Kubernetes and Docker Administration

Agenda

1. Docker-Basics

   - Overview Architecture
   - What is a container ?
   - What are container images
   - Container vs. Virtual machines
   - What is a Dockerfile

2. Docker-Installation

   - Installation Docker under Ubuntu with snap

3. Docker-Commands

   - The most important commands
   - docker run
   - Logs anschauen - docker logs - mit Beispiel nginx
   - 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

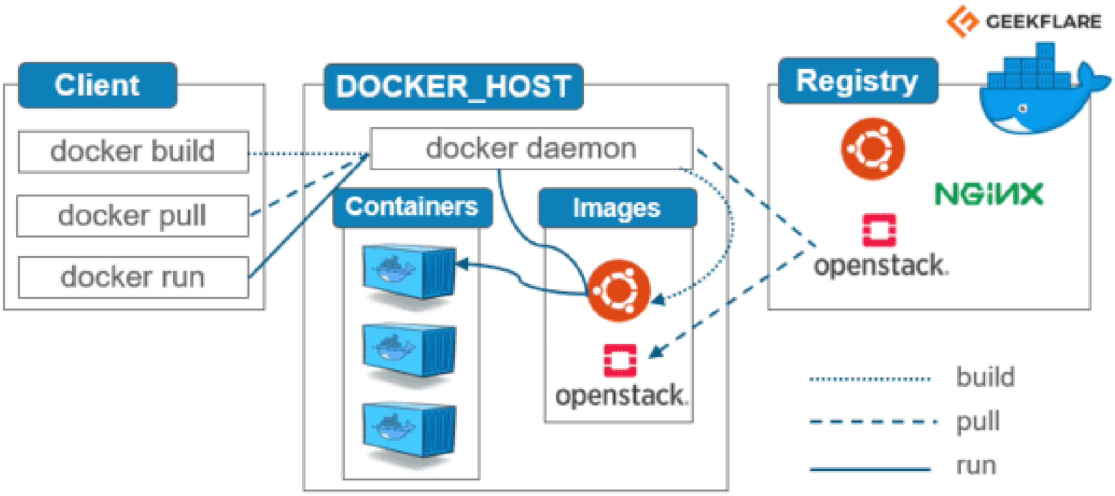
   - Netzwerk

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

Docker-Basics

Overview Architecture



What is a container ?

```
contains all of these:
- software
- libraries
- tools
- configuration data
- no own kernel
- good for executing applications in different enviroments

benefits:
- Containers are decoupled (entkoppelt)
- Containers are independent from each other
- They are able to communicate through a well defined interface to exchange
information

by decoupling containers:
- solve problems with libraries, tools, databases (different applications needing
different version)
```

What are container images

- container images are needed, to create container instances at runtime
- In docker container images are getting docker container, wenn they are executed as a process in the docker engine
- think docker images are templates
  - They used to create a docker container as copy

Container vs. Virtual machines

```
VM's virtualized hardware
containers are virtualized operating system

```

What is a Dockerfile

- Textfile, that holds Linux commands
    - you could also execute on the command line
    - these complete tasks, that are necessary, to assemble an image
    - the images are created with docker build

Docker-Installation

Installation Docker under Ubuntu with snap

```

sudo su -
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

## is the docker service being started on next book (autostart in windows)
systemctl is-enabled snap.docker.dockerd.service
```

Docker-Commands

The most important commands

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

docker run <image>
## z.b. // gets image from docker hub
## hub.docker.com
docker run hello-world

## images die lokal vorhanden
docker images

```

```
## container (running)
docker container ls
## container (present, but exited)
docker container ls -a

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

docker run

Example (bind it to a 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 is there
docker images

## exec in /into container
docker exec -it my_xenial bash
docker exec my_xenial cat /etc/os-release
##
```

Logs anschauen - docker logs - mit Beispiel nginx

in general

```
## start nginx and container-id will be shown
docker run -d nginx
a234
docker logs a234 # a234 are first 4 chars of container id
```

ongoing Log-output

```
docker logs -f a234
## stop ->  CTRL + c
```

Docker container/image stoppen/löschen

```
## stop it
docker stop my_xenial
docker container ls -a


## start it
docker start my_xenial
docker container ls

## Kill it if it cannot be stopped -be careful
docker kill my_xenial

## Works only, if container is not running
docker rm my_xenial

## remove container when running (force)
docker rm -f my_xenial

## delete image
docker rmi ubuntu:xenial

## if container is present but not running
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 run -t -d --name my_container ubuntu:latest
docker inspect my_container # my_container = container name
```

Docker container in den Vordergrund bringen - attach

docker attach - walkthrough

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

docker attach 1a4d

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

interactiven Prozess nicht beenden (statt exit)

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

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

Reference:

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

Aufräumen - container und images löschen

Alle nicht verwendeten container und images löschen

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

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

Nginx mit portfreigabe laufen lassen

Internal ip

```
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
```

External

```
docker run --name test-nginx -d -p 8080:80 nginx
lsof -i
cat /etc/services | grep 8080

## ip - address ? -> e.g. 134.209.247.10
ip a

## use browser on local machine (our desktop)
## and open <ip> http://<ip>:8080

## after the stop docker container docker, it is exposing anymore
docker stop test-nginx
## browser will complain
```

Dockerfile - Examples

Ubuntu mit hello world

Simple Version

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

### Step 2:
## nano Dockerfile
FROM ubuntu:latest

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

### Step 3:
nano hello.sh
##!/bin/bash
echo hello-docker

### Step 4:
## docker build -t dockertrainereu/<your-name>-hello-docker .
## Example
docker build -t dockertrainereu/<your-name>-hello-docker .
docker images
docker run dockertrainereu/<your-name>-hello-docker
```

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

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

## and look if we find it
## in browser
https://hub.docker.com/u/dockertrainereu
```

Loop Version (runs endlessly)

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

### Schritt 2:
```

```
## nano Dockerfile
FROM ubuntu:latest

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

### Schritt 3:
nano hello.sh
##!/bin/bash
while true
do
  echo hello-docker
  sleep 5
done

### Schritt 4:
## Beispiel
docker build -t my_echo .
docker images
docker run -d -t --name hello my_echo
docker exec -it hello sh
```

Ubuntu mit ping

```
## create a build folder
cd
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 is needed to run in the background
## also with -d (ohne -t) the bash is being executed, but  "the terminal" is stopped
## -> the container stops to run.
docker run -d -t --name container-ubuntu myubuntu
docker container ls
## enter the container by name myubuntu
docker exec -it container-ubuntu bash
ls -la
```

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

## docker inspect to find out ip of other container
## 172.17.0.3
```

```
docker inspect <container-id>

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

Nginx mit content aus html-ordner

Schritt 1: Simple Example

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

Schritt 2: Push build

```

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

Schritt 3: dokcer laufen lassen

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

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

```
curl http://localhost:8080


##
docker rm -f hello-web
```

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
```

Docker-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
docker container ls -a

## have a look in the volume.
## find out where the folder is saved on filesystem
docker inspect container test-vol

## create new container and check for /test_data/README
docker run -it --name=container-test-vol --mount target=/test_data2,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

Overview

```
3 types

o none
o bridge (Standard-network)
o host
```

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

command

```
## show networks
docker network ls

## show bridged network
## also show the ips of the container
docker inspect bridge

## network from the container perspective
docker inspect ubuntu-container
```

create network

```
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
```

delete network /

```
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?

```
## better. 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: "3.8"

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

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

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

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

Example with Wordpress / Nginx / MariadB

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

```
version: "3.7"

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

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


### now start the system
docker-compose up -d
### we can do some test if db is reachable
docker exec -it wp_wordpress_1 bash
### within shell do
apt update
apt-get install -y telnet
## this should work
telnet database 3306

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

Example with Ubuntu and Dockerfile

```
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
## image is being created and container started
## When Dockerfile changes you have to use the parameter --build
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

Why Kubernetes, what is kubernetes doing

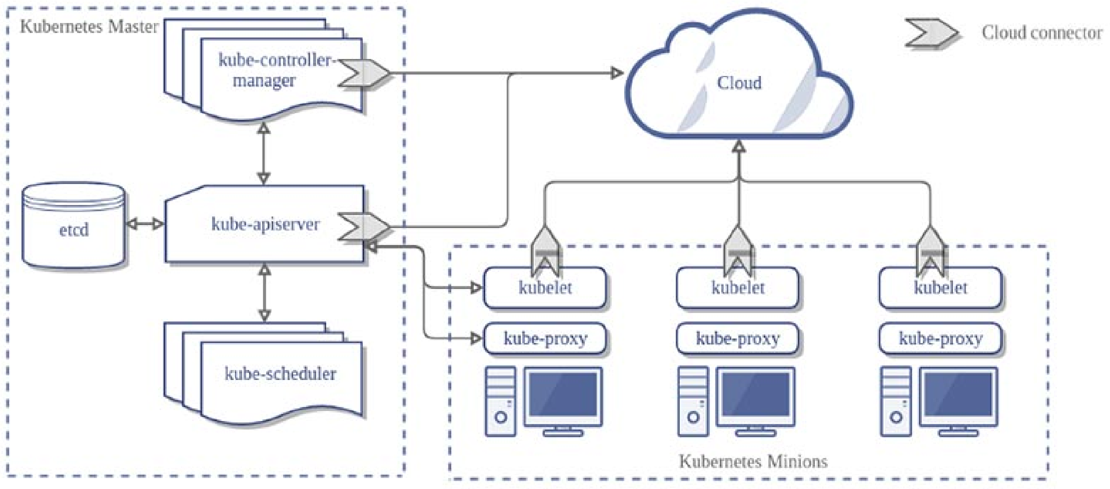
- Virtualiuzation of Operating Systems - 5times more efficient
- Google als Ausgangspunkt
- Software 2014 available as Open Source
- Use Hardware in a effective way, hundreds to thousands of services on a couple of mahines (Cluster)
- Immutable - System
- Self headling

Wny use Kubernetes ?

- Orchestration of Containern
- most commonly use with Docker

Architecture

Diagram



Components

Master (Control Plane)

Tasks / Jobs

- The master coordinates all the activities within the cluster
  - planning of applications (deployment of those)
  - management of the desired state of the applications
  - scaling of the applications
  - rollout of new updates.

components of the master

ETCD

- management of the cluster configuration (key/value - pairs)

KUBE-CONTROLLER-MANAGER

- in charge of monitoring the state of the cluster with endless loops
- communicated with the cluster through the kubernetes-api (provided by the kube-api-server)

KUBE-API-SERVER

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

KUBE-SCHEDULER

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

Nodes

- Nodes are workers (machines), that execute applications
- Ref: https://kubernetes.io/de/docs/concepts/architecture/nodes/

Pod/Pods

- pods are the smalles usable unit, that can be created and managed
- a pod (=group) is a group of one or more containers
  - the use mutual storage and network ressources
  - they are always on the same physical/virtual 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)
He is in charge, that container in a pod are executed
```

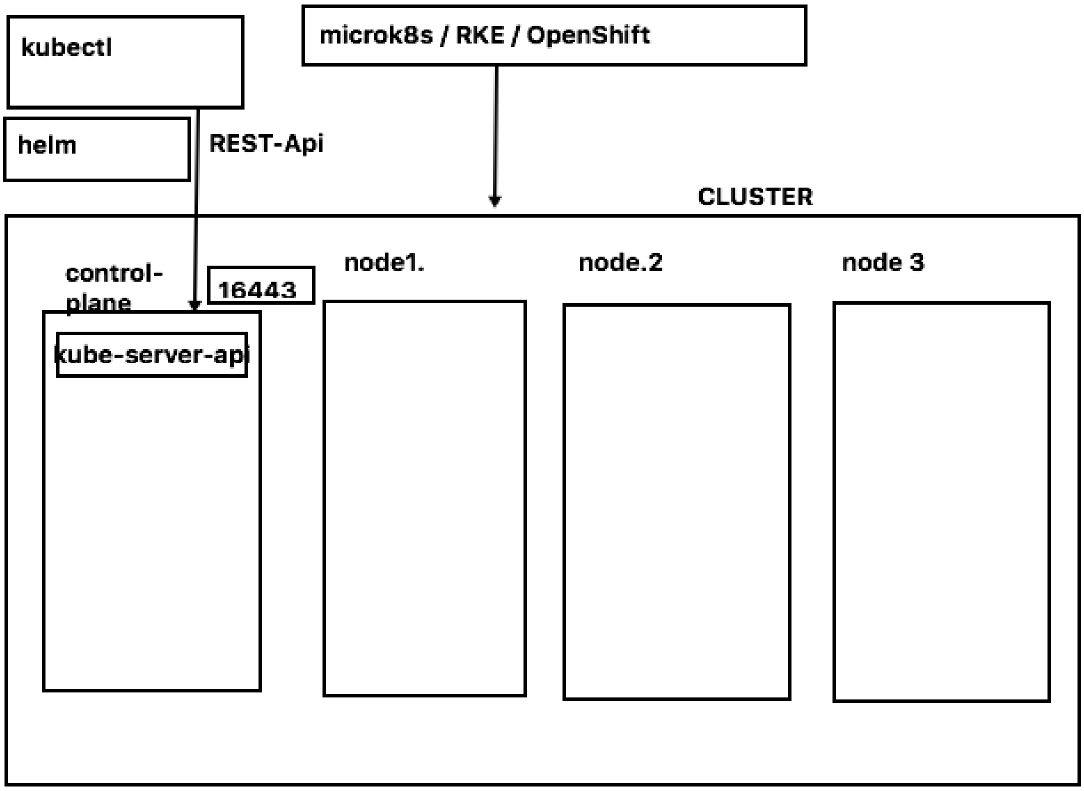
Kube-proxy

- runs on every node
- = network proxy for the kubernetes-network-services.
- Kube-proxy manages the network communication inside and (to/from) the outside of the cluster

source:

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

Aufbau mit helm,OpenShift,Rancher(RKE),microk8s



Welches System ? (minikube, micro8ks etc.)

Überblick der Systeme

General

```
kubernetes itself has no convenient way of doing specific stuff like
creating the kubernetes cluster.

So there are other tools/distributions 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 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             164.92.255.234/20 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
## namensaufloesung fuer pods
microk8s enable dns
```

```
## Funktioniert microk8s
microk8s status
```

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

```
## Was macht das ?
## 1. Basisnutzer (11trainingdo) - 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$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONw
 /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 Scripte

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

## Digitalocean - unter user_data reingepastet beim Einrichten

##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$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONw
 /etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo
```

Kubernetes - Installer - microk8s (Installation und Management)

Installation Ubuntu - snap

Walkthrough

```
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
```

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 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

Installation Kubernetes Dashboard

Reference:

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

Kubernetes - doks (digitaloean) (Installation and Management)

Setup Cluster with kubectl client

what is doks ?

- Managed digital ocean kubernetes cluster

Step 1: Setup Cluster online

```
https://cloud.digitalocean.com/kubernetes/clusters/new
* 3 nodes
* at least 8 GB Ram per Node
* No HA for the control plan

Takes a while till it ready... take some coffee
```

Step 2: Install and configure client

```
## on an ubuntu machine using it as client
## being root - user
snap install --classic kubectl
kubectl completion bash > /etc/bash_completion.d/kubectl
## relogin for the config to take effect
su -

#### Get the configuration from the cluster
## in the kubernetes interface on digitalocean download
## Dashboard of cluster in digitalocean -> window (configuration) -> download config

### on client-machine as root
#### Save it in ~/.kube/config
cd
mkdir .kube
cd .kube
scp 11trainingdo@10.135.0.7:/home/11trainingdo/config .

### test the connection - should have a proper connection now
kubectl cluster-info
```

Kubernetes - Installer - Rancher

Installation on digitalocean

Prerequisites

- You need to create an api -key (with write access under) : Left Menu -> Api -> Applications & API

Walktrough

```
1. Create a new droplet with os -> Rancher OS
use userdata as mentioned in the the document:
https://rancher.com/docs/os/v1.x/en/installation/cloud/do/

2. Connect to the created droplet with the local browser using the created ip
3. Set credentials
4. Create Cluster
```

Create Cluster

- Decide which cluster you want to create (on gke, digitalocean a.s.o)

- Depending on choice provide necessary data

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops) - Cheatsheet

Allgemein

```
## Show information about the cluster
## check connection
kubectl cluster-info

## Which api-resources are available ?
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 natürluch 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
```

Working with 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

## set a new default namespace
kubectl config set-context --current --namespace=<yourname>



## 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 example with run

Example (that does work)

```
## Show the pods that are running
kubectl get pods

## 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/manifest/pod



### Walkthrough
```

cd

mkdir -p manifests

cd manifests

mkdir web

cd web

nano nginx-static.yml

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

- name: web image: nginx

```
```

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

show config

kubectl get pod/nginx-static-web -o yaml kubectl get pod/nginx-static-web -o wide

```
### kubectl/manifest/replicaset
```

cd cd manifests mkdir 02-rs cd 02-rs nano rs.yml

apiVersion: apps/v1 kind: ReplicaSet metadata: name: nginx-replica-set spec: replicas: 2 selector: matchLabels: tier: frontend template: metadata: name: template-nginx-replica-set labels: tier: frontend spec: containers: - name: nginx image: "nginx:latest" ports: - containerPort: 80

kubectl apply -f rs.yml

```
### kubectl/manifest/deployments
```

cd cd manifests mkdir 03-deploy cd 03-deploy nano deploy.yml

vi deploy.yml

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

kubectl apply -f deploy.yml

```
### Services - Structure


![Services Aufbau](/images/kubernetes-services.drawio.svg)

### kubectl/manifest/service


### Prepare
```

cd cd manifests mkdir 04-svc cd 04-svc

```
### Version 1: all objects in one file (not recommended)
```

nano svc.yml

apiVersion: apps/v1 kind: Deployment metadata: name: my-nginx spec: selector: matchLabels: run: my-nginx replicas: 3 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: svc: nginx spec: ports:

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

kubectl apply -f .

```

### Version 2: Recommended
```

nano 01-deploy.yml

apiVersion: apps/v1 kind: Deployment metadata: name: my-nginx spec: selector: matchLabels: run: my-nginx replicas: 3 template: metadata: labels: run: my-nginx spec: containers: - name: my-nginx image: nginx ports: - containerPort: 80

nano 02-service.yml

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

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

```
kubectl apply -f .
```

Ref.

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

Background IngressController

Ref. / Dokumentation

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

Documentation for default ingress nginx

- https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/

Beispiel Ingress

Prerequisits

```
## 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-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
```

Install Ingress On Digitalocean DOKS

Basics

- Works on other platforms as, if you do not have ingress controller installed
- A bit more complicated, if you have the choice, use process/plugin from installer, e.g. microk8s enable ingress

Prerequisites

- kubectl needs to be installed and configured

Walkthrough (Setup Ingress Controller)

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm show values ingress-nginx/ingress-nginx

## It will be setup with type loadbalancer - so waiting to retrieve an ip from the
external loadbalancer
## This will take a little.
helm install nginx-ingress ingress-nginx/ingress-nginx --namespace ingress --create-
namespace --set controller.publishService.enabled=true

## See when the external ip comes available
```

```
kubectl -n ingress get all
kubectl --namespace ingress get services -o wide -w nginx-ingress-ingress-nginx-
controller

## Output
NAME                                     TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)                      AGE     SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer   10.245.78.34   157.245.20.222
80:31588/TCP,443:30704/TCP   4m39s
app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-
ingress,app.kubernetes.io/name=ingress-nginx

## Now setup wildcard - domain for training purpose
## inwx.com
*.lab1.t3isp.de A 157.245.20.222
```

Ingress Example with hostnames

Prerequisits

```
## Ingress Controller muss aktiviert sein
### Nur der Fall wenn man microk8s zum Einrichten verwendet
### Ubuntu
microk8s enable ingress
```

Walkthrough

```
mkdir abi
cd abi
```

```
## 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-<your-name>"
---

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-<your-name>"

---

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: /
    # with the ingress controller from helm, you need to set an annotation
    # otherwice it does not know, which controller to use
    # old version... use ingressClassName instead
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
```

```
  - host: "<euername>.lab<nr>.t3isp.de"
    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: /
    # with the ingress controller from helm, you need to set an annotation
    # old version useClassName instead
    # otherwice it does not know, which controller to use
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "app12.lab.t3isp.de"
```

```
    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
```

Achtung: Ingress mit Helm - annotations

Permanente Weiterleitung mit Ingress

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
-

ConfigMap Example

Step 1: configmap preparation

```
### 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-configmap
data:
  # als Wertepaare
  database: mongodb
  database_uri: mongodb://localhost:27017

  # als Inhalte
  keys: |
    image.public.key=771
    rsa.public.key=42
```

```
kubectl apply -f 01-configmap.yml
kubectl get cm
kubectl get cm -o yaml
```

Step 2: Example as file

```
nano 02-pod.yml
```

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-mit-configmap

spec:
  # Add the ConfigMap as a volume to the Pod
  volumes:
    # `name` here must match the name
    # specified in the volume mount
    - name: example-configmap-volume
      # Populate the volume with config map data
      configMap:
        # `name` here must match the name
        # specified in the ConfigMap's YAML
        name: example-configmap

  containers:
    - name: container-configmap
      image: nginx:latest
      # Mount the volume that contains the configuration data
      # into your container filesystem
      volumeMounts:
        # `name` here must match the name
        # from the volumes section of this pod
        - name: example-configmap-volume
          mountPath: /etc/config
```

```
kubectl apply -f 02-pod.yml
```

```
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-mit-configmap -- ls -la /etc/config
kubectl exec -it pod-mit-configmap --  bash
## ls -la /etc/config
```

Step 3: Beispiel. ConfigMap als env-variablen

```
## 03-pod-mit-env.yml
kind: Pod
apiVersion: v1
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:latest
      envFrom:
        - configMapRef:
            name: example-configmap
```

```
kubectl apply -f 03-pod-mit-env.yml
```

```
## und wir schauen uns das an
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-env-var -- env
kubectl exec -it pod-env-var --  bash
## env
```

Reference:

- https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html

Kubernetes - RBAC

Nutzer einrichten

Enable RBAC in microk8s

```
## This is important, if not enable every user on the system is allowed to do
everything
microk8s enable rbac
```

Schritt 1: Nutzer-Account auf Server anlegen / in Client

```
cd
mkdir -p manifests/rbac
cd manifests/rbac
```

Mini-Schritt 1: Definition für Nutzer

```
## vi service-account.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: training
  namespace: default


kubectl apply -f service-account.yml
```

Mini-Schritt 2: ClusterRolle festlegen - Dies gilt für alle namespaces, muss aber noch zugewiesen werden

```
### Bevor sie zugewiesen ist, funktioniert sie nicht - da sie keinem Nutzer zugewiesen
ist

## vi pods-clusterrole.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-clusterrole
rules:
- apiGroups: [""] # "" indicates the core API group
```

```
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

kubectl apply -f pods-clusterrole.yml
```

Mini-Schritt 3: Die ClusterRolle den entsprechenden Nutzern über RoleBinding zu ordnen

```
## vi rb-training-ns-default-pods.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rolebinding-ns-default-pods
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pods-clusterrole
subjects:
- kind: ServiceAccount
  name: training
  namespace: default

kubectl apply -f rb-training-ns-default-pods.yml
```

Mini-Schritt 4: Testen (klappt der Zugang)

```
kubectl auth can-i get pods -n default --as system:serviceaccount:default:training
```

Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen

Mini-Schritt 1: kubeconfig setzen

```
kubectl config set-context training-ctx --cluster microk8s-cluster --user training

## extract name of the token from here
TOKEN_NAME=`kubectl get serviceaccount training -o jsonpath='{.secrets[0].name}'`

TOKEN=`kubectl get secret $TOKEN_NAME -o jsonpath='{.data.token}' | base64 --decode`
echo $TOKEN
kubectl config set-credentials training --token=$TOKEN
kubectl config use-context training-ctx

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

Mini-Schritt 2:

```
kubectl config use-context training-ctx
kubectl get pods
```

Refs:

- https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm
- https://microk8s.io/docs/multi-user
- https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286

Kubernetes - Netzwerk (CNI's) + Mesh

Übersicht Netzwerke

CNI

- Common Network Interface

- fixed Definition, how containers with network libraries are communicating

- ### Docker - Container oder andere

- Container is started -> over CNI -> get pod-ip

- Container is stopped -> over CNI -> pod - IP is released

Which are there ?

- Flanel
- Canal
- Calico
- Cilium

Flanel

Overlay - Network

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

Advantage

- good and easy start
- reduced to only one binary flanneld

Disadvantages

- no network policies allowed
- hard to debug, because no classical network (virtual), cannot use classic network tools

Canal

General

- Auch ein Overlay - Netzwerk
- Unterstüzt auch policies

Calico

Generally

- 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

Cilium

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.
```

Callico - 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

Callico - client-backend-ui-example

Walkthrough

```
cd
mkdir -p manifests/callico/example1
cd manifests/callico/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 possilbe
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/
```

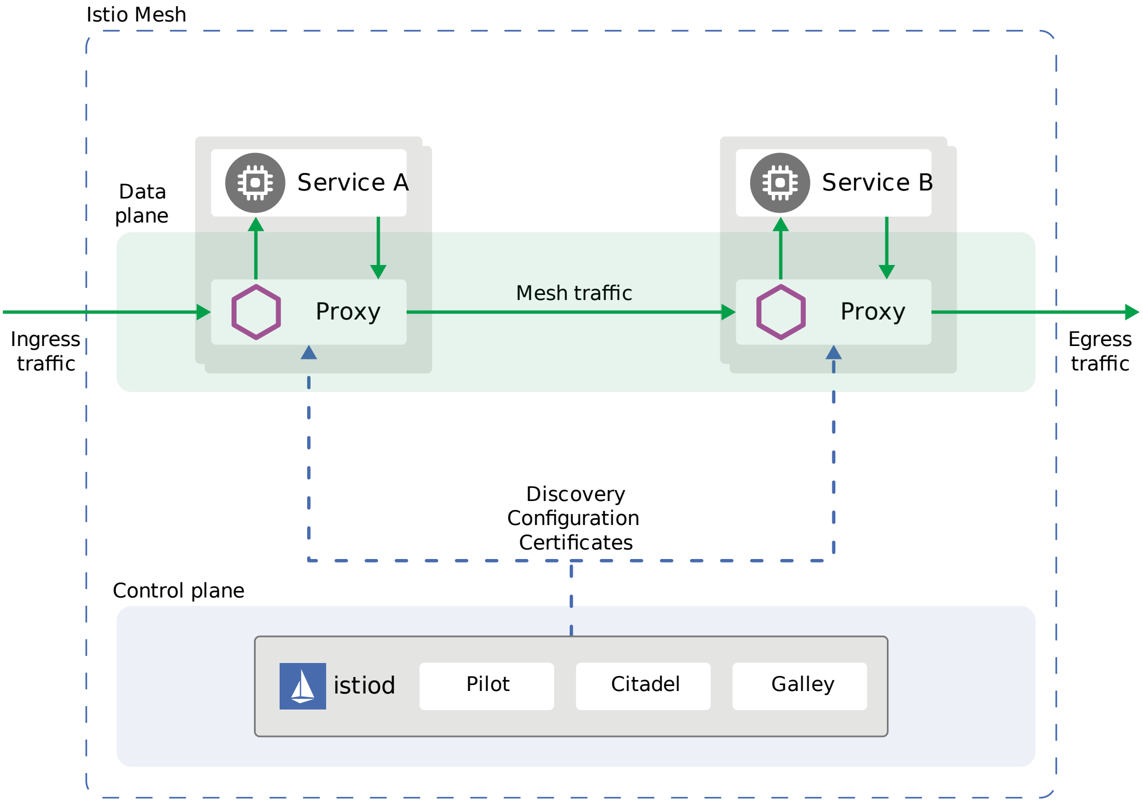
```
## Alles wieder löschen
kubectl delete ns client stars management-ui
```

Reference

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

Mesh - istio as a type of implementation

Overview



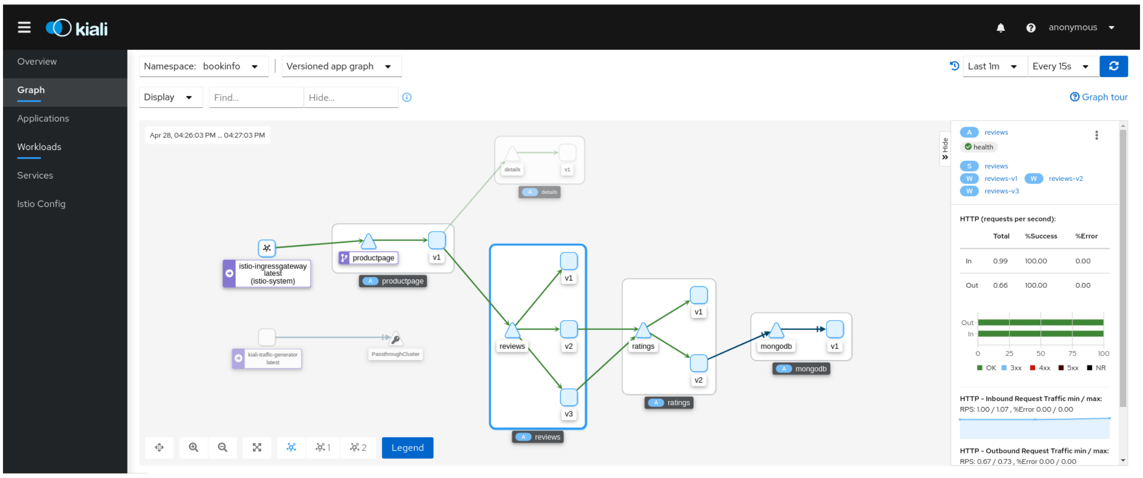
Istio

```
## Visualization
## with kiali (included in istio)
https://istio.io/latest/docs/tasks/observability/kiali/kiali-graph.png

## Example
## https://istio.io/latest/docs/examples/bookinfo/
The sidecars are injected in all pods within the namespace by labeling the namespace
like so:
kubectl label namespace default istio-injection=enabled

## Gateway (like Ingress in vanilla Kubernetes)
kubectl label namespace default istio-injection=enabled
```

Visualization of Services (kiali - included in istio)



istio - the next generation without sidecar

- https://istio.io/latest/blog/2022/introducing-ambient-mesh/

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

prometheus

checkmk plugin

- https://docs.checkmk.com/latest/de/monitoring_kubernetes.html

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

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

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

Shared Volumes with nfs/multipe

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 nodes (needed for production)

```
##
apt install nfs-common
```

On all nodes (only for testing)

```
#### Please do this on all servers (if you have access by ssh)
### find out, if connection to nfs works !

## 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
```

Persistent Storage-Step 1: Setup PersistentVolume in cluster

```
cd
cd manifests
mkdir -p nfs

```

```
cd nfs
nano 01-pv.yml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln<nr>
  labels:
    volume: nfs-data-volume-tln<nr>
spec:
  capacity:
    # storage size
    storage: 1Gi
  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/tln<nr>/nginx
    server: 10.135.0.14
    readOnly: false
  storageClassName: ""
```

```
kubectl apply -f 01-pv.yml
kubectl get pv
```

Persistent Storage-Step 2: Create Persistent Volume Claim

```
nano 02-pvs.yml
```

```
## vi 02-pvs.yml
## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim-tln<nr>
spec:
  storageClassName: ""
  volumeName: pv-nfs-tln<nr>
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

```
kubectl apply -f 02-pvs.yml
kubectl get pvc
```

Persistent Storage-Step 3: Deployment

```
## deployment including mount
## vi 03-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # tells deployment to run 4 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:

      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80

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

      volumes:
      - name: nfsvol
        persistentVolumeClaim:
          claimName: pv-nfs-claim-tln<tln>

```

```
kubectl apply -f 03-deploy.yml
```

Persistent Storage Step 4: service

```
## now testing it with a service
## cat 04-service.yml
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
  labels:
    run: svc-my-nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

```
kubectl apply -f 04-service.yml

## connect to the container and add index.html - data
kubectl exec -it deploy/nginx-deployment -- bash
## in container
echo "hello dear friend" > /usr/share/nginx/html/index.html
exit

## now try to connect
kubectl get svc

## connect with ip and port
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit

## now destroy deployment
kubectl delete -f 03-deploy.yml

## Try again - no connection
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit


## now start deployment again
kubectl apply -f 03-deploy.yml

## and try connection again
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit
```

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 replicasets 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 "$(<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 Probes (Liveness and Readiness)

Übung Liveness-Probe

Übung 1: Liveness (command)

```
What does it do ?

* At the beginning pod is ready (first 30 seconds)
* Check will be done after 5 seconds of pod being startet
* Check will be done periodically every 5 minutes and will check
  * for /tmp/healthy
  * if file is there will return: 0
  * if file is not there will return: 1
* After 30 seconds container will be killed
* After 35 seconds container will be restarted
```

```
## cd
## mkdir -p manifests/probes
## cd manifests/probes
## vi 01-pod-liveness-command.yml

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
## apply and test
kubectl apply -f 01-pod-liveness-command.yml
kubectl describe -l test=liveness pods
sleep 30
kubectl describe -l test=liveness pods
sleep 5
kubectl describe -l test=liveness pods
```

```
## cleanup
kubectl delete -f 01-pod-liveness-command.yml
```

Übung 2: Liveness Probe (HTTP)

```
## Step 0: Understanding Prerequisite:
This is how this image works:
## after 10 seconds it returns code 500
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([](fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]("ok"))
    }
})
```

```
## Step 1: Pod  - manifest
## vi 02-pod-liveness-http.yml
## status-code >=200 and < 400 o.k.
## else failure
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

```
## Step 2: apply and test
kubectl apply -f 02-pod-liveness-http.yml
## after 10 seconds port should have been started
sleep 10
kubectl describe pod liveness-http
```

Reference:

- https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

Funktionsweise Readiness-Probe vs. Liveness-Probe

Why / Howto / Difference to LiveNess

- Readiness checks, if container is ready and if it's not READY
  - SENDS NO TRAFFIC to the container
- They are configured exactly the same, but use another keyword
  - readinessProbe instead of livenessProbe

Example

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Reference

- https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/#define-readiness-probes

configmaps / secrets

configmap

Step 1: configmap preparation

```
### 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-configmap
data:
  # als Wertepaare
  database: mongodb
  database_uri: mongodb://localhost:27017

  # als Inhalte
  keys: |
    image.public.key=771
    rsa.public.key=42
```

```
kubectl apply -f 01-configmap.yml
kubectl get cm
kubectl get cm -o yaml
```

Step 2: Example as file

```
nano 02-pod.yml
```

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-mit-configmap

spec:
  # Add the ConfigMap as a volume to the Pod
  volumes:
    # `name` here must match the name
    # specified in the volume mount
    - name: example-configmap-volume
      # Populate the volume with config map data
      configMap:
        # `name` here must match the name
        # specified in the ConfigMap's YAML
        name: example-configmap

  containers:
    - name: container-configmap
      image: nginx:latest
      # Mount the volume that contains the configuration data
      # into your container filesystem
      volumeMounts:
        # `name` here must match the name
        # from the volumes section of this pod
        - name: example-configmap-volume
          mountPath: /etc/config
```

```
kubectl apply -f 02-pod.yml
```

```
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-mit-configmap -- ls -la /etc/config
kubectl exec -it pod-mit-configmap --  bash
## ls -la /etc/config
```

Step 3: Beispiel. ConfigMap als env-variablen

```
## 03-pod-mit-env.yml
kind: Pod
apiVersion: v1
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:latest
      envFrom:
        - configMapRef:
            name: example-configmap
```

```
kubectl apply -f 03-pod-mit-env.yml
```

```
## und wir schauen uns das an
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-env-var -- env
kubectl exec -it pod-env-var --  bash
## env
```

Reference:

- https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html

configmap-mysql-realworld example

```
## 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: mysql-configmap
data:
  # als Wertepaare
  MYSQL_ROOT_PASSWORD: abcmaster
## 03-pod-mit-env.yml
kind: Pod
apiVersion: v1
metadata:
  name: mysql
spec:
  containers:
    - name: mysql
      image: mysql:5.7
      envFrom:
        - configMapRef:
            name: mysql-configmap

```

Working with secrets

Exercise 1 - set ENV variables from Secrets

```
## Schritt 1: Secret anlegen.
## Diesmal noch nicht encoded - base64
## vi 06-secret-unencoded.yml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
stringData:
    APP_PASSWORD: "s3c3tp@ss"
    APP_EMAIL: "mail@domain.com"
```

```
## Schritt 2: Apply'en und anschauen
kubectl apply -f 06-secret-unencoded.yml
## ist zwar encoded, aber last_applied ist im Klartext
## das könnte ich nur nur umgehen, in dem ich es encoded speichere
kubectl get secret mysecret -o yaml
```

```
## Schritt 3:
## vi 07-print-envs-complete.yml
apiVersion: v1
kind: Pod
metadata:
  name: print-envs-complete
spec:
  containers:
  - name: env-ref-demo
    image: nginx
    env:
    - name: APP_VERSION
      value: 1.21.1
    - name: APP_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: APP_PASSWORD
    - name: APP_EMAIL
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: APP_EMAIL
```

```
## Schritt 4:
kubectl apply -f 07-print-envs-complete.yml
kubectl exec -it print-envs-complete -- bash
##env | grep -e APP_ -e MYSQL
```

Bitnami

2 Komponenten

- Sealed Secrets consists of 2 components
  - kubeseal, to encrypt passwords
  - The Operator (ein Controller) is in charge for decrypting

Schritt 1: Walkthrough - Client Installation (als root)

```
## download binary for linux
## Achtung: Immer die neueste Version von den Releases nehmen, siehe unten:
## Install as root
cd /usr/src
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.17.5/kubeseal-0.17.5-linux-amd64.tar.gz
tar xzvf kubeseal-0.17.5-linux-amd64.tar.gz
install -m 755 kubeseal /usr/local/bin/kubeseal
```

Schritt 2: Walkthrough - Server Installation with kubectl client

```
## auf dem Client
## cd
## mkdir manifests/seal-controller/ #
## cd manifests/seal-controller
## Neueste Version
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.17.5/controller.yaml
kubectl apply -f controller.yaml
```

Schritt 3: Walkthrough - Verwendung (als normaler/unpriviligierter Nutzer)

```
kubeseal --fetch-cert

## Secret - config erstellen mit dry-run, wird nicht auf Server angewendet (nicht an Kube-Api-Server geschickt)
kubectl create secret generic basic-auth --from-literal=APP_USER=admin --from-literal=APP_PASS=change-me --dry-run=client -o yaml > basic-auth.yaml
cat basic-auth.yaml

## öffentlichen Schlüssel zum Signieren holen
kubeseal --fetch-cert > pub-sealed-secrets.pem
cat pub-sealed-secrets.pem

kubeseal --format=yaml --cert=pub-sealed-secrets.pem < basic-auth.yaml > basic-auth-sealed.yaml
cat basic-auth-sealed.yaml

## Ausgangsfile von dry-run löschen
rm basic-auth.yaml

## Ist das secret basic-auth vorher da ?
kubectl get secrets basic-auth

kubectl apply -f basic-auth-sealed.yaml

## Kurz danach erstellt der Controller aus dem sealed secret das secret
kubectl get secret
kubectl get secret -o yaml
```

```
## Ich kann dieses jetzt ganz normal in meinem pod verwenden.
## Step 3: setup another pod to use it in addition
## vi 02-secret-app.yml
apiVersion: v1
kind: Pod
metadata:
  name: secret-app
spec:
  containers:
    - name: env-ref-demo
      image: nginx
      envFrom:
```

```
      - secretRef:
          name: basic-auth
```

Hinweis: Ubuntu snaps

```
Installation über snap funktioniert nur, wenn ich auf meinem Client
ausschliesslich als root arbeite
```

Wie kann man sicherstellen, dass nach der automatischen Änderung des Secretes, der Pod bzw. Deployment neu gestartet wird ?

- https://github.com/stakater/Reloader

Ref:

- Controller: https://github.com/bitnami-labs/sealed-secrets/releases/

Helm (Kubernetes Package Manager)

Basics

Where ?

```
artifacts helm
https://artifacthub.io/
```

components

```
Chart - contains description and components
tar.gz - Format
or a folder

when executen a chart a release will be created.
(parallel: image -> container, analog: chart -> release
```

Installation

```
## Beispiel ubuntu
## snap install --classic helm

## Cluster muss vorhanden, aber nicht notwendig wo helm installiert

## Voraussetzung auf dem Client-Rechner (helm ist nichts als anderes als ein Client-
Programm)
Ein lauffähiges kubectl auf dem lokalen System (welches sich mit dem Cluster
verbinden.
-> saubere -> .kube/config

## Test
kubectl cluster-info
```

Example with kubernetes package manager

Prerequisites

- kubectl needs to be installed and configured to access cluster
- Good: helm works as unprivileged user as well - Good for our setup
- install helm on ubuntu (client) as root: snap install --classic helm
  - this installs helm3
- Please only use: helm3. No server-side components needed (in cluster)
  - Get away from examples using helm2 (hint: helm init) - uses tiller

Important commands

```
## Repo hinzufpgen
helm repo add bitnami https://charts.bitnami.com/bitnami
## gecachte Informationen aktualieren
helm repo update

## search in configured repos
helm search repo mysql

## search in artifacts hub
helm search hub mysql

## Chart runterziehen ohne installieren
helm pull bitnami/mysql


helm install release-name bitnami/mysql



## Release anzeigen zu lassen
helm list

## Status einer Release / Achtung, heisst nicht unbedingt nicht, dass pod läuft
helm status my-mysql

## zweiten Release
helm install neuer-release-name bitnami/mysql
```

Under the hood

```
## Helm speichert Informationen über die Releases in den Secrets
kubectl get secrets | grep helm
```

Example 1: - To get know the structure

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami
helm repo update
helm pull bitnami/mysql
tar xzvf mysql-9.0.0.tgz
```

Example 2: We will setup mysql without persistent storage (not helpful in production ;o()

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami
helm repo update

helm install my-mysql bitnami/mysql

```

Example 2 - continue - fehlerbehebung

```
helm uninstall my-mysql
## Install with persistentStorage disabled - Setting a specific value
helm install my-mysql --set primary.persistence.enabled=false bitnami/mysql

## just as notice
## helm uninstall my-mysql
```

Example 2b: using a values file

```
## mkdir helm-mysql
## cd helm-mysql
## vi values.yml
primary:
  persistence:
    enabled: false
```

```
helm uninstall my-mysql
helm install my-mysql bitnami/mysql -f values.yml
```

Referenced

- https://github.com/bitnami/charts/tree/master/bitnami/mysql/#installing-the-chart
- https://helm.sh/docs/intro/quickstart/

Kustomize

Kustomize Exercise/Example

concepts Overlay

- Base + Overlay = Patched manifests

Example 1: Walkthrough

```
## Step 1:
## Create the structure
## kustomize-example1
## L base
## | - kustomization.yml
## L overlays
##.    L dev
##       - kustomization.yml
##.    L prod
##.      - kustomization.yml
cd; mkdir -p manifests/kustomize-example1/base; mkdir -p manifests/kustomize-example1/overlays/prod; cd manifests/kustomize-example1
```

```
## Step 2: base dir with files
## now create the base kustomization file
## vi base/kustomization.yml
resources:
- service.yml
```

```
## Step 3: Create the service - file
## vi base/service.yml
kind: Service
apiVersion: v1
metadata:
  name: service-app
spec:
  type: ClusterIP
  selector:
    app: simple-app
  ports:
  - name: http
    port: 80
```

```
## See how it looks like
## only used, if you want to test the functionality
## kubectl kustomize ./base
```

```
## Step 4: create the customization file accordingly
##vi overlays/prod/kustomization.yaml
bases:
- ../../base
patches:
- service-ports.yaml
```

```
## Step 5: create overlay (patch files)
## vi overlays/prod/service-ports.yaml
kind: Service
apiVersion: v1
metadata:
  #Name the to be patched resource
  name: service-app
spec:
  # Changed to Nodeport
  type: NodePort
  ports: #Die Porteinstellungen werden überschrieben
  - name: https
    port: 443
```

```
## Step 6:
kubectl kustomize overlays/prod

## or apply it directly
kubectl apply -k overlays/prod/
```

```
## Step 7:
## mkdir -p overlays/dev
## vi overlays/dev/kustomization.yml
bases:
- ../../base
```

```
## Step 8:
## statt mit der base zu arbeiten
kubectl kustomize overlays/dev
```

Example 2: Advanced Patching with patchesJson6902 (You need to have done example 1 firstly)

```
## Schritt 1:
## Replace overlays/prod/kustomization.yml with the following syntax
bases:
- ../../base
patchesJson6902:
- target:
    version: v1
    kind: Service
    name: service-app
  path: service-patch.yaml
```

```
## Schritt 2:
## vi overlays/prod/service-patch.yaml
- op: remove
  path: /spec/ports
  value:
  - name: http
    port: 80
- op: add
  path: /spec/ports
  value:
  - name: https
    port: 443
```

```
## Schritt 3:
kubectl kustomize overlays/prod
```

Special Use Case: Change the metadata.name

```
## Same as Example 2, but patch-file is a bit different
## vi overlays/prod/service-patch.yaml
- op: remove
  path: /spec/ports
  value:
  - name: http
    port: 80

- op: add
  path: /spec/ports
  value:
  - name: https
    port: 443

- op: replace
  path: /metadata/name
  value: svc-app-test
```

```
kubectl kustomize overlays/prod
```

Ref:

- https://blog.ordix.de/kubernetes-anwendungen-mit-kustomize

Marriage between helm and kustomize

Option 1: unpack helm chart and patch it

```
helm add repo bitnami https://charts.bitnami.com/bitnami
cd base
helm template my-release bitnami/mysql > all-resources.yaml
## patchen kustomize
cd ..
kustomize build overlay/prod
kubectl apply -k overlay/prod
```

Option 2: packe helm chart aus

```
## pull
helm pull
tar xvf mysql-9.0.34.tgz
## templates werden
kubectl kustomize build overlay/prod
helm install mysql-release mysql # 2. mysql wäre das chart-verzeichnis lokal im
filesystem
## Vorteile
## ich kann das ganze auch wieder so installieren
## ich kann ein update durch führen
```

Option 3: helm --post-renderer

```
## erst wird template erstellt und dann dann ein weiteres script ausgeführt
## und dann erst installiert.
helm install --post-renderer=./patch.sh

## im shell-script
## kubectl kustomize
## https://austindewey.com/2020/07/27/patch-any-helm-chart-template-using-a-kustomize-post-renderer/
```

Option 4: kustomize lädt helm - chart

```
## kustomization.yml
https://github.com/kubernetes-sigs/kustomize/blob/master/examples/chart.md
```

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 debugging ClusterIP/PodIP

Situation

- No access to the nodes, to test the connection to pods and services

Solution

```
## using busybox (swiss army knife of embedded)
## busytester is the name
## long version
kubectl run -it --rm --image=busybox busytester
## wget <pod-ip-des-ziels>
## exit


## quick and dirty
kubectl run -it --rm --image=busybox busytester -- wget <pod-ip-des-ziels>
```

Debugging pods

How ?

1. Which pod is in charge
2. Problems when starting: kubectl describe po mypod
3. Problems while running: kubectl logs mypod

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/

Helpful to learn - Kubernetes

- https://kubernetes.io/docs/tasks/

Environment to learn

- https://killercoda.com/killer-shell-cks

Youtube Channel

- https://www.youtube.com/watch?v=01qcYSck1c4

Defining defaultBackend for IngressController

- https://dev.to/kenmoini/custom-kubernetes-ingress-default-backend-and-error-pages-3alh

Kubernetes -Wann / Wann nicht

Kubernetes Wann / Wann nicht

Frage: Kubernetes: Sollen wir das machen und was kost' mich das ?

Rechtliche Regulatorien

Nationale Grenzen

Cloud oder onPrem (private Cloud)

Gegenfragen:

```
1. Monolithisches System (SAP Rx) <-> oder stark modulares System (Web-Applikation mit
microservices)

   Kubernetes : weniger sinnvoll  <-> sehr sinnvoll.
```

Kosten:

```
  o Konzeption / Planung
  o Cluster / Manpower (Cluster-Kompetenz)
  o Neue Backup-Strategie / Software
  o Monitoring (ELK / EFK - STack (Elastich Search / Logstash-Fluent))
```

Anforderungen an Last

- Statisch (immer gleich)
- Dynamisch (stark wechselnd) - Einsparpotential durch Features Cloudanbieter (nur so viel bezahlen wie ich nutze)

Nutzt mir Skailierung und kann ich skalieren

- Gibt meine Applikation
- Habe durch mehr Webservice der gleichen Typs eine bessere Performance

Kubernetes -> Kategorien. Warum ?

- Kosten durch Umstellung auf Cloud senken ?
- Automatisches Skalieren meiner Software bei Hochlast / Bedarf (verbunden mit dynamische Kosten)
- Erleichtertes Handling Updates (schnelleres Time-To-Market -> neuere Versioninierung)

Kubernetes - Hardening

Kubernetes Tipps Hardening

PSA (Pod Security Admission)

```
Policies defined by namespace.
e.g. not allowed to run container as root.

Will complain/deny when creating such a pod with that container type
```

Example (seccomp / security context)

```
A. seccomp - profile
https://github.com/docker/docker/blob/master/profiles/seccomp/default.json
```

```
apiVersion: v1
kind: Pod
metadata:
  name: audit-pod
  labels:
    app: audit-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json

  containers:

  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

SecurityContext (auf Pod Ebene)

```
kubectl explain pod.spec.containers.securityContext
```

NetworkPolicy

```
## Firewall Kubernetes
```

Kubernetes run as non root

Note:

```
USER does not need to exist on the system.
The settings

securityContext:
  runasuser: 12000

overwrites the settings under which the docker container runs
Directive: USER

BUT: Problems will occur when docker cannot start the software defined by ENTRYPOINT
or CMD and the contaner stops and cannot be started
```

Example 1: (normal mit root)

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 01-privileged.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi1
spec:
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true
```

```
## Schritt 2:
## Ausführen
kubectl apply -f 01-privileged.yml
kubectl exec -it ubsi1 -- bash
## id
```

Example 2: (als nobody: 65534)

```
## Step 1:
## mkdir runtest
## cd runtest
## vi 02-nobody-privileged.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi2
spec:
  securityContext:
    runAsUser: 65534
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true
```

```
## Step 2:
## Execute
kubectl apply -f 01-nobody-privileged.yml
kubectl exec -it ubsi2 -- bash
## id
## touch testfile
## ls -la
```

Example 3: (as 1001 - user does not exist)

```
## Step 1:
## mkdir runtest
## cd runtest
## vi 03-user-1001.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi3
spec:
  securityContext:
    runAsUser: 1001
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true
```

```
## Step 2:
## Execute
kubectl apply -f 03-user-1001.yml
kubectl exec -it ubsi3 -- bash
## id
## touch testfile
## ls -la
```

Example 4: including group

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 04-user-group-1001.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: ubsi4
spec:
  securityContext:
    runAsUser: 1001
    runAsGroup: 1001
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true

```

Kubernetes find unprivileged images

```
https://hub.docker.com/u/nginxinc
https://hub.docker.com/search?q=bitnami
```

Kubernetes Deployment Scenarios

Deployment green/blue,canary,rolling update

Canary Deployment

```
A small group of the user base will see the new application
(e.g. 1000 out of 100.000), all the others will still see the old version

From: a canary was used to test if the air was good in the mine
(like a test balloon)
```

Blue / Green Deployment

```
The current version is the Blue one
The new version is the Green one

New Version (GREEN) will be tested and if it works
the traffic will be switch completey to the new version (GREEN)

Old version can either be deleted or will function as fallback
```

A/B Deployment/Testing

```
2 Different versions are online, e.g. to test a new design / new feature
You can configure the weight (how much traffic to one or the other)
by the number of pods
```

Example Calculation

```
e.g. Deployment1: 10 pods
Deployment2: 5 pods

Both have a common label,
The service will access them through this label
```

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/