

# Kubernetes und Docker Administration und Orchestrierung

## 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](#)

### 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](#)
- [Docker container in den Vordergrund bringen - attach](#)
- [Aufräumen - container und images löschen](#)
- [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. Docker-Daten persistent machen / Shared Volumes

- [Überblick](#)
- [Volumes](#)

### 7. Docker-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](#)
- [docker-compose und replicas](#)

#### 9. Docker Swarm

- [Docker Swarm Beispiele](#)

#### 10. Docker - Dokumentation

- [Vulnerability Scanner with docker](#)
- [Vulnerability Scanner mit snyk](#)
- [Parent/Base - Image bauen für Docker](#)

#### 11. Kubernetes - Überblick

- [Warum Kubernetes, was macht Kubernetes](#)
- [Aufbau](#)
- [Welches System ? \(minikube, micro8s etc.\)](#)
- [Installation - Welche Komponenten from scratch](#)

#### 12. Kubernetes - microk8s (Installation und Management)

- [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](#)
- [Installation Kubernetes Dashboard](#)

#### 13. Kubernetes - API - Objekte

- [Welche API-Objekte gibt es? \(Kommando\)](#)
- [Api Versionierung Lifetime](#)
- [Was sind Deployments](#)
- [Service - Objekt und IP](#)
- [Ingress -> Nginx Proxy](#)

#### 14. Kubernetes - RBAC

- [Nutzer einrichten](#)

#### 15. Kubernetes - Netzwerk (CNI's)

- [Übersicht Netzwerke](#)
- [Calico - nginx example](#)
- [Calico - client-backend-ui-example](#)

#### 16. kubectl

- [Start pod \(container with run && examples\)](#)
- [Bash completion for kubectl](#)
- [kubectl Spickzettel](#)
- [Tipps&Tricks zu Deployment - Rollout](#)

#### 17. kubectl - manifest - examples

- [02 Pod nginx mit Port und IP innerhalb des Clusters](#)
- [03b Example with service and nginx](#)
- [04 Ingress mit einfachem Beispiel](#)

- [05 Ingress mit Permanent Redirect](#)

#### 18. Kubernetes - Monitoring (microk8s und vanilla)

- [metrics-server aktivieren \(microk8s und vanilla\)](#)

#### 19. Kubernetes - Shared Volumes

- [Shared Volumes with nfs](#)

#### 20. Kubernetes - Backups

- [Kubernetes Aware Cloud Backup - kasten.io](#)

#### 21. Kubernetes - Wartung

- [kubectl drain/uncordon](#)
- [Alte manifeste konvertieren mit convert plugin](#)

#### 22. Kubernetes - Tipps & Tricks

- [Assigning Pods to Nodes](#)

#### 23. Kubernetes - Documentation

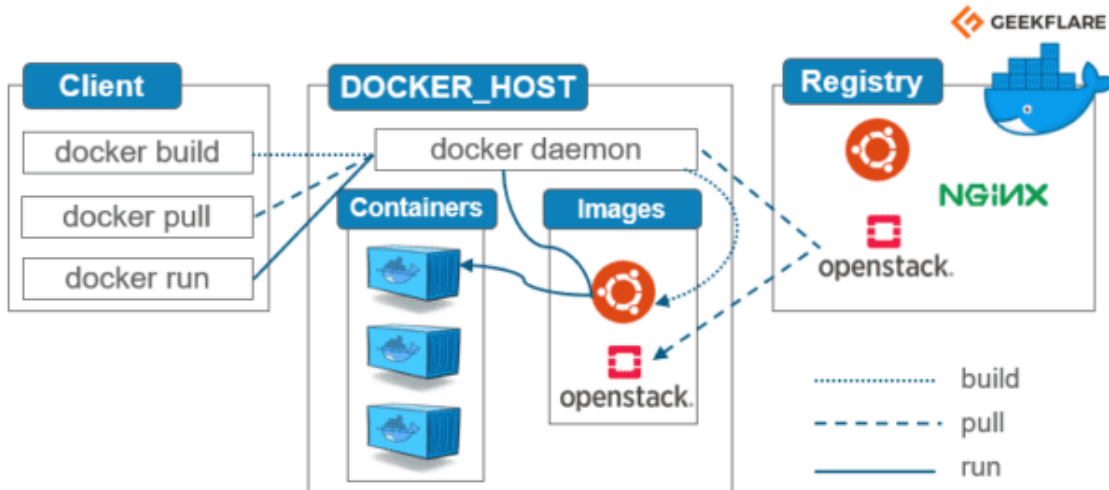
- [Documentation zu microk8s plugins/addons](#)
- [LDAP-Anbindung](#)
- [Shared Volumes - Welche gibt es ?](#)

#### 24. Linux und Docker Tipps & Tricks allgemein

- [Auf ubuntu root-benutzer werden](#)
- [IP - Adresse abfragen](#)
- [Hostname setzen](#)
- [Proxy für Docker setzen](#)
- [vim einrückung für yaml-dateien](#)
- [YAML Linter Online](#)
- [Läuft der ssh-server](#)
- [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 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.
  - o 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

```
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 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
```

## 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
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

## 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 container analysieren

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

```
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 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
## -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
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

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

## 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/
## vi index.html
Text, den du rein haben möchtest

cd ..
vi Dockerfile

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

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

```

## Schritt 2: Push build

```

## eventually you are not logged in
docker login
docker push dockertrainereu/jml-hello-web
##aus spass geloescht
docker rmi dockertrainereu/jml-hello-web

```

## Schritt 3: dokcer laufen lassen

```

## und direkt aus der Registry wieder runterladen
docker run --name hello-web -p 8080:80 -d dockertrainereu/jml-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, wie das Netz jetzt aussieht
docker network inspect demo_net
```

## Docker-Daten persistent machen / Shared Volumes

### Überblick

#### Overview

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

#### Disadvantages

```
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
```

## Docker-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

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

## 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.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:
```

## 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 bauteast
cd bauteast
```

```
## 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
##~/bauteast
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/>

## 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 - 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/>

## 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 gebräuchlichsten aktuell Docker

## Aufbau

### 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-frontent 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>

#### 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 schwierig 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.

## Installation - Welche Komponenten from scratch

### Step 1: Server 1 (manuell installiert -> microk8s)

```
## Installation Ubuntu - Server

## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers lltrainingdo per
ssh)

## Server 1 - manuell
## Ubuntu 20.04 LTS - Grundinstallation

## minimal Netzwerk - öffentlichen IP
## nichts besonderes eingerichtet - Standard Digitalocean

## Standard vo Installation microk8s
lo                UNKNOWN          127.0.0.1/8 ::1/128
## public ip / interne
eth0              UP                164.92.255.234/20 10.19.0.6/16
fe80::c:66ff:fec4:cbce/64
## private ip
eth1              UP                10.135.0.3/16 fe80::8081:aaff:feaa:780/64

snap install microk8s --classic
## namensauflösung fuer pods
microk8s enable dns

## Funktioniert microk8s
microk8s status
```

### Steps 2: Server 2+3 (automatische Installation -> microk8s )

```
## Was macht das ?
## 1. Basisnutzer (lltrainingdo) - keine Voraussetzung für microk8s
## 2. Installation von microk8s
## .>>>>>> microk8s installiert <<<<<<<
## - snap install --classic microk8s
## >>>>>> Zuordnung zur Gruppe microk8s - notwendig für bestimmte plugins (z.B. helm)
## usermod -a -G microk8s root
## >>>>>> Setzen des .kube - Verzeichnisses auf den Nutzer microk8s -> nicht zwingend
erforderlich
## chown -R -R microk8s ~/.kube
## >>>>>> REQUIRED .. DNS aktivieren, wichtig für Namensauflösungen innerhalb der
PODS
## >>>>>> sonst funktioniert das nicht !!!
## microk8s enable dns
```

```
## >>>>>> kubectl alias gesetzt, damit man nicht immer microk8s kubectl eingeben muss
## - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

## cloud-init script
## s.u. MITMICROK8S (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo
per ssh)
##cloud-config
users:
  - name: 11trainingdo
    shell: /bin/bash

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g"
/etc/ssh/sshd_config
  - echo " " >> /etc/ssh/sshd_config
  - echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
  - echo "AllowUsers root" >> /etc/ssh/sshd_config
  - systemctl reload sshd
  - sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hg13d6ATbr2kEu9zMOfwLxkYMO.AJF526mZONwc
/etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo

  - echo "Installing microk8s"
  - snap install --classic microk8s
  - usermod -a -G microk8s root
  - chown -f -R microk8s ~/.kube
  - microk8s enable dns
  - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

## Prüfen ob microk8s - wird automatisch nach Installation gestartet
## kann eine Weile dauern
microk8s status
```

### Step 3: Client - Maschine (wir sollten nicht auf control-plane oder cluster - node arbeiten)

```
Weiteren Server hochgezogen.
Vanilla + BASIS

## Installation Ubuntu - Server

## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per
ssh)

## Server 1 - manuell
## Ubuntu 20.04 LTS - Grundinstallation

## minimal Netzwerk - öffentlichen IP
## nichts besonderes eingerichtet - Standard Digitalocean
```



```
## Standard vo Installation microk8s
lo                UNKNOWN          127.0.0.1/8 ::1/128
## public ip / interne
eth0              UP                164.92.255.232/20 10.19.0.6/16
fe80::c:66ff:fec4:cbce/64
## private ip
eth1              UP                10.135.0.5/16 fe80::8081:aaff:feaa:780/64
```

```
##### Installation von kubectl aus dem snap
## NICHT .. keine microk8s - keine control-plane / worker-node
## NUR Client zum Arbeiten
snap install kubectl --classic

##### .kube/config
## Damit ein Zugriff auf die kube-server-api möglich
## d.h. REST-API Interface, um das Cluster verwalten.
## Hier haben uns für den ersten Control-Node entschieden
## Alternativ wäre round-robin per dns möglich

## Mini-Schritt 1:
## Auf dem Server 1: kubeconfig ausspielen
microk8s config > /root/kube-config
## auf das Zielsystem gebracht (client 1)
scp /root/kubeconfig 11trainingdo@10.135.0.5:/home/11trainingdo

## Mini-Schritt 2:
## Auf dem Client 1 (diese Maschine) kubeconfig an die richtige Stelle bringen
## Standardmäßig der Client nach eine Konfigurationsdatei sucht in ~/.kube/config
sudo su -
cd
mkdir .kube
cd .kube
mv /home/11trainingdo/kube-config config

## Verbindungstest gemacht
## Damit feststellen ob das funktioniert.
kubectl cluster-info
```

#### Schritt 4: Auf allen Servern IP's hinterlegen und richtigen Hostnamen überprüfen

```
## Auf jedem Server
hostnamectl
## evtl. hostname setzen
## z.B. - auf jedem Server eindeutig
hostnamectl set-hostname n1.training.local

## Gleiche hosts auf allen server einrichten.
## Wichtig, um Traffic zu minimieren verwenden, die interne (private) IP

/etc/hosts
10.135.0.3 n1.training.local n1
```

```
10.135.0.4 n2.training.local n2
10.135.0.5 n3.training.local n3
```

## Schritt 5: Cluster aufbauen

```
## Mini-Schritt 1:
## Server 1: connection - string (token)
microk8s add-node
## Zeigt Liste und wir nehmen den Eintrag mit der lokalen / öffentlichen ip
## Dieser Token kann nur 1x verwendet werden und wir auf dem ANDEREN node ausgeführt
## microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 2:
## Dauert eine Weile, bis das durch ist.
## Server 2: Den Node hinzufügen durch den JOIN - Befehl
microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 3:
## Server 1: token besorgen für node 3
microk8s add-node

## Mini-Schritt 4:
## Server 3: Den Node hinzufügen durch den JOIN-Befehl
microk8s join 10.135.0.3:25000/09c96e57ec12af45b2752fb45450530c/bcad1949221a

## Mini-Schritt 5: Überprüfen ob HA-Cluster läuft
Server 1: (es kann auf jedem der 3 Server überprüft werden, auf einem reicht
microk8s status | grep high-availability
high-availability: yes
```

## Ergänzend nicht notwendige Skripte

```
## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers lltrainingdo per
ssh)

## Digitalocean - unter user_data reingepastet beim Einrichten

##cloud-config
users:
  - name: lltrainingdo
    shell: /bin/bash

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g"
/etc/ssh/sshd_config
  - echo " " >> /etc/ssh/sshd_config
  - echo "AllowUsers lltrainingdo" >> /etc/ssh/sshd_config
  - echo "AllowUsers root" >> /etc/ssh/sshd_config
  - systemctl reload sshd
  - sed -i '/lltrainingdo/c
```

```
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hg13d6ATbr2kEu9zMOfwLxkYMO.AJF526mZONwc
/etc/shadow
- echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
- chmod 0440 /etc/sudoers.d/11trainingdo
```

## Kubernetes - microk8s (Installation und Management)

### Installation Ubuntu - snap

#### Walkthrough

```
sudo snap install microk8s --classic
## Important enable dns // otherwise 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
```

#### Working with snaps

```
snap info microk8s
```

#### Ref:

- <https://microk8s.io/docs/setting-snap-channel>

#### 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 1ltrainingdo
scp 1ltrainingdo@192.168.56.102:/home/1ltrainingdo/remote_config config

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

## Ultimate 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

### Installation Kuberernetes Dashboard

#### Reference:

- <https://blog.tippybits.com/installing-kubernetes-in-virtualbox-3d49f666b4d6>

## Kubernetes - API - Objekte

## Welche API-Objekte gibt es? (Kommando)

```
kubectl api-resources
```

## Api Versionierung Lifetime

### Wie ist die deprecation policy ?

- <https://kubernetes.io/docs/reference/using-api/deprecation-policy/>

### Was ist wann deprecated ?

- <https://kubernetes.io/docs/reference/using-api/deprecation-guide/>

### Reference:

- <https://kubernetes.io/docs/reference/using-api/>

## 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 there 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/>

## Service - Objekt und IP

### Was ?

Stellt eine Netzwerkverbindung zu verschiedenen Pods her,  
auf Basis eines Labels

### Warum ?

```
service (-controller) überprüft welche Nodes mit entsprechenden
Label zur Verfügung stehen und übernimmt das Routing
```

```
standardmäßig: round robin
```

## What are services ?

- Services help you to connect to the pods seamlessly
- Service knows which pods are available

## service - types

```
The type defines how the connection is done (what kind of network/ip/port is provided
to connect to the service
```

```
ClusterIP
```

```
NodePort
```

```
LoadBalancer - an external balancer is used (that is mainly the case in
```

## Reference:

- <https://kubernetes.io/docs/concepts/services-networking/service/>

## Ingress -> Nginx Proxy

## Ref. / Dokumentation

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

## Kubernetes - RBAC

### Nutzer einrichten

### Enable RBAC in microk8s

```
microk8s enable rbac
```

## Schritt 1: Systemaccount anlegen und in kubeconfig hinterlegen

```
kubectl create -n kube-system serviceaccount training
## extract name of the token from here
kubectl get -n kube-system serviceaccount training -o yaml

## Secret auslesen und base64 dekodieren
## $() geht nur in der bash
TOKEN=$(kubectl get -n kube-system secret training-token-nxdfl -o
jsonpath='{.data.token}' | base64 --decode)
echo $TOKEN
kubectl config set-credentials training --token=$TOKEN
## trainingc vorher in .kube/config eingetragen
kubectl config use-context trainingc

## Hier reichen die Rechte nicht aus
```

```
kubectl get pods
## Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:kube-
system:training" cannot list # resource "pods" in API group "" in the namespace
"default"
```

## Schritt 2: Role und Rolebinding festlegen

```
cd
mkdir -p manifests/rbac
### Minischritt 1:

## namespace.yml:
apiVersion: v1
kind: Namespace
metadata:
  name: training
  labels:
    name: training

kubectl apply -f namespace.yaml

### Minischritt 2:
## vi role.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: training
  name: role-training-pods
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

kubectl apply -f role.yaml

### Minischritt 3:
### Verknüpfung Role zu Benutzer (=rolebinding)

kubectl create rolebinding rolebinding-training-pods --role role-training-pods --user
training

### Minischritt 4:
kubectl config use-context trainingc
kubectl get pods
```

### Ref:

- <https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm>
- <https://microk8s.io/docs/multi-user>

## Kubernetes - Netzwerk (CNI's)

## Übersicht Netzwerke

### CNI

- Common Network Interface
- Fest Definition, wie Container mit Netzwerk-Bibliotheken kommunizieren

### Docker - Container oder andere

- Container wird hochgefahren -> über CNI -> zieht Netzwerk - IP hoch.
- Container wird runtergefahren -> über CNI -> Netzwerk - IP wird released

### Welche gibt es ?

- Flannel
- Canal
- Calico

### Flannel

#### Overlay - Netzwerk

- virtuelles Netzwerk was sich oben drüber und eigentlich auf Netzwerkebene nicht existiert
- VXLAN

#### Vorteile

- Guter einfacher Einstieg
- reduziert auf eine Binary flanneld

#### Nachteile

- keine Firewall - Policies möglich
- keine klassischen Netzwerk-Tools zum Debuggen möglich.

### Canal

#### General

- Auch ein Overlay - Netzwerk
- Unterstützt auch policies

### Calico

#### Generell

- klassische Netzwerk (BGP)

#### Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

#### Vorteile

- ISTIO integrierbar (Mesh - Netz)
- Performance etwas besser als Flannel (weil keine Encapsulation)

#### Referenz

- <https://projectcalico.docs.tigera.io/security/calico-network-policy>

### microk8s Vergleich

- <https://microk8s.io/compare>



```
snap.microk8s.daemon-flanneld
```

Flannel is a CNI which gives a subnet to each host for use with container runtimes.

Flanneld runs if ha-cluster is not enabled. If ha-cluster is enabled, calico is run instead.

The flannel daemon is started using the arguments in `${SNAP_DATA}/args/flanneld`. For more information on the configuration, see the flannel documentation.

## Calico - nginx example

```
## Schritt 1:
```

```
kubectl create ns policy-demo
```

```
kubectl create deployment --namespace=policy-demo nginx --image=nginx
```

```
kubectl expose --namespace=policy-demo deployment nginx --port=80
```

```
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
```

```
kubectl run --namespace=policy-demo access --rm -ti --image busybox /bin/sh
```

```
## innerhalb der shell
```

```
wget -q nginx -O -
```

```
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
```

```
## in diesem namespace: policy-demo
```

```
kubectl create -f - <<EOF
```

```
kind: NetworkPolicy
```

```
apiVersion: networking.k8s.io/v1
```

```
metadata:
```

```
  name: default-deny
```

```
  namespace: policy-demo
```

```
spec:
```

```
  podSelector:
```

```
    matchLabels: {}
```

```
EOF
```

```
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
```

```
kubectl run --namespace=policy-demo access --rm -ti --image busybox /bin/sh
```

```
## innerhalb der shell
```

```
wget -q nginx -O -
```

```
## Schritt 3: Zugriff erlauben von pods mit dem Label run=access
```

```
kubectl create -f - <<EOF
```

```
kind: NetworkPolicy
```

```
apiVersion: networking.k8s.io/v1
```

```
metadata:
```

```
  name: access-nginx
```

```
  namespace: policy-demo
```

```
spec:
```

```
  podSelector:
```

```
    matchLabels:
```

```
      app: nginx
```

```
  ingress:
```

```
- from:
  - podSelector:
      matchLabels:
        run: access
EOF

## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
## pod hat durch run -> access automatisch das label run:access zugewiesen
kubectl run --namespace=policy-demo access --rm -ti --image busybox /bin/sh
```

```
## innerhalb der shell
wget -q nginx -O -
```

```
kubectl run --namespace=policy-demo no-access --rm -ti --image busybox /bin/sh
```

```
## in der shell
wget -q nginx -O -
```

```
kubectl delete ns policy-demo
```

#### Ref:

- <https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic>

### Calico - client-backend-ui-example

#### Walkthrough

```
cd
mkdir -p manifests/calico/example1
cd manifests/calico/example1
```

```
### Step 1: Create containers
```

```
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/00-namespace.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/01-management-ui.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/02-backend.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/03-frontend.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/04-client.yaml
```

```
kubectl get pods --all-namespaces --watch
kubectl get ns
```

```
### Step 2: Check connections in the browser (ui)
### Use IP of one of your nodes here
http://164.92.255.234:30002/
```

```
### Step 3: Download default-deny rules
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/default-deny.yaml
### Let us have look into it
### Deny all pods
cat default-deny.yaml
### Apply this for 2 namespaces created in Step 1
kubectl -n client apply -f default-deny.yaml
kubectl -n stars apply -f default-deny.yaml
```

```
### Step 4: Refresh UI and see, that there are no connections possible
http://164.92.255.234:30002/
```

```
### Step 5:
### Allow traffic by policy
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/allow-ui.yaml
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/allow-ui-client.yaml
### Let us look into this:
cat allow-ui.yaml
cat allow-ui-client.yaml
kubectl apply -f allow-ui.yaml
kubectl apply -f allow-ui-client.yaml
```

```
### Step 6:
### Refresh management ui
### Now all traffic is allowed
http://164.92.255.234:30002/
```

```
### Step 7:
### Restrict traffic to backend
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/backend-policy.yaml
cat backend-policy.yaml
kubectl apply -f backend-policy.yaml
```

```
### Step 8:
### Refresh
## The frontend can now access the backend (on TCP port 6379 only).
## The backend cannot access the frontend at all.
## The client cannot access the frontend, nor can it access the backend
http://164.92.255.234:30002/
```

```
### Step 9:
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/frontend-policy.yaml
cat frontend-policy.yaml
kubectl apply -f frontend-policy.yaml
```

```
### Step 10:
## Refresh ui
## Client can now access Frontend
http://164.92.255.234:30002/
```

```
## Alles wieder löschen
kubectl delete ns client stars management-ui
```

## Reference

- <https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/kubernetes-demo>

## 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

### Bash completion for kubectl

### Walkthrough
```

apt install bash-completion source /usr/share/bash-completion/bash\_completion

## is it installed properly

type \_init\_completion

## activate for all users

```
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
```

## verifizieren - neue login shell

```
su -
```

## zum Testen

```
kubectl g kubectl get
```

```
### Alternative für k als alias für kubectl
```

```
source <(kubectl completion bash) complete -F __start_kubectl k
```

```
### Reference
```

```
* https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/
```

```
### kubectl Spickzettel
```

```
### Allgemein
```

## Zeige Information über das Cluster

```
kubectl cluster-info
```

## Welche api-resources gibt es ?

```
kubectl api-resources
```

## Hilfe zu object und eigenschaften bekommen

```
kubectl explain pod kubectl explain pod.metadata kubectl explain pod.metadata.name
```

```
### Arbeiten mit manifesten
```

```
kubectl apply -f nginx-replicaset.yml
```

## Wie ist aktuell die hinterlegte config im system

```
kubectl get -o yaml -f nginx-replicaset.yml
```

## Änderung in nginx-replicaset.yml z.B. replicas: 4

## dry-run - was wird geändert

```
kubectl diff -f nginx-replicaset.yml
```

## anwenden

```
kubectl apply -f nginx-replicaset.yml
```

## Alle Objekte aus manifest löschen

```
kubectl delete -f nginx-replicaset.yml
```

```
### Ausgabeformate
```

## Ausgabe kann in verschiedenen Formaten erfolgen

```
kubectl get pods -o wide # weitere Informationen
```

## im json format

```
kubectl get pods -o json
```

## gilt natürlich auch für andere kommandos

```
kubectl get deploy -o json kubectl get deploy -o yaml
```

```
### 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 describe 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/

### Tipps&Tricks zu Deployment - Rollout

### Warum
```

Rückgängig machen von deploys, Deploys neu unstossen. (Das sind die wichtigsten Fähigkeiten)

```
### Beispiele
```

## Deployment nochmal durchführen

## z.B. nach kubectl uncordon n12.training.local

`kubectl rollout restart deploy nginx-deployment`

## Rollout rückgängig machen

`kubectl rollout undo deploy nginx-deployment`

```
## 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
```

```
### Prerequisites
```

## Ingress Controller muss aktiviert sein

microk8s enable ingress

```
### 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-resources

## 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

```

## 05 Ingress mit Permanent Redirect

### Example

```

## redirect.yml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace

---

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.de
    nginx.ingress.kubernetes.io/permanent-redirect-code: "308"
  creationTimestamp: null
  name: destination-home
  namespace: my-namespace
spec:
  rules:
  - host: web.training.local
    http:
      paths:
      - backend:
          service:
            name: http-svc
            port:
              number: 80
        path: /source
        pathType: ImplementationSpecific

```

Achtung: host-eintrag auf Rechner machen, von dem aus man zugreift

```

/etc/hosts
45.23.12.12 web.training.local

```

```

curl -I http://web.training.local/source
HTTP/1.1 308
Permanent Redirect

```

## Umbauen zu google ;o)

```
This annotation allows to return a permanent redirect instead of sending data to the
upstream. For example nginx.ingress.kubernetes.io/permanent-redirect:
https://www.google.com would redirect everything to Google.
```

### Refs:

- <https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-guide/nginx-configuration/annotations.md#permanent-redirect>
- 

## Kubernetes - Monitoring (microk8s und vanilla)

### metrics-server aktivieren (microk8s und vanilla)

#### Warum ? Was macht er ?

```
Der Metrics-Server sammelt Informationen von den einzelnen Nodes und Pods
Er bietet mit

kubectl top pods
kubectl top nodes

ein einfaches Interface, um einen ersten Eindruck über die Auslastung zu bekommen.
```

### Walktrough

```
## Auf einem der Nodes im Cluster (HA-Cluster)
microk8s enable metrics-server

## Es dauert jetzt einen Moment bis dieser aktiv ist auch nach der Installation
## Auf dem Client
kubectl top nodes
kubectl top pods
```

### Kubernetes

- <https://kubernetes-sigs.github.io/metrics-server/>
- kubectl apply -f <https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>

## Kubernetes - Shared Volumes

### Shared Volumes with nfs

#### Create new server and install nfs-server

```
## on Ubuntu 20.04LTS
apt install nfs-kernel-server
systemctl status nfs-server
```

```

vi /etc/exports
## adjust ip's of kubernetes master and nodes
## kmaster
/var/nfs/ 192.168.56.101(rw,sync,no_root_squash,no_subtree_check)
## knode1
/var/nfs/ 192.168.56.103(rw,sync,no_root_squash,no_subtree_check)
## knode 2
/var/nfs/ 192.168.56.105(rw,sync,no_root_squash,no_subtree_check)

exportfs -av

```

## On all clients

```

#### Please do this on all servers

apt install nfs-common
## for testing
mkdir /mnt/nfs
## 192.168.56.106 is our nfs-server
mount -t nfs 192.168.56.106:/var/nfs /mnt/nfs
ls -la /mnt/nfs
umount /mnt/nfs

```

## Setup PersistentVolume and PersistentVolumeClaim in cluster

```

## vi nfs.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs
  labels:
    volume: nfs-data-volume
spec:
  capacity:
    # storage size
    storage: 5Gi
  accessModes:
    # ReadWriteMany(RW from multi nodes), ReadWriteOnce(RW from a node),
    # ReadOnlyMany(R from multi nodes)
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    # retain even if pods terminate
    Retain
  nfs:
    # NFS server's definition
    path: /var/nfs/nginx
    server: 192.168.56.106
    readOnly: false
    storageClassName: ""
---
```

```

## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim
spec:
  storageClassName: ""
  volumeName: pv-nfs
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi

```

```
kubectl apply -f nfs.yml
```

```

## deployment including mount
## vi deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: nfsvol
          persistentVolumeClaim:
            claimName: pv-nfs-claim

      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
          livenessProbe:
            exec:
              command:
                - /bin/sh
                - -c
                - "[ -f /run/nginx.pid ]"
            initialDelaySeconds: 10
            periodSeconds: 5

```

```
readinessProbe:
  httpGet:
    scheme: HTTP
    path: /
    port: 80
  initialDelaySeconds: 10
  periodSeconds: 5

volumeMounts:
- name: nfsvol
  mountPath: "/usr/share/nginx/html"
```

```
kubectl apply -f deploy.yml
```

## Kubernetes - Backups

## Kubernetes - Wartung

### kubectl drain/uncordon

```
## Achtung, bitte keine pods verwenden, dies können "ge"-drained (ausgetrocknet)
werden
kubectl drain <node-name>
z.B.
## Daemonsets ignorieren, da diese nicht gelöscht werden
kubectl drain n17 --ignore-daemonsets

## Alle pods von replicaset werden jetzt auf andere nodes verschoben
## Ich kann jetzt wartungsarbeiten durchführen

## Wenn fertig bin:
kubectl uncordon n17

## Achtung: deployments werden nicht neu ausgerollt, dass muss ich anstossen.
## z.B.
kubectl rollout restart deploy/webserver
```

### Alte manifeste konvertieren mit convert plugin

#### What is about?

- Plugins needs to be installed seperately on Client (or where you have your manifests)

#### Walkthrough

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert"
## Validate the checksum
curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert.sha256"
echo "$(cat kubectl-convert.sha256) kubectl-convert" | sha256sum --check
```

```
## install
sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert

## Does it work
kubectl convert --help

## Works like so
## Convert to the newest version
## kubectl convert -f pod.yaml
```

## Reference

- <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-convert-plugin>

## Kubernetes - Tipps & Tricks

### Assigning Pods to Nodes

#### Walkthrough

```
## leave n3 as is
kubectl label nodes n7 rechenzentrum=rz1
kubectl label nodes n17 rechenzentrum=rz2
kubectl label nodes n27 rechenzentrum=rz2

kubectl get nodes --show-labels

## nginx-deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 9 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
      nodeSelector:
        rechenzentrum: rz2

## Let's rewrite that to deployment
apiVersion: v1
kind: Pod
```



```
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    rechenzentrum=rz2
```

#### Ref:

- <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

## Kubernetes - Documentation

### Documentation zu microk8s plugins/addons

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

### LDAP-Anbindung

- <https://github.com/apprenda-kismatic/kubernetes-ldap>

### Shared Volumes - Welche gibt es ?

- <https://kubernetes.io/docs/concepts/storage/volumes/>

## 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 -
```

### 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/>

## 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/>

## 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/>