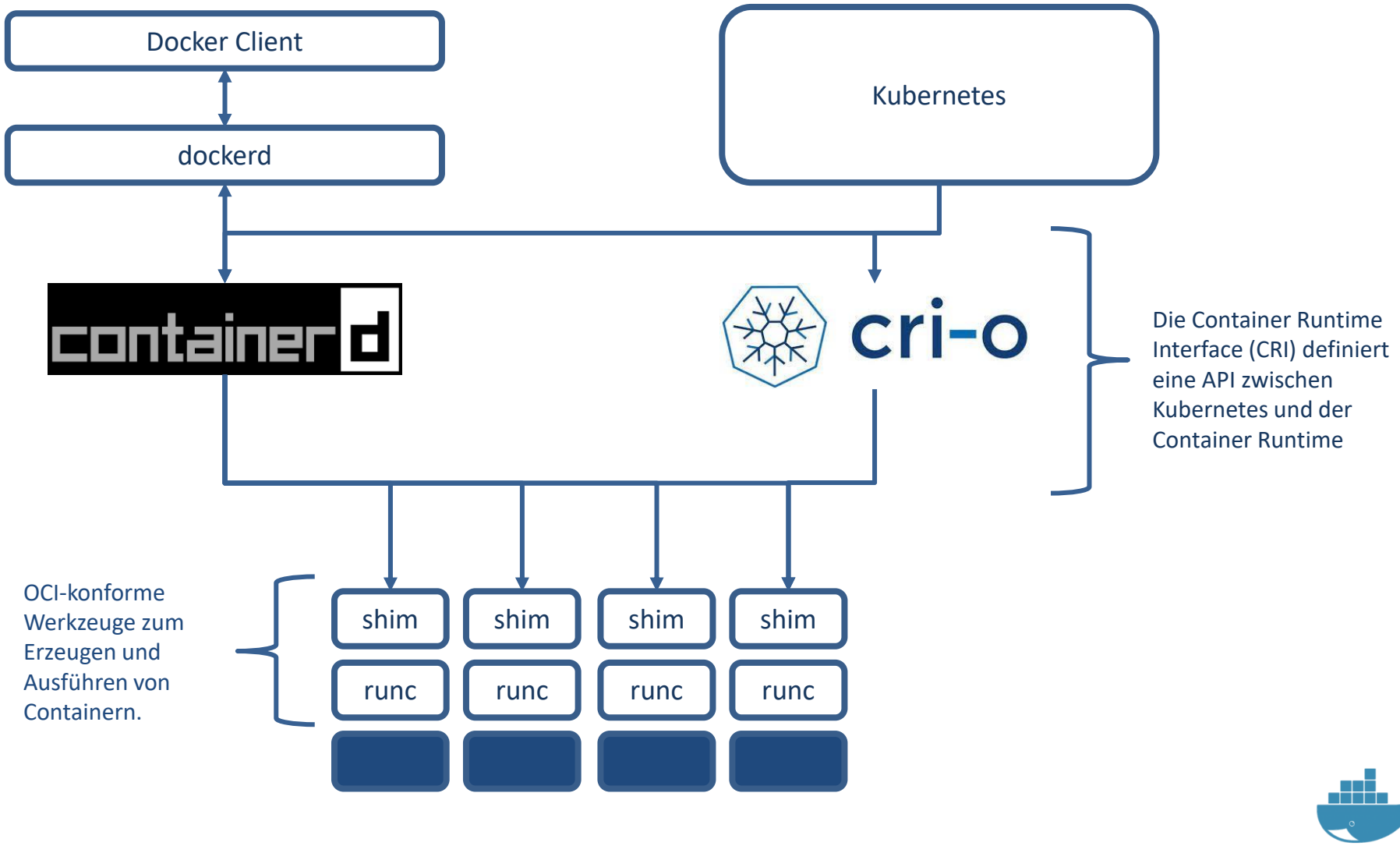
Ein Docker-Image ist ein transportables Read-only-Template für Container.

Etwas technischer Hintergrund.

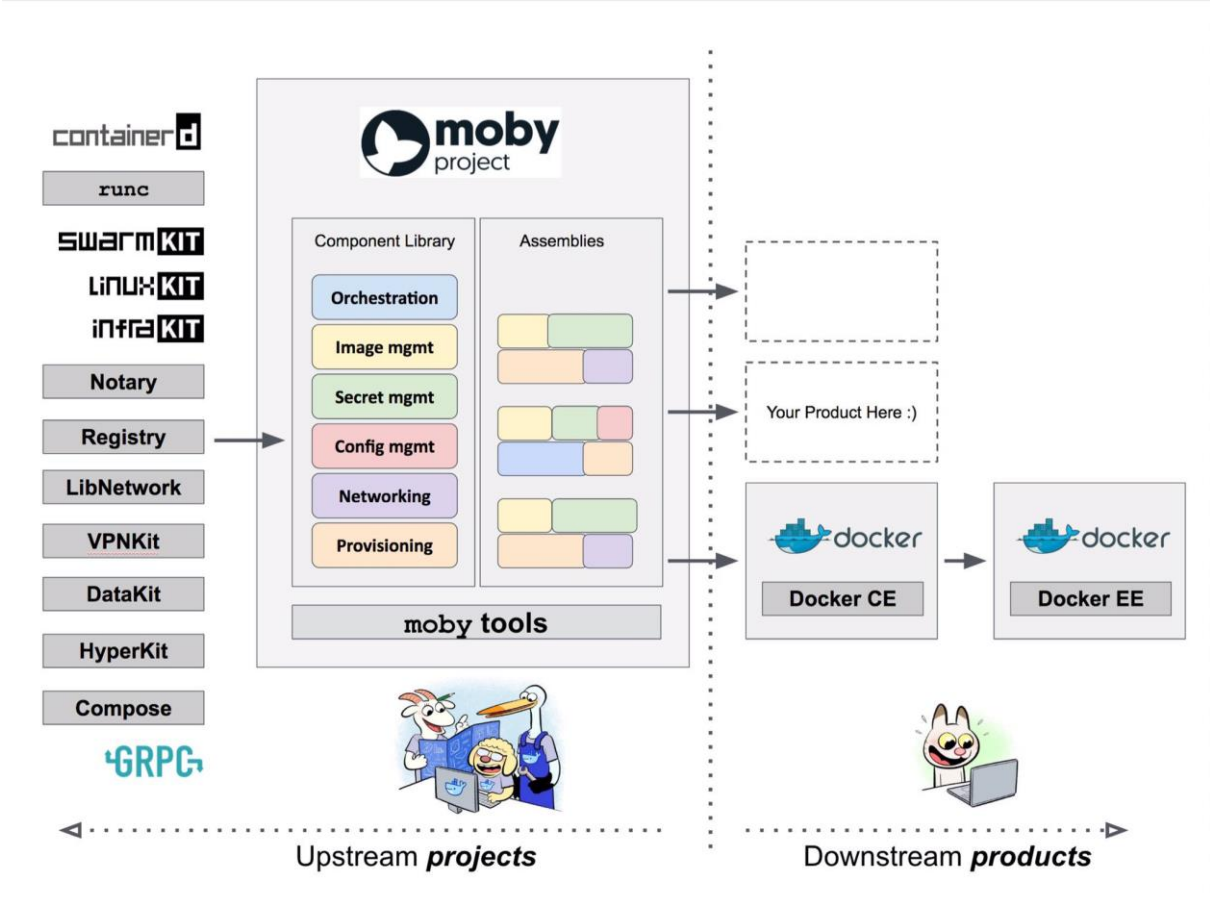


- Schnittstelle zum Kernel
- CLI
- Rest-API
- Socket-API

Noch etwas etwas technischer Hintergrund – Docker vs. Kubernetes.

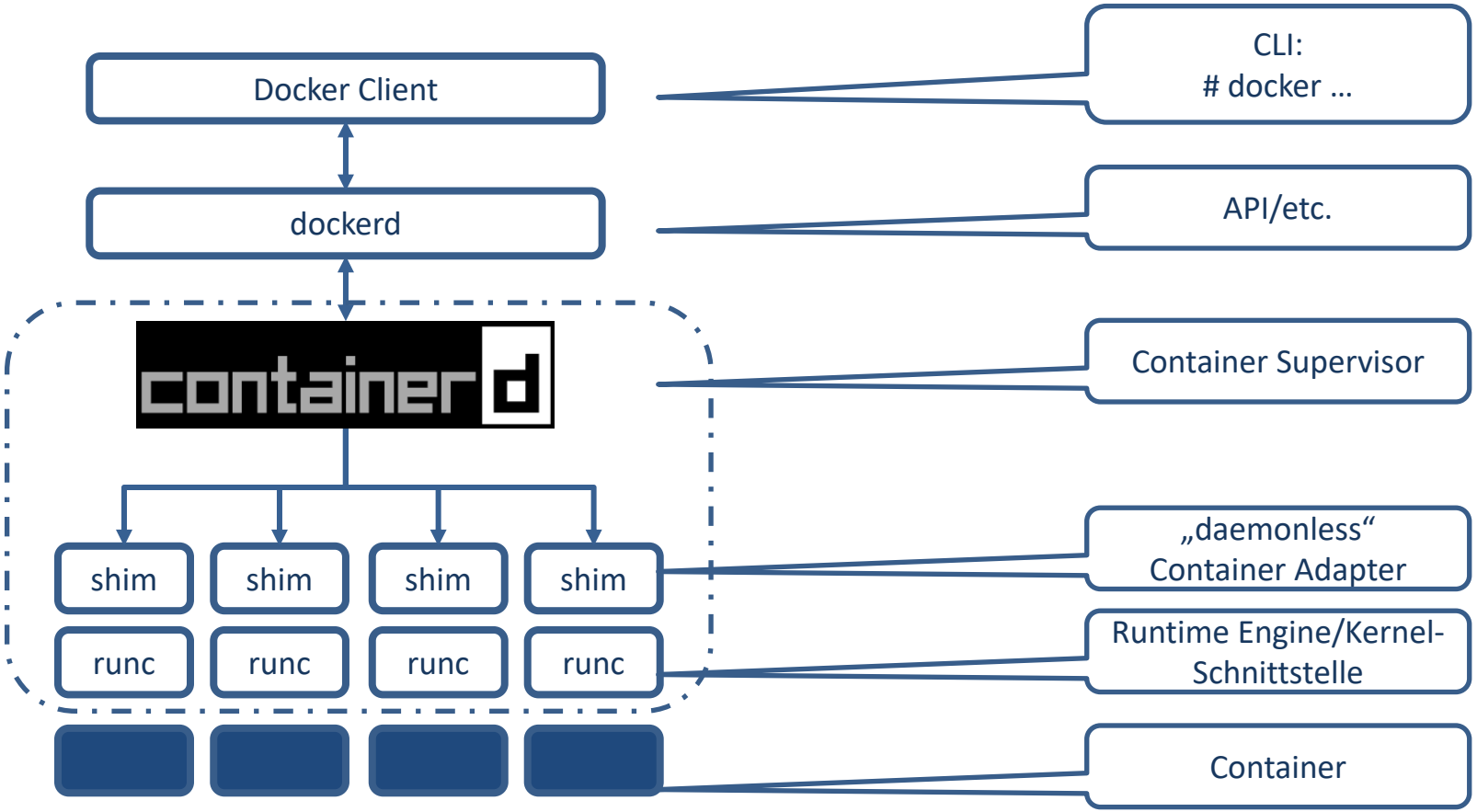


Project Moby.

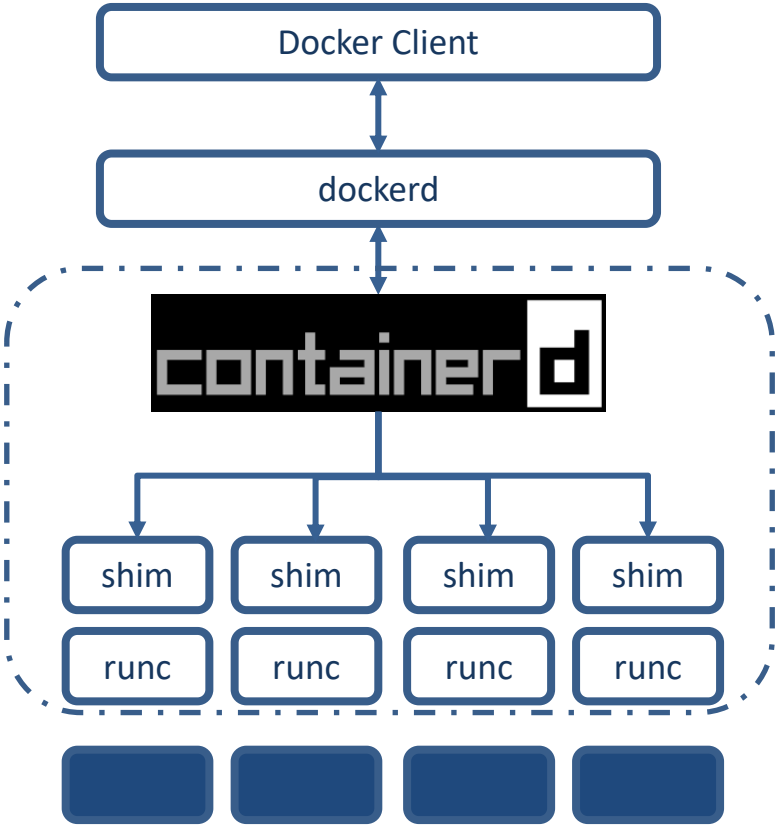


Bisher ist war der Docker-Stack ein größtenteils monolithisches Aufbau.
Mit Moby werden die Anwendungen in einzelne Bestandteile aufgeteilt, wie z.B die Laufzeitumgebung containerd.
Diese einzelnen Teile sollen künftig im Moby-Projekt gepflegt werden.

Noch etwas etwas technischer Hintergrund – Docker Engine.(1)



Noch etwas etwas technischer Hintergrund – Docker Engine.(2)



docker run … wird an API-Endpoint (dockerd/socket) gesendet

dockerd instruiert containerd zum Start eines neuen Container

Linux-Daemon, der Container verwaltet und startet. Er lädt Images aus dem Registry herunter, verwaltet Speicher und Netzwerke und überwacht die Ausführung von Containern.

Shims ermöglichen u.a. das Beenden einer Laufzeit (runC) nach dem Start des Containers. containerd ausfällt, hält es STDIO für den Container offen. Ohne diese Funktion würde sich der Container beenden.

runC ist eine Low-Level-Container-Runtime, die aus den übergebenen Images OCI-kompatible Container erstellt und ausführt. (via libcontainer, eine native Go-basierte Implementierung zum Erstellen von Containern)

Docker-Komponenten.

- dockerd
- containerd
- runC
- containerd-ctr
- containerd-shim

Docker-Komponenten - CLI.

- **Docker CLI (docker)**
  - /usr/bin/docker
  - docker-cli ist für die Kommunikation mit dockerd verantwortlich

```
<1008>root@dev:~ >docker version
Client: Docker Engine - Community
 Version:           20.10.18
 API version:       1.41
 Go version:        go1.18.6
 Git commit:        b40c2f6
 Built:             Thu Sep  8 23:11:43 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.18
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.6
  Git commit:       e42327a
  Built:            Thu Sep  8 23:09:30 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.8
  GitCommit:        9cd3357b7fd7218e4aec3eae239db1f68a5a6ec6
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Docker-Komponenten - CLI.

- **Docker CLI (wo steht was)**
  - https://docs.docker.com/engine/reference/run/
  - https://docs.docker.com/engine/reference/commandline/cli/
  - https://docs.docker.com/engine/reference/commandline/docker/

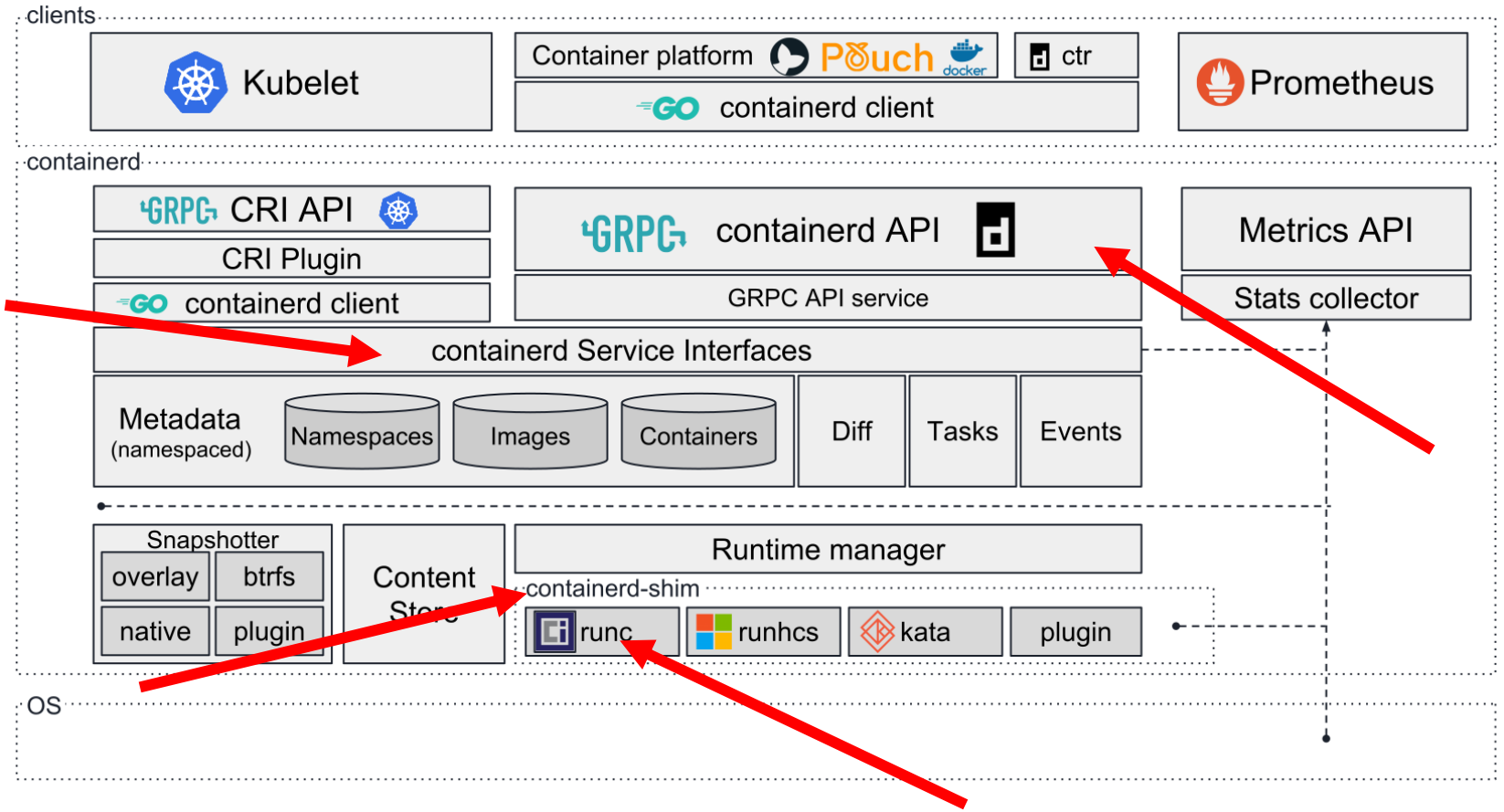
Docker-Komponenten - Dockerd.

- **Dockerd**
  - /usr/bin/dockerd
  - /var/run/docker.sock
  - dockerd ist der API-Endpoint für Docker-API-Anfragen
  - dockerd kann über drei verschiedene Socket-Typen agesprochen werden:
    - unix, tcp, fd
    - /var/run/docker.sock (Unix-Domänen-Socket)
      - Root-Berechtigung oder eine Docker-Gruppenzugehörigkeit
      - systemd: via systemd socket activation -> dockerd -H fd://
  - Konfigfile /etc/docker/daemon.json

Docker-Komponenten - Containerd.

- **Containerd**
  - /usr/bin/containerd
  - containerd
    - Verwaltung (start/stop/pause/…)/Lifecycle der Container via Aufruf runC
    - Image push/pull
    - Storageverwaltung
    - Netzwerkverwaltung
    - gRPC API-Schnittstelle
      - gRPC – Open Source universal RPC framework

Noch etwas etwas technischer Hintergrund – containerd (CNCF).



Docker-Komponenten - RunC.

- **RunC**
  - /usr/bin/docker-runc
  - /usr/sbin/runc
  - /runc (OCI-Laufzeit) kann als Bestandteil von containerd betrachtet werden.

```
<1040>root@dev:/usr/bin >runc --help
NAME:
   runc - Open Container Initiative runtime

runc is a command line client for running applications packaged according to
the Open Container Initiative (OCI) format and is a compliant implementation of the
Open Container Initiative specification.
…
```

Docker-Komponenten - containerd-ctr.

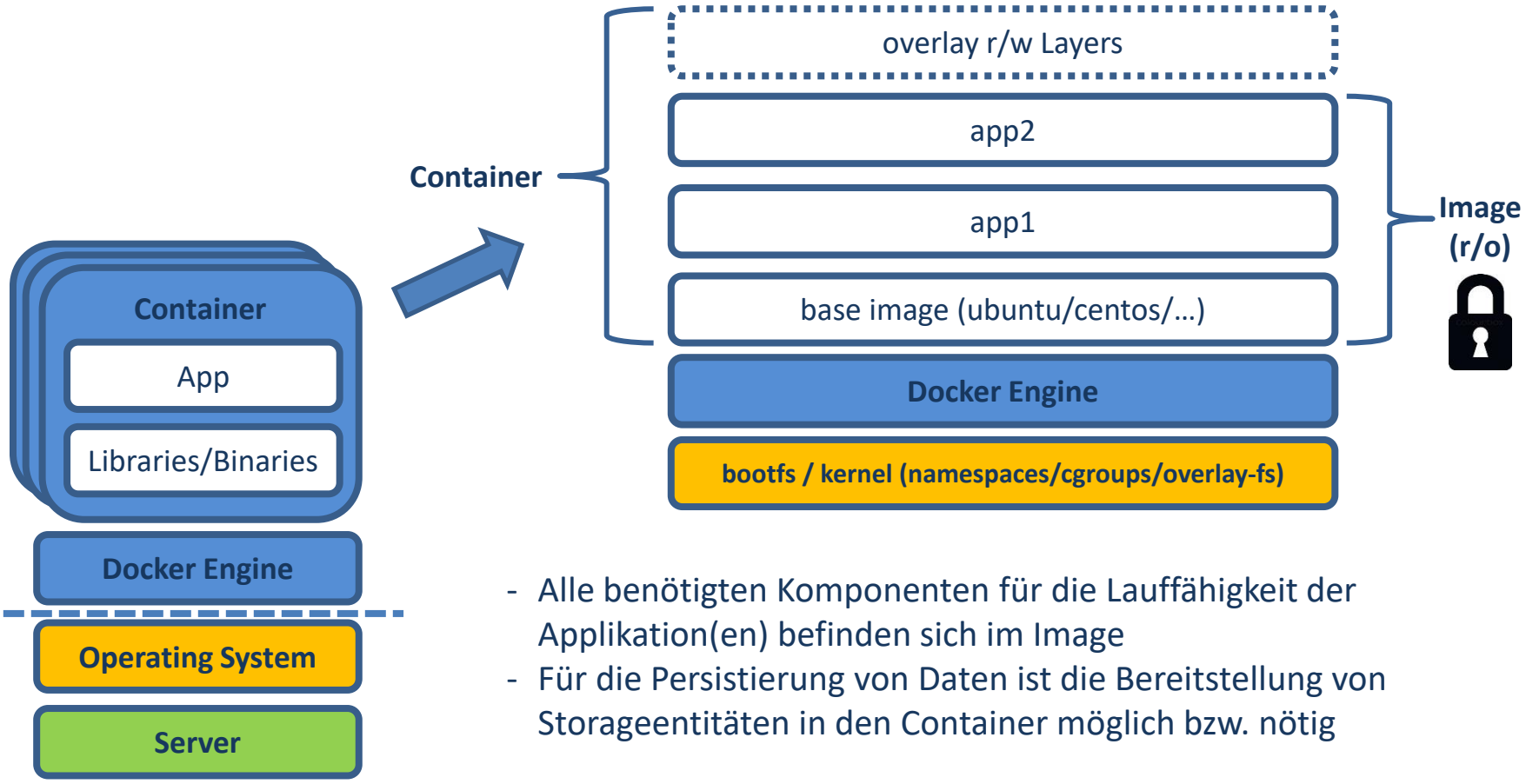
- **containerd-ctr**
  - /usr/bin/ctr
  - ctr ist ein CLI, das speziell für Entwicklungs- und Debuggingzwecke zur direkten Kommunikation mit containerd entwickelt wurde
  - ctr ist in den Versionen von containerd enthalten

```
<1040>root@dev:/usr/bin >ctr
NAME:
   ctr -

        __
  _____/ /_____
 / ___/ __/ ___/
/ /__/ /_/ /
\___/\__/_/

containerd CLI


USAGE:
   ctr [global options] command [command options] [arguments...]
…
```

Docker-Komponenten - containerd-shim.

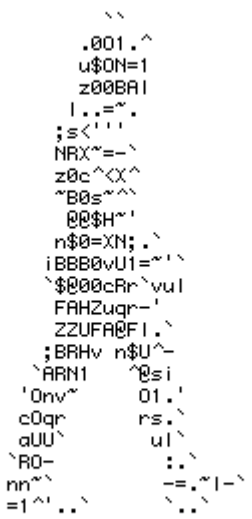
- **containerd-shim**
  - /usr/bin/containerd-shim
  - Entkoppelung der Runtime Engine(runC) von gestarteten Container
  - STDIO/fd bleiben geöffnet, auch wenn containerd/dockerd “sterben”
  - -> docker kann bei laufenden Containern geupdatet werden !

Docker in Schichten.



- Alle benötigten Komponenten für die Lauffähigkeit der Applikation(en) befinden sich im Image
- Für die Persistierung von Daten ist die Bereitstellung von Storageentitäten in den Container möglich bzw. nötig

Docker Grundlagen.



Konfiguration docker

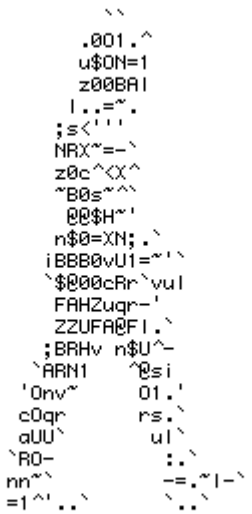
Docker - Konfiguration.

- **/etc/docker/daemon.json**
  - https://docs.docker.com/config/daemon/

```
 #cat /etc/docker/daemon.json
{
  "insecure-registries": [],
  "debug": true,
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

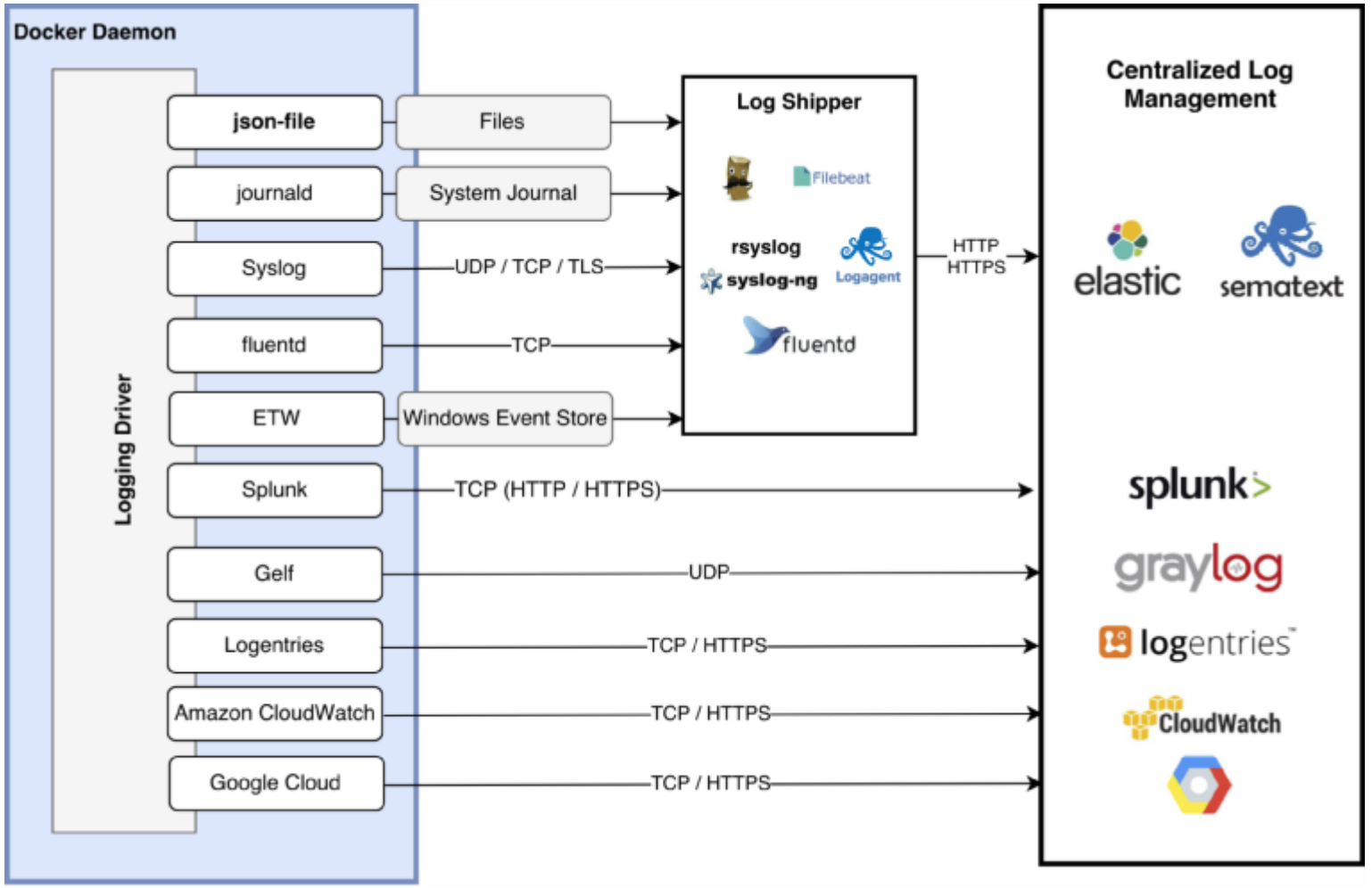
- **/lib/systemd/system/docker.service**
  - Nicht empfohlen ...
  - https://docs.docker.com/engine/reference/commandline/dockerd/

Docker Grundlagen.



Logging

Docker – Logging-Driver.



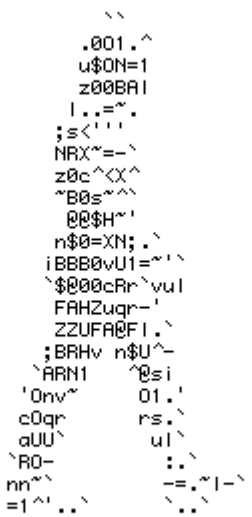
Docker – Logging-Driver.

- Logging Treiber (der Container, nicht des dockerd …)
- docker logs …
- Binäre plug-in Logging-Treiber
  - Bestandteil des Docker-Daemon
  - Dedizierte Prozesse
- Docker-Logging-Treiber leiten Container-Logs weiter
  - json-file ist der Default

```
<1040>root@dev:/usr/bin > docker inspect -f {{.HostConfig.LogConfig.Type}} <container>
json-file

<1041>root@dev:/usr/bin > docker inspect -f {{.LogPath}} <container>
/var/lib/docker/containers/9952f18d789d639d5bab11fbe3a127f526bb18f4822e3395b2a0d6485f900d26/9952f18d789d639d5ba
b11fbe3a127f526bb18f4822e3395b2a0d6485f900d26-json.log…
```

Hands On - 2.

Docker Grundlagen.



Registry, Repository usw.

Docker – Registry vs. Repository vs. Images vs. Tags.

- Die Ablage der Containerimages ist ein hierarchische organisiertes System:
  - **Registry**
    - Ein Dienst, der für das Hosten und Verteilen von Images verantwortlich ist.
  - **Repository**
    - Eine Sammlung zusammengehöriger Images (die in der Regel verschiedene Versionen derselben Anwendung oder desselben Dienstes bereitstellen).
  - **Tag**
    - Eine alphanumerische Kennung, die Images innerhalb eines Repositorys zugeordnet ist.

```
<636>hs@dn:~ >docker pull docker.io/alpine:1.0.3
```

Docker – Registry vs. Repository vs. Images vs. Tags.

- Eine private Docker-Registry
  - ermöglicht es benutzerdefinierten Basis-Images abzulegen und zu teilen
  - ist eine konsistente, private und zentralisierte Quelle
  - Eine private Docker-Registry bietet bessere Leistungen für hochfrequente Rollouts sowie zusätzliche Funktionen wie die Zugangsauthentifizierung

- In einem Docker-Repository
  - können 1 oder mehrere Versionen eines bestimmten Docker-Images gespeichern werden.
  - Ein Image kann 1 oder mehrere Versionen (Tags) haben.

Docker – Image Name Schema Convention.

<image-registry>/<project>/<image-name>:<tag>

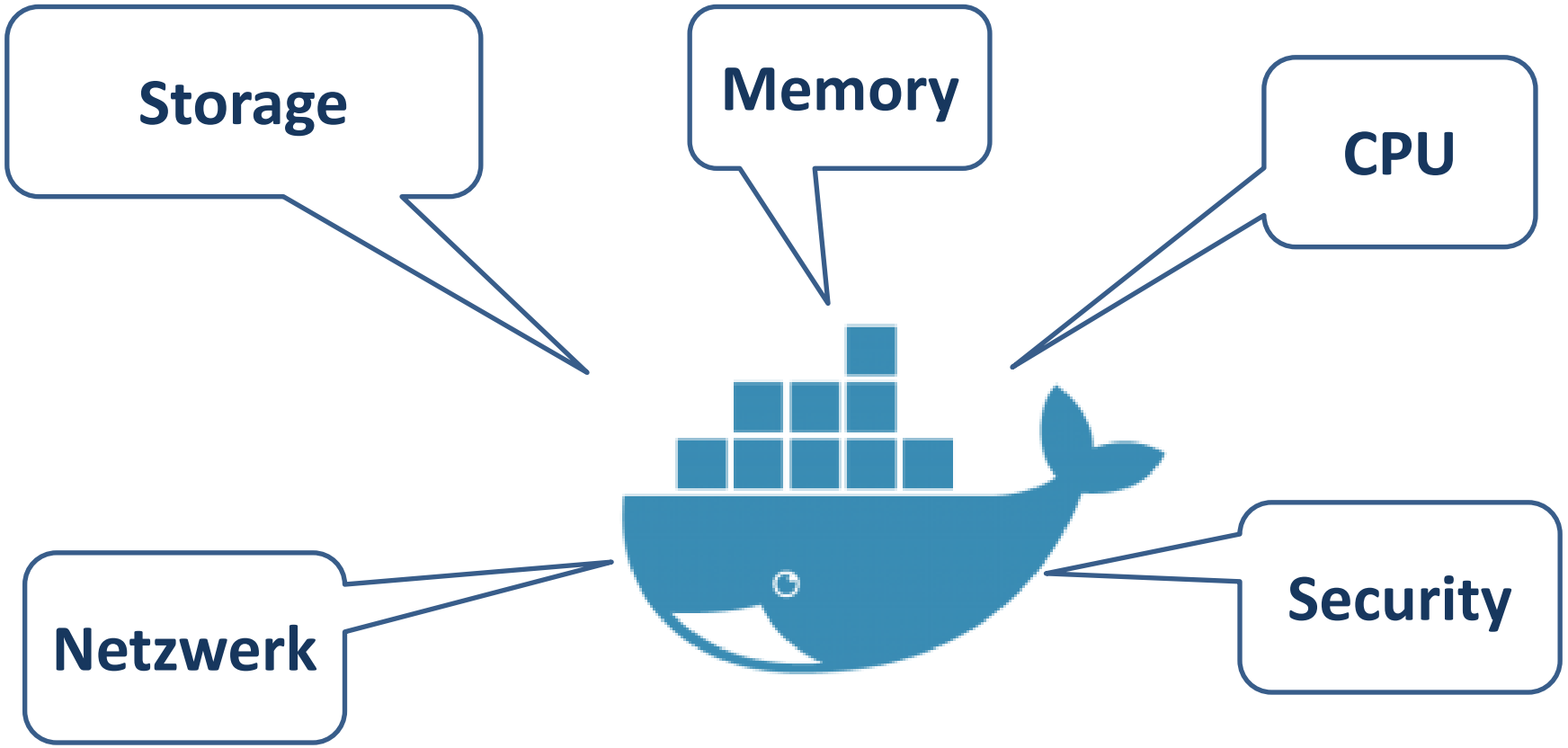
```
<635>hs@dn:~ >docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
Digest: sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
Status: Image is up to date for alpine:latest
docker.io/library/alpine:latest

<636>hs@dn:~ >docker pull docker.io/alpine:latest
latest: Pulling from library/alpine
Digest: sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
Status: Image is up to date for alpine:latest
docker.io/library/alpine:latest

<637>hs@dn:~ >docker pull \
docker.io/alpine@sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
```

<image-registry>/<project>/<image-name>@<image digest>

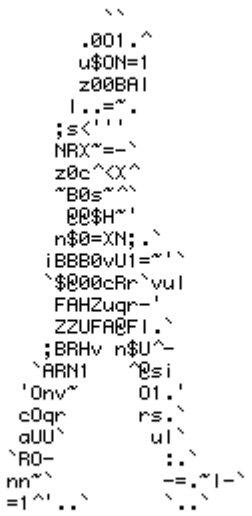
```
<637>hs@dn:~ >docker pull \
docker.io/alpine@sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a8d9730fc2ad
```

Hands On - 3.

Docker – Generelle Themen.



Docker Grundlagen.



Memory

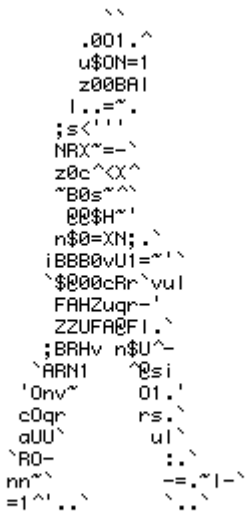
Memory.

- Size Memory (resident) definieren / limitieren
  - memory-reservation, memory-swap
  - memory-swappiness, kernel-memory
  - oom-kill-disable
- Size Shared Memory definieren / erhöhen
  - shm-size
- Java
  - Innerhalb des Container immer Heap-Size definieren / MaxRAM / +UseCGroupMemoryLimitForHeap / …

```
# docker run -d -it --memory=2g --memory-swap=6g --name MYDB …

# docker run --memory=1G --rm store/oracle/serverjre:8 java -XX:MaxRAM=1024m …
```

Wir sind Teil der teccle group.

Docker Grundlagen.



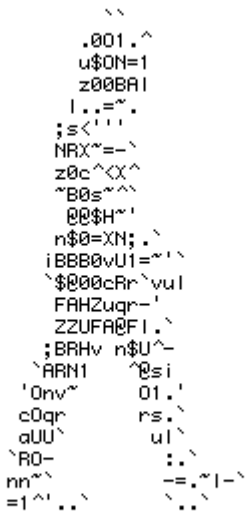
CPU

CPU.

- CPU-Nutzung limitieren, um die Kompromittierung des Dockerhost zu vermeiden (… gemeinsamer Kernel, gemeinsame Resourcen)
  - cpus, cpu-period, cpuset-cpus, cpu-shares, …
- Nicht Lizenzrelevant !!!

```
- Alle 50ms Nutzung von 50% aller Cpu's
# docker run -ti --cpu-period=50000 --cpu-quota=25000 …

- Nutzung von CPU1, CPU2
# docker run -ti --cpuset-cpus="1,3" …

-  Nutzung von 50 % aller CPU's pro Sekunde
# docker run -ti –cpu=.5 …
```

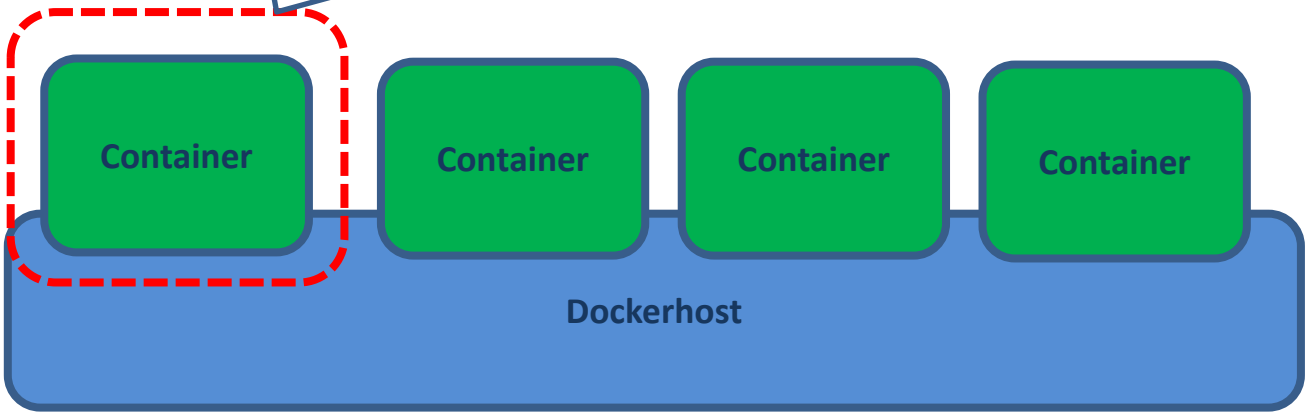
Docker Grundlagen.



Netzwerk

Netzwerk.

- Kommunikation von Datenbanken, weiteren Services etc. zwischen Containern via Overlay-Netzwerke absichern
  - vxlan
  - iptables
- Loadbalancing im Swarm-Mode via Ingress Network
- Externe Loadbalancer können den Service via **PublishedPort** adressieren



Netzwerk Driver.

- Bridge
- Host
- Overlay
- Macvlan
- None
- Network plugins
  - https://docs.docker.com/network/#network-drivers

! ! Nicht alle Treiber sind für alle Plattformen supportet !!

Netzwerk Driver - none.

- **Closed Network / none**
  - Wird verwendet, wenn der Docker Container keine Verbindung zu einem Netzwerk benötigt.
  - Der Container besitzt keine virtuelle NIC und arbeitet isoliert von allen anderen Containern auf dem Host.
  - Man nennt diese dann auch closed container.

- **Vorteile**
  - Geeignet, wenn der Container ein Höchstmaß an Netzwerksicherheit erfordert und kein Netzwerkzugriff erforderlich ist

- **Nachteile**
  - Keine gute Wahl, wenn eine Netzwerk- oder Internetverbindung erforderlich ist. ☺

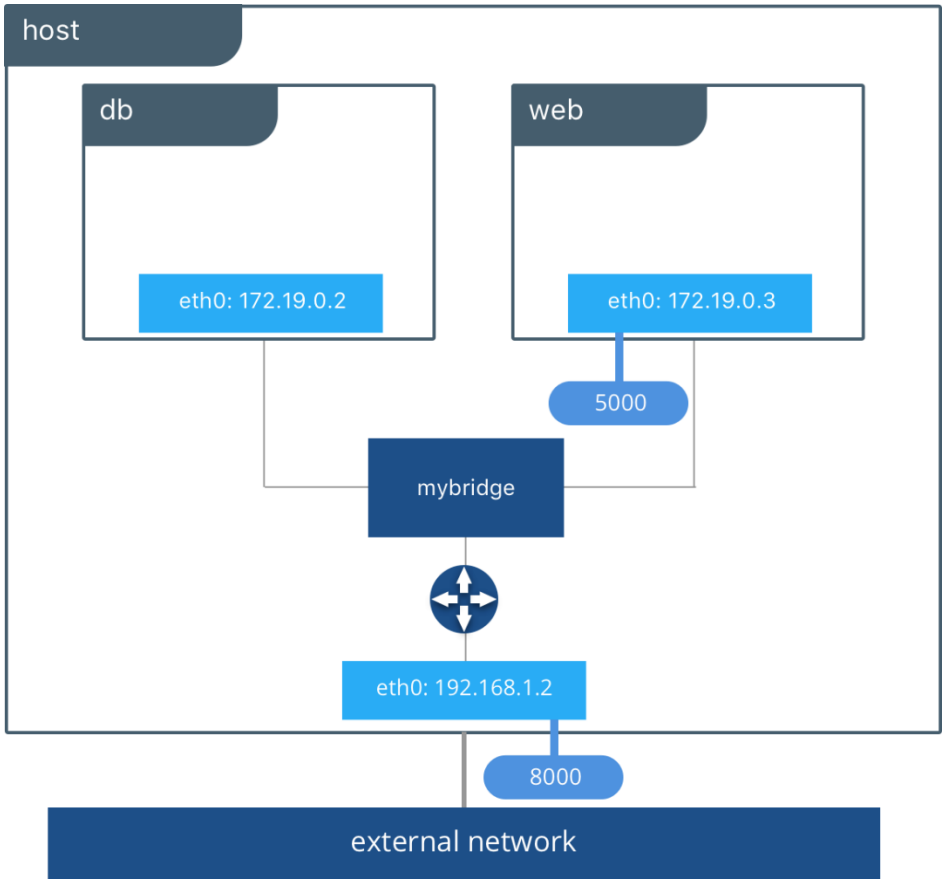
Netzwerk Driver - bridge.

- **Bridge Network**
  - Das Bridge Network ist das Standard Docker Network.
  - Über diese Netzwerk können die Container untereinander und mit der Außenwelt kommunizieren.

- **Vorteil:**
  - Verbindung ins Internet
  - Verbindung von Containern untereinander

- **Nachteil:**
  - Reduzierung der Netzwerksicherheit

Netzwerk Driver - bridge.



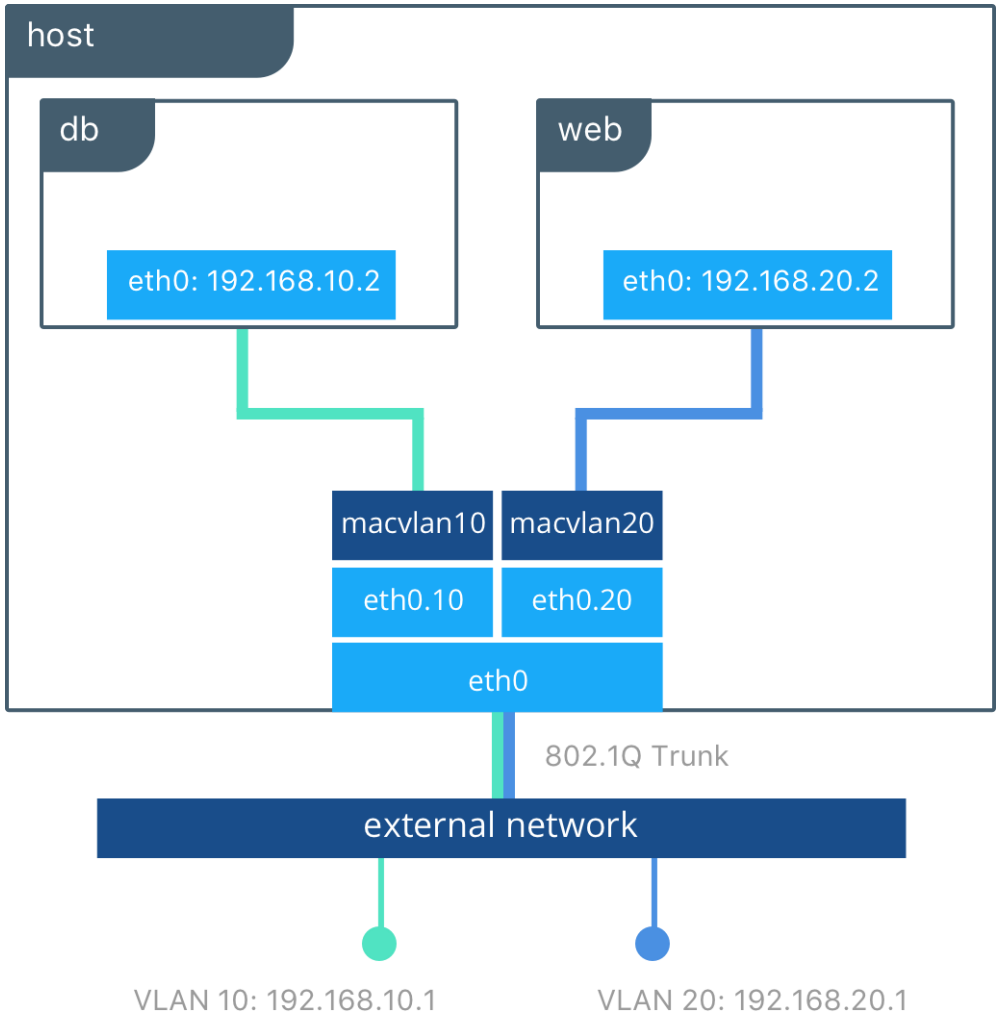
Netzwerk Driver - host.

- **Host Network**
  - Bei dieser Netzwerkart werden dem Docker Container die Netzwerkeigenschaften des Docker Hosts übertragen. Zum Starten eines Docker Containers im Host Network verwendet man den Parameter –net host.
- **Vorteil:**
  - Einfach
- **Nachteil:**
  - Massive Reduzierung der Netzwerksicherheit

Netzwerk Driver - macvlan.

- **Macvlan**
  - In Macvlan-Netzwerken können Containern eine MAC-Adresse zugewiesen werden, sodass er als physisches Gerät in Ihrem Netzwerk angezeigt wird.
  - Der Docker-Dämon leitet den Datenverkehr anhand seiner MAC-Adressen an die Container weiter. .
- **Vorteil:**
  - ggf. die beste Wahl, wenn ältere Anwendungen direkt mit dem physischen Netzwerk verbunden sein sollen
- **Nachteil:**
  - Statische Konfiguration

Netzwerk Driver - macvlan.



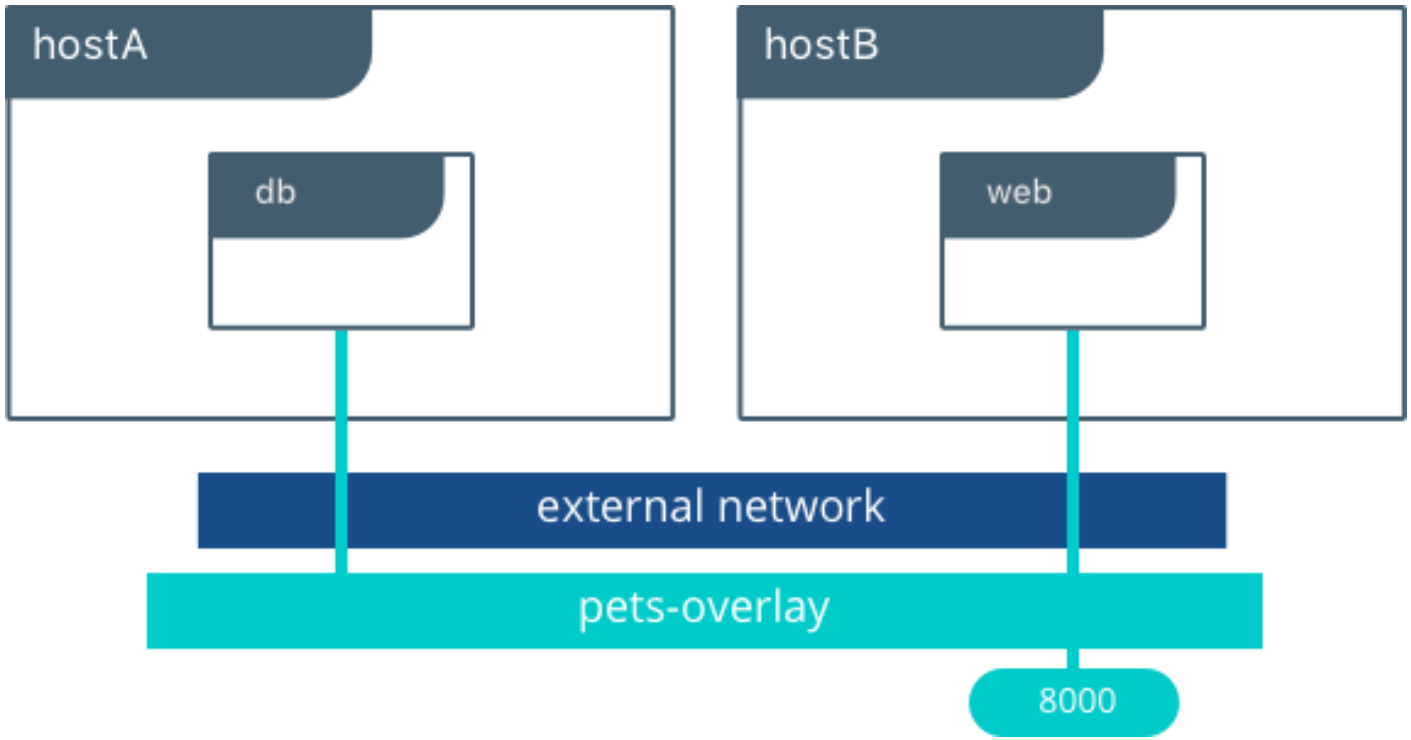
Netzwerk Driver – overlay (swarm).

- **Overlay (swarm)**
  - Overlay-Netzwerke verbinden mehrere Docker-Daemons miteinander und ermöglichen die Kommunikation zwischen Schwarmdiensten.
  - Overlay-Netzwerke können auch zur Kommunikation zwischen einem Schwarmdienst und einem eigenständigen Container oder zwischen zwei eigenständigen Containern auf verschiedenen Docker-Daemons verwendet werden.
  - Diese Strategie macht das Routing auf Betriebssystemebene zwischen diesen Containern überflüssig.

- **Vorteil:**
  - Transparent.
  - Geringer Konfigurationsaufwand.

- **Nachteil:**
  - Reduzierung der Netzwerksicherheit

Netzwerk Driver – overlay (swarm).



Netzwerk Driver - Summary.

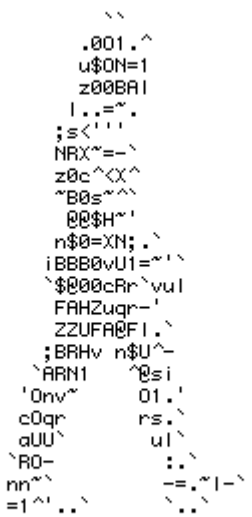
- Benutzerdefinierte Bridge-Netzwerke
  - Für die Kommunikation von Containern auf dem selben Docker-Host.
- Host-Netzwerke
  - Für Anwendungen ohne Isolation vom Netzwerk des Docker-Host.
- Overlay-Netzwerke (swarm)
  - Für Kommunikation über verschiedenen Docker-Hosts bzw. wenn mehrere Anwendungen mithilfe von Swarm-Diensten zusammenarbeiten.
- Macvlan-Netzwerke
  - Migration von einem VM-Setup o. legacy Apps, wenn Container wie physische Hosts im Netzwerk mit jeweils einer eindeutigen MAC-Adresse betrachtet werden sollen.
- Mit Netzwerk-Plugins von Drittanbietern
  - können Container in spezielle Netzwerkstacks integriert werden.

Netzwerk.

```
- 2 Bridges
# docker network create --driver=bridge --subnet=10.0.0.0/8   --ip-
range=10.0.0.0/8  --gateway=10.0.0.1 -o "com.docker.network.bridge.name"="br-
app1" br-app1

# docker network create --driver=bridge --subnet=100.0.0.0/8   --ip-
range=100.0.0.0/8  --gateway=100.0.0.1 -o "com.docker.network.bridge.name"="br-
app2" br-app2

# docker network list
NETWORK ID          NAME                DRIVER              SCOPE
8155e827baf9        br-app1             bridge              local
fe4562467e00        br-app2             bridge              local
76a3d6ca6b94        bridge              bridge              local
20260843402e        host                host                local
b863fdcb3e5e        none                null                local

# docker run -dt  -h ap-2-net --name ap-2-net --rm --net br-app1 d-
reg:5000/alpine

# docker network connect br-app2 ap-2-net
```

Hands On - 4.

Docker Grundlagen.



Storage / Volumes

Storage – 2 Fragen.

Welcher Storagedriver ?

- Files im Container …
- Achtung, Unterschiede in den Default-Storagedriver Docker EE/CE für verschiedene Linux-Derivate
  - https://success.docker.com/article/Compatibility_Matrix

Welcher Mount-Typ?

- Oder wohin mit den Data-Files ?

Storage – Storagedriver, WER für WAS.

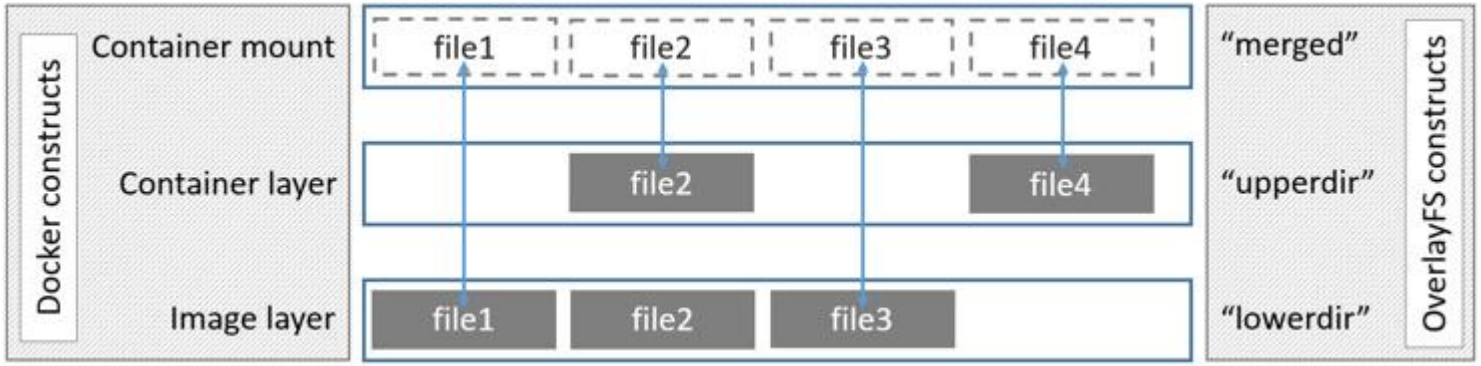
AUFS	stable	production-ready	good memory use	smooth Docker experience	high write activity	PaaS-type work
Devicemapper (loop)	stable	in mainline kernel	smooth Docker experience	production	performance	lab testing
Devicemapper (direct-lvm)	stable	production-ready	in mainline kernel	smooth Docker experience	PaaS-type work	
Btrfs	in mainline kernel	high write activity	container churn	build pools		
Overlay	stable	good memory use	in mainline kernel	container churn	lab testing	
ZFS native (ZoL)	PaaS-type work					
ZFS FUSE	stable	lab testing	production			

Key

Has attribute	attribute
If good for use case	use case
If bad for use case	use case

https://docs.docker.com/engine/userguide/storagedriver/selectadriver/

Docker in Schichten – Overlay2.




```
<1008>root@dev:~ >docker image inspect alpine -f "{{json .GraphDriver.Data }}" | jq
{
    "MergedDir": "/var/lib/docker/overlay2/be38d732a348745d391c536d4a44158d9638c58d28ea2557c7b36df56d2d9fca/merged",
    "UpperDir": "/var/lib/docker/overlay2/be38d732a348745d391c536d4a44158d9638c58d28ea2557c7b36df56d2d9fca/diff",
    "WorkDir": "/var/lib/docker/overlay2/be38d732a348745d391c536d4a44158d9638c58d28ea2557c7b36df56d2d9fca/work"
}
```

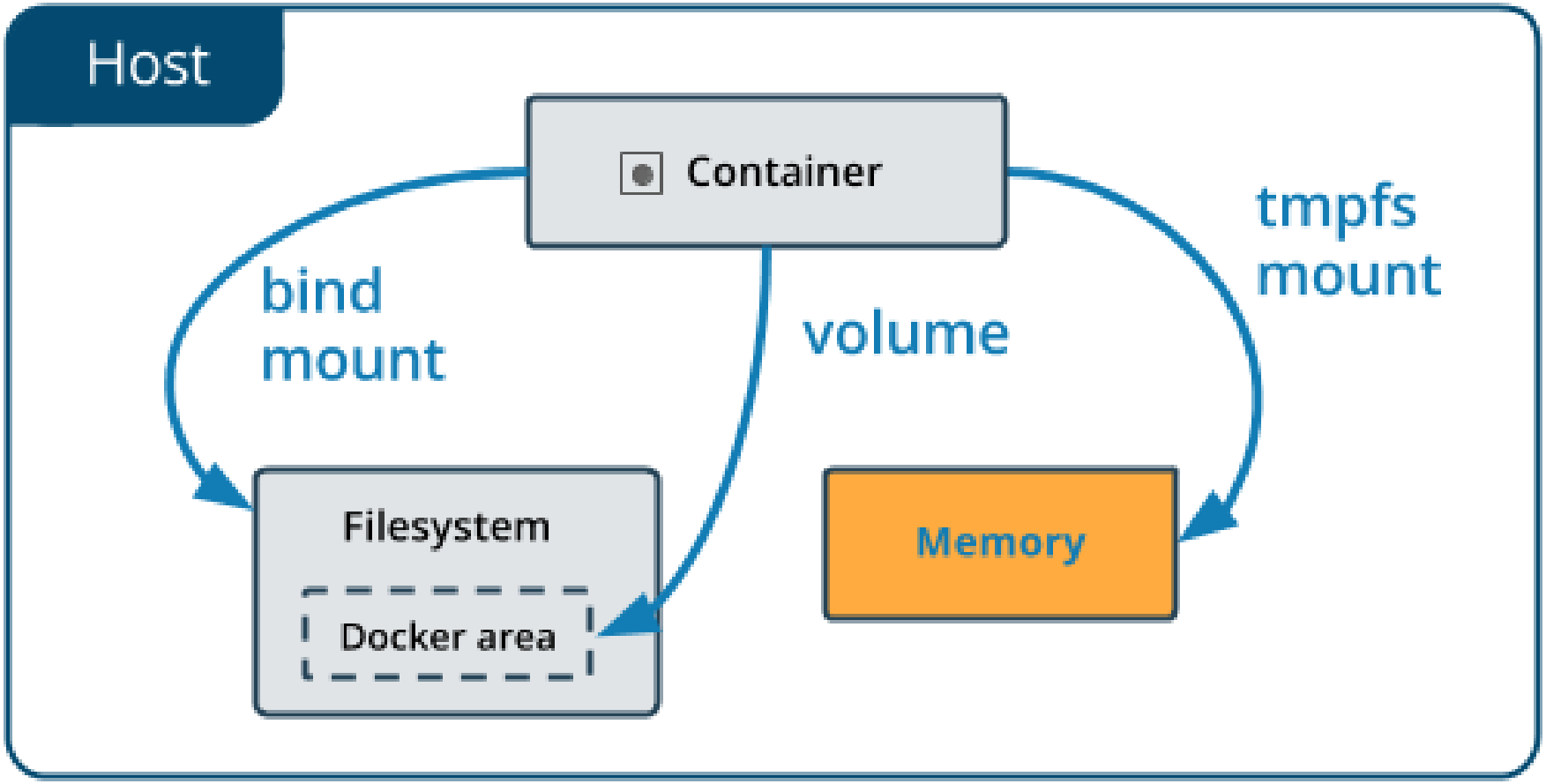
https://docs.docker.com/storage/storagedriver/overlayfs-driver/

Storage – Storagedriver, WER für WAS.

	EE 17.03	**EE 17.06**
Storage Driver	CentOS: devicemapper	CentOS: devicemapper
	Oracle Linux: devicemapper	Oracle Linux: devicemapper
	RHEL: devicemapper	RHEL: devicemapper, overlay2[6]
	SLES: btrfs	SLES: btrfs
	Ubuntu: aufs3	Ubuntu: aufs3

https://success.docker.com/article/Compatibility_Matrix

Storage.



Storage – Host Volume.

- Werden implizit durch docker angelegt.
- Befindet sich auf dem Dateisystem des Docker-Hosts und wird über den Container angesprochen
- Ablage einem einem definierbaren statischen Pfad
- Flexible Ablage der Daten bezüglich des genutzten Verzeichnisses und somit der genutzten Devices (SAN/local/…)
- Vorsicht bei Shared Storage Architekturen !
- Vor Nutzung als Datastore sind die Berechtigungen entspechend zu setzen

```
# docker run -v /path/on/host:/path/in/container ...
```

Storage – Anonymous Volumen.

- Werden implizit durch Docker angelegt und verwaltet
- Einfach in der Erstellung und Nutzung
- Nachnutzung in anderen Container ist kompliziert

```
# docker run -v /path/in/container ...
```

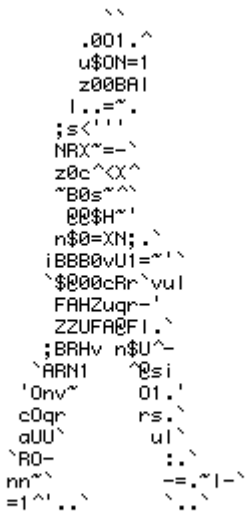
Storage – Named Volumes.

- Named Volumes sind in Prinzip ähnlich zu Anonymous Volumes
- Werden explizit durch docker angelegt und verwaltet, aber mit einem eindeutigen Namen versehen
- Ablage unter /var/lib/docker/volumes/<name>
- Gehören per default root

```
# docker volume create namedv

# docker run -v namedv:/path/in/container ...

# docker volume inspect namedv
Output
[ { "CreatedAt": "2018-01-21T21:02:53Z", "Driver": "local", "Labels": {},
"Mountpoint": "/var/lib/docker/volumes/namedv/_data", "Name": "DataVolume1",
"Options": {}, "Scope": "local" } ]
```

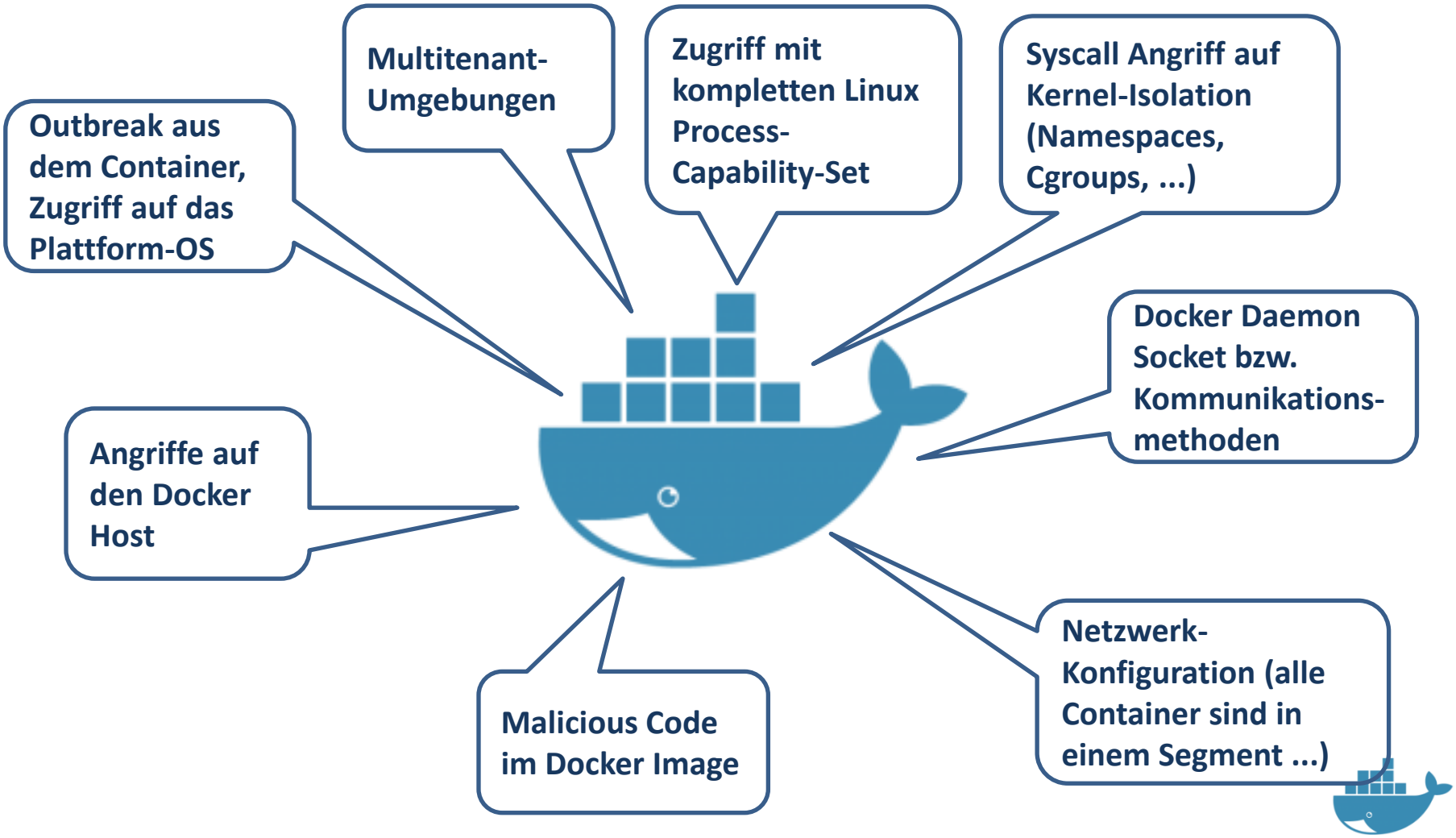
Hands On - 5.

Docker Grundlagen.

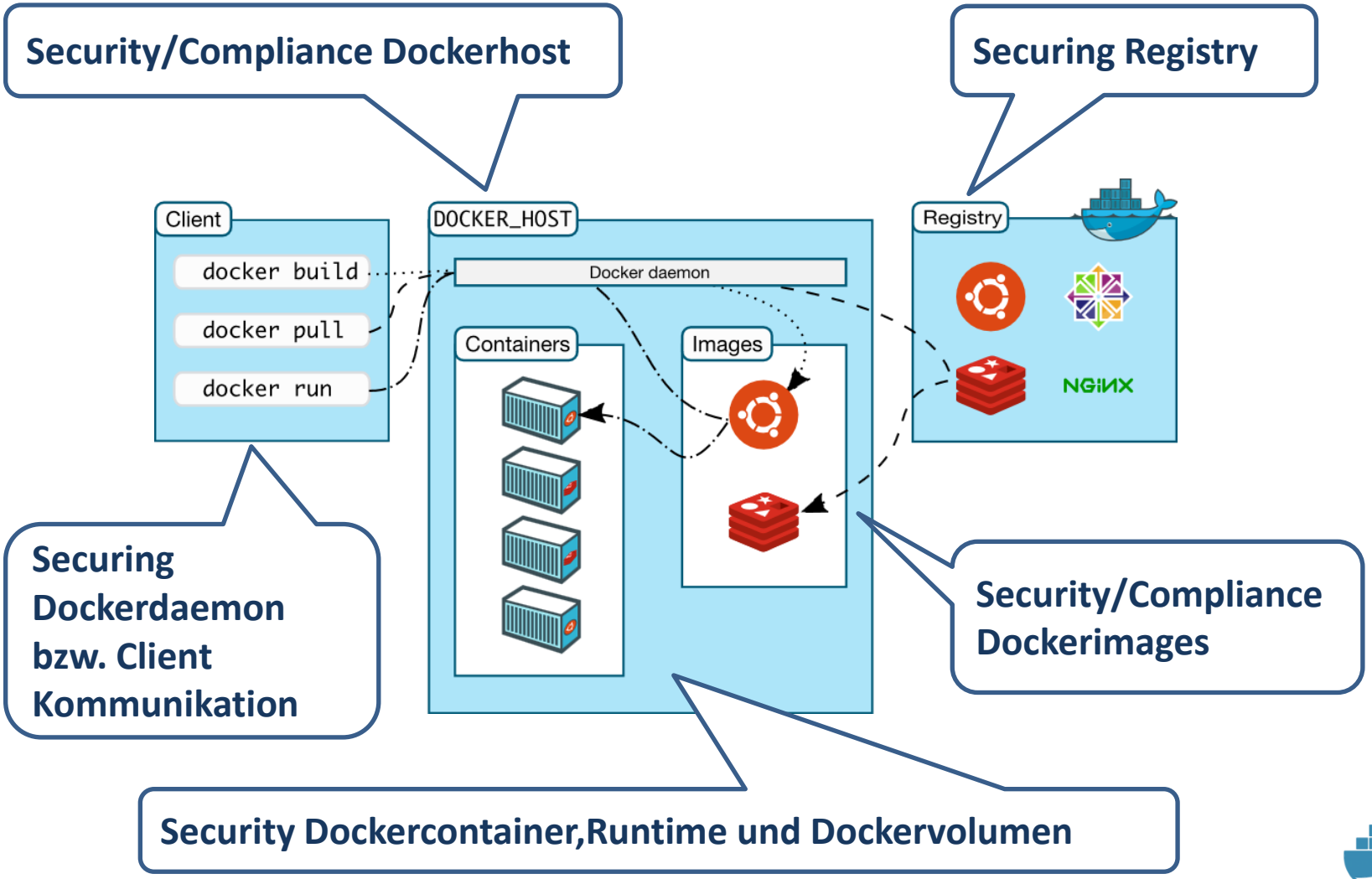


Security

Bedrohungsszenarien/Risiken.



Umsetzung Security & Compliance.



Grundsätzliches – Konstruktionsbedingte Risiken.

- Isolierte Umgebungen via Linux Kernel-Funktionalitäten(cgroups, namespaces)
- Aber ... Zugriff auf Kernel-Subsysteme (u.a /proc, /sys) + identische Usernamespaces zwischen allen Containern und der Plattform

Root-privelegierte Processe und Tasks innerhalb des Containers haben partiellen Zugriff auf den Kernel und ggf. intergrierte/gebundene Filesysteme.

Grundsätzliches – Konstruktionsbedingte Risiken.

- Container Interaktion werden über den Docker Daemon unter Nutzung **eines** Unix-Domain-Socket abgewickelt

```
# ls -laF /var/run/docker.sock
srw-rw----. 1 root docker 0 Oct 12 13:07 /var/run/docker.sock=
```

- Der Docker Daemon benötigt root-Privilegien (… dockergroup … hilft nicht wirklich)

- Desweiteren ist ein Zugriff via REST möglich, im Default http …

Alle Nutzer des Daemons haben privilegierten Zugriff auf die Plattform, Kommunikation ist per Default unverschlüsselt.

Capabilities und Docker.

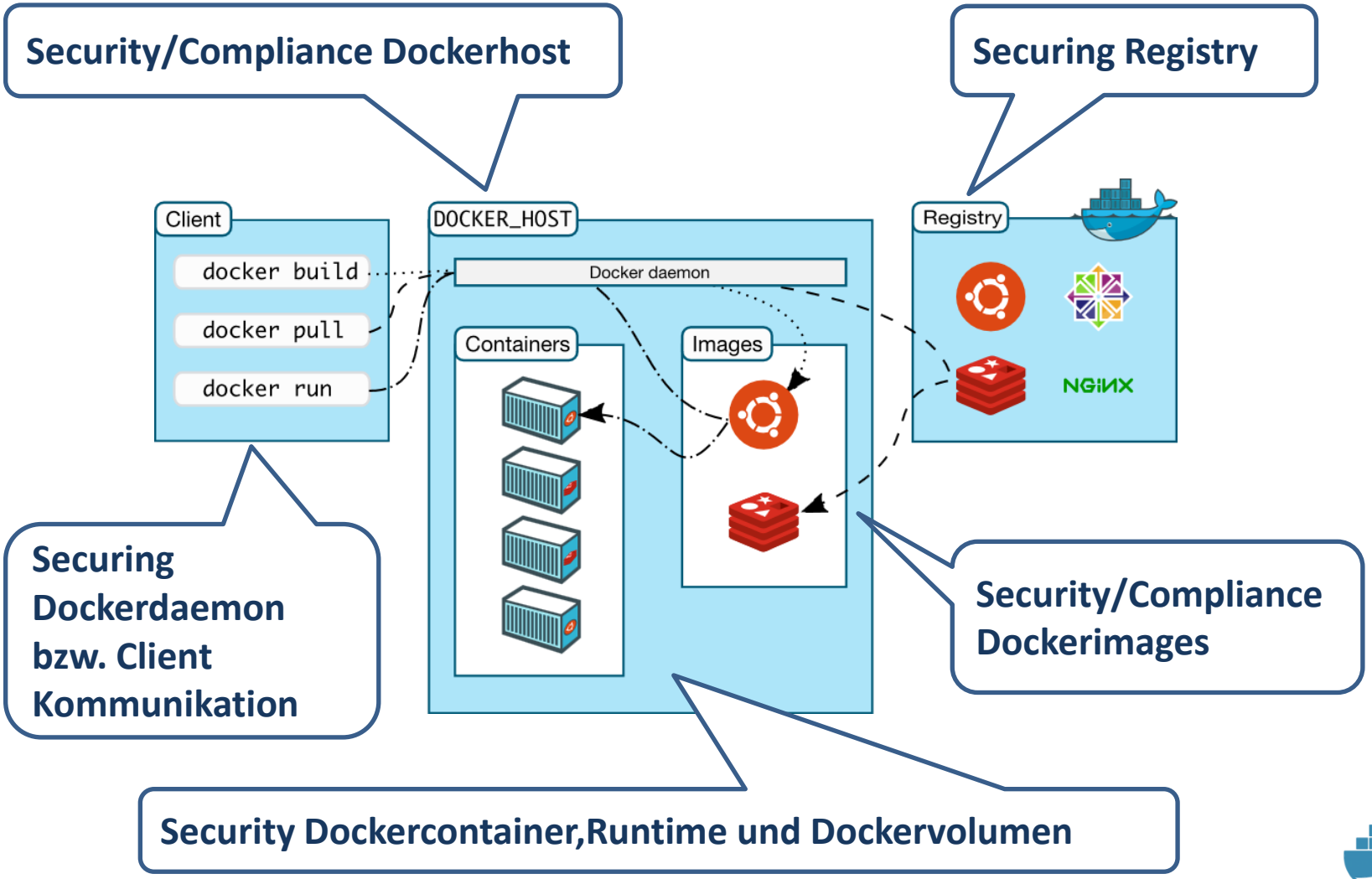
- Mit Linux-Kernel 2.2 wurden Capabilities eingeführt, fein granulierbare, diskrete Prozess-Privilegien

- Ein Großteil der unter Linux implementierten Capabilities sind jedoch auch potentiell nutzbar, um ein System zu kompromittieren.

- Beispielsweise basiert der Shocker-Exploit (06/2014,docker 0.11) mit welchem privelegierter Zugriff auf /etc/shadow der Plattform erreicht wurde.
  - Via CAP_DAC_OVERRIDE bzw. CAP_DAC_READ_SEARCH und der Nutzung des Syscalls *open_by_handle_at()*

Potentielle große Angriffsfläche in durch ein eigentlich gegenteilig geplantes Feature.

Umsetzung Security & Compliance.



Security/Compliance Dockerhost.

- **System Optimisierung und Hardening**
  - Minimalisierung der Plattforminstallation
  - SELinux
  - Seccomp
  - Grsecurity
  - Firewall/Iptables
- **Securing Services**
- **Service Optimierung**
- **Security/Compliance Software**
  - OpenScap
  - Black Duck

Securing Registry/Security & Compliance Dockerimages. (1)

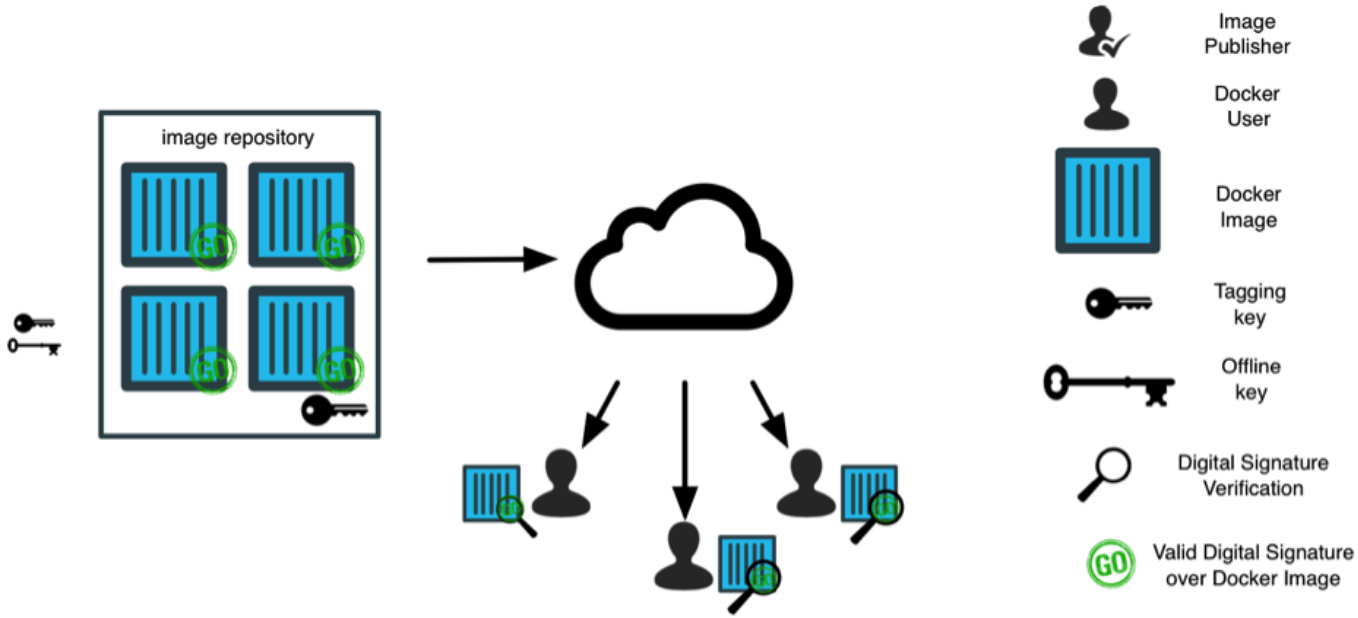
- **Nutzung einer secured private Image Registry**
  - Absicherung der Kommunikation via TLS/Domain Registry
    - openssl ...

```
docker run -d -p 443:5000 --restart=always --name registry \
  -v /my-certs/registry-certs:/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/registry-ca.pem \
  -e REGISTRY_HTTP_TLS_KEY=/certs/registry-key.pem \
  registry:2
```

  - Ggf. Nutzung zusätzlicher Docker Registry Storage Driver für die Auslagerung von der Docker Registry
    - S3, Azure, Swift, GCS, ...

Securing Registry/Security & Compliance Dockerimages. (2)

- **Nutzung selbst erstellter Images**
- **Nutzung von signierten Images**
  - Docker Content Trust (DCT) ist ab Docker Version 1.8 verfügbar
  - Signierte, mit Tags versehene Images und vertrauliche Publisher schützen vor Nutzung und Import unbekannter Images



Securing Registry/Security & Compliance Dockerimages. (3)

- **Sinnvoll ist:**
  - Nur eine Applikation/Service pro Image
  - Keine relevanten Informationen in Images einbinden

- **Patching Dockercontainer**
  - siehe Dockerfile

```
FROM scratch
MAINTAINER Heiko Stein hs@etomer.com
ADD oraclelinux-7.2-rootfs.tar.xz /
RUN yum update -y httpd httpd-devel
...
```

Securing Registry/Security & Compliance Dockerimages. (4)

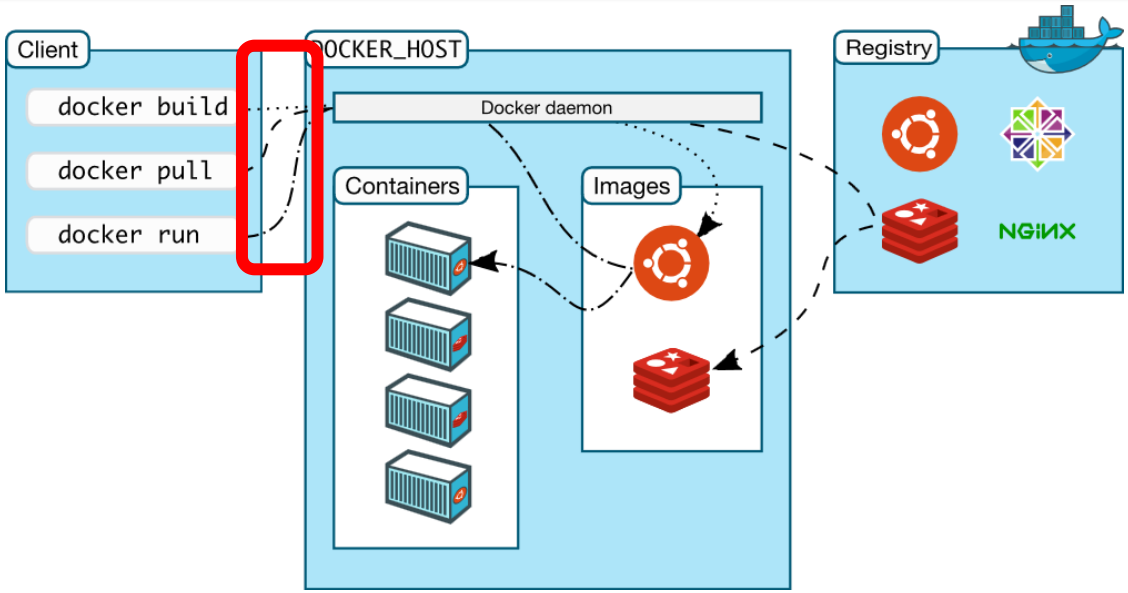
- **Docker Security Scanning**
  - CVE-Scanning der Images/Container
  - Nutzung freier Tools z.B.
    - OpenScap
    - Clair

```
# oscap-docker image oraclelinux oval eval --report report.html \
/usr/share/xml/scap/ssg/content/ssg-rhel7-oval.xml
```

Security Dockerdaemon.

- **Secure Docker Client Kommunikation**
  - Kein Zugriff auf den Dockerdaemon direkt auf dem Dockerhost
  - Absicherung via TLS
  - Zugriff via Rest/https nur von vertrauenswürdigen, dedizierten Clients

```
# dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem \
--tlskey=server-key.pem -H=0.0.0.0:2376
```



Security Dockercontainer, Runtime und Dockervolumen . (1)

- **Nicht alle Umgebungen/Use-Cases sind für Docker geeignet**
  - Vermeidung von ssh-Login
  - Vermeidung von --privileged Mode für Container
  - Saubere Trennung/Isolation von genutzten Volumen

- **Container mit dediziertem User starten (!= root)**

```
# docker run --user hs -i oracleunix
```

- **Limitierung des externen Traffic zu den Containern via iptables**
  - Interface docker0 wird als Bridge von allen Containern als Default genutzt via veth

Security Dockercontainer, Runtime und Dockervolumen . (2)

- **Syscallfilter via seccomp**

```
# docker run -i --security-opt "seccomp=/profiles/docker.json" ...
```

- **Nutzung von AppArmor-Profilen**
  - Zuordnung/imitierung von Rechten bzw. Implementation von Mandatory Access Control (MAC) für Applikationen im Container

```
# docker run -i --security-opt "apparmor=docker-profil" ...
```

- **Separation von Dockerhost und Container**
  - Einsatz von SE-Linux Label, Roles usw.

```
# docker run -i --security-opt "label:type:svirt_apache" ...
```

Security Dockercontainer, Runtime und Dockervolumen . (3)

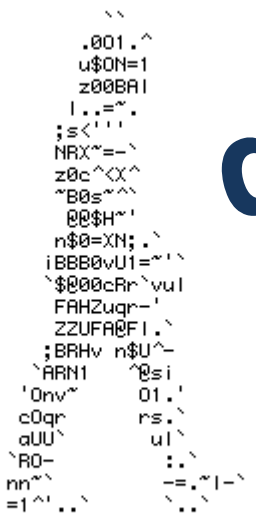
- **Limitierung/komplementäre Erweiterung der Capabilities im Container**
  - --cap-add / --cap-drop

```
# docker run -it --cap-drop=KILL ...
```

- **Nutzung von Constraints zur Quotierung von Systemresourcen**
  - Memory/CPU-Shares/ etc.
  - Schutz/Isolation der Resourcen des Dockerhost

```
# docker run -it -m 200M --memory-swap 400M ...
```

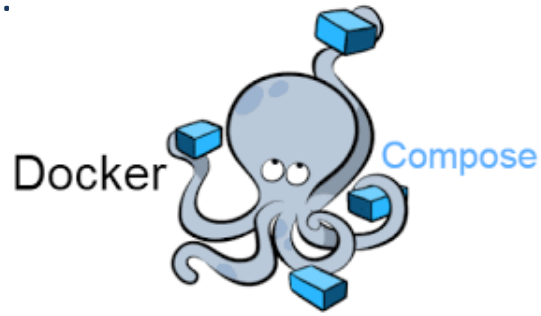
Docker Grundlagen.



docker-compose

Was ist docker-compose ?.

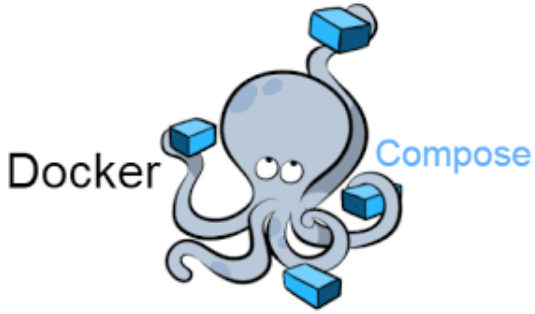
- **Wofür ?**
  - Für den Einsatz von Docker im Deployment oder auch für komplexere Aufgaben fehlen noch die Orchestrierung, also die Koordination mehrerer Docker-Container, die gemeinsam aufgesetzt, gestartet und beendet werden um bestimmte Aufgabenstellungen zu bewältigen.



- **Wie also ?**
  - Compose ist ein Tool zur Definition und Ausführung von Multi-Container-Docker-Anwendungen.
  - Mit Compose wird in einer oder mehreren YAML-Datein, Dienste der Anwendung zu konfiguriert.

Features von docker-compose.

- Starten, Stoppen und Wiederherstellen von Diensten
- Anzeigen des Status laufender Dienste
- Streaming der Log-Ausgabe von laufenden Diensten
- Ausführen eines einmaligen Befehls für einen Dienst



- Mehrere isolierte Umgebungen auf einem einzigen Host
- Beibehaltung der Volume-Daten bei der Erstellung von Containern
- Nur Container neu erstellen, die sich geändert haben
- Unterstützt Variablen und das Verschieben einer Komposition zwischen Umgebungen

Hands On - 6.

docker-compose (1).

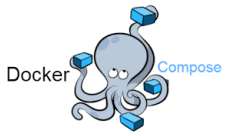
docker-compose.yml.

- In dieser Datei werden die Container definiert und ihre Beziehungen untereinander.
- Verzeichnisse/Volumen, Ports freigeben
- ....

```
version: "3.9"
services:
  container1:
    image: nginx
```

```
version: "3.9"
services:
  container1:
    image: nginx
  container2:
    image: nginx
```

https://docs.docker.com/compose/compose-file/compose-file-v3/



docker-compose (2).

docker-compose.yml.

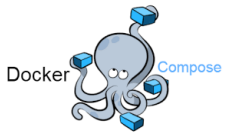
- In dieser Datei werden die Container definiert und ihre Beziehungen untereinander.
- Verzeichnisse/Volumen, Ports freigeben

- Referenz auf zu verwendende Images

```
version: "3.9"
services:
  container1:
    image: nginx
```

```
version: "3.9"
services:
  container1:
    image: nginx
  container2:
    image: nginx
```

https://docs.docker.com/compose/compose-file/compose-file-v3/



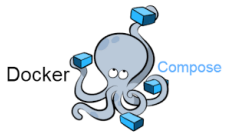
docker-compose (2).

- Referenz auf zu verwendendes Dockerfile

```
version: "3.9"
services:
  container1:
    build:
      context: ./websrv
      dockerfile: DOCKERFILE.ubuntu_linked
```

- Portdefinition

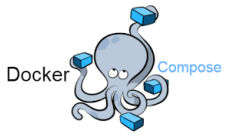
```
version: "3.9"
services:
  container1:
    build:
      context: ./websrv
    ports:
      - "8890:80"
      - "49125:22"
```



docker-compose (3).

- Definition Volumes

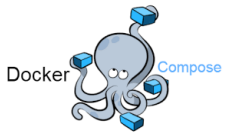
```
version: "3.9"
services:
  container1:
    build:
      context: ./websrv
    ports:
      - "8890:80"
      - "49125:22"
    volumes:
      - "/data/1:/root/data1"
      - type: bind
        source: "/certs"
        target: "/root/certs"
```



docker-compose (4).

- Links und Dependencies

```
version: "3.9"
services:
  ubuntu_linked:
    container_name: ubuntu_linked
    links:
      - nginx-8889
    depends_on:
      - nginx-8889
    command: "sleep 9000"
    build:
      context: ./dockerfile
      dockerfile: DOCKERFILE.ubuntu_linked

  nginx-8889:
    container_name: nginx-8889
    image: nginx
    ports:
      - 8889:80
    volumes:
      - type: bind
        source: ./html
        target: /usr/share/nginx/html
```



docker-compose (4).

- Commands

```
# docker-compose
Usage:  docker compose [OPTIONS] COMMAND
  build       Build or rebuild services
  convert     Converts the compose file to platform's canonical format
  cp          Copy files/folders between a service container and the local filesystem
  create      Creates containers for a service.
  down        Stop and remove containers, networks
  events      Receive real time events from containers.
  exec        Execute a command in a running container.
  images      List images used by the created containers
  kill        Force stop service containers.
  logs        View output from containers
  ls          List running compose projects
  pause       Pause services
  port        Print the public port for a port binding.
  ps          List containers
  pull        Pull service images
  push        Push service images
  restart     Restart service containers
  rm          Removes stopped service containers
  run         Run a one-off command on a service.
  start       Start services
  stop        Stop services
  top         Display the running processes
  unpause     Unpause services
  up          Create and start containers
  version     Show the Docker Compose version information
```

Diskussion.

Fragen ?

Quellen.

- Docker Dokumentation
  - https://docs.docker.com/
- Docker security
  - https://docs.docker.com/engine/security/security/
- Docker Blogs
  - https://blog.docker.com/
- Docker Release Notes
  - https://github.com/moby/moby/releases/
- Security compliance of RHEL7 Docker containers
  - https://www.open-scap.org/resources/documentation/security-compliance-of-rhel7-docker-containers/
- Shocker / Docker Breakout PoC
  - https://github.com/gabrtv/shocker
- Compose
  - https://docs.docker.com/compose/

Quellen.

- OCI
  - https://www.opencontainers.org/
- Cloud Native Computing Foundation
  - https://www.cncf.io/
- Containerd
  - https://containerd.io/
  - https://mobyproject.org/
- Moby Project
  - https://mobyproject.org/
- gRPC
  - https://grpc.io/
- Issues
  - https://github.com/moby/moby/issues/