Kubernetes und Docker Administration und Orchestrierung

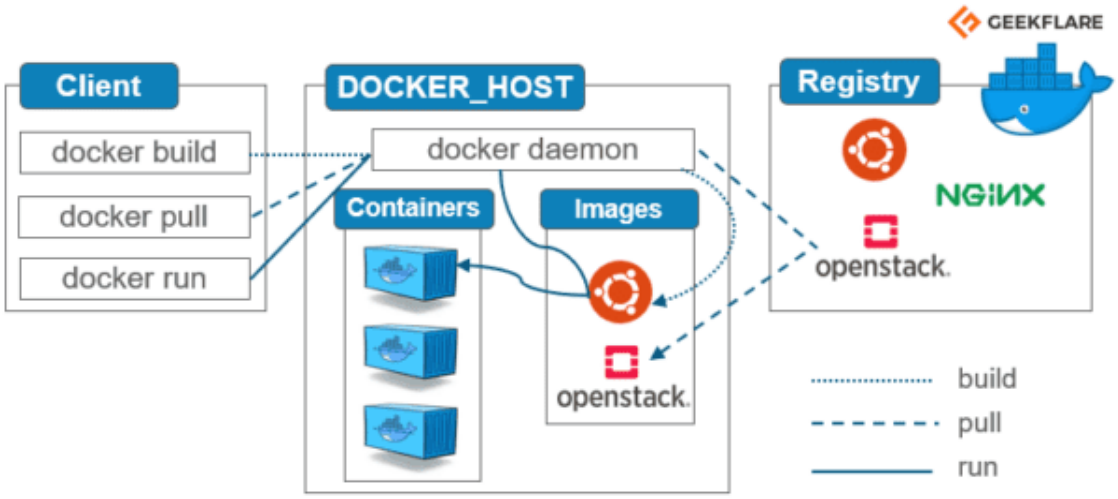
Agenda

1. Grundlagen
    - Übersicht Architektur
    - Was ist ein Container ?
    - Was sind container images
    - Container vs. Virtuelle Maschine
    - Was ist ein Dockerfile
2. Installation
    - Installation Docker unter Ubuntu mit snap
3. Docker-Befehle
    - Die wichtigsten Befehle
    - Logs anschauen - docker logs - mit Beispiel nginx
    - docker run
    - Docker container/image stoppen/löschen
    - Docker containerliste anzeigen
    - Docker container analysieren
    - Nginx mit portfreigabe laufen lassen
4. Dockerfile - Examples
    - Ubuntu mit hello world
    - Ubuntu mit ping
    - Nginx mit content aus html-ordner
    - ssh server
5. Docker-Container Examples
    - 2 Container mit Netzwerk anpingen
    - Container mit eigenem privatem Netz erstellen
6. Daten persistent machen / Shared Volumes
    - Überblick
    - Volumes
7. Netzwerk
    - Netzwerk
8. 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

9. Docker Swarm
    - Docker Swarm Beispiele
10. Tipps & Tricks allgemein / docker
    - Auf ubuntu root-benutzer werden
    - IP - Adresse abfragen
    - Hostname setzen
    - Proxy für Docker setzen
    - YAML Linter Online
    - Läuft der ssh-server
    - Basis/Parent - Image erstellen
    - Eigenes unsichere Registry-Verwenden. ohne https
11. Kubernetes - Überblick
    - Warum Kubernetes, was macht Kubernetes
    - Aufbau
    - Welches System ? (minikube, micro8ks etc.)
12. Kubernetes - API - Objekte
    - Welche API-Objekte gibt es? (Kommando)
    - Was sind Deployments
13. microk8s
    - Installation Ubuntu - snap
    - Patch to next major release - cluster
    - Remote-Verbindung zu Kubernetes (microk8s) einrichten
    - Create a cluster with microk8s
    - Ingress controller in microk8s aktivieren
    - Arbeiten mit der Registry
14. kubectl
    - Start pod (container with run && examples)
    - kubectl Spickzettle
15. kubectl - manifest - examples
    - 02 Pod nginx mit Port und IP innerhalb des Clusters
    - 03b Example with service and nginx
    - 04 Ingress mit einfachem Beispiel
16. Documentation
    - Vulnerability Scanner with docker
    - Vulnerability Scanner mit snyk
    - Parent/Base - Image bauen für Docker
    - Documentation zu microk8s plugins/addons

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 Image 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
- 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

Installation

Installation Docker unter Ubuntu mit snap

```
snap install docker

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

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

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
```

Docker-Befehle

Die wichtigsten Befehle

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

## 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

## docker hub durchsuchen
```

```
docker search hello-world
```

Logs anschauen - docker logs - mit Beispiel nginx

```
## 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
```

docker run

Beispiel (binden an ein terminal), detached

```
## before that we did
docker pull ubuntu:xenial
docker run -t -d --name my_xenial ubuntu:xenial
## will wollen überprüfen, ob der container läuft
docker container ls
## image vorhanden
docker images

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

Docker container/image stoppen/löschen

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

docker rm ubuntu-container

## ooder alternative
docker rm -f ubuntu-container


## image löschen
docker rmi ubuntu:xenial
```

Docker containerliste anzeigen

```
## besser
docker container ls
docker container ls -a


## deprecated
docker ps
```

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

Docker container analysieren

```
docker inspect hello-web # hello-web = container name
```

Nginx mit portfreigabe laufen lassen

```
docker run --name test-nginx -d -p 8080:80 nginx

docker container ls
lsof -i
cat /etc/services | grep 8080
curl http://localhost:8080
docker container ls
## wenn der container gestoppt wird, keine ausgabe mehr, weil kein webserver
docker stop test-nginx
curl http://localhost:8080
```

Dockerfile - Examples

Ubuntu mit hello world

```
### Schritt 1:
cd
mkdir 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
echo hello-docker

### Schritt 4:
## docker build -t dockertrainereu/<dein-name>-hello-docker .
## Beispiel
docker build -t dockertrainereu/jm-hello-docker .
docker run -i -t dockertrainereu/<dein-name>-hello-docker
```

```
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

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

Nginx mit content aus html-ordner

```
## das gleich wie cd ~
## Heimatverzeichnis des Benutzers root
cd
mkdir nginx-test
cd nginx-test
mkdir html
cd html/
## vi index.html
Text, den du rein haben möchtest

cd ..
vi Dockerfile

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

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

```
docker images
## eventually you are not logged in
docker login
docker push dockertrainereu/jm1-hello-web
##aus spass geloescht
docker rmi dockertrainereu/jm1-hello-web
## und direkt aus der Registry wieder runterladen
docker run --name hello-web -p 8888:80 -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
```

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
```

Daten persistent machen / Shared Volumes

Überblick

Overview

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

Disadvantags

```
stored only on one node
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

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

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

Storage volume löschen

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

Netzwerk

Netzwerk

Übersicht

```
3 Typen

o none
o bridge (Standard-Netzwerk)
o host

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

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
```

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

### löschen erst möglich, wenn es keine Endpoints
### d.h. container die das Netzwerk verwenden
```

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

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

```
## docker-compose.yaml
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:
```

Example with Wordpress / Nginx / MariadB

```
mkdir wordpress-mit-docker-compose
cd wordpress-mit-docker-compose
## 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 wordpress_compose_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

```
cd
mkdir bautest
cd bautest
```

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

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

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

```
cd ../
## wichtig, im docker-compose - Ordner seiend
```

```
##pwd
##~/bautest
docker-compose up -d
## wird image gebaut und container gestartet
```

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 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/

Tipps & Tricks allgemein / docker

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 -
```

Proxy für Docker setzen

Walktrough

```
## 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/

YAML Linter Online

- http://www.yamllint.com/

Läuft der ssh-server

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

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/

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 Container
- am gebräuchlisten aktuell Docker

Aufbau

Welches System ? (minikube, micro8ks etc.)

Überblick der Systeme

General

```
kubernetes itself has not convenient way of doing specific stuff like
creating the kubernetes cluster.

So there are other tools/distri around helping you with that.
```

Kubeadm

General

- The official CNCF (https://www.cncf.io/) tool for provisioning Kubernetes clusters (variety of shapes and forms (e.g. single-node, multi-node, HA, self-hosted))
- Most manual way to create and manage a cluster

Disadvantages

- Plugins sind oftmals etwas schwierige zu aktivieren

microk8s

General

- Created by Canonical (Ubuntu)
- Runs on Linux
- Runs only as snap

- In the meantime it is also available for Windows/Mac
- HA-Cluster

Production-Ready ?

- Short answer: YES

```
Quote canonical (2020):

MicroK8s is a powerful, lightweight, reliable production-ready Kubernetes
distribution. It is an enterprise-grade Kubernetes distribution that has a small disk
and memory footprint while offering carefully selected add-ons out-the-box, such as
Istio, Knative, Grafana, Cilium and more. Whether you are running a production
environment or interested in exploring K8s, MicroK8s serves your needs.

Ref: https://ubuntu.com/blog/introduction-to-microk8s-part-1-2
```

Advantages

- Easy to setup HA-Cluster (multi-node control plane)
- Easy to manage

minikube

Disadvantages

- Not usable / intended for production

Advantages

- Easy to set up on local systems for testing/development (Laptop, PC)
- Multi-Node cluster is possible
- Runs und Linux/Windows/Mac
- Supports plugin (Different name ?)

k3s

kind (Kubernetes-In-Docker)

General

- Runs in docker container

For Production ?

```
Having a footprint, where kubernetes runs within docker
and the applikations run within docker as docker containers
it is not suitable for production.
```

Kubernetes - API - Objekte

Welche API-Objekte gibt es? (Kommando)

```
kubectl api-resources
```

Was sind Deployments

Hierarchy

```
deployment
  replicaset
    pod


Deployment :: create a new replicaset, when needed (e.g. new version of image comes
out)
Replicaset :: manage the state - take care, that the are always x-pods running (e.g.
3)
Pod :: create the containers
```

What are deployments

- Help to manage updates of pods / replicaset (rolling update)

Example

```
## Deploy a sample from k8s.io
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
```

Refs:

- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

microk8s

Installation Ubuntu - snap

```
sudo snap install microk8s --classic
## Important enable dns // otherwice not dns lookup is possible
microk8s enable dns
microk8s status

## Execute kubectl commands like so
microk8s kubectl
microk8s kubectl cluster-info

## Make it easier with an alias
echo "alias kubectl='microk8s kubectl'" >> ~/.bashrc
source ~/.bashrc
kubectl
```

Patch to next major release - cluster

Remote-Verbindung zu Kubernetes (microk8s) einrichten

```
## on CLIENT install kubectl
sudo snap install kubectl --classic

## On MASTER -server get config
## als root
cd
microk8s config > /home/kurs/remote_config
```

```
## Download (scp config file) and store in .kube - folder
cd ~
mkdir .kube
cd .kube  # Wichtig: config muss nachher im verzeichnis .kube liegen
## scp kurs@master_server:/path/to/remote_config config
## z.B.
scp kurs@192.168.56.102:/home/kurs/remote_config config
## oder benutzer 11trainingdo
scp 11trainingdo@192.168.56.102:/home/11trainingdo/remote_config config

##### Evtl. IP-Adresse in config zum Server aendern

## Ultimative 1. Test auf CLIENT
kubectl cluster-info

## or if using kubectl or alias
kubectl get pods

## if you want to use a different kube config file, you can do like so
kubectl --kubeconfig /home/myuser/.kube/myconfig
```

Create a cluster with microk8s

Walkthrough

```
## auf master (jeweils für jedes node neu ausführen)
microk8s add-node

## dann auf jeweiligem node vorigen Befehl der ausgegeben wurde ausführen
## Kann mehr als 60 sekunden dauern ! Geduld...Geduld..Geduld
##z.B. -> ACHTUNG evtl. IP ändern
microk8s join 10.128.63.86:25000/567a21bdfc9a64738ef4b3286b2b8a69
```

Auf einem Node addon aktivieren z.B. ingress

```
gucken, ob es auf dem anderen node auch aktiv ist.
```

Ref:

- https://microk8s.io/docs/high-availability

Ingress controller in microk8s aktivieren

Aktivieren

```
microk8s enable ingress
```

Referenz

- https://microk8s.io/docs/addon-ingress

Arbeiten mit der Registry

kubectl

Start pod (container with run && examples)

Example (that does work)

```
## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx

kubectl get pods
## on which node does it run ?
kubectl get pods -o wide
```

Example (that does not work)

```
kubectl run foo2 --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
## Weitere status - info
kubectl describe pods foo2


### Ref:

  * https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#run

### kubectl Spickzettle


### Allgemein
```

Zeige Information über das Cluster

kubectl cluster-info

Welche api-resources gibt es ?

kubectl api-resources

Hilfe zu object und eigenschaften bekommen

kubectl describe pod kubectl describe pod.metadata kubectl describe pod.metadata.name

```
### Ausgabeformate
```

Ausgabe kann in verschiedenen Formaten erfolgen

kubectl get pods -o wide # weitere informationen

im json format

kubectl get pods -o json

gilt natürluch auch für andere kommandos

kubectl get deploy -o json

```
### Zu den Pods
```

Start einen pod // BESSER: direkt manifest verwenden

kubectl run podname image=imagename

kubectl run nginx image=nginx

Pods anzeigen

kubectl get pods kubectl get pod

Format weitere Information

kubectl get pod -o wide

Zeige labels der Pods

kubectl get pods --show-labels

Zeige pods mit einem bestimmten label

kubectl get pods -l app=nginx

Status eines Pods anzeigen

kubectl explain pod nginx

Pod löschen

kubectl delete pod nginx

Kommando in pod ausführen

kubectl exec -it nginx -- bash

```
### Arbeiten mit namespaces
```

Welche namespaces auf dem System

kubectl get ns kubectl get namespaces

Standardmäßig wird immer der default namespace verwendet

wenn man kommandos aufruft

kubectl get deployments

Möchte ich z.B. deployment vom kube-system (installation) aufrufen,

kann ich den namespace angeben

kubectl get deployments --namespace=kube-system kubectl get deployments -n kube-system

```
### Referenz

  * https://kubernetes.io/de/docs/reference/kubectl/cheatsheet/

## kubectl - manifest - examples

### 02 Pod nginx mit Port und IP innerhalb des Clusters


### What is containerPort (from kubectl explain) ?
```

containerPort -required- Number of port to expose on the pod's IP address. This must be a valid port number, 0 < x < 65536.

```
### Walkthrough
```

vi nginx-static-expose.yml

apiVersion: v1 kind: Pod metadata: name: nginx-static-web labels: webserver: nginx spec: containers:

- name: web image: nginx ports:
  - name: web containerPort: 80 protocol: TCP

kubectl apply -f nginx-static-expose.yml kubectl describe nginx-static-web

show config

kubectl get pod/nginx-static-web -o yml

```
### 03b Example with service and nginx
```

apiVersion: apps/v1 kind: Deployment metadata: name: my-nginx spec: selector: matchLabels: run: my-nginx replicas: 2 template: metadata: labels: run: my-nginx spec: containers: - name: my-nginx image: nginx ports: - containerPort: 80

---

apiVersion: v1 kind: Service metadata: name: my-nginx labels: run: my-nginx spec: ports:

- port: 80 protocol: TCP selector: run: my-nginx

```
### Ref.

  * https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

### 04 Ingress mit einfachem Beispiel


### Walkthrough
```

mkdir apple-banana-ingress

apple.yml

vi apple.yml

kind: Pod apiVersion: v1 metadata: name: apple-app labels: app: apple spec: containers: - name: apple-app image: hashicorp/http-echo args: - "-text=apple"

---

kind: Service apiVersion: v1 metadata: name: apple-service spec: selector: app: apple ports: - protocol: TCP port: 80 targetPort: 5678 # Default port for image

kubectl apply -f apple.yml

banana

vi banana.yml

kind: Pod apiVersion: v1 metadata: name: banana-app labels: app: banana spec: containers: - name: banana-app image: hashicorp/http-echo args: - "-text=banana"

---

kind: Service apiVersion: v1 metadata: name: banana-service spec: selector: app: banana ports: - port: 80 targetPort: 5678 # Default port for image

kubectl apply -f banana.yml

Ingress

apiVersion: extensions/v1beta1 kind: Ingress metadata: name: example-ingress annotations: ingress.kubernetes.io/rewrite-target: / spec: rules:

- http: paths:

```
- path: /apple
  backend:
    serviceName: apple-service
    servicePort: 80
- path: /banana
  backend:
    serviceName: banana-service
    servicePort: 80
```

```
## ingress
kubectl apply -f ingress.yml
kubectl get ing
```

Reference

- https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html

Find the problem

```
## Hints

## 1. Which resources does our version of kubectl support
## Can we find Ingress as "Kind" here.
kubectl api-ressources

## 2. Let's see, how the configuration works
kubectl explain --api-version=networking.k8s.io/v1
ingress.spec.rules.http.paths.backend.service

## now we can adjust our config
```

Solution

```
## in kubernetes 1.22.2 - ingress.yml needs to be modified like so.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
        - path: /apple
```

```
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

Documentation

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/

Documentation zu microk8s plugins/addons

- https://microk8s.io/docs/addons