Docker Kompaktkurs

Agenda

1. Docker-Grundlagen
    - Übersicht Architektur
    - Was ist ein Container ?
    - Was sind container images
    - Container vs. Virtuelle Maschine
    - Was ist ein Dockerfile
2. Docker-Installation
    - Installation Docker unter Ubuntu mit snap
    - Installation Docker unter SLES 15
    - Installation Docker unter Ubuntu mit Docker Repo
3. Docker-Befehle
    - Docker Grundlagen run
    - Die einzelnen Image-Schichten anschauen
    - Logs anschauen - docker logs - mit Beispiel nginx
    - docker run
    - Docker container/image stoppen/löschen
    - Docker containerliste anzeigen
    - Docker nicht verwendete Images/Container löschen
    - Docker container analysieren
    - Docker container in den Vordergrund bringen - attach
    - Aufräumen - container und images löschen
    - Nginx mit portfreigabe laufen lassen
    - docker-commit
    - images taggen und tags löschen
4. Docker - Szenarien
    - Verbindung zu nginx mit anderem Container testen

1. Dockerfile - Examples
    - Ubuntu mit hello world
    - Ubuntu mit ping
    - Nginx mit content aus html-ordner
    - ssh server
2. Docker-Container Examples
    - 2 Container mit Netzwerk anpingen
    - Container mit eigenem privatem Netz erstellen
3. Container-Image bei Laufzeit konfigurieren
    - Container-Image bei Laufzeit konfigurieren
4. Docker-Daten persistent machen / Shared Volumes
    - Überblick
    - Volumes

- bind-mounts

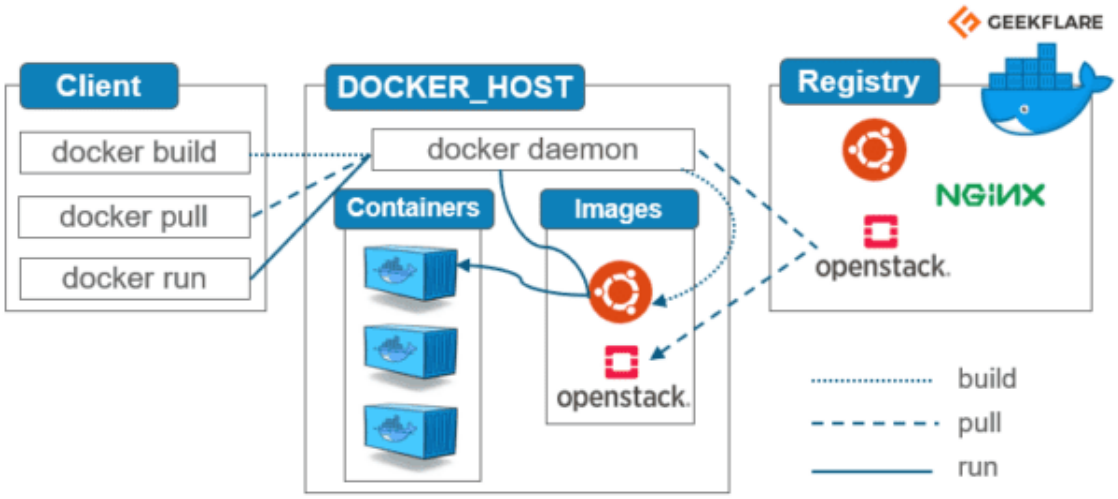
5. Docker-Netzwerk
   - Netzwerk
   - Internes Netz ohne Internetzugang aufsetzen
6. Docker Compose
   - yaml-format
   - Ist docker-compose installiert?
   - Example with Wordpress / MySQL
   - Example with Wordpress / Nginx / MariadB
   - Example with Ubuntu and Dockerfile
   - Logs in docker - compose
   - docker-compose und replicas
   - restart policies
7. Docker Swarm
   - Docker Swarm Beispiele
8. Docker Registry
   - Privater Registry Server
9. Docker Reverse Proxy (vorgeschaltet)
   - Docker mit Reverse Proxy verwenden
10. Docker Security
    - Docker Security
11. Docker Monitoring
    - Docker Monitoring - cAdvisor/Prometheus
    - Prometheus Funktionsweise
12. Docker - Dokumentation
    - Vulnerability Scanner with docker
    - Vulnerability Scanner mit snyk
    - Parent/Base - Image bauen für Docker
13. Linux und Docker Tipps & Tricks allgemein
    - Auf ubuntu root-benutzer werden
    - IP - Adresse abfragen
    - Hostname setzen
    - Läuft der ssh-server
14. VirtualBox Tipps & Tricks
    - VirtualBox 6.1. - Ubuntu - Entwicklungsserver für Docker aufsetzen
    - VirtualBox 6.1. - Shared folder aktivieren
15. Kubernetes - Überblick
    - Warum Kubernetes, was macht Kubernetes
    - Aufbau Allgemein

Backlog

1. Linux und Docker Tipps & Tricks allgemein
    - Proxy für Docker setzen
    - vim einrückung für yaml-dateien
    - YAML Linter Online
    - Basis/Parent - Image erstellen
    - Eigenes unsichere Registry-Verwenden. ohne https

Docker-Grundlagen

Übersicht Architektur



Was ist ein Container ?

```
- vereint in sich Software
- Bibliotheken
- Tools
- Konfigurationsdateien
- keinen eigenen Kernel
- gut zum Ausführen von Anwendungen auf verschiedenen Umgebungen

- Container sind entkoppelt
- Container sind voneinander unabhängig
- Können über wohldefinierte Kommunikationskanäle untereinander Informationen
austauschen

- Durch Entkopplung von Containern:
  o Unverträglichkeiten von Bibliotheken, Tools oder Datenbank können umgangen werden,
wenn diese von den Applikationen in unterschiedlichen Versionen benötigt werden.
```

Was sind container images

- Container Images werden benötigt, um zur Laufzeit Container-Instanzen zu erzeugen
- Bei Docker werden Docker Images zu Docker Containern, wenn Sie auf einer Docker Engine als Prozess ausgeführt werden.
- Man kann sich ein Docker Image als Kopiervorlage vorstellen.
    - Diese wird genutzt, um damit einen Docker Container als Kopie zu erstellen

Container vs. Virtuelle Maschine

```
VM's virtualisieren Hardware
Container virtualisieren Betriebssystem
```

Was ist ein Dockerfile

- Textdatei, die Linux - Kommandos enthält
  - die man auch auf der Kommandozeile ausführen könnte
  - Diese erledigen alle Aufgaben, die nötig sind, um ein Image zusammenzustellen
  - mit docker build wird dieses image erstellt

Docker-Installation

Installation Docker unter Ubuntu mit snap

Step 1: install

```
sudo su -
snap install docker

### Attention: if you want to use bind-mount to access directories
### all over the place, you need to install it differently
### See:
### https://snapcraft.io/docs/install-modes
snap install --devmode docker
```

Step 2: Validate if service is running and enabled ?

```
systemctl status snap.docker.dockerd.service
## oder (aber veraltet)
service snap.docker.dockerd status

systemctl is-enabled snap.docker.dockerd.service
## Dienst aktivieren (er wird dann auch beim nächsten Boot gestartet) - Windows
Autostart
systemctl enable snap.docker.dockerd.service
```

Optional: Find the name of the service

```
## for information retrieval
snap info docker
systemctl list-units
systemctl list-units -t service
systemctl list-units -t service | grep docker
```

```
systemctl stop snap.docker.dockerd.service
systemctl status snap.docker.dockerd.service
systemctl start snap.docker.dockerd.service

## wird der docker-dienst beim nächsten reboot oder starten des Server gestartet ?
systemctl is-enabled snap.docker.dockerd.service
```

Optional: Access to docker-daemon aus unprvileged user

```
sudo addgroup --system docker
sudo adduser $USER docker
## lädt die neue Gruppe in den User hinein
newgrp docker
sudo snap disable docker
sudo snap enable docker
```

Installation Docker unter SLES 15

Walkthrough

```
sudo zypper search -v docker*
sudo zypper install docker

## Dem Nutzer /z.B. Nutzer kurs die Gruppe docker hinzufügen
## damit auch dieser den Docker-daemon verwenden darf
sudo groupadd docker
sudo usermod -aG docker $USER

### Unter SLES werden Dienste nicht automatisch aktiviert und gestartet !!!
## Service für start nach Boot aktivieren
newgrp docker
sudo systemctl enable docker.service
## Docker dienst starten
sudo systemctl start docker.service
```

Ausführlich mit Ausgaben

```
sudo zypper search -v docker*

Repository-Daten werden geladen...
Installierte Pakete werden gelesen...

sudo zypper install docker

Dienst 'Basesystem_Module_x86_64' wird aktualisiert.
Dienst 'Containers_Module_x86_64' wird aktualisiert.
Dienst 'Desktop_Applications_Module_x86_64' wird aktualisiert.
Dienst 'Development_Tools_Module_x86_64' wird aktualisiert.
Dienst 'SUSE_Linux_Enterprise_Server_x86_64' wird aktualisiert.
Dienst 'Server_Applications_Module_x86_64' wird aktualisiert.
Repository-Daten werden geladen...
Installierte Pakete werden gelesen...
Paketabhängigkeiten werden aufgelöst...

Das folgende empfohlene Paket wurde automatisch gewählt:
  git-core

Die folgenden 7 NEUEN Pakete werden installiert:
  catatonit containerd docker docker-bash-completion git-core libsha1detectcoll1 runc
```

```
7 neue Pakete zu installieren.
Gesamtgröße des Downloads: 52,2 MiB. Bereits im Cache gespeichert: 0 B. Nach der
Operation werden zusätzlich 242,1 MiB belegt.
Fortfahren? [j/n/v/...? zeigt alle Optionen](j): j
Paket libsha1detectcoll1-1.0.3-2.18.x86_64 abrufen
(1/7),  23,2 KiB ( 45,8 KiB entpackt)
Abrufen: libsha1detectcoll1-1.0.3-2.18.x86_64.rpm
.....................................................................................
[fertig]
Paket catatonit-0.1.5-3.3.2.x86_64 abrufen
(2/7), 257,2 KiB (696,5 KiB entpackt)
Abrufen: catatonit-0.1.5-3.3.2.x86_64.rpm
.....................................................................................
[fertig]
Paket runc-1.1.4-150000.33.4.x86_64 abrufen
(3/7),   2,6 MiB (  9,1 MiB entpackt)
Abrufen: runc-1.1.4-150000.33.4.x86_64.rpm
.....................................................................................
[fertig]
Paket containerd-1.6.6-150000.73.2.x86_64 abrufen
(4/7),  17,7 MiB ( 74,2 MiB entpackt)
Abrufen: containerd-1.6.6-150000.73.2.x86_64.rpm
.....................................................................................
[fertig]
Paket git-core-2.35.3-150300.10.15.1.x86_64 abrufen
(5/7),   4,8 MiB ( 26,6 MiB entpackt)
Abrufen: git-core-2.35.3-150300.10.15.1.x86_64.rpm
.....................................................................................
[fertig]
Paket docker-20.10.17_ce-150000.166.1.x86_64 abrufen
(6/7),  26,6 MiB (131,4 MiB entpackt)
Abrufen: docker-20.10.17_ce-150000.166.1.x86_64.rpm
.....................................................................................
[fertig]
Paket docker-bash-completion-20.10.17_ce-150000.166.1.noarch abrufen
(7/7), 121,3 KiB (113,6 KiB entpackt)
Abrufen: docker-bash-completion-20.10.17_ce-150000.166.1.noarch.rpm
.....................................................................................
[fertig]

Überprüfung auf Dateikonflikte läuft:
.....................................................................................
[fertig]
(1/7) Installieren: libsha1detectcoll1-1.0.3-2.18.x86_64
.....................................................................................
[fertig]
(2/7) Installieren: catatonit-0.1.5-3.3.2.x86_64
.....................................................................................
[fertig]
(3/7) Installieren: runc-1.1.4-150000.33.4.x86_64
.....................................................................................
```

```
[fertig]
(4/7) Installieren: containerd-1.6.6-150000.73.2.x86_64
.....................................................................................
[fertig]
(5/7) Installieren: git-core-2.35.3-150300.10.15.1.x86_64
.....................................................................................
[fertig]
Updating /etc/sysconfig/docker ...
(6/7) Installieren: docker-20.10.17_ce-150000.166.1.x86_64
.....................................................................................
[fertig]
(7/7) Installieren: docker-bash-completion-20.10.17_ce-150000.166.1.noarch
.....................................................................................
[fertig]

sudo groupadd docker
sudo usermod -aG docker $USER
// logout

newgrp docker
sudo systemctl enable docker.service
sudo systemctl start docker.service
```

Installation Docker unter Ubuntu mit Docker Repo

Walkthrough

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Läuft der Dienst (dockerd)

```
systemctl status docker
```

docker-compose ?

```
## herausfinden, ob docker compose installieren
docker compose version
```

Docker-Befehle

Docker Grundlagen run

```
## docker hub durchsuchen
docker search hello-world

docker run <image>
## z.b. // Zieht das image aus docker hub
## hub.docker.com
docker run hello-world

## images die lokal vorhanden
docker images

## container (laufende)
docker container ls
## container (vorhanden, aber beendet)
docker container ls -a

## z.b hilfe für docker run
docker help run
```

Die einzelnen Image-Schichten anschauen

```
## Beispiel
docker pull ubuntu
docker images
docker history ubuntu
```

Logs anschauen - docker logs - mit Beispiel nginx

Allgemein

```
## Erstmal nginx starten und container-id wird ausgegeben
docker run -d nginx
a234
docker logs a234 # a234 sind die ersten 4 Ziffern der Container ID
```

Laufende Log-Ausgabe

```
docker logs -f a234
## Abbrechen CTRL + c
```

docker run

Beispiel (binden an ein terminal), detached

```
## before that we did
docker pull ubuntu:xenial
## image vorhanden
docker images

docker run -t -d --name my_xenial ubuntu:xenial
## will wollen überprüfen, ob der container läuft
docker container ls


## in den Container reinwechsel
docker exec -it my_xenial bash
docker exec my_xenial cat /etc/os-release
##
```

Docker container/image stoppen/löschen

```
docker stop ubuntu-container
## Kill it if it cannot be stopped -be careful
docker kill ubuntu-container

## Get nur, wenn der Container nicht mehr läuft
docker rm ubuntu-container

## oder alternative
docker rm -f ubuntu-container


## image löschen
docker rmi ubuntu:xenial

## falls Container noch vorhanden aber nicht laufend
docker rmi -f ubuntu:xenial
```

Docker containerliste anzeigen

```
## besser
docker container ls
## Alle Container, auch die, die beendet worden sind
docker container ls -a


## deprecated
docker ps
```

```
## -a auch solche die nicht mehr laufen
docker ps -a
```

Docker nicht verwendete Images/Container löschen

```
DOES not always work
```

```
docker system prune
## Löscht möglicherweise nicht alles

## d.h. danach nochmal prüfen ob noch images da sind
docker images
## und händisch löschen
docker rmi <image-name>
```

Docker container analysieren

```
docker run -t -d --name mein_container ubuntu:latest
docker inspect mein_container # mein_container = container name
```

Docker container in den Vordergrund bringen - attach

docker attach - walkthrough

```
docker run -d ubuntu
1a4d...

docker attach 1a4d

## Es ist leider mit dem Aufruf run nicht möglich, den prozess wieder in den
Hintergrund zu bringen
```

interactiven Prozess nicht beenden (statt exit)

```
docker run -it ubuntu bash
## ein exit würde jetzt den Prozess beenden
## exit

## Alternativ ohne beenden (detach)
## Geht aber nur beim start mit run -it
CTRL + P, dann CTRL + Q
```

Reference:

- https://docs.docker.com/engine/reference/commandline/attach/

Aufräumen - container und images löschen

Alle nicht verwendeten container und images löschen

```
## Alle container, die nicht laufen löschen
docker container prune

## Alle images, die nicht an eine container gebunden sind, löschen
docker images prune
```

Nginx mit portfreigabe laufen lassen

Schritt 1: Container erstellen

```
## Container 1
docker run --name test-nginx -d -p 8080:80 nginx
## Container 2
docker run --name test-nginx2 -d -p 80:80 nginx
```

Schritt 2: Tests durchführen

```
docker container ls
lsof -i
cat /etc/services | grep 8080
## Mit Browser in Remote Desktop testen: 192.168.56.104:8080
## oder
curl http://localhost:8080
docker container ls
```

Schritt 3: Container stoppen und probieren zu erreichen

```
## wenn der container gestoppt wird, keine ausgabe mehr, weil kein webserver
docker stop test-nginx
## Mit Browser in Remote Desktop testen: 192.168.56.104:8080
## oder
curl http://localhost:8080
```

docker-commit

Schritt 1: beispiel

```
## start clean
docker images | grep ubuntu
## eventually delete images

## ubuntu starts bash by default
docker run -it --name=ubuntu-customized ubuntu:latest
## innerhalb der shell
apt update
apt install telnet
exit

## to get container-id
docker container ls -a
```

```
docker commit -m "included telnet" ubuntu-customized ubuntu:latest
docker images
```

Schritt 2: aufräumen

```
docker rmi ubuntu:latest
```

images taggen und tags löschen

```
docker images
docker tag -d 9f myubuntu:latest
docker tag --help
## deletes tag and keeps image, if other tags are set on that image-id
docker rmi mybuntu:latest
docker images
```

Docker - Szenarien

Verbindung zu nginx mit anderem Container testen

Step 1:

```
## nginx - server starten
docker run --name=nginx-server -d nginx

## Container - IP dieses containers it.
docker inspect nginx-server
```

Step 2:

```
## Assuming nginx-server has pod-ip 172.17.0.2
## connects to sh by default
docker run -it --rm busybox
/ # ping 172.17.02
/ # wget -O - http://172.17.0.2
```

Dockerfile - Examples

Ubuntu mit hello world

Simple Version

Schritt 1:

```
cd
mkdir Hello-World
cd Hello-World
```

```
### Schritt 2:
## nano Dockerfile
FROM ubuntu:latest
```

```
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]
```

Schritt 2:

```
nano hello.sh
```

```
##!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done
```

Schritt 3:

```
## dockertrainereu/<dein-name>-hello-docker .
## Beispiel
docker build -t dockertrainereu/jm1810-hello-docker .
docker images
docker run -d dockertrainereu/<dein-name>-hello-docker
```

Schritt 4: hochladen

```
docker login
user: dockertrainereu
pass: --bekommt ihr vom trainer--

## docker push dockertrainereu/<dein-name>-hello-docker
## z.B.
## docker push registry.domain.de/markusg/jm1810-hello-docker
docker push dockertrainereu/jm1810-hello-docker

## und wir schauen online, ob wir das dort finden
```

Advanced Version

```
### Schritt 1:
cd
mkdir Hello-World
cd Hello-World

### Schritt 2:
## nano Dockerfile
FROM ubuntu:latest
```

```
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]

### Schritt 3:
nano hello.sh
##!/bin/bash
while true
do
  echo hello-docker
done

### Schritt 4:
## docker build -t dockertrainereu/<dein-name>-hello-docker .
## Beispiel
docker build -t dockertrainereu/jm-hello-docker .
docker images
docker run -d -t --name hello dockertrainereu/<dein-name>-hello-docker
docker exec -it hello sh

docker login
user: dockertrainereu
pass: --bekommt ihr vom trainer--

## docker push dockertrainereu/<dein-name>-hello-docker
## z.B.
docker push dockertrainereu/jm-hello-docker

## und wir schauen online, ob wir das dort finden
```

Ubuntu mit ping

Schritt 1: Container bauen

```
mkdir myubuntu
cd myubuntu/
```

```
## nano Dockerfile
FROM ubuntu:latest
RUN apt-get update; apt-get install -y inetutils-ping
CMD ["/bin/bash"]
```

```
docker build -t myubuntu .
docker images
## -t wird benötigt, damit bash WEITER im Hintergrund im läuft.
## auch mit -d (ohne -t) wird die bash ausgeführt, aber "das Terminal" dann direkt
beendet
## -> container läuft dann nicht mehr
```

Schritt 1b: Dockerfile aufhübschen (best practice)

```
FROM ubuntu:latest
RUN apt-get update && \
  apt-get install -y inetutils-ping && \
  rm -rf /var/lib/apt/lists/*
```

```
## With comments
FROM ubuntu:latest
## Better use && and new lines
## RUN apt-get update; apt-get install -y inetutils-ping
RUN apt-get update && \
  apt-get install -y inetutils && \
  rm -rf /var/lib/apt/lists/*

## is not needed, because CMD["bash"] is default in ubuntu:latest
## CMD ["/bin/bash"]
```

Schritt 2: Container testen

```
docker run -d -t --name container-ubuntu myubuntu
docker container ls
## in den container reingehen mit dem namen des Containers: myubuntu
docker exec -it container-ubuntu bash
ls -la
```

Schritt 3: ping von neuem container zu altem Container

```
## Zweiten Container starten
docker run -d -t --name container-ubuntu2 myubuntu

## docker inspect to find out ip of other container
## 172.17.0.3
docker inspect <container-id>

## Ersten Container -> 2. anpingen
docker exec -it container-ubuntu bash
## Jeder container hat eine eigene IP
ping 172.17.0.3
```

Nginx mit content aus html-ordner

Schritt 1: Simple Example

```
## das gleich wie cd ~
## Heimatverzeichnis des Benutzers root
cd
mkdir nginx-test
cd nginx-test
mkdir html
cd html/
nano index.html
```

```
## Text reinschreiben und speichern
Text, den du rein haben möchtest
```

```
cd ..
nano Dockerfile
```

```
FROM nginx:latest
COPY html /usr/share/nginx/html
```

```
## nameskürzel z.B. jm1
docker build -t dockertrainereu/jm1-hello-web .
docker images
```

Schritt 2: Push build

```

## eventually you are not logged in
## docker login
## DURCH euren Namen ersetzen -> jm1
docker push dockertrainereu/jm1-hello-web
##aus spass geloescht
docker rmi dockertrainereu/jm1-hello-web
```

Schritt 3: docker laufen lassen

```
## und direkt aus der Registry wieder runterladen
docker run --name hello-web -p 8080:80 -d dockertrainereu/jm1-hello-web

## laufenden Container anzeigen lassen
docker container ls
## oder alt: deprecated
docker ps

curl http://localhost:8080


##
docker rm -f hello-web
docker rmi dockertrainereu/jm1-hello-web
```

ssh server

```
cd
mkdir devubuntu
cd devubuntu
## vi Dockerfile
```

```
FROM ubuntu:latest


```

```
RUN apt-get update && \
    DEBIAN_FRONTEND="noninteractive" apt-get install -y inetutils-ping openssh-server
&& \
    rm -rf /var/lib/apt/lists/*

RUN mkdir /run/sshd && \
    echo 'root:root' | chpasswd && \
    sed -ri 's/^#?PermitRootLogin\s+.*/PermitRootLogin yes/' /etc/ssh/sshd_config && \
    sed -ri 's/UsePAM yes/#UsePAM yes/g' /etc/ssh/sshd_config && \
    mkdir /root/.ssh

EXPOSE 22/tcp

CMD ["/usr/sbin/sshd","-D"]
```

```
docker build -t devubuntu .
docker run --name=devjoy -p 2222:22  -d -t devubuntu3

ssh root@localhost -p 2222
## example, if your docker host ist 192.168.56.101 v
ssh root@192.168.56.101 -p 2222
```

Docker-Container Examples

2 Container mit Netzwerk anpingen

```
clear
docker run --name dockerserver1 -dit ubuntu
docker run --name dockerserver2 -dit ubuntu
docker network ls
docker network inspect bridge
## dockerserver1 - 172.17.0.2
## dockerserver2 - 172.17.0.3
docker container ls
docker exec -it dockerserver1 bash
## im container
apt update; apt install -y iputils-ping
ping 172.17.0.3
```

Container mit eigenem privatem Netz erstellen

```
clear
## use bridge as type
## docker network create -d bridge test_net
## by bridge is default
docker network create test_net
docker network ls
docker network inspect test_net

## Container mit netzwerk starten
docker container run -d --name nginx1 --network test_net nginx
```

```
docker network inspect test_net

## Weiteres Netzwerk (bridged) erstellen
docker network create demo_net
docker network connect demo_net nginx1

## Analyse
docker network inspect demo_net
docker inspect nginx1

## Verbindung lösen
docker network disconnect demo_net nginx1

## Schauen, wir das Netz jetzt aussieht
docker network inspect demo_net
```

Container-Image bei Laufzeit konfigurieren

Container-Image bei Laufzeit konfigurieren

```
docker run --detach --name some-mariadb --env MARIADB_USER=example-user --env
MARIADB_PASSWORD=my_cool_secret --env MARIADB_ROOT_PASSWORD=my-secret-pw
mariadb:latest
## kurzform
docker run --detach --name some-mariadb -e MARIADB_USER=example-user -e
MARIADB_PASSWORD=my_cool_secret -e MARIADB_ROOT_PASSWORD=my-secret-pw  mariadb:latest
## kurzform
```

Docker-Daten persistent machen / Shared Volumes

Überblick

Overview

```
bind-mount  # not recommended
volumes
tmpfs
nfs-mounts
```

Disadvantages

```
stored only on one node (besides nfs)
Does not work well in cluster
```

Alternative for cluster

```
glusterfs
cephfs
nfs
```

```
## Stichwort
ReadWriteMany
```

Volumes

Storage volumes verwalten

```
docker volume ls
docker volume create test-vol
docker volume ls
docker volume inspect test-vol
```

Storage volumes in container einhängen

```
## Schritt 1
docker run -it --name=container-test-vol --mount target=/test_data,source=test-vol
ubuntu bash
1234ad# touch /test_data/README
exit
## stops container
docker container ls -a
```

```
## Schritt 2
Inhalt im Filesystem betrachten.
docker volume inspect test-vol
## ziehen wir den pfad raus
ls -la /var/snap/docker/common/var-lib-docker/volumes/test-vol/_data
cat /var/snap/docker/common/var-lib-docker/volumes/test-vol/_data/README
```

```
## Schritt 3:
## create new container and check for /test_data/README
docker run -it --name=container-test-vol2 --mount target=/test_data_neu,source=test-
vol ubuntu bash
ab45# ls -la /test_data_neu/README
```

Storage volume löschen

```
## Zunächst container löschen
docker rm container-test-vol
docker rm container-test-vol2
docker volume rm test-vol
```

bind-mounts

```
## andere Verzeichnis als das Heimatverzeichnis von root funktionieren aktuell nicht
mit
## snap install docker
## wg. des Confinements
docker run -d -it  --name devtest --mount type=bind,source=/root,target=/app
nginx:latest
```

```
docker exec -it devtest bash
/# cd /app
```

Docker-Netzwerk

Netzwerk

1. Übersicht

```
3 Typen

o none
o bridge (Standard-Netzwerk)
o host

### Additionally possible to install
o overlay (needed for multi-node)
```

2. Kommandos

```
## Netzwerk anzeigen
docker network ls

## bridge netzwerk anschauen
## Zeigt auch ip der docker container an
docker inspect bridge

## im container sehen wir es auch
docker inspect ubuntu-container
```

3. Test mit eigenem Netz (bridged)

Schritt 1: Netzwerk erstellen

```
docker network create -d bridge app_net
docker inspect app_net
```

Schritt 2: Netzwerk testen

```
docker run -dit --name alpine1 --network app_net alpine ash
docker run -dit --name alpine2 --network app_net alpine ash
docker run -dit --name alpine3 alpine ash

## ip rausfinden
docker inspect alpine3 | grep -i ipaddress

docker exec -it alpine1 ash
/ # ping -c2 alpine2
/ # ping -c2 alpine3
/ # ping -c2 <ip-addr>
/ # exit
```

```
docker exec -it alpine3 ash
/ # ping -c2 alpine1
/ # ping -c2 <ip-addresse>
```

Schritt 3: Test von container mit 2 Netzwerken

```
docker run -dit --name alpine4 alpine ash
docker network connect app_net alpine4
docker inspect alpine4

docker exec -it alpine4 ash
/# ping alpine1
## alpine3 lässt soch nicht mit Namen anpingen, weil mit der der default -> bridge
verbunden
## dort gibt es keine Namensauflösung
/# ping alpine3
/# ping <ip-address-alpine3>
```

4. Test mit Host-Netz

```
## start first container
docker run --network host -d nginx
## will bind to port 80
lsof -i
## start second container
docker run --network host -d nginx
## Will run only the first
docker container ls -a

CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS     NAMES
acaffd1c8ac9   nginx     "/docker-entrypoint.…"   2 minutes ago    Exited (1) 2
minutes ago             priceless_kapitsa
476a52395eb9   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes
focused_buck

docker rm -f ac 47

## und jetzt ist der Server von aussen erreichbar
```

5. Port nach aussen (ausserhalb des Server) zur Verfügung stellen

```
## bridged netzwerk wird aufgebaut.
## und auf server wird der port 80 geöffnet
docker run --name nginx_extern -p 80:80 -d nginx
docker run --name nginx_extern8080 -p 8080:80 -d nginx

## Entweder browser (d.h. auf remote desktop) oder curl / wget
## nginx muss antworten
```

```
wget -O - http://localhost:8080
wget -O - http://localhost:80
```

Eigenes Netz erstellen

```
docker network create -d bridge test_net
docker network ls

docker container run -d --name nginx --network test_net nginx
docker container run -d --name nginx_no_net --network none nginx

docker network inspect none
docker network inspect test_net

docker inspect nginx
docker inspect nginx_no_net
```

Netzwerk rausnehmen / hinzufügen

```
docker network disconnect none nginx_no_net
docker network connect test_net nginx_no_net

### Das Löschen von Netzwerken ist erst möglich, wenn es keine Endpoints
### d.h. container die das Netzwerk verwenden
docker network rm test_net
```

Internes Netz ohne Internetzugang aufsetzen

Why ?

- When containers are in an internal net, they are not able to reach the outside world (aka internet)

Schritt 1: Setup / Walkthrough

```
cd
mkdir internal
cd internal/
nano docker-compose.yml
```

```
services:
   partner1:
     image: nginx
     networks:
       - private
   partner2:
     image: nginx
     networks:
       - private

networks:
```

```
  private:
    internal: true
```

```
docker compose up -d
docker network ls
docker compose ps
## Update will not work now
docker exec -it internal-partner1-1 bash
/ # apt update
```

Schritt 2: Connect to the outside world

```
docker network create -d bridge app-public
docker network connect app-public internal-partner1-1
## Now it works
docker exec -it internal-partner1-1 bash
/ # apt update
```

Docker Compose

yaml-format

```
## Kommentare

## Listen
- rot
- gruen
- blau

## Mappings
Version: 3.7

## Mappings können auch Listen enthalten
expose:
  - "3000"
  - "8000"

## Verschachtelte Mappings
build:
  context: .
  labels:
    label1: "bunt"
    label2: "hell"
```

Ist docker-compose installiert?

```
## besser. mehr infos
docker-compose version
docker-compose --version
```

Example with Wordpress / MySQL

Schritt 1:

```
clear
cd
mkdir wp
cd wp
nano docker-compose.yml
```

Schritt 2:

```
## docker-compose.yaml
version: "3.8"

services:
  database:
    image: mysql:5.7
    volumes:
      - database_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: mypassword
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    image: wordpress:latest
    depends_on:
      - database
    ports:
      - 8080:80
    restart: always
    environment:
      WORDPRESS_DB_HOST: database:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
    volumes:
      - wordpress_plugins:/var/www/html/wp-content/plugins
      - wordpress_themes:/var/www/html/wp-content/themes
      - wordpress_uploads:/var/www/html/wp-content/uploads

volumes:
  database_data:
  wordpress_plugins:
  wordpress_themes:
  wordpress_uploads:
```

Schritt 3:

```
docker-compose up -d
```

Schritt 4:

```
## use same network to debug
docker run --network wp_default -it --name alpinechecker alpine
```

Example with Wordpress / Nginx / MariadB

```
mkdir wp
cd wp
## nano docker-compose.yml
```

```
version: "3.7"

services:
    database:
        image: mysql:5.7
        volumes:
            - database_data:/var/lib/mysql
        restart: always
        environment:
            MYSQL_ROOT_PASSWORD: mypassword
            MYSQL_DATABASE: wordpress
            MYSQL_USER: wordpress
            MYSQL_PASSWORD: wordpress

    wordpress:
        image: wordpress:latest
        depends_on:
            - database
        ports:
            - 8080:80
        restart: always
        environment:
            WORDPRESS_DB_HOST: database:3306
            WORDPRESS_DB_USER: wordpress
            WORDPRESS_DB_PASSWORD: wordpress
        volumes:
            - wordpress_plugins:/var/www/html/wp-content/plugins
            - wordpress_themes:/var/www/html/wp-content/themes
            - wordpress_uploads:/var/www/html/wp-content/uploads
volumes:
    database_data:
    wordpress_plugins:
    wordpress_themes:
    wordpress_uploads:


### now start the system
docker-compose up -d
```

```
### we can do some test if db is reachable
docker exec -it wp_wordpress_1 bash
### within shell do
apt update
apt-get install -y telnet
## this should work
telnet database 3306

## and we even have logs
docker-compose logs
```

Example with Ubuntu and Dockerfile

Schritt 1:

```
cd
mkdir bautest
cd bautest
```

Schritt 2:

```
## nano docker-compose.yml
version: "3.8"

services:
  myubuntu:
    build: ./myubuntu
    restart: always
```

Schritt 3:

```
mkdir myubuntu
cd myubuntu
```

```
nano hello.sh
```

```
##!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done
```

```
## nano Dockerfile
FROM ubuntu:latest
RUN apt-get update; apt-get install -y inetutils-ping
COPY hello.sh .
```

```
RUN chmod u+x hello.sh
CMD ["/hello.sh"]
```

Schritt 4:

```
cd ../
## wichtig, im docker-compose - Ordner seiend
##pwd
##~/bautest
docker-compose up -d
## wird image gebaut und container gestartet

## Bei Veränderung vom Dockerfile, muss man den Parameter --build mitangeben
docker-compose up -d --build
```

Logs in docker - compose

```
##Im Ordner des Projektes
##z.B wordpress-mysql-compose-project
cd ~/wordpress-mysql-compose-project
docker-compose logs
## jetzt werden alle logs aller services angezeigt
```

docker-compose und replicas

Beispiel

```
version: "3.9"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    configs:
      - my_config
      - my_other_config
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

Ref:

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

restart policies

what is possible

```
no: Containers will not restart automatically
on-failure[:max-retries]: Restart the container if it exits with a non-zero exit code
```

```
and provide a maximum number of attempts for the Docker daemon to restart the
container
always: Always restart the container if it stops
unless-stopped: Always restart the container unless it was stopped arbitrarily or by
the Docker daemon
```

Example

```
## docker-compose.yml
services:
  web:
    image: 'gitlab/${GITLAB_VERSION}'
    restart: unless-stopped
```

Reference

- https://www.baeldung.com/ops/docker-compose-restart-policies

Docker Swarm

Docker Swarm Beispiele

Generic examples

```
## should be at least version 1.24
docker info

## only for one network interface
docker swarm init

## in our case, we need to decide what interface
docker swarm init --advertise-addr 192.168.56.101

## is swarm active
docker info | grep -i swarm
## When it is -> node command works
docker node ls
## is the current node the manager
docker info | grep -i "is manager"

## docker create additional overlay network
docker network ls

## what about my own node -> self
docker node inspect self
docker node inspect --pretty self
docker node inspect --pretty self | less
```

```
## Create our first service
docker service create redis
docker images
docker service ls
```

```
## if service-id start with  j
docker service inspect j
docker service ps j
docker service rm j
docker service ls
```

```
## Start with multiple replicas and name
docker service create --name my_redis --replicas 4 redis
docker service ls
## Welche tasks
docker service ps my_redis
docker container ls
docker service inspect my_redis

## delete service
docker service rm
```

Add additional node

```
## on first node, get join token
docker swarm join-token manager

## on second node execute join command
docker swarm join --token SWMTKN-1-07jy3ym29au7u3isf1hfhgd7wpfggc1nia2kwtqfnfc8hxfczw-
2kuhwlnr9i0nkje8lz437d2d5 192.168.56.101:2377

## check with node command
docker node ls

## Make node a simple worker
## Does not make, because no highavailable after crush node 1
## Take at LEAST 3 NODES
docker node demote <node-name>
```

expose port

```
docker service create --name my_web \
                        --replicas 3 \
                        --publish published=8080,target=80 \
                        nginx
```

Ref

- https://docs.docker.com/engine/swarm/services/

Docker Registry

Privater Registry Server

Walkthrough

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
docker pull ubuntu:16.04
docker tag ubuntu:16.04 localhost:5000/my-ubuntu
docker push localhost:5000/my-ubuntu

docker image remove ubuntu:16.04
docker image remove localhost:5000/my-ubuntu
docker pull localhost:5000/my-ubuntu
```

Reference

- https://docs.docker.com/registry/deploying/

Docker Reverse Proxy (vorgeschaltet)

Docker mit Reverse Proxy verwenden

Walktrough

```
cd
mkdir jwilder-nginx-proxy
cd jwilder-nginx-proxy
nano docker-compose.yml
```

```
version: '3'

services:
  nginx-proxy:
    image: jwilder/nginx-proxy
    ports:
      - "80:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro

  whoami:
    image: jwilder/whoami
    environment:
      - VIRTUAL_HOST=whoami.local,whoami.training.local
```

```
docker compose up -d
```

```
## Eintrag in meinem Desktop-Client gemacht
## Windows->System32->drivers->etc->hosts
## mit notepad++ -> als Administrator starten
## ip meines server
192.168.56.107 whoami.training.local  whoami
```

```
## im Browser achtung: http:// verwenden
http://whoami.training.local
```

Docker Security

Docker Security

Run container under specific user:

```
## user with id 40000 does not need to exist in container
docker run -it -u 40000 alpine

## user kurs needs to exist in container (/etc/passwd)
docker run -it -u kurs alpine
```

Default capabilities

- Set everytime a new container is started as default
- https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

Run container with less capabilities

```
cd
mkdir captest
cd captest
```

```
nano docker-compose.yml
```

```
services:
  nginx:
    image: nginx
    cap_drop:
      - CHOWN
```

```
docker compose up -d
## start and exits
docker compose ps
## what happened -> wants to do chown, but it is not allowed
docker logs captest_nginx_1
o=rwx /tmp/foo
```

```
docker compose down
```

Reference:

- https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html
- https://www.redhat.com/en/blog/secure-your-containers-one-weird-trick
- man capabilities

Docker Monitoring

Docker Monitoring - cAdvisor/Prometheus

Überblick

```
cAdvisor + Prometheus + Grafana

https://github.com/google/cadvisor
https://prometheus.io/docs/guides/cadvisor/

a. Was ist cAdvisor ?
Sammelt Metriken von allen docker containern und stellt sie zentral
zur Verfügung

b. Warum cAdvisor und Prometheus ?
cAdvisor exposes container and hardware statistics as Prometheus metrics out of the
box. By default, these metrics are served under the /metrics HTTP endpoint. This
endpoint may be customized by setting the -prometheus_endpoint and -disable_metrics or
-enable_metrics command-line flag

c. Warum Prometheus ?
Speichert zeitserien sehr gut in einen zeitreihen - datenbank (influxdb)
-> Hier Komponenten vorstellen.

d. Evtl. Grafana ?
Hier werden die Daten hübsch in Grafiken aufbereitet.
```

Aufbau / Funktionsweise prometheus

```
```

Reference:

- https://prometheus.io/docs/guides/cadvisor/

Prometheus Funktionsweise

What does it do ?

- It monitors your system by collecting data
- Data is pulled from your system by defined endpoints (http) from your cluster
- To provide data on your system, a lot of exporters are available, that
  - collect the data and provide it in Prometheus

Technical

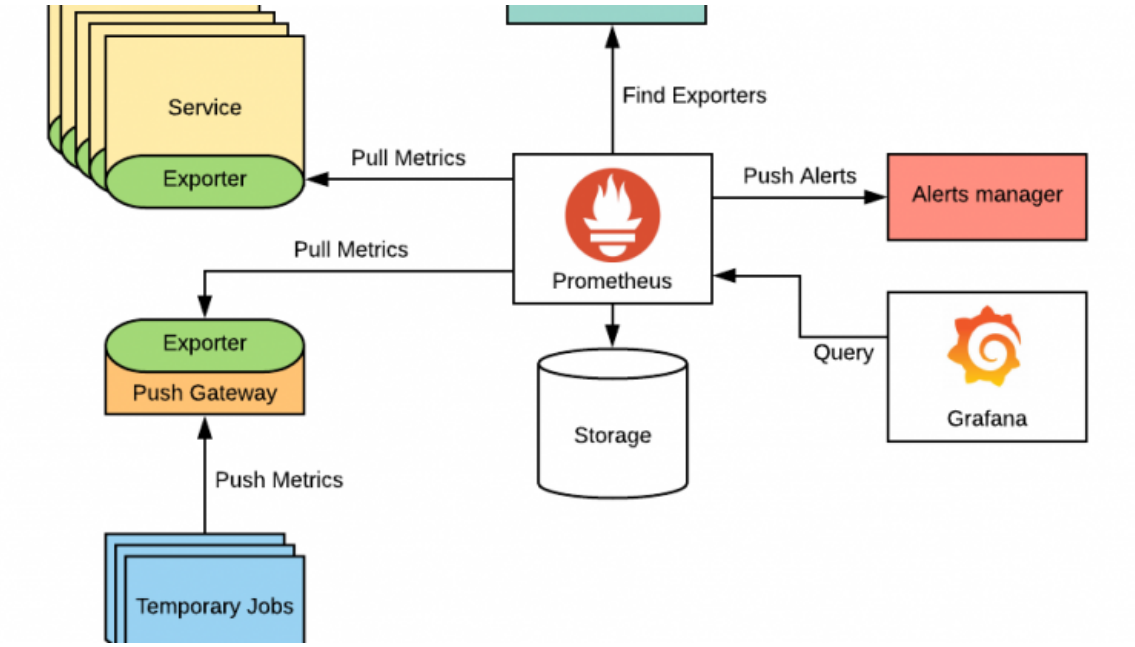
- Prometheus has a TDB (Time Series Database) and is good as storing time series with data
- Prometheus includes a local on-disk time series database, but also optionally integrates with remote storage systems.
- Prometheus's local time series database stores data in a custom, highly efficient format on local storage.
- Ref: https://prometheus.io/docs/prometheus/latest/storage/

What are time series ?

- A time series is a sequence of data points that occur in successive order over some period of time.
- Beispiel:
  - Du willst die täglichen Schlusspreise für eine Aktie für ein Jahr dokumentieren
  - Damit willst Du weitere Analysen machen

- Du würdest das Paar Datum/Preis dann in der Datumsreihenfolge sortieren und so ausgeben
- Dies wäre eine "time series"

Kompenenten von Prometheus



Quelle: https://www.devopsschool.com/

Prometheus Server

1. Retrieval (Sammeln)
   - Data Retrieval Worker
     - pull metrics data
2. Storage
   - Time Series Database (TDB)
     - stores metrics data
3. HTTP Server
   - Accepts PromQL - Queries (e.g. from Grafana)
     - accept queries

Grafana ?

- Grafana wird meist verwendet um die grafische Auswertung zu machen.
- Mit Grafana kann ich einfach Dashboards verwenden
- Ich kann sehr leicht festlegen (Durch Data Sources), so meine Daten herkommen

Docker - Dokumentation

Vulnerability Scanner with docker

- https://docs.docker.com/engine/scan/#prerequisites

Vulnerability Scanner mit snyk

- https://snyk.io/plans/

Parent/Base - Image bauen für Docker

- https://docs.docker.com/develop/develop-images/baseimages/

Linux und Docker Tipps & Tricks allgemein

Auf ubuntu root-benutzer werden

```
## kurs>
sudo su -
## password von kurs eingegeben
## wenn wir vorher der benutzer kurs waren
```

IP - Adresse abfragen

```
## IP-Adresse abfragen
ip a
```

Hostname setzen

```
## als root
hostnamectl set-hostname server.training.local
## damit ist auch sichtbar im prompt
su -
```

Läuft der ssh-server

```
systemctl status sshd
systemctl status ssh
```

VirtualBox Tipps & Tricks

VirtualBox 6.1. - Ubuntu - Entwicklungsserver für Docker aufsetzen

Vorbereitung

- Ubuntu Server 22.04 LTS - ISO herunterladen

Schritt 1: Virtuelle Maschine erstellen

```
In VirtualBox Manager -> Menu -> Maschine -> Neu (Oder Neu icon)

Seite 1:
Bei Name Ubuntu Server eingeben (dadurch wird gleich das richtige ausgewählt, bei den
Selects)
Alles andere so lassen.
Weiter

Seite 2:
```

```
Hauptspeicher mindest 4 GB , d.h. 4096 auswählen (füpr Kubernetes / microk8s)
Weiter

Seite 3:
Festplatte erzeugen ausgewählt lassen
Weiter

Seite 4:
Dateityp der Festplatte: VDI ausgewählt lassen
Weiter

Seite 5:
Art der Speicherung -> dynamisch alloziert ausgewählt lassen
Weiter

Seite 6:
Dateiname und Größe -> bei Größe mindestens 30 GB einstellen (bei Bedarf größer)
-> Erzeugen
```

Schritt 2: ISO einhängen / Netzwerk und starten / installieren

```
Neuen Server anklicken und ändern klicken:

1.
Massenspeicher -> Controller IDE -> CD (Leer) klicken
CD - Symbol rechts neben -> Optisches Laufwerk (sekundärer Master) -> klicken ->
Abbild auswählen
Downgeloadetes ISO Ubuntu 22.04 auswählen -> Öffnen klicken

2.
Netzwerk -> Adapter 2 (Reiter) anklicken -> Netzwerkadapter aktivieren
Angeschlossen an -> Host-only - Adapter

3.
unten rechts -> ok klicken
```

Schritt 3: Starten klicken und damit Installationsprozess beginnen

```
Try or install Ubuntu Server -> ausgewählt lassen

Seite 1:
Use up -.... Select your language
-> English lassen
Enter eingeben

Seite 2: Keyboard Configuration
Layout auswählen (durch Navigieren mit Tab-Taste) -> Return
German auswählen (Pfeiltaste nach unten bis German, dann return)
Identify Keyboard -> Return
Keyboard Detection starting -> Ok
Jetzt die gewünschten tasten drücken und Fragen beantworten
```

```
Layout - Variante bestätigen mit OK

-> Done

Seite 3: Choose type of install
Ubuntu - Server ausgewählt lassen

-> Done

Seite 4: Erkennung der Netzwerkkarten
(192.168.56.1x) sollte auftauchen

-> Done

Seite 5: Proxy

leer lassen

-> Done

Seite 6: Mirror Address

kann so bleiben

-> Done

Seite 7:

Guided Storage konfiguration
Entire Disk

-> Done

Seite 8: File System Summary

-> Done

Seite 9: Popup: Confirm destructive action
Bestätigen, dass gesamte Festplatte überschrieben wird
(kein Problem, da Festplatte ohnehin leer und virtuell)

-> Continue

Seite 10: Profile Setup

User eingeben / einrichten
Servernamen einrichten

-> Done

Seite 11: SSH Setup
```

```
Haken bei: Install OpenSSH Server
setzen

-> Done

Seite 12: Featured Server Snaps

Hier brauchen wir nichts auswählen, alles kann später installiert werden

-> Done

Seit 13:  Installation

Warten bis Installation Complete und dies auch unten angezeigt wird (Reboot Now):
(es dauert hier etwas bis alle Updates (unattended-upgrades) im Hintergrund
durchgeführt worden sind)

-> Reboot Now

Wenn "Failed unmounting /cdrom" kommt
dann einfach Server stoppen
-> Virtual Box Manager -> Virtuelle Maschine auswählen -> Rechte Maustaste ->
Schliessen ->  Ausschalten
```

Schritt 4: Starten des Gast-Systems in virtualbox

```
* Im VirtualBox Manager auf virtuelle Maschine klicken
* Neben dem Start - Pfeil -> Dreieck anklicken und Ohne Gui starten wählen
* System startet dann im Hintergrund (kein 2. Fenster)
```

Erklärung

- Console wird nicht benötigt, da wir mit putty (ssh) arbeiten zum Administrieren des Clusters
- Putty-Verbindung muss nur auf sein, wenn wir administrieren
- Verwendung des Clusters (nutzer/Entwickler) erfolgt ausschliesslich über kubectl in powershell !!

VirtualBox 6.1. - Shared folder aktivieren

Prepare

```
Walkthrough

## At the top menu of the virtual machine
## Menu -> Geräte -> Gasterweiterung einlegen

## In the console do a
sudo mkdir -p /mnt/platte
sudo mount /dev/cdrom /mnt/platte
cd /mnt/platte
sudo apt-get install -y build-essential linux-headers-`uname -r`
sudo ./VBoxLinuxAdditions.run
```

```
sudo reboot
```

Configure

```
Geräte -> Gemeinsame Ordner
Hinzufügen (blaues Ordnersymbol mit + ) ->
Ordner-Pfad: C:\Linux (Ordner muss auf Windows angelegt sein)
Ordner-Name: linux
checkbox nicht ausgewählt bei : automatisch einbinden, nur lesbar
checkbox ausgewählt bei: Permanent erzeugen

## aus meiner Sicht nicht notwendig zu rebooten
### Dann rebooten

In der virtuellen Maschine:
sudo su -
mkdir /linux
## linux ist der vergebene Ordnername
mount -t vboxsf linux /linux

## Optional, falls du nicht zugreifen kannst:
sudo usermod -aG vboxsf <your-user>
```

persistent setzen (beim booten mounten)

```
echo "linux    /linux    vboxsf    defaults    0    0" >> /etc/fstab
reboot
```

Reference:

- https://gist.github.com/estorgio/1d679f962e8209f8a9232f7593683265

Kubernetes - Überblick

Warum Kubernetes, was macht Kubernetes

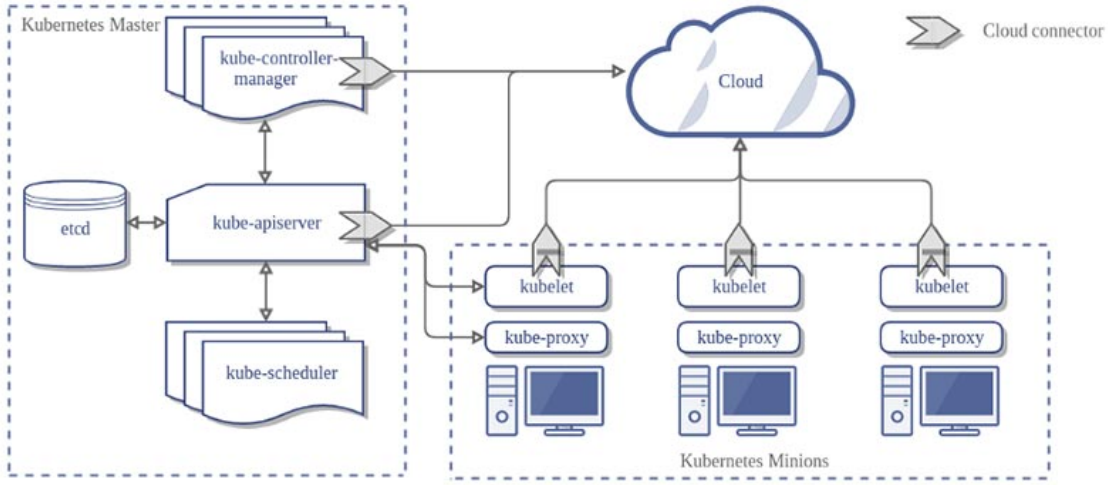
- Virtualisierung von Hardware - 5fache bessere Auslastung
- Google als Ausgangspunkt
- Software 2014 als OpenSource zur Verfügung gestellt
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbstheilend

Wozu dient Kubernetes

- Orchestrierung von Containern
- am gebräuchlisten aktuell Docker

Aufbau Allgemein

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

- Der Master koordiniert den Cluster
- Der Master koordiniert alle Aktivitäten in Ihrem Cluster
  - Planen von Anwendungen
  - Verwalten des gewünschten Status der Anwendungen
  - Skalieren von Anwendungen
  - Rollout neuer Updates.

Komponenten des Masters

ETCD

- Verwalten der Konfiguration des Clusters (key/value - pairs)

KUBE-CONTROLLER-MANAGER

- Zuständig für die Überwachung der Stati im Cluster mit Hilfe von endlos loops.
- kommuniziert mit dem Cluster über die kubernetes-api (bereitgestellt vom kube-api-server)

KUBE-API-SERVER

- provides api-frontend for administration (no gui)
- Exposes an HTTP API (users, parts of the cluster and external components communicate with it)
- REST API

KUBE-SCHEDULER

- assigns Pods to Nodes.
- scheduler determines which Nodes are valid placements for each Pod in the scheduling queue (according to constraints and available resources)
- The scheduler then ranks each valid Node and binds the Pod to a suitable Node.
- Reference implementation (other schedulers can be used)

Nodes

- Nodes (Knoten) sind die Arbeiter (Maschinen), die Anwendungen ausführen

- Ref: https://kubernetes.io/de/docs/concepts/architecture/nodes/

Pod/Pods

- Pods sind die kleinsten einsetzbaren Einheiten, die in Kubernetes erstellt und verwaltet werden können.
- Ein Pod (übersetzt Gruppe) ist eine Gruppe von einem oder mehreren Containern
  - gemeinsam genutzter Speicher- und Netzwerkressourcen
  - Befinden sich immer auf dem gleich virtuellen Server

Control Plane Node (former: master) - components

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

```
Node Agent that runs on every node (worker)
Er stellt sicher, dass Container in einem Pod ausgeführt werden.
```

Kube-proxy

- Läuft auf jedem Node
- = Netzwerk-Proxy für die Kubernetes-Netzwerk-Services.
- Kube-proxy verwaltet die Netzwerkkommunikation innerhalb oder außerhalb Ihres Clusters.

Referenzen

- https://www.redhat.com/de/topics/containers/kubernetes-architecture

Linux und Docker Tipps & Tricks allgemein

Proxy für Docker setzen

Für docker über repo installiert

- https://www.baeldung.com/ops/docker-setting-proxy

Walktrough (snap -> docker)

```
## as root
systemctl list-units -t service | grep docker
systemctl cat snap.docker.dockerd.service
systemctl edit snap.docker.dockerd.service
## in edit folgendes reinschreiben
[Service]
Environment="HTTP_PROXY=http://user01:password@10.10.10.10:8080/"
Environment="HTTPS_PROXY=https://user01:password@10.10.10.10:8080/"
Environment="NO_PROXY= hostname.example.com,172.10.10.10"

systemctl show snap.docker.dockerd.service --property Environment
systemctl restart snap.docker.dockerd.service
systemctl cat snap.docker.dockerd.service
cd /etc/systemd/system/snap.docker.dockerd.service.d/
```

```
ls -la
cat override.conf
```

Ref

- https://www.thegeekdiary.com/how-to-configure-docker-to-use-proxy/

vim einrückung für yaml-dateien

Ubuntu (im Unterverzeichnis /etc/vim - systemweit)

```
hi CursorColumn cterm=NONE ctermbg=lightred ctermfg=white
autocmd FileType y?ml setlocal ts=2 sts=2 sw=2 ai number expandtab cursorline
cursorcolumn
```

Testen

```
vim test.yml
Eigenschaft: <return> # springt eingerückt in die nächste Zeile um 2 spaces eingerückt

## evtl funktioniert vi test.yml auf manchen Systemen nicht, weil kein vim (vi
improved)
```

YAML Linter Online

- http://www.yamllint.com/

Basis/Parent - Image erstellen

Auf Basis von debootstrap

```
## Auf einem Debian oder Ubuntu - System
## folgende Schritte ausführen
## z.B. virtualbox -> Ubuntu 20.04.

### alles mit root durchführen
apt install debootstrap
cd
debootstrap focal focal > /dev/null
tar -C focal -c . | docker import - focal

## er gibt eine checksumme des images
## so kann ich das sehen
## müsste focal:latest heissen
docker images

## teilchen starten
docker run --name my_focal2 -dit focal:latest bash

## Dann kann ich danach reinwechseln
docker exec -it my_focal2 bash
```

Virtuelle Maschine Windows/OSX mit Vagrant erstellen

```
## Installieren.
https://vagrantup.com
## ins terminal
cd
cd Documents
mkdir ubuntu_20_04_test
cd ubuntu_20_04_test
vagrant init ubuntu/focal64
vagrant up
## Wenn die Maschine oben ist, kann direkt reinwechseln
vagrant ssh
## in der Maschine kein pass notwendig zum Wechseln
sudo su -

## wenn ich raus will
exit
exit

## Danach kann ich die maschine wieder zerstören
vagrant destroy -f
```

Ref:

- https://docs.docker.com/develop/develop-images/baseimages/

Eigenes unsichere Registry-Verwenden. ohne https

Setup insecure registry (snap)

```

systemctl restart
```

Spiegel - Server (mirror -> registry-mirror)

```
https://docs.docker.com/registry/recipes/mirror/
```

Ref:

- https://docs.docker.com/registry/insecure/