

Kubernetes Einführung

Agenda

1. Docker-Grundlagen
 - [Übersicht Architektur](#)
 - [Was ist ein Container ?](#)
 - [Was sind container images](#)
 - [Container vs. Virtuelle Maschine](#)
 - [Was ist ein Dockerfile](#)
 - [Dockerfile - image kleinhalten](#)
2. Kubernetes Installation
 - [Kubernetes Installation mit Proxmox und kubespray](#)
3. Kubernetes Workloads
 - [Welche Wege gibt es Kubernetes Workloads auszurollen](#)
4. Kubernetes Infrastructure (Performance)
 - [Performance of etcd for setup](#)
5. Kubernetes - Überblick
 - [Warum Kubernetes, was macht Kubernetes](#)
 - [Aufbau Allgemein](#)
 - [Kubernetes Architektur Deep-Dive](#)
 - [Ausbaustuifen Kubernetes](#)
 - [Wann macht Kubernetes Sinn, wann nicht?](#)
 - [Aufbau mit helm, OpenShift, Rancher\(RKE\), microk8s](#)
 - [Welches System ? \(minikube, micro8ks etc.\)](#)
 - [Installer für grosse Cluster](#)
 - [Installation - Welche Komponenten from scratch](#)
6. Kubernetes - Überblick
 - [Liste wichtiger/sinnvoller Client-Tools](#)
7. Kubernetes - Hochverfügbarkeit
 - [Strategien für Hochverfügbarkeit](#)
8. Kubernetes - Authentication
 - [oidc mit kubectl](#)
 - [traefik authentication mit oidc](#)
9. Kubernetes Deployment - Internals
 - [Strategy when creating and terminating pods in Deployment - RollingUpdate - maxSurge, maxUnavailability](#)
10. kubectl
 - [kubectl einrichten mit namespace](#)
 - [kubectl cheatsheet kubernetes](#)
 - [kubectl mit verschiedenen Clustern arbeiten](#)
11. Kubernetes Ingress (Eingehender Trafik ins Cluster)
 - [Wann LoadBalancer, wann Ingress](#)
12. Kubernetes Praxis API-Objekte
 - [Das Tool kubectl \(Devs/Ops\) - Spickzettel](#)
 - [kubectl example with run](#)
 - [Bauen einer Applikation mit Resource Objekten](#)
 - [Anatomie einer Webanwendungen](#)
 - [kubectl/manifest/pod](#)
 - [ReplicaSets \(Theorie\) - \(Devs/Ops\)](#)
 - [kubectl/manifest/replicaset](#)
 - [Deployments \(Devs/Ops\)](#)
 - [kubectl/manifest/deployments](#)
 - [Debugging](#)
 - [Netzwerkverbindung zum Pod testen](#)
 - [Services \(Devs/Ops\)](#)
 - [kubectl/manifest/service](#)
 - [DaemonSets \(Devs/Ops\)](#)
 - [ConfigMap Example](#)
 - [ConfigMap Example MariaDB](#)
 - [Secrets Example MariaDB](#)
 - [Connect to external database](#)
13. Kubernetes Ingress (Grundlagen)

- [Hintergrund Ingress](#)

14. Kubernetes Ingress (Nginx - deprecated)

- [Ingress Controller auf Digitalocean \(doks\) mit helm installieren](#)
- [Documentation for default ingress nginx](#)
- [Beispiel Ingress](#)
- [Beispiel mit Hostnamen](#)
- [Beispiel Deployment mit Ingress und Hostnamen](#)
- [Achtung: Ingress mit Helm - annotations](#)
- [Permanente Weiterleitung mit Ingress](#)

15. Kubernetes Ingress (Traefik)

- [Install Traefik-IngressController](#)
- [Ingress mit traefik](#)
- [ingress mit traefik, letsencrypt und cert-manager](#)

16. Kubernetes Praxis (Stateful Sets)

- [Hintergrund statefulsets](#)
- [Example stateful set](#)

17. Kubernetes Secrets und Encrypting von z.B. Credentials

- [Kubernetes secrets Typen](#)
- [Sealed Secrets - bitnami](#)
- [Exercise Sealed Secret mariadb](#)
- [registry mit secret auth](#)

18. Kubernetes API-Objekte (Teil 2)

- [Jobs](#)
- [Cronjobs](#)
- [DaemonSet - einfaches Beispiel](#)
- [Daemonset with HostPort](#)
- [Daemonset with HostNetwork](#)

19. Kubernetes Praxis

- [Befehle in pod ausführen - Übung](#)
- [Welche Pods mit Namen gehören zu einem Service](#)

20. Security

- [ServiceLinks nicht in env in Pod einbinden](#)

21. Helm (Kubernetes Paketmanager)

- [Helm - Was kann Helm](#)
- [Helm Grundlagen](#)
- [Helm Warum ?](#)
- [Helm Example](#)
- [Installation, Upgrade, Uninstall helm-Chart exercise - simple \(mariadb-cloudpirates\)](#)
- [Helm Exercise with nginx](#)
- [Helm Spickzettel](#)

22. Helm Charts erstellen und analysieren

- [Eigene Helm-Chart erstellen](#)
- [Chart zur Analyse runterladen und entpacken](#)

23. Helm - Fehleranalyse

- [Beispiel Cloudpirates - helm chart nginx](#)

24. Helpful plugins

- [Use shortnames for kubectl - commands](#)

25. Kubernetes Debugging

- [Probleme über Logs identifiziert - z.B. non-root image](#)

26. Weiter lernen

- [Lernumgebung](#)
- [Kubernetes Doku - Bestimmte Tasks lernen](#)
- [Kubernetes Videos mit Hands On](#)

27. Kubernetes Storage (CSI)

- [Überblick Persistant Volumes \(CSI\)](#)
- [Liste der Treiber mit Features \(CSI\)](#)
- [Übung Persistant Storage](#)
- [Beispiel mariadb](#)

28. Kubernetes Installation

- [k3s installation](#)

29. Kubernetes Monitoring

- [Prometheus Monitoring Server \(Overview\)](#)
- [Prometheus / Grafana Stack installieren](#)

30. Kubernetes QoS / HealthChecks / Live / Readiness

- [Quality of Service - evict pods](#)
- [LiveNess/Readiness - Probe / HealthChecks](#)
- [Taints / Tolerations](#)

31. Installation mit microk8s

- [Schritt 1: auf 3 Maschinen mit Ubuntu 24.04 LTS](#)
- [Schritt 2: cluster - node2 + node3 einbinden - master ist node 1](#)
- [Schritt 3: Remote Verbindung einrichten](#)

32. Installation mit kubeadm

- [Schritt für Schritt mit kubeadm](#)

33. Tipps & Tricks

- [Pods bleiben im terminate-mode stehen](#)

Backlog

1. Podman

- [Podman vs. Docker](#)

2. ServiceMesh

- [Why a ServiceMesh ?](#)
- [How does a ServiceMesh work? \(example istio\)](#)
- [istio vs. ingress](#)
- [istio security features](#)
- [istio-service mesh - ambient mode](#)
- [Performance comparison - baseline, sidecar, ambient](#)

3. Kubernetes Ingress

- [Ingress HA-Proxy Sticky Session](#)
- [Nginx Ingress Session Stickyness](#)
- [https mit ingressController und Letsencrypt](#)

4. Kubernetes Pod Termination

- [LifeCycle Termination](#)
- [preStopHook](#)
- [How to wait till a pod gets terminated](#)

5. Kubernetes Security

- [Best practices security, pods](#)
- [Best practices in general](#)
- [Images in kubernetes von privatem Repo verwenden](#)

6. Kubernetes Monitoring/Security

- [Überwachung, ob Images veraltet sind, direkt in Kubernetes](#)

7. Helm (IDE - Support)

- [Kubernetes-Plugin IntelliJ](#)
- [IntelliJ - Helm Support Through Kubernetes Plugin](#)

8. Helm - Charts entwickeln

- [Unser erstes Helm Chart erstellen](#)
- [Wie starte ich am besten - Übung](#)

9. Helm und Kustomize kombinieren

- [Helm und Kustomize kombinieren](#)

10. LoadBalancer on Premise (metallb)

- [Metallb](#)

11. Helm mit gitlab ci/cd

- [Helm mit gitlab ci/cd ausrollen](#)

12. Kubernetes Verlässlichkeit erreichen

- [Keine 2 pods auf gleichem Node - PodAntiAffinity](#)

13. Metrics-Server / GröÙe Cluster

- [Metrics-Server mit helm installieren](#)
- [Speichernutzung und CPU berechnen für Anwendungen](#)

14. Kubernetes -> High Availability Cluster (multi-data center)

- [High Availability multiple data-centers](#)
- [PodAntiAffinity für Hochverfügbarkeit](#)
- [PodAffinity](#)

15. Kubernetes -> etcd

- [etcd - cleaning of events](#)
- [etcd in multi-data-center setup](#)

16. Kubernetes Storage

- [Praxis. Beispiel \(Dev/Ops\)](#)

17. Kubernetes Netzwerk

- [Kubernetes Netzwerke Übersicht](#)
- [DNS - Resolution - Services](#)
- [Kubernetes Firewall / Cilium Calico](#)
- [Sammlung istio/mesh](#)

18. Kubernetes NetworkPolicy (Firewall)

- [Kubernetes Network Policy Beispiel](#)

19. Kubernetes Autoscaling

- [Kubernetes Autoscaling](#)

20. Kubernetes Secrets / ConfigMap

- [Configmap Example 1](#)
- [Secrets Example 1](#)
- [Änderung in ConfigMap erkennen und anwenden](#)

21. Kubernetes RBAC (Role based access control)

- [RBAC Übung kubectl](#)

22. Kubernetes Operator Konzept

- [Ueberblick](#)

23. Kubernetes Deployment Strategies

- [Deployment green/blue.canary.rolling update](#)
- [Praxis-Übung A/B Deployment](#)

24. Kubernetes Monitoring

- [Prometheus / blackbox exporter](#)
- [Kubernetes Metrics Server verwenden](#)

25. Tipps & Tricks

- [Netzwerkverbindung zum Pod testen](#)
- [Debug Container neben Container erstellen](#)
- [Debug Pod auf Node erstellen](#)

26. Kubernetes Administration /Upgrades

- [Kubernetes Administration / Upgrades](#)
- [Terminierung von Container vermeiden](#)
- [Praktische Umsetzung RBAC anhand eines Beispiels \(Ops\)](#)

27. Documentation (Use Cases)

- [Case Studies Kubernetes](#)
- [Use Cases](#)

28. Interna von Kubernetes

- [OCI Container Images Standards](#)

29. Andere Systeme / Verschiedenes

- [Kubernetes vs. Cloudfoundry](#)
- [Kubernetes Alternativen](#)
- [Hyperscaler vs. Kubernetes on Premise](#)

30. Lokal Kubernetes verwenden

- [Kubernetes in ubuntu installieren z.B. innerhalb virtualbox](#)
- [minikube](#)
- [rancher for desktop](#)

31. Microservices

- [Microservices vs. Monolith](#)
- [Monolith schneiden/aufteilen](#)
- [Strategic Patterns - wie monolith praktisch umbauen](#)
- [Literatur von Monolith zu Microservices](#)

32. Extras

- [Install minikube on wsl2](#)
- [kustomize - gute Struktur für größere Projekte](#)
- [kustomize with helm](#)

33. Documentation

- [References](#)
- [Tasks Documentation - Good one !](#)

34. AWS

- [ECS \(managed containers\) vs. Kubernetes](#)

35. Documentation for Settings right resources/limits

- [Goldilocks](#)

Backlog

1. Kubernetes - Überblick

- [Allgemeine Einführung in Container \(Devs/Ops\)](#)
- [Microservices \(Warum ? Wie ?\) \(Devs/Ops\)](#)
- [Wann macht Kubernetes Sinn, wann nicht?](#)
- [Aufbau Allgemein](#)
- [Aufbau mit helm, OpenShift, Rancher\(RKE\), microk8s](#)
- [Welches System ? \(minikube, micro8ks etc.\)](#)
- [Installation - Welche Komponenten from scratch](#)

2. Kubernetes - microk8s (Installation und Management)

- [Installation Ubuntu - snap](#)
- [Remote-Verbindung zu Kubernetes \(microk8s\) einrichten](#)
- [Create a cluster with microk8s](#)
- [Ingress controller in microk8s aktivieren](#)
- [Arbeiten mit der Registry](#)
- [Installation Kubernetes Dashboard](#)

3. Kubernetes Praxis API-Objekte

- [Das Tool kubectl \(Devs/Ops\) - Spickzettel](#)
- [kubectl example with run](#)
- Arbeiten mit manifests (Devs/Ops)
- Pods (Devs/Ops)
- [kubectl/manifest/pod](#)
- ReplicaSets (Theorie) - (Devs/Ops)
- [kubectl/manifest/replicaset](#)
- Deployments (Devs/Ops)
- [kubectl/manifest/deployments](#)
- Services (Devs/Ops)
- [kubectl/manifest/service](#)
- DaemonSets (Devs/Ops)
- IngressController (Devs/Ops)
- [Hintergrund Ingress](#)
- [Documentation for default ingress nginx](#)
- [Beispiel Ingress](#)
- [Beispiel mit Hostnamen](#)
- [Achtung: Ingress mit Helm - annotations](#)
- [Permanente Weiterleitung mit Ingress](#)
- [ConfigMap Example](#)

4. Kubernetes - ENV - Variablen für den Container setzen

- [ENV - Variablen - Übung](#)

5. Kubernetes - Arbeiten mit einer lokalen Registry (microk8s)

- [microk8s lokale Registry](#)

6. Kubernetes Praxis Scaling/Rolling Updates/Wartung

- Rolling Updates (Devs/Ops)
- Scaling von Deployments (Devs/Ops)
- [Wartung mit drain / uncordon \(Ops\)](#)
- [Ausblick AutoScaling \(Ops\)](#)

7. Kubernetes Storage

- Grundlagen (Dev/Ops)
- Objekte PersistentVolume / PersistentVolumeClaim (Dev/Ops)
- [Praxis. Beispiel \(Dev/Ops\)](#)

8. Kubernetes Networking

- [Überblick](#)
- Pod to Pod
- Webbasierte Dienste (Ingress)
- IP per Pod
- Inter Pod Communication ClusterDNS
- [Beispiel NetworkPolicies](#)

9. Kubernetes Paketmanagement (Helm)

- [Warum ? \(Dev/Ops\)](#)
- [Grundlagen / Aufbau / Verwendung \(Dev/Ops\)](#)
- [Praktisches Beispiel bitnami/mysql \(Dev/Ops\)](#)

10. Kustomize

- [Beispiel ConfigMap - Generator](#)
- [Beispiel Overlay und Patching](#)
- [Resources](#)

11. Kubernetes Rechteverwaltung (RBAC)

- Warum ? (Ops)
- [Wie aktivieren?](#)
- Rollen und Rollenzuordnung (Ops)
- Service Accounts (Ops)
- [Praktische Umsetzung anhand eines Beispiels \(Ops\)](#)

12. Kubernetes Backups

- [Kubernetes Backup](#)
- [Kasten.io overview](#)

13. Kubernetes Monitoring

- [Debugging von Ingress](#)
- [Ebenen des Loggings](#)
- [Working with kubectl logs](#)
- [Built-In Monitoring tools - kubectl top pods/nodes](#)
- [Protokollieren mit Elasticsearch und Fluentd \(Devs/Ops\)](#)
- [Long Installation step-by-step - DigitalOcean](#)
- Container Level Monitoring (Devs/Ops)
- [Setting up metrics-server - microk8s](#)

14. Kubernetes Security

- [Grundlagen und Beispiel \(Praktisch\)](#)

15. Kubernetes GUI

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

16. Kubernetes CI/CD (Optional)

- Canary Deployment (Devs/Ops)
- Blue Green Deployment (Devs/Ops)

17. Tipps & Tricks

- [Ubuntu client aufsetzen](#)
- [bash-completion](#)
- [Alias in Linux kubectl get -o wide](#)
- [vim einrückung für yaml-dateien](#)
- [kubectl spickzettel](#)
- [alte manifests migrieren](#)
- [X-Forward-Header-For setzen in Ingress](#)

18. Übungen

- [Übung Tag 3](#)
- [Übung Tag 4](#)

19. Fragen

- [Q and A](#)
- [Kubernetes und Ansible](#)

20. Documentation

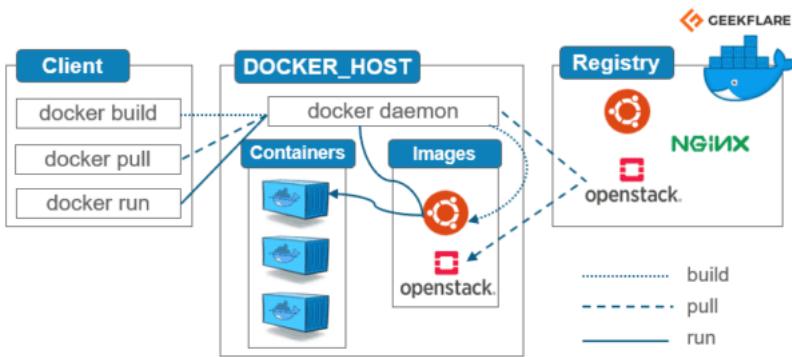
- [Kubernetes mit VisualStudio Code](#)
- [Kube Api Ressources - Versionierungsschema](#)
- [Kubernetes Labels and Selector](#)

21. Documentation - Sources

- [controller manager](#)

Docker-Grundlagen

Übersicht Architektur



Was ist ein Container ?

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

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

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

Was sind container images

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

Container vs. Virtuelle Maschine

VM's virtualisieren Hardware
Container virtualisieren Betriebssystem

Was ist ein Dockerfile

Grundlagen

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

Beispiel

```
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
## Übersetzt: node src/index.js
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Dockerfile - image kleinhalten

- Delete all files that are not needed in image

Example

```

### Delete files needed for installation
### Right after the installation of the necessary
## Variante 2
## nano Dockerfile
FROM ubuntu:22.04
RUN apt-get update && \
    apt-get install -y inetutils-ping && \
    rm -rf /var/lib/apt/lists/*
## CMD ["/bin/bash"]

```

Example 2: Start from scratch

- <https://codeburst.io/docker-from-scratch-2a84552470c8>

Kubernetes Installation

Kubernetes Installation mit Proxmox und kubespray

Schritt 1: virtuellen Maschine deployen

- 4GB (control nodes eher 8GB) - minimaler Arbeitsspeicher
- Debian als Betriebssystem
- minimale Installation
- ssh-server installiert (openssh-server)
- sudo benutzer ohne Passwort für privilege Escalation (z.B. admin darf als root arbeiten)

Info 1.1 Netzwerk

- Alle virtuellen Maschinen im gleichen Netzwerk (kein VLAN)
- Kubernetes mit eigenem VLAN
- Alternativ neuerdings: mit SDN (Performance beobachten !)

Schritt 2: maschine für ansible deployen/nutzen

- private/public key erstellen und den public auf die maschinen aus Schritt 1 verteilen
- ansible und git installieren
- kubespray clonen oder docker image verwenden (dann braucht man kein ansible installieren)
- apt update -y; apt install docker.io -y
- Inventory rauskopieren und anpassen

```

[all]
## Master/Control Plane Node
kube-master ansible_host=192.168.1.10
## Worker Nodes
kube-worker1 ansible_host=192.168.1.11
kube-worker2 ansible_host=192.168.1.12

[kube_control_plane]
kube-master

[etcd]
kube-master

[kube_node]
kube-worker1
kube-worker2

## evtl config anpassen vornehmen in den group vars
## z.B.
https://github.com/kubernetes-sigs/kubespray/blob/master/inventory/sample/group_vars/k8s_cluster/k8s-cluster.yml

```

```

ansible-playbook -i inventory/mycluster/ cluster.yml -b -v \
--private-key=~/ssh/private_key

```

```

## zum hostsystem verbinden und die kubeconfig
## in der Regel
cd /home/<user-mit-dem-ich-installiert-habe>/.kube
cat config

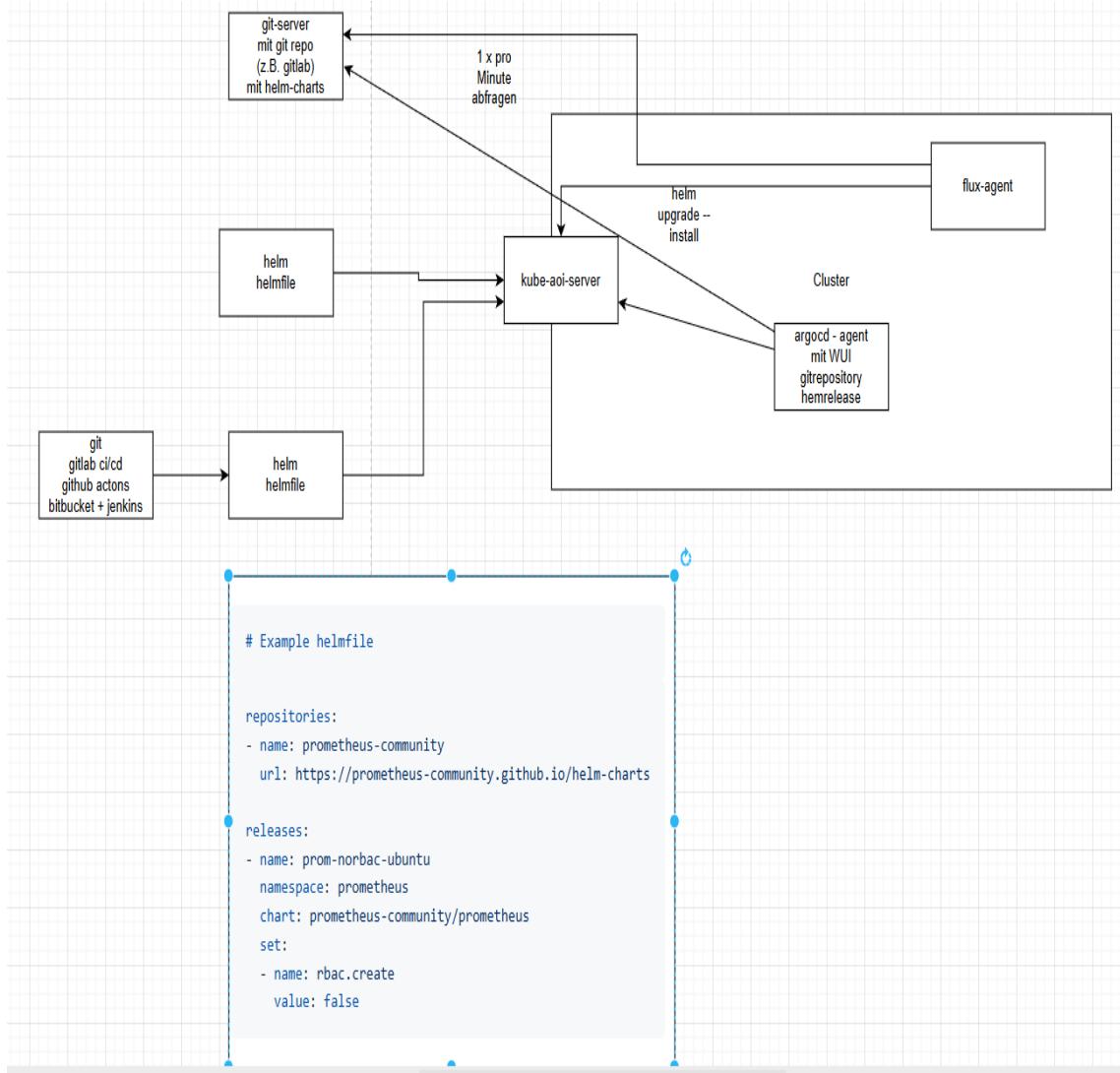
```

Kubernetes Workloads

Welche Wege gibt es Kubernetes Workloads auszurollen

- gitlab ci/cd
- github actions
- bitbucket + jenkins
- ArgoCD
- Flux
- helm/helmfile

Grafik



Kubernetes Infrastructure (Performance)

Performance of etcd for setup

```

etcdctl check perf --load="s"
(s steht für small:

https://andreaskaris.github.io/blog/openshift/etcd_perf/
The performance check's workload model. Accepted workloads: s(small), m(medium), l(large), xl(xLarge)
  
```

Kubernetes - Überblick

Warum Kubernetes, was macht Kubernetes

Ausgangslage

- Ich habe jetzt einen Haufen Images, aber:
 - Wie bekomme ich die auf die Systeme.
 - Und wie halte ich den Verwaltungsaufwand in Grenzen.
- Lösung: Kubernetes -> ein Orchestrierungstool

Hintergründe

- Gegenüber Virtualisierung von Hardware - x-fache bessere Auslastung

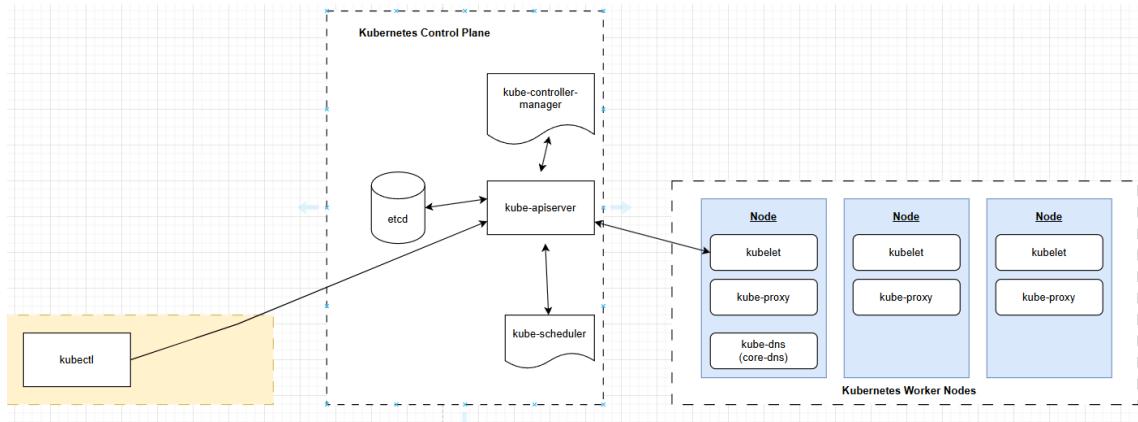
- Google als Ausgangspunkt (Borg)
- Software 2014 als OpenSource zur Verfügung gestellt
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbsterheilend

Wozu dient Kubernetes

- Orchestrierung von Containern
- am gebräuchlichsten aktuell Docker -Images

Aufbau Allgemein

Schaubild



Komponenten / Grundbegriffe

Control Plane (Master)

Aufgaben

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

Komponenten des Masters

ETCD

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

KUBE-CONTROLLER-MANAGER

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

KUBE-API-SERVER

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

KUBE-SCHEDULER

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

Nodes

- Worker Nodes (Knoten) sind die Arbeiter (Maschinen), die Anwendungen ausführen
- Ref: <https://kubernetes.io/de/docs/concepts/architecture/nodes/>

Pod/Pods

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

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

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

Kube-proxy

- Läuft auf jedem Node
- = Netzwerk-Proxy für die Kubernetes-Netzwerk-Services.
- Kube-proxy verwaltet die Netzwerkkommunikation der Services innerhalb des Clusters

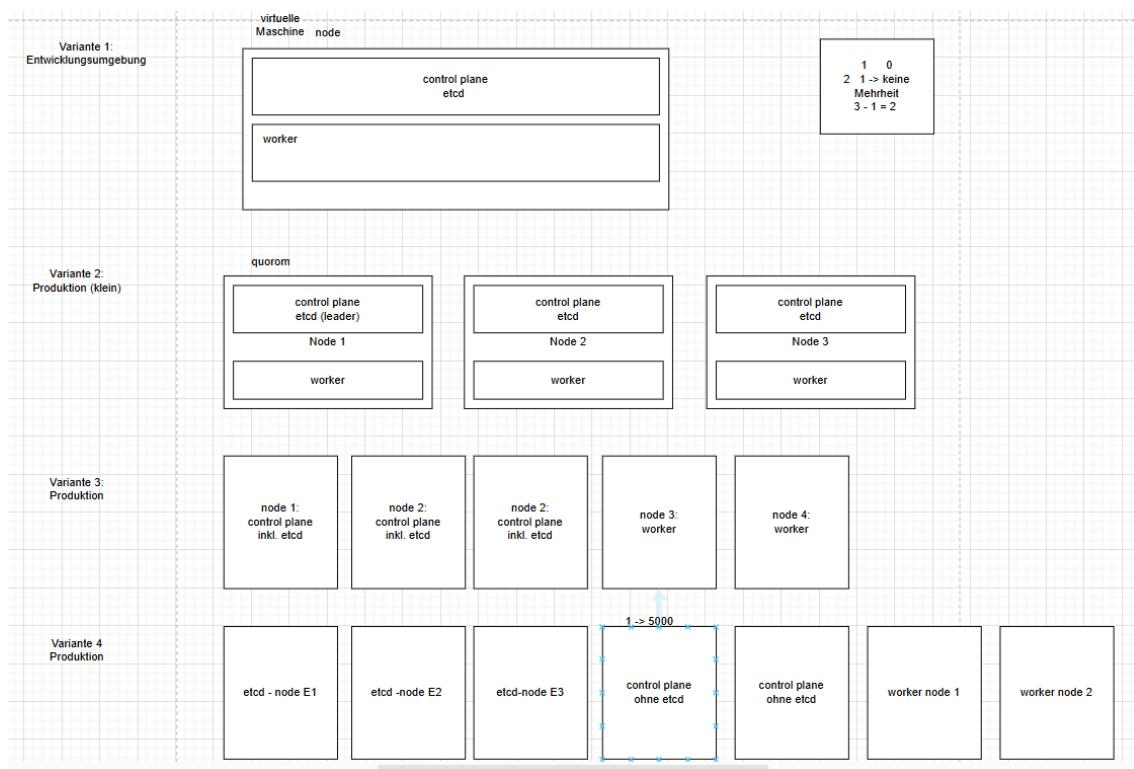
Referenzen

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

Kubernetes Architektur Deep-Dive

- <https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/1ca0d174-f354-43b2-81cc-67af8498b56c>

Ausbaustufen Kubernetes



Wann macht Kubernetes Sinn, wann nicht?

Wann nicht sinnvoll ?

- Anwendung, die ich nicht in Container "verpackt" habe
- Spielt der Dienstleister mit (Wartungsvertrag)
- Kosten / Nutzenverhältnis (Umstellen von Container zu teuer)
- Anwendung lässt sich nicht skalieren
 - z.B. Bottleneck Datenbank
 - Mehr Container bringen nicht mehr (des gleichen Typs)

Wo spielt Kubernetes seine Stärken aus ?

- Skalieren von Anwendungen.
- bessere Hochverfügbarkeit out-of-the-box
- Heilen von Systemen (neu starten von Containern)
- Automatische Überwachung (mit deklarativem Management) - ich beschreibe, was ich will
- Neue Versionen auszurollen (Canary Deployment, Blue/Green Deployment)

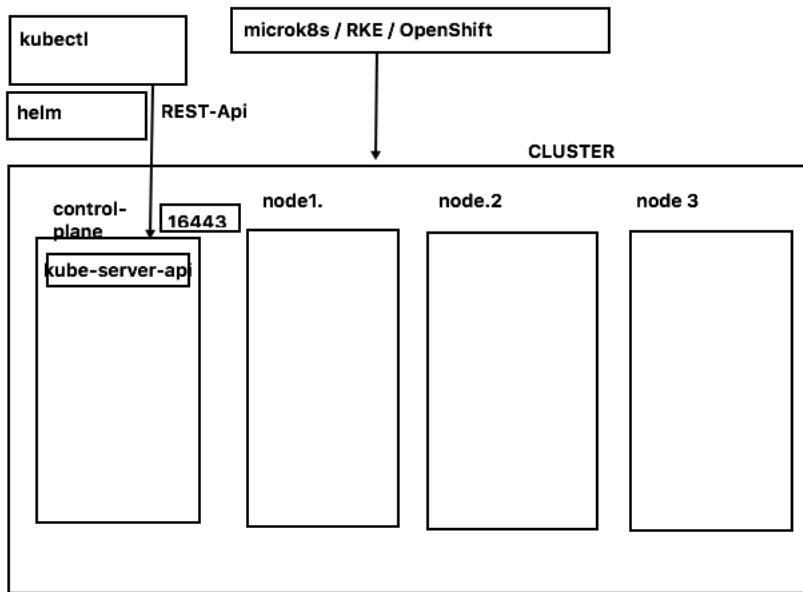
Mögliche Nachteile

- Steigert die Komplexität.
- Debugging wird u.U. schwieriger
- Mit Kubernetes erkaufe ich mir auch, die Notwendigkeit.
 - Über adequate Backup-Lösungen nachzudenken (Moving Target, Kubernetes Aware Backups)
 - Bereitsstellung von Monitoring
 - Bereitsstellung Observability (Log-Aggregationslösung, Tracing)

Klassische Anwendungsfällen (wo Kubernetes von Vorteil)

- Webbasierte Anwendungen (z.B. auch API's bzw. Web)
- Ausser Problematik: Session StickyNess

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



Welches System ? (minikube, micro8ks etc.)

Überblick der Systeme

General

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

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

Kubeadm

General

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

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 (control plane)

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

Disadvantages

- Nicht so flexible wie kubeadm
- z.B. freie Wahl des CNI - Providers (z.B Calico)
- nicht so flexibel bei speziell config (z.B. andere IP-Ranges)

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 on Linux/Windows/Mac
- Supports plugin (Different name ?)

k3s (wsl oder virtuelle Maschine)

- sehr schlank.
- lokal installierbar (eine node, ca 5 minuten)
- ein einziges binary
- <https://docs.k3s.io/quick-start>

kind (Kubernetes-In-Docker)

General

- Runs in docker container

For Production ?

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

Installer für grosse Cluster

Tanzuh (vmware)

- Lizenzkosten

Alternative (Cluster API)

- 1 Management Cluster
- jedes weiteres wird vom Management Cluster ausgerollt.
- Beschreibung Deines Clusters als Konfiguration
- feststehende Images für die Basis des Clusters

Nachteile:

- nur auf der Kommandozeile
- keinen Support

Rancherlabs Ranger (SuSE)

- Grafische Weboberfläche
- kann eine oder mehrere Clusters verwalten

OpenStack (Alternative: vmware) - OpenSource

* API für OpenStack (Nutzung dieser API über Terraform oder OpenTofu) -> Terraform -> Infrastruktur as code. (.tf)

Schritt 1: virtuellen Maschinen ausrollen.

Schritt 2: Kubernetes ausrollen

* Ansible (leichter bestimmte zu Konfigurieren)
* kubeadmin

Proxmox

Schritt 1: virtuellen Maschinen ausrollen.

Schritt 2: Kubernetes ausrollen

* Kubespray (verwendet auch ansible aber direkt auf ansible abgestimmt)
* Ansible (leichter bestimmte zu Konfigurieren)
* kubeadm

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:fea4:cbce/64
## private ip
eth1    UP          10.135.0.3/16 fe80::8081:aaff:fea4:780/64

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

## Funktioniert microk8s
microk8s status

```

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

```

## Was macht das ?
## 1. Basisnutzer (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$4xSfDFQjKWfAoGkZF3LFAXM4hg13d6ATbr2kEu9zMOfwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/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 mikrok8s
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:fed4: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 mikrok8s - 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 verwälten.
## 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
mikrok8s 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 einer 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)
mikrok8s 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
## mikrok8s 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.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
- echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
- chmod 0440 /etc/sudoers.d/11trainingdo

```

Kubernetes - Überblick

Liste wichtiger/sinnvoller Client-Tools

- <https://github.com/metzger/training-kubernetes-einfuehrung/blob/main/tools/liste-client-tools.md>

Kubernetes - Hochverfügbarkeit

Strategien für Hochverfügbarkeit

Kernprinzip: Control Plane benötigt RTT < 10ms zwischen etcd-Nodes für stabilen Quorum-Betrieb. Workloads können über höhere Latenzen hinweg verteilt werden. (Fall 1 - Fall 3)

Legende

RTO

- Recovery Time Objective
- Die maximale akzeptable Zeit, bis ein System nach einem Ausfall wieder verfügbar sein muss.

RPO

- Recovery Point Objective (häufig zusammen genannt)
- Der maximale akzeptable Datenverlust, gemessen in Zeit.

FALL 1: Single-Cluster in einem Rechenzentrum

Ausgangssituation

- 3 Control Plane Nodes (stacked etcd)
- 1+n Worker Nodes
- Alle Nodes im selben RZ

Sinnvoll wenn:

- RTO: 1-5 Minuten (Wie lange bis wieder verfügbar)
- RPO: Nahe Null (bei korrekter Storage-Konfiguration)
- Budget: Gering bis mittel
- Keine regulatorischen Anforderungen für Geo-Redundanz
- Primäres Ziel: Schutz vor Hardware-/Software-Ausfällen

HA-Strategie Control Plane

etcd-Cluster (3 Nodes)

```
## Unbedingt odd number: 3 oder 5 nodes
## 3 Nodes tolerieren 1 Ausfall
## 5 Nodes tolerieren 2 Ausfälle
```

- Separate Hosts für etcd (empfohlen) oder stacked etcd
- Separate Disks mit niedriger Latenz (NVMe SSD)
- Monitoring: etcd-Metriken (Leader Elections, DB Size)
- Backup-Strategie: Regelmäßige etcd-Snapshots

API Server Load Balancing

- Redundante kube-apiserver auf allen Control Plane Nodes
- Virtual IP ((z.B kube-vip) und/oder Hardware Load Balancer vor API Servers
- Tools: kube-vip, HAProxy, keepalived

Scheduler & Controller Manager

- Laufen auf allen Control Plane Nodes
- Leader Election über Leases in kube-system
- Automatisches Failover bei Node-Ausfall

HA-Strategie Workload

```
## In diesem Fall Kann-Option
## Wenn eine Node wegbricht, wird ohnehin auf eine andere Node verteilt
```

Pod-Replikation

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3 # Minimum für HA
  template:
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: kubernetes.io/hostname
          whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            app: web-app
```

- maxSkew = 1: Unterschied der einzelnen Pods auf Nodes darf maximal 1 sein, d.h. z.B. node1: 1, node2: 2,
- topologyKey: Verteilung über Nodes

PodDisruptionBudgets (PDB)

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: web-app-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: web-app
```

- Es müssen insgesamt mindestens noch 2 pods laufen

Anti-Affinity Rules

- Pods auf unterschiedliche Worker Nodes verteilen
- `requiredDuringSchedulingIgnoredDuringExecution` für strikte Trennung

Storage HA

Lokale Storage-Lösungen:

- Rook-Ceph: Distributed Storage mit Replikation
- Longhorn: Leichtgewichtige Alternative (schlechte Performance)
- OpenEBS: Verschiedene Storage Engines (nur readwriteonce)

Externe Storage:

- NFS mit HA (z.B. über DRBD)
- iSCSI mit Multipathing (nur readwriteonce)
- Cloud-Provider CSI-Driver (bei Managed Kubernetes)

Backup:

- Velero für Cluster-Backups
- Regelmäßige Application-Level Backups
- Externe Backup-Location (S3-kompatibel)

Netzwerk HA

CNI-Redundanz:

- Calico, Cilium oder Flannel mit HA-Konfiguration
- Redundante Netzwerk-Interfaces auf Nodes

Load Balancer:

- MetalLB für Bare-Metal (Layer 2 oder BGP-Mode)
- Cloud-Provider Load Balancer (bei Managed Services)

Ingress HA:

```
## Nginx Ingress mit mindestens 2 Replicas
helm install ingress-nginx ingress-nginx/ingress-nginx \
--set controller.replicaCount=3 \
--set controller.service.externalTrafficPolicy=Local
```

Single Points of Failure (SPOFs)

Komponente	Risiko	Mitigation
Load Balancer vor API Server	Hoch	kube-vip mit VRRP oder redundante Hardware-LB
Storage Backend	Hoch	Distributed Storage (Ceph, Longhorn)
Gesamtes RZ	Hoch	Nur durch Multi-RZ-Setup adressierbar

Tools & Komponenten

- **kube-vip**: VIP für API Server (Layer 2 oder BGP)
- **HAProxy/keepalived**: Alternative zu kube-vip
- **MetalLB**: Load Balancer für Bare-Metal
- **Velero**: Backup und Restore
- **Prometheus + Alertmanager**: Monitoring
- **etcdctl**: etcd-Management und Backups

Vor-/Nachteile

Vorteile:

- Einfachste HA-Lösung
- Niedrige Latenz innerhalb Cluster
- Geringste Komplexität
- Moderate Kosten

Nachteile:

- Kein Schutz vor RZ-Ausfall
- Limitierte geografische Redundanz

Kostenabschätzung

- **Relativ**: Basis (1x)
- **Absolute Kosten**: 3 Control Plane + 3+ Worker Nodes + Storage

RTO/RPO

- **RTO**: 1-5 Minuten (bei automatischem Pod-Failover)
- **RPO**: < 1 Minute (bei korrekter Storage-Replikation)

Quellen

- [Kubernetes HA Topologies](#)
- [etcd Administration Guide](#)
- [kube-vip Documentation](#)

FALL 2: Multi-Zone Cluster (RTT < 10ms)

Ausgangssituation

- 3-5 Control Plane Nodes über Availability Zones verteilt
- Worker Nodes in allen Zones
- RTT zwischen Zones: < 10ms (typisch < 2ms innerhalb Region)
- Cloud-Provider oder Metro-RZ mit dedizierten Zonen

Sinnvoll wenn:

- RTO: < 1 Minute
- RPO: Nahe Null
- Budget: Mittel
- Cloud-Native Deployment (AWS, GCP, Azure)
- Schutz gegen Zone-Ausfall erforderlich

- Keine strengen Latenzanforderungen zwischen Zones

HA-Strategie Control Plane

etcd über Zones verteilt

- **Kritisch:** RTT muss < 10ms bleiben für etcd-Quorum
- Mindestens 3 Zones mit jeweils 1 Control Plane Node
- Bei 5 Nodes: 2-2-1 oder 2-1-2 Distribution

Latenz-Monitoring:

```
## etcd Performance-Check
etcdctl check perf --load="s"
## Warn bei > 10ms Backend Commit Duration
```

HA-Strategie Workload

Topology Spread Constraints

```
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 6
  template:
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: topology.kubernetes.io/zone
          whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            app: web-app
        - maxSkew: 2
          topologyKey: kubernetes.io/hostname
          whenUnsatisfiable: ScheduleAnyway
```

Zone-Aware PodDisruptionBudgets

```
apiVersion: policy/v1
kind: PodDisruptionBudget
spec:
  minAvailable: 4 # Bei 6 Replicas über 3 Zones
  selector:
    matchLabels:
      app: web-app
  unhealthyPodEvictionPolicy: AlwaysAllow
```

Storage HA

Zone-Aware Storage Classes

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: zone-redundant
provisioner: kubernetes.io/aws-ebs # Beispiel AWS
parameters:
  type: gp3
  iops: "3000"
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
    - key: topology.kubernetes.io/zone
      values:
        - eu-central-1a
        - eu-central-1b
        - eu-central-1c
```

Replizierte Storage-Lösungen:

- Rook-Ceph mit Zone-Awareness
- Portworx mit Zone-Replication
- Cloud-Provider replizierte Volumes

Cloud-Provider Integration

AWS:

- EKS mit Multi-AZ Control Plane (Managed)

- EBS Multi-Attach für Shared Storage
- ELB/ALB automatisch über Zones verteilt

GCP:

- GKE Regional Clusters (Multi-Zonal Control Plane)
- Regional Persistent Disks
- Cloud Load Balancer über Zones

Azure:

- AKS mit Availability Zones
- Zone-Redundant Storage (ZRS)
- Azure Load Balancer Standard (Zone-Redundant)

Tools & Komponenten

- **Cloud-Provider CCM:** Automatische Zone-Awareness
- **CSI-Driver:** Zone-Aware Storage Provisioning
- **Cluster Autoscaler:** Zone-Aware Scaling
- **Velero:** Multi-Zone Backups

Vor-/Nachteile

Vorteile:

- Schutz gegen Zone-Ausfall
- Automatisches Failover
- Native Cloud-Integration
- Transparenz für Workloads

Nachteile:

- Höhere Cloud-Kosten (Cross-Zone Traffic)
- Potenzielle Latenz zwischen Zones
- Abhängigkeit von Cloud-Provider

Kostenabschätzung

- **Relativ:** 1.5-2x (wegen Cross-Zone Traffic)
- **Zusatzkosten:** Data Transfer zwischen Zones (~0.01-0.02 €/GB)

RTO/RPO

- **RTO:** < 1 Minute (automatisches Failover)
- **RPO:** 0 (synchrone Replikation)

Quellen

- [Kubernetes Zone-Aware Scheduling](#)
- [AWS EKS Best Practices](#)
- [GCP GKE Multi-Zonal Clusters](#)

FALL 3: Stretched Cluster über RZ (RTT < 10ms)

Ausgangssituation

- 2-3 physisch getrennte Rechenzentren
- RTT zwischen RZ: < 10ms (Metro-Cluster Szenario)
- Dedierte Glasfaser-Verbindung zwischen RZ
- Geografische Distanz: typisch < 50-100 km

Sinnvoll wenn:

- RTO: < 1 Minute
- RPO: Nahe Null
- Budget: Hoch
- Regulatorische Anforderungen für Geo-Redundanz
- Synchrone Daten-Replikation erforderlich
- Sehr niedrige Latenz zwischen RZ verfügbar

HA-Strategie Control Plane

Stretched etcd-Cluster mit Witness

```

RZ1: 2 etcd Nodes
RZ2: 2 etcd Nodes
RZ3 (optional): 1 Witness Node

Total: 5 Nodes für 2-Node Ausfall-Toleranz
  
```

Kritische Anforderungen:

- RTT zwischen allen etcd-Nodes: < 10ms
- Stabile, dedizierte Netzwerk-Verbindung
- Monitoring für Network Partitions

Split-Brain Prevention:

- Odd number of etcd nodes

- Optional: Witness-Node in drittem Standort
- Fencing-Mechanismen auf Storage-Level

HA-Strategie Workload

Site-Aware Scheduling ``yaml apiVersion: v1 kind: Node metadata: labels: topology.kubernetes.io/region: eu-central topology.kubernetes.io/zone: fra1 # RZ1 site: primary

apiVersion: apps/v1 kind: Deployment spec: replicas: 6 template: spec: topologySpreadConstraints: - maxSkew: 1 topologyKey: site whenUnsatisfiable: DoNotSchedule

```
#### Storage-Replikation zwischen RZ

**Synchronre Replikation:***
- **Rook-Ceph** mit stretched cluster mode
  - Min. 5 OSD Nodes (2-2-1 über Sites)
  - Replicated Pools mit site-awareness

- **Portworx** mit DR-License
  - Synchronous Replication zwischen Sites
  - Automatic Failover

- **DRBD** für Block-Storage
  - Kernel-Level Replication
  - Dual-Primary Mode möglich

#### Netzwerk-Design

**Anforderungen:***
- Dedizierte Layer 2/3 Verbindung zwischen RZ
- BGP-Routing für Pod-Network
- Redundante Uplinks (LACP)

**CNI-Empfehlungen:***
- **Calico** mit BGP zwischen Sites
- **Cilium** mit Cluster Mesh (single cluster mode)

#### Risiken & Mitigationen

| Risiko | Impact | Mitigation |
|-----|-----|-----|
| Network Partition | Kritisch | Witness Node, Fencing |
| Latenz-Spikes | Hoch | SLA für Interconnect, Monitoring |
| Split-Brain Storage | Kritisch | Quorum-basierte Systeme |
| Asynchroner etcd | Kritisch | Automatisches Health-Check, Rollback |

#### Tools & Komponenten

- **Rook-Ceph**: Distributed Storage
- **Portworx**: Enterprise Storage mit DR
- **DRBD**: Block-Level Replication
- **Cilium/Calico**: Advanced Networking
- **Disaster Recovery Tools**: Velero mit Multi-Site Backups

#### Vor-/Nachteile

**Vorteile:***
- Transparente RZ-Redundanz
- Ein logischer Cluster
- Niedrige RTT für Workloads
- Automatisches Failover

**Nachteile:***
- Sehr hohe Komplexität
- Abhängigkeit von stabiler, niedriger Latenz
- Risiko für Split-Brain
- Hohe Kosten für Interconnect
- Schwierige Fehlerdiagnose

#### Kostenabschätzung

- **Relativ:** 3-4x (Infrastructure, Interconnect, Storage-Lizenzen)
- **Zusatzkosten:** Dedizierte Glasfaser, HA-Storage-Lizenzen

#### RTO/RPO

- **RTO:** < 1 Minute (bei korrekter Konfiguration)
- **RPO:** 0 (synchrone Replikation)

#### Warnung
```

⚠ **Stretched Clusters sind komplex und fehleranfällig.** Nur bei zwingenden Business-Anforderungen und entsprechender Expertise empfohlen. Multi-Cluster-Setups (Fall 4) sind oft die bessere Wahl.

Quellen

- [etcd Latency Requirements] (<https://etcd.io/docs/v3.5/op-guide/hardware/>)
- [Ceph Stretched Cluster Mode] (<https://docs.ceph.com/en/latest/rados/operations/stretch-mode/>)
- [Portworx Disaster Recovery] (<https://docs.portworx.com/portworx-enterprise/operations/operate-kubernetes/disaster-recovery>)

FALL 4: Multi-Cluster (RTT > 10ms oder unabhängige RZ)

Ausgangssituation

- Separate Kubernetes-Cluster pro RZ/Region
- RTT zwischen RZ: > 10ms (typisch 30-200ms)
- Geografisch verteilte Standorte
- Jeder Cluster ist unabhängig lauffähig

Sinnvoll wenn:

- RTO: 1-10 Minuten (manuell) oder < 1 Minute (automatisch)
- RPO: Sekunden bis Minuten (je nach Replikation)
- Budget: Mittel bis hoch
- Geo-Redundanz über weite Distanzen
- Disaster Recovery Anforderung
- Latenz-Optimierung für User (Edge-Deployment)
- Regulatorische Anforderungen (Data Residency)

Multi-Cluster Architektur-Patterns

4.1 Active-Passive (Cold Standby)

Setup:

- Primary Cluster: Alle Workloads aktiv
- Secondary Cluster: Bereit, aber idle
- DNS-Failover zu Secondary bei Primary-Ausfall

Komponenten:

- **GitOps**: ArgoCD oder Flux auf beiden Clustern
- **Backup**: Velero für Disaster Recovery
- **DNS**: External-DNS oder GSLB (Global Server Load Balancing. Eine DNS-basierte Load-Balancing-Lösung für geografisch verteilte Systeme/Cluster).

Vor-/Nachteile:

- ✓ Einfachste Multi-Cluster Lösung
- ✓ Niedrige Laufkosten
- ✗ RTO: 5-15 Minuten (manuelle Aktivierung)
- ✗ Secondary-Cluster ungenutzt

4.2 Active-Active (Hot Standby)

Setup:

- Traffic auf beide Cluster verteilt (z.B. 50/50 oder Geo-basiert)
- Automatisches Failover bei Cluster-Ausfall

Komponenten:

- **Global Load Balancer**: AWS Route53, Cloudflare, F5 GTM
- **Service Mesh**: Istio Multi-Cluster oder Linkerd
- **Data Replication**: Anwendungsspezifisch oder DB-Level

Traffic Distribution:

```
```yaml
Beispieldokument: External-DNS mit GeoDNS
apiVersion: v1
kind: Service
metadata:
 name: web-app
 annotations:
 external-dns.alpha.kubernetes.io/hostname: app.example.com
 external-dns.alpha.kubernetes.io/aws-geolocation-routing-policy: "eu-central-1"
spec:
 type: LoadBalancer
```

```

Vor-/Nachteile:

- ✓ RTO: < 1 Minute (automatisch)
- ✓ Optimale Resource-Nutzung
- ✓ Geo-Latenz-Optimierung
- ✗ Hohe Komplexität

- ✕ Daten-Konsistenz-Herausforderungen

Multi-Cluster Networking

Submariner

- Submariner ist ein Open-Source-Projekt, das sichere IP-Tunnels zwischen Kubernetes-Clustern erstellt und das Netzwerk im Wesentlichen "abflacht", sodass Pods und Services in verschiedenen Clustern direkt kommunizieren können
- Open-Source Multi-Cluster Networking
- Direct Pod-to-Pod Communication
- Cross-Cluster Service Discovery

```
## Installation
subctl deploy-broker --kubeconfig cluster1-config
subctl join --kubeconfig cluster1-config broker-info.subm
subctl join --kubeconfig cluster2-config broker-info.subm

## Service Export
subctl export service web-app -n production
```

Eigenschaften:

- IPsec/WireGuard Tunnel zwischen Clustern
- Automatisches Route-Advertisement
- Unterstützt unterschiedliche CNIs

Cilium Cluster Mesh

Setup:

```
## Cluster 1
cilium clustermesh enable
cilium clustermesh connect --destination-context cluster2

## Shared Service
kubectl annotate service web-app io.cilium/shared-service="true"
```

Eigenschaften:

- Native Pod-to-Pod Communication ohne Tunnel
- Service Affinity (bevorzuge lokale Pods)
- eBPF-basiert, sehr performant

Multi-Cluster GitOps

ArgoCD Multi-Cluster

Setup:

```
## Cluster registrieren
argocd cluster add cluster1-context
argocd cluster add cluster2-context

## ApplicationSet für beide Cluster
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: web-app
spec:
  generators:
  - clusters:
    selector:
      matchLabels:
        env: production
  template:
    spec:
      source:
        repoURL: https://github.com/org/repo
        path: apps/web-app
      destination:
        name: '{{name}}'
        namespace: production
```

Strategien:

- **Pull-Model:** Jeder Cluster hat eigenes ArgoCD
- **Push-Model:** Zentrales ArgoCD verwaltet alle Cluster

Flux Multi-Cluster

```
## Flux Configuration per Cluster
apiVersion: kustomize.toolkit.fluxcd.io/v1
```

```

kind: Kustomization
metadata:
  name: apps
spec:
  interval: 5m
  path: ./clusters/cluster1
  prune: true
  sourceRef:
    kind: GitRepository
    name: fleet-infra

```

Daten-Replikation Strategien

Anwendungsebene

- Application-Managed Replication (z.B. Kafka Multi-DC)
- Event Sourcing mit Cross-Cluster Event Streams
- CQRS mit regionalen Read-Replicas

Datenbank-Replikation

PostgreSQL:

```

## CrunchyData Postgres Operator mit Standby
apiVersion: postgres-operator.crunchydata.com/v1beta1
kind: PostgresCluster
spec:
  instances:
  - name: instance1
    replicas: 3
    dataVolumeClaimSpec:
      accessModes:
      - ReadWriteOnce
    patroni:
      dynamicConfiguration:
        postgresql:
          parameters:
            wal_level: logical
    standby:
      enabled: true
      repoName: repo1

```

MariaDB:

- (Galera Cluster für synchrone Multi-Master) - performance Probleme bei zu hoher RTT

NoSQL:

- MongoDB Replica Sets über Regions
- Cassandra Multi-DC Replication
- Redis Sentinel mit Cross-DC Replication

Global Load Balancing (GSLB)

Cloudflare Load Balancer

```
{
  "name": "app.example.com",
  "default_pools": ["eu-central-pool", "us-east-pool"],
  "region_pools": {
    "WEUR": ["eu-central-pool"],
    "EEU": ["eu-central-pool"],
    "NAM": ["us-east-pool"]
  },
  "steering_policy": "geo"
}
```

AWS Route53

```

## Terraform Beispiel
resource "aws_route53_record" "app" {
  zone_id = var.zone_id
  name    = "app.example.com"
  type    = "A"

  geolocation_routing_policy {
    continent = "EU"
  }

  alias {
    name           = aws_lb.eu_cluster.dns_name
    zone_id       = aws_lb.eu_cluster.zone_id
  }
}

```

```

        evaluate_target_health = true
    }
}

```

Multi-Cluster Monitoring

Prometheus Federation

```

## Zentrale Prometheus-Instanz
scrape_configs:
- job_name: 'federate-cluster1'
  honor_labels: true
  metrics_path: '/federate'
  params:
    'match[]':
      - '{job="kubernetes-pods"}'
  static_configs:
  - targets:
    - 'prometheus-cluster1.monitoring:9090'
    labels:
      cluster: cluster1

```

Thanos für Multi-Cluster

- Zentrale Object Storage für Metriken
- Global Query Layer
- Cross-Cluster Alerting

Grafana Multi-Cluster Dashboards

- Cluster-Selector per Variable
- Aggregierte Ansichten über Cluster
- Geo-Maps für Traffic-Distribution

Failover-Strategien

DNS-basiert

```

## External-DNS mit Healthcheck
apiVersion: v1
kind: Service
metadata:
  annotations:
    external-dns.alpha.kubernetes.io/hostname: app.example.com
    external-dns.alpha.kubernetes.io/set-identifier: cluster1
    external-dns.alpha.kubernetes.io/aws-weight: "100"
    external-dns.alpha.kubernetes.io/aws-health-check-id: <id>

```

Eigenschaften:

- RTO: TTL-abhängig (typisch 60-300s)
- Einfach zu implementieren
- Keine zusätzliche Infrastruktur

Application-Level Failover

- Circuit Breaker in Service Mesh
- Retry-Logic mit Fallback-Cluster
- Client-Side Load Balancing

Tools & Komponenten Übersicht

| Kategorie | Tool | Zweck |
|--------------------|----------------------|------------------------------|
| Cluster Management | Rancher | Multi-Cluster UI, Management |
| | KubeFed (deprecated) | Cluster Federation (Legacy) |
| Networking | Submariner | Pod-to-Pod Communication |
| | Cilium Cluster Mesh | eBPF Multi-Cluster |
| GitOps | ArgoCD | Multi-Cluster Deployments |
| | Flux | GitOps Engine |
| Load Balancing | Cloudflare | GSLB, DDoS Protection |
| | F5 GTM | Enterprise GSLB |
| Monitoring | AWS Route53 | Geo-Routing, Healthchecks |
| | Thanos | Multi-Cluster Prometheus |
| | Grafana | Unified Dashboards |

| Disaster Recovery | Velero | Backup/Restore |
|-------------------|------------|-------------------|
| | Kasten K10 | Enterprise Backup |

Implementierungs-Beispiel: 2-Cluster Active-Active

Architektur:

```

Internet
|
[Cloudflare GSLB]
|
└── EU Cluster (Frankfurt)
    ├── Ingress (Nginx)
    ├── Application Pods (3x)
    └── PostgreSQL (Primary)
|
└── US Cluster (Virginia)
    ├── Ingress (Nginx)
    ├── Application Pods (3x)
    └── PostgreSQL (Read Replica)

```

Deployment:

1. Basis-Setup:

```
## Beide Cluster mit ArgoCD verbinden
argocd cluster add eu-cluster
argocd cluster add us-cluster
```

2. ApplicationSet:

```

apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: web-app
spec:
  generators:
  - clusters:
    selector:
      matchLabels:
        env: production
  template:
    spec:
      project: default
      source:
        repoURL: https://github.com/org/apps
        path: web-app/overlays/{{values.cluster}}
        targetRevision: main
      destination:
        name: '{{name}}'
        namespace: production

```

3. Datenbank-Replikation (PostgreSQL):

```
## Primary Cluster (EU)
apiVersion: postgres-operator.crunchydata.com/v1beta1
kind: PostgresCluster
metadata:
  name: web-db
spec:
  postgresVersion: 15
  instances:
  - name: instance1
    replicas: 2

## Replica Cluster (US) - Logical Replication
## Subscription zu EU Cluster
```

4. GSLB (Cloudflare):

```
## Cloudflare Load Balancer mit Geo-Steering
curl -X POST "https://api.cloudflare.com/client/v4/zones/<zone>/load_balancers" \
-H "Authorization: Bearer <token>" \
-d '{
  "name": "app.example.com",
```

```

    "default_pools": ["eu-pool", "us-pool"],
    "region_pools": {
        "WEUR": ["eu-pool"],
        "NAM": ["us-pool"]
    },
    "steering_policy": "geo"
}

```

Vor-/Nachteile

Vorteile:

- Echte Geo-Redundanz (RZ-unabhängig)
- Keine etcd-Latency-Limitierung
- Unabhängige Cluster-Updates
- Flexible Failover-Strategien
- Data Residency Compliance möglich

Nachteile:

- Höchste Komplexität
- Daten-Konsistenz Herausforderungen
- Höhere Betriebskosten
- Mehr Tooling erforderlich
- Manueller Aufwand für Synchronisation

Kostenabschätzung

- **Relativ:** 2-3x pro Cluster (Minimum 2 Cluster)
- **Zusatzkosten:** GSLB, Service Mesh, GitOps-Tooling

RTO/RPO

| Strategie | RTO | RPO |
|--------------------------|----------|------------------|
| Active-Passive (manuell) | 5-15 min | 1-5 min |
| Active-Passive (auto) | 1-5 min | 1-5 min |
| Active-Active | < 1 min | Sekunden-Minuten |

Quellen

- [Kubernetes Multi-Cluster Networking](#)
- [Submariner Documentation](#)
- [Istio Multi-Cluster](#)
- [ArgoCD ApplicationSet](#)
- [Flux Multi-Cluster](#)

FALL 5: Hybrid/Edge Scenarios

Ausgangssituation

- Kombination aus Cloud und On-Premise
- Edge-Locations mit eingeschränkter Connectivity
- IoT/Industrial Use Cases
- Retail Stores, Remote Sites

Sinnvoll wenn:

- Data Sovereignty erforderlich (On-Premise)
- Latenz-kritische Workloads am Edge
- Offline-Fähigkeit erforderlich
- Schrittweise Cloud-Migration
- Budget: Variabel

Lightweight Kubernetes Distributionen

K3s

```

## Master Installation
curl -sfL https://get.k3s.io | sh -

## Agent (Edge) Installation
curl -sfL https://get.k3s.io | K3S_URL=https://master:6443 \
K3S_TOKEN=<token> sh -

```

Eigenschaften:

- Minimaler Footprint (<512 MB RAM)
- SQLite statt etcd (für single-master)
- Integrierter Load Balancer (ServiceLB)
- Ideal für Edge-Deployments

MicroK8s

```
## Installation
snap install microk8s --classic

## HA-Cluster
microk8s add-node
microk8s join <master-ip>:<port>/<token>
```

Eigenschaften:

- Snap-basiert, Auto-Updates
- Add-Ons für DNS, Storage, Ingress
- Gut für Desktop/Dev-Environments

KubeEdge

```
## Cloud-Seite
keadm init --advertise-address=<cloud-ip>

## Edge-Seite
keadm join --cloudcore-ipport=<cloud-ip>:10000 \
--token=<token>
```

Eigenschaften:

- Speziell für IoT/Edge
- Offline-Autonomie
- Lightweight Edge-Runtime
- Cloud-Edge Message Bus

Hybrid-Cluster Management

Rancher

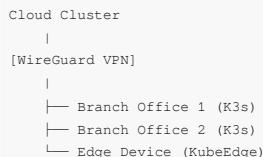
- Zentrale UI für alle Cluster (Cloud + On-Prem)
- Multi-Cluster App Catalog
- RBAC und Policy Management
- Monitoring über Cluster hinweg

VMware Tanzu

- Enterprise Kubernetes für Hybrid Cloud
- Integration mit vSphere
- Consistent Operations überall

Connectivity-Patterns

VPN-Mesh:



Hub-and-Spoke:

- Zentraler Hub-Cluster (Cloud)
- Spoke-Cluster (Edge/Branch)
- Submariner für Pod-Connectivity

Edge-Specific Patterns

Edge-Autonomy:

```
## Application läuft lokal, synchronisiert bei Connectivity
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edge-app
  annotations:
    edge.kubernetes.io/offline-capable: "true"
spec:
  replicas: 1
  template:
    spec:
      nodeSelector:
        edge-location: retail-store-42
```

Data Sync:

- Lokale Datenerfassung am Edge

- Batch-Upload bei Connectivity
- Conflict Resolution bei Sync

Tools & Komponenten

- **K3s**: Lightweight Kubernetes
- **MicroK8s**: Snap-based Kubernetes
- **KubeEdge**: Cloud-Edge Framework
- **Rancher**: Multi-Cluster Management
- **Submariner**: Hybrid Networking
- **Flux/ArgoCD**: GitOps auch für Edge

Vor-/Nachteile

Vorteile:

- Flexible Deployment-Optionen
- On-Premise Data Residency
- Niedrige Latenz am Edge
- Offline-Fähigkeit

Nachteile:

- Management-Komplexität
- Heterogene Infrastruktur
- Connectivity-Abhängigkeit
- Update-Management schwierig

Kostenabschätzung

- **Cloud**: 1-2x Standard-Cluster
- **Edge**: Hardware + K3s (minimal)
- **Gesamt**: Stark use-case abhängig

RTO/RPO

- **Cloud-Edge**: Abhängig von Connectivity
- **Edge-Autonomie**: RTO < 1min (lokal), RPO variabel

Quellen

- [K3s Documentation](#)
- [MicroK8s Documentation](#)
- [KubeEdge Documentation](#)
- [Rancher Documentation](#)

Entscheidungsmatrix

| Kriterium | Fall 1 | Fall 2 | Fall 3 | Fall 4 | Fall 5 |
|--------------------------|----------|----------------|-----------------|--------------|----------|
| RZ-Anzahl | 1 | 1 (Multi-Zone) | 2-3 | 2+ | Hybrid |
| RTT Control Plane | < 1ms | < 10ms | < 10ms | > 10ms | Variabel |
| RTO | 1-5 min | < 1 min | < 1 min | 1-10 min | Variabel |
| RPO | < 1 min | 0 | 0 | Sek-Min | Variabel |
| Komplexität | Niedrig | Mittel | Sehr Hoch | Hoch | Hoch |
| Kosten (relativ) | 1x | 1.5-2x | 3-4x | 2-3x/Cluster | Variabel |
| Geo-Redundanz | ✗ | ✓ (Zones) | ✓ (RZ) | ✓✓ | ✓ |
| Empfehlung | Standard | Cloud-Native | ⚠ Nur bei Zwang | ✓ Empfohlen | Use-Case |

Tooling Comparison

GitOps: ArgoCD vs Flux

| Feature | ArgoCD | Flux |
|-----------------------|-----------------|----------------------------------|
| UI | ✓ Web UI | ✗ CLI only (FluxCD UI 3rd party) |
| Multi-Cluster | ✓ Native | ✓ Multi-tenancy |
| RBAC | ✓ Integriert | ⚠ Kubernetes RBAC |
| Sync Waves | ✓ Hooks | ✓ Dependencies |
| Helm | ✓ Native | ✓ HelmRelease CRD |
| Kustomize | ✓ | ✓ |
| Image Updates | ⚠ Image Updater | ✓ Image Automation |
| Resource Usage | Höher | Niedriger |

Empfehlung: ArgoCD für Enterprise mit UI-Bedarf, Flux für GitOps-Puristen

Zusammenfassung & Empfehlungen

Quick Decision Tree

```
Start
|
└ Ein RZ ausreichend?
  |   └ JA → Fall 1 (Single-Cluster)
  |
  └ Cloud-Provider mit Zones?
    |   └ JA → Fall 2 (Multi-Zone)
    |
    └ RTT < 10ms zwischen RZ?
      |   └ JA → △ Fall 3 (Stretched) - nur bei Zwang!
      |   └ NEIN → Fall 4 (Multi-Cluster) ✓
      |
      └ Edge/Hybrid?
        └ JA → Fall 5 (Hybrid/Edge)
```

Pragmatische Empfehlungen

1. **Start Simple:** Fall 1 (Single-Cluster) ist für 80% ausreichend
2. **Cloud-Native:** Fall 2 (Multi-Zone) bei Cloud-Deployment
3. **Avoid Stretched Clusters:** Fall 3 nur bei regulatorischen Zwängen
4. **Go Multi-Cluster:** Fall 4 für echte Geo-Redundanz
5. **Edge wenn nötig:** Fall 5 nur für spezifische Use Cases

Typische Fehler vermeiden

- ✗ **Stretched Cluster bei hoher Latenz** → Split-Brain Risiko
- ✓ **Multi-Cluster stattdessen**
- ✗ **Keine PodDisruptionBudgets** → Outage bei Node-Drain
- ✓ **PDBs für alle kritischen Apps**
- ✗ **Single Replica für kritische Services** → No HA
- ✓ **Min. 3 Replicas + Anti-Affinity**
- ✗ **Keine Backup-Tests** → DR funktioniert nicht
- ✓ **Regelmäßige Restore-Drills**
- ✗ **etcd ohne Monitoring** → Unerkannte Performance-Issues
- ✓ **etcd-Metriken + Alerts**

Weiterführende Ressourcen

Offizielle Dokumentation

- [Kubernetes Production Best Practices](#)

Tools & Projekte

- [Awesome Kubernetes](#)
- [Kubernetes Failure Stories](#)

Kubernetes - Authentication

oidc mit kubectl

Voraussetzung (allgemein)

- IDP-Provider muss vorhanden sein

Einrichtung auf dem Kubernetes Server (so -> seit 1.30 (GA 1.32))

Auf dem Server-Host (Control Plane)

```
## Erstellen als
## /etc/kubernetes/auth-config.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AuthenticationConfiguration
jwt:
- issuer:
  url: https://provider1.example.com
  audiences:
  - kubernetes
  claimMappings:
  username:
    claim: email
  groups:
    claim: groups
```

```

- issuer:
  url: https://provider2.example.com
  audiences:
  - k8s-prod
  claimMappings:
  username:
    claim: sub
  groups:
    claim: roles

## API - Flag setzen
## Diese Datei liegt bereits vor und muss angepasst werden
## Static Pod
## /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - --authentication-config=/etc/kubernetes/auth-config.yaml
    volumeMounts:
    - name: auth-config
      mountPath: /etc/kubernetes/auth-config.yaml
      readOnly: true
  volumes:
  - name: auth-config
    hostPath:
      path: /etc/kubernetes/auth-config.yaml

```

- Es handelt sich im einen static - pod, wenn die Datei geändert wurde wird der Pod automatisch neu erstellt

Voraussetzung (Client-Seite)

- krew (Plugin Manager für kubectl muss installiert sein)

```

## Linux/macOS
(
  set -x; cd "$(mktemp -d)" &&
  OS=$(uname | tr '[:upper:]' '[lower:]')" &&
  ARCH=$(uname -m | sed -e 's/x86_64/amd64/' -e 's/\(arm\)\(64\)\?/\1\2/' -e 's/aarch64$/arm64/')" &&
  KREW="krew-$OS_-$ARCH" &&
  curl -fsSLO "https://github.com/kubernetes-sigs/krew/releases/latest/download/$KREW.tar.gz" &&
  tar zxvf "$KREW.tar.gz" &&
  ./"$KREW" install krew
)

## PATH erweitern
export PATH="$KREW_ROOT:$HOME/.krew/bin:$PATH"

```

Aufsetzen auf dem Client

```

## OIDC-Plugin installieren (falls noch nicht vorhanden)
kubectl krew install oidc-login
## Kubeconfig anpassen
kubectl config set-credentials oidc-user \
  --exec-api-version=client.authentication.k8s.io/v1 \
  --exec-command=kubectl \
  --exec-arg=oidc-login \
  --exec-arg=get-token \
  --exec-arg--oidc-issuer-url=https://dein-idp.example.com \
  --exec-arg--oidc-client-id=kubernetes \
  --exec-arg--oidc-client-secret=SECRET

## Das führt zu folgender kubeconfig
users:
- name: oidc-user
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1
      command: kubectl
      args:
      - oidc-login
      - get-token
      - --oidc-issuer-url=https://keycloak.example.com/realm/myrealm
      - --oidc-client-id=kubernetes
      - --oidc-client-secret=SECRET # optional

```

```

## neuen context erstellen
## Cluster-Eintrag muss vorhanden sein
kubectl config set-context oidc-context \
--cluster=dein-cluster \
--user=oidc-user

## Context verwenden
kubectl config use-context oidc-context

## jetzt kannst du ganz normal Befehle verwenden
## z. B.
kubectl get pods

```

Wie funktioniert das ganze ?

```
kubectl get pods
```

2. kubectl prüft Kubeconfig

- Findet OIDC-Config (exec-Plugin: `kubectl oidc-login`)
- **Kein Token vorhanden** → startet Auth-Flow

3. Browser-Login

- Plugin öffnet Browser automatisch
- Du loggst dich bei deinem OIDC-Provider ein (z.B. Keycloak, Google)
- Provider gibt **ID-Token** (JWT) zurück
- Token wird **lokal gecacht** (`~/.kube/cache/oidc-login/`)

4. kubectl → API-Server

```
Authorization: Bearer eyJhbGc... (JWT-Token)
```

Wie ist ein jwt aufgebaut ?

```

3 Teile
1. Header
2. Payload
3. Signature

```

Wie sieht der Payload aus ?

```
{
  "iss": "https://provider1.example.com",
  "aud": "kubernetes",
  "email": "jochen@example.com",
  "groups": ["admins", "developers"]
}
```

traefik authentication mit oidc

1. OAuth2-Proxy Deployment (wird für die Authentifizierung verwendet)

- In go geschrieben
- Besser: Mit helm - Chart ausrollen

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: oauth2-proxy
  namespace: traefik
spec:
  replicas: 1
  selector:
    matchLabels:
      app: oauth2-proxy
  template:
    metadata:
      labels:
        app: oauth2-proxy
    spec:
      containers:
        - name: oauth2-proxy
          image: quay.io/oauth2-proxy/oauth2-proxy:latest
          args:
            - --provider=oidc
            - --oidc-issuer-url=https://keycloak.example.com/realm/myrealm
            - --client-id=traefik

```

```

- --client-secret=your-secret-here
- --cookie-secret=random-32-byte-string
- --email-domain=*
- --upstream=static:/200
- --http-address=0.0.0.0:4180
- --redirect-url=https://auth.example.com/oauth2/callback
ports:
- containerPort: 4180
---
apiVersion: v1
kind: Service
metadata:
  name: oauth2-proxy
  namespace: traefik
spec:
  selector:
    app: oauth2-proxy
  ports:
- port: 4180
  targetPort: 4180

```

2. Traefik Middleware

```

apiVersion: traefik.io/v1alpha1
kind: Middleware
metadata:
  name: oidc-auth
  namespace: default
spec:
  forwardAuth:
    address: http://oauth2-proxy.traefik.svc.cluster.local:4180
    authResponseHeaders:
      - X-Auth-Request-User
      - X-Auth-Request-Email
      - X-Auth-Request-Access-Token

```

3. IngressRoute für OAuth2-Proxy selbst

```

apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: oauth2-proxy
  namespace: default
spec:
  entryPoints:
  - websecure
  routes:
  - match: Host(`auth.example.com`)
    kind: Rule
    services:
    - name: oauth2-proxy
      port: 4180
  tls:
    certResolver: letsencrypt

```

4. Geschützte App mit Middleware

```

apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: protected-app
  namespace: default
spec:
  entryPoints:
  - websecure
  routes:
  - match: Host(`app.example.com`)
    kind: Rule
    middlewares:
    - name: oidc-auth
      namespace: default
    services:
    - name: my-app
      port: 80

```

```
  tls:  
    certResolver: letsencrypt
```

Keycloak Client Setup

In Keycloak (oder anderem OIDC Provider):

- **Client ID:** traefik
- **Valid Redirect URLs:** <https://auth.example.com/oauth2/callback>
- **Access Type:** confidential
- **Standard Flow:** enabled

Cookie-Secret generieren:

```
python -c 'import os,base64; print(base64.urlsafe_b64encode(os.urandom(32)).decode())'
```

Alternativ: Statt eigenen Pod als auth2-middleware möglich direkt in Traefik

- Über die Plugins beim Installieren des Helm-Charts
- <https://artifacthub.io/packages/helm/traefik/traefik>
- Plugin-Catalog: <https://artifacthub.io/packages/helm/traefik/traefik>
- Man kann z.B. verwenden:
 - <https://plugins.traefik.io/plugins/6613338ea28c508f411a44d5/traefik-oidc>

```
experimental.plugins
```

Kubernetes Deployment - Internals

Strategy when creating and terminating pods in Deployment - RollingUpdate - maxSurge, maxUnavailability

Ausgangssituation

- replicas auf 8 eingestellt im Deployment
- änderung auf image, dass es nicht gibt

Welche Werte spielen hier eine Rolle

- deployment.spec.strategy.rollingUpdate.maxUnavailability
- deployment.spec.strategy.rollingUpdate.maxSurge

Wie sind die Standardwerte

- maxSurge: 25%
- maxUnavailability: 25%

```
strategy:  
  rollingUpdate:  
    maxSurge: 25%  
    maxUnavailable: 25%  
  type: RollingUpdate
```

Ablauf. RollingUpdate

Runde 1: Rolling Update legt los

- Ausgangszustand: 8 pods laufen
- Herausfinden, wieviel Pods im alten Replicaset sofort terminiert werden (maxUnavailability)
 - Beispiel: 8 Replicas (25% Unavailability) = 2 Pods -> dürfen sofort terminiert werden
- Herausfinden, wieviele Pods im neuen Replicaset dazu gestartet werden dürfen von vor
 - Insgesamt dürfen replicas: 8 + 25% gestartet werden, d.h. gesamt 10
 - Aktuell laufen 8, also noch 2

Ende von Runde 1:

- 6 Pods im alten Replicaset
- 2 Pods im neuen Replicaset

Runde 1: Nächste Runde

- Ausgangszustand: 6 Pods laufen (von 8 replicas)
- Wieviel dürfen gestoppt werden im alten Replicaset (replicas 8, davon 25% -> 2)
 - Also insgesamt müssen im 6 laufen
 - Es können keinen weiteren terminiert werden
- Wieviel dürfen im neuen Replicaset jetzt noch gestartet werden
 - Insgesamt dürfen replicas: 8 + 25% gestartet werden, d.h. gesamt 10
 - Insgesamt laufen im alten Replicaset 6 und im neuen Replicaset 2, also dürfen noch 2 gestartet werden

Referenz:

- <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/deployment-v1/>

```
strategy.rollingUpdate.maxUnavailability:  
- scaled down to .... pods immediately when the rolling update
```

kubectl

kubectl einrichten mit namespace

config einrichten

```
cd  
mkdir .kube  
cd .kube  
cp /tmp/config config  
ls -la  
## Alternative: nano config befüllen  
## das bekommt ihr aus Eurem Cluster Management Tool
```

```
kubectl cluster-info
```

Arbeitsbereich konfigurieren

```
kubectl create ns <euernname>  
kubectl get ns  
kubectl config set-context --current --namespace <euernname>  
kubectl get pods  
  
## Beispiel  
## kubectl create ns jochen  
## kubectl get ns  
## kubectl config set-context --current --namespace jochen  
## kubectl get pods
```

kubectl cheatsheet kubernetes

- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

kubectl mit verschiedenen Clustern arbeiten

```
### Zwei config in KUBECONFIG env variable  
export KUBECONFIG=~/.kube/config:~/.kube/config.single  
kubectl config view  
cp config config.bkup  
kbuectl config view --flatten > config.yaml  
kubectl config get-contexts  
kubectl config use-context do-frai-single  
kubectl get nodes  
kubectl config use-context do-fra-bka-cluster
```

Kubernetes Ingress (Eingehender Traffik ins Cluster)

Wann LoadBalancer, wann Ingress

Vorteile Load-Balancer

- Ich brauche keine https Termination
- Es ist mir egal wieviel IP-Adresse ich verbrate (weil 1 (eine !) IP pro Service)
- kein http-Dienst sondern tcp stream or grpc

Nachteile sind

- Keine Routing von Pfaden (d.h. <http://meine-domain.de> und <http://meine-domain.de/backend> gehen beide zum gleich Service)
- Kein Auswertung von HTTP-Headern
- Sehr schlechte Stickyess - Features (maximal binding über die IP)

Vorteile von Ingress (gut für http und https)

- Routing von Pfaden (d.h. <http://meine-domain.de> und <http://meine-domain.de/backend> gehen beide zu unterschiedlichen Services)
- Auswertung von HTTP-Headern
- Anbieter mit sehr guter Session Stickyess
- Nur eine IP für alle Dienste (die IP, die Ingress selbst benötigt, um erreichbar zu sein)
- HTTPS Termination

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops) - Spickzettel

Hilfe

```
## Hilfe zu befehl  
kubectl help config  
## Hilfe nächste Ebene  
kubectl config set-context --help
```

Allgemein

```
## Zeige Informationen über das Cluster  
kubectl cluster-info  
  
## Welche Ressourcen / Objekte gibt es, z.B. Pod  
kubectl api-resources  
kubectl api-resources | grep namespaces  
  
## Hilfe zu object und eigenschaften bekommen  
kubectl explain pod  
kubectl explain pod.metadata  
kubectl explain pod.metadata.name
```

namespaces

```
kubectl get ns  
kubectl get namespaces  
  
## namespace wechseln, z.B. nach Ingress  
kubectl config set-context --current --namespace=ingress  
## jetzt werden alle Objekte im Namespace Ingress angezeigt  
kubectl get all,configmaps  
  
## wieder zurückwechseln.  
## der standardmäßige Namespace ist 'default'  
kubectl config set-context --current --namespace=default
```

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  
  
## Recursive Löschen  
cd ~/manifests  
## multiple subfolders subfolders present  
kubectl delete -f . -R
```

Ausgabeformate / Spezielle Informationen

```
## Ausgabe kann in verschiedenen Formaten erfolgen  
kubectl get pods -o wide # weitere informationen  
## im json format  
kubectl get pods -o json  
  
## gilt natürlich auch für andere kommandos  
kubectl get deploy -o json  
kubectl get deploy -o yaml  
  
## Label anzeigen  
kubectl get deploy --show-labels
```

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

## Pods in allen namespaces anzeigen
kubectl get pods -A

## 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
## Löscht alle Pods im eigenen Namespace bzw. Default
kubectl delete pods --all

## Kommando in pod ausführen
kubectl exec -it nginx -- bash

```

Alle Objekte anzeigen

```

## Nur die wichtigsten Objekte werden mit all angezeigt
kubectl get all
## Dies, kann ich wie folgt um weitere ergänzen
kubectl get all,configmaps

## Über alle Namespaces hinweg
kubectl get all -A

```

Logs

```

kubectl logs <container>
kubectl logs <deployment>
## e.g.
## kubectl logs -n namespace8 deploy/nginx
## with timestamp
kubectl logs --timestamps -n namespace8 deploy/nginx
## continuously show output
kubectl logs -f <pod>
## letzten x Zeilen anschauen aus log anschauen
kubectl logs --tail=5 <your pod>

```

Referenz

- <https://kubernetes.io/de/docs/reference/kubectl/cheatsheet/>

kubectl example with run

Example (that does work)

```

## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx:1.23

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

```

Example (that does not work)

```

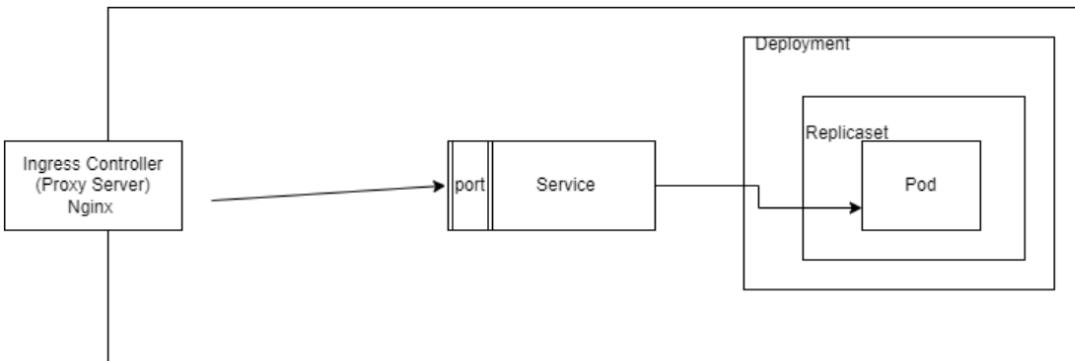
kubectl run testpod --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
## Weitere status - info
kubectl describe pods testpod

```

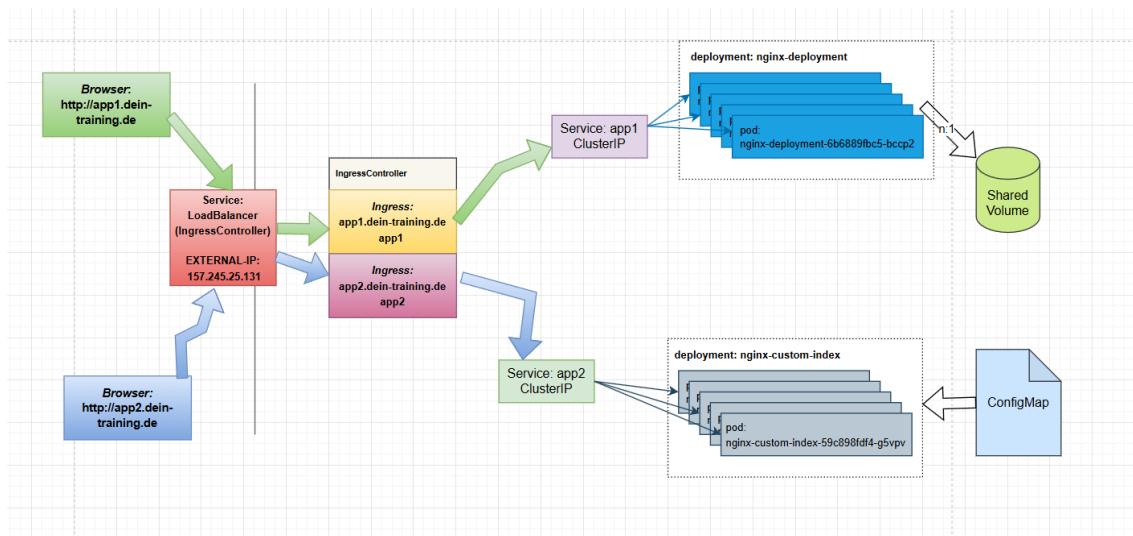
Ref:

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

Bauen einer Applikation mit Resource Objekten



Anatomie einer Webanwendungen



kubectl/manifest/pod

Walkthrough

```

cd
mkdir -p manifests
cd manifests/
mkdir -p 01-web
cd 01-web
nano nginx-static.yml
  
```

```

## vi nginx-static.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-web
  labels:
    webserver: nginx
spec:
  containers:
  - name: web
    image: nginx:1.23
  
```

```

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

```
kubectl get pod/nginx-static-web -o wide  
kubectl describe pod nginx-static-web  
## show config  
kubectl get pod/nginx-static-web -o yaml
```

Aufräumen

```
kubectl delete -f .
```

kubectl/manifest/replicaset

Walkthrough Erstellen

```
cd  
mkdir -p manifests  
cd manifests  
mkdir 02-rs  
cd 02-rs  
nano rs.yaml
```

```
apiVersion: apps/v1  
kind: ReplicaSet  
metadata:  
  name: nginx-replica-set  
spec:  
  replicas: 5  
  selector:  
    matchLabels:  
      tier: frontend  
  template:  
    metadata:  
      name: template-nginx-replica-set  
      labels:  
        tier: frontend  
    spec:  
      containers:  
        - name: nginx  
          image: nginx:1.23  
          ports:  
            - containerPort: 80
```

```
kubectl apply -f .  
kubectl get all  
kubectl get pods -l tier=frontend  
kubectl get pods --show-labels  
## name anpassen  
kubectl describe pod/nginx-replica-set-lpkbs
```

Pod löschen, was passiert

```
## kubectl delete po nginx-r<TAB>  
## einfach einen pod raussuchen und löschen  
## z.B.  
kubectl delete po nginx-replica-set-xg8jp  
  
## gucken, welches sind die neuesten ?  
kubectl get pods
```

Walkthrough Skalieren

```
nano rs.yaml
```

```
## Ändern  
## replicas: 5  
## -> ändern in  
## replicas: 8
```

```
kubectl apply -f .  
kubectl get pods
```

Aufräumen des replicaset

```
kubectl delete -f .
```

kubectl/manifest/deployments

Prepare

```
cd  
mkdir -p manifests  
cd manifests  
mkdir 03-deploy  
cd 03-deploy  
nano nginx-deployment.yml
```

```
## vi nginx-deployment.yml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: nginx-deployment  
spec:  
  selector:  
    matchLabels:  
      app: nginx  
  replicas: 8 # tells deployment to run 8 pods matching the template  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
      - name: nginx  
        image: nginxinc/nginx-unprivileged:1.28  
        ports:  
        - containerPort: 8080
```

```
kubectl apply -f .
```

Explore

```
kubectl get all
```

Optional: Change image - Version

```
nano nginx-deployment.yml
```

Version 1: (optical nicer)

```
## Ändern des images von nginxinc/nginx-unprivileged:1.28 -> auf 1.29  
## danach  
kubectl apply -f . && watch kubectl get pods
```

Version 2:

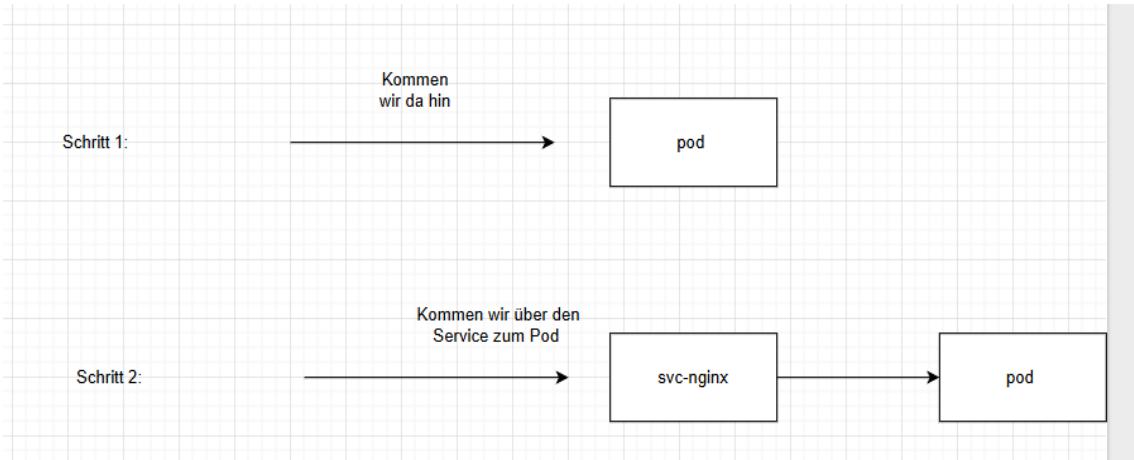
```
## Ändern des images von nginxinc/nginx-unprivileged:1.28 -> auf 1.29  
## danach  
kubectl apply -f .  
kubectl get all  
kubectl get pods -w
```

Netzwerkverbindung zum Pod testen

Situation

```
Managed Cluster und ich kann nicht auf einzelne Nodes per ssh zugreifen
```

Was wollen wir testen (auf der Verbindungsebene) ?



Beispiel: Eigenen Pod starten mit busybox

```
## der einfachste Weg
kubectl run podtest --rm -it --image busybox

## Alternative
kubectl run podtest --rm -it --image busybox -- /bin/sh
```

Example test connection

```
## wget befehl zum Kopieren
ping -c4 10.244.0.99
wget -O - http://10.244.0.99

## -O -> Output (grosses O (buchstabe))
kubectl run podtest --rm -ti --image busybox -- /bin/sh
/ # wget -O - http://10.244.0.99
/ # exit
```

kubectl/manifest/service

Warum Services ?

- Wenn in einem Deployment bei einem Wechsel des images neue Pods erstellt werden, erhalten diese eine neue IP-Adresse
- Nachteil: Man müsste diese dann in allen Applikationen ändern, die auf die Pods zugreifen.
- Lösung: Wir schalten einen Service davor !

Hintergrund IP-Wechsel

```
tln1@client:~/manifests/04-service$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE
web-nginx-69cc9f8d49-9r6wb   1/1     Running   0          5m37s   10.108.0.200   node
-p8s6u         <none>   <none>
web-nginx-69cc9f8d49-z6fxf   1/1     Running   0          5m37s   10.108.0.25    node
-n8s6r         <none>   <none>

• Image-Version wurde jetzt in Deployment geändert, Ergebnis:
```

```
tln1@client:~/manifests/04-service$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
web-nginx-754794c6d6-8pj99   1/1     Running   0          8s     10.108.1.42   node-p
3s68          <none>   <none>
web-nginx-754794c6d6-swckk   1/1     Running   0          10s    10.108.0.121  node-p
266r          <none>   <none>
```

Example I : Service with ClusterIP

Schritt 1: Vorbereitung

```
cd
mkdir -p manifests
```

```
cd manifests  
mkdir 04-service  
cd 04-service
```

Schritt 2: Deployment erstellen

```
nano deploy.yml
```



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

```
nano service.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
spec:
  type: ClusterIP
  ports:
    - port: 80
      protocol: TCP
  selector:
    web: my-nginx
```

```
kubectl apply -f .
## wie ist die ClusterIP ?
kubectl get all
kubectl get svc svc-nginx
## Find endpoints / did svc find pods ?
kubectl describe svc svc-nginx
```

Schritt 3: Deployment löschen

```
kubectl delete -f deploy.yml
## Keine endpunkte mehr
kubectl describe svc svc-nginx
```

Schritt 4: Deployment wieder erstellen

```
kubectl apply -f .
## Endpunkte wieder da
kubectl describe svc svc-nginx
```

Example II : Short version (NodePort)

```
## Wo sind wir ?
## cd; cd manifests/04-service

nano service.yml
## in Zeile type:
## ClusterIP ersetzt durch NodePort

kubectl apply -f .
## NodePort ab 30.000 ausfindig machen
kubectl get svc
```

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|-----------|----------|---------------|-------------|--------------|-----|
| svc-nginx | NodePort | 10.109.24.227 | <none> | 80:32708/TCP | 18h |

```
kubectl get nodes -o wide
```

| NAME | STATUS | ROLES | AGE | VERSION | INTERNAL-IP | EXTERNAL-IP | OS-IMAGE |
|-------------------------|--------|--------|-----|----------------|---------------------|----------------|----------|
| | | | | KERNEL-VERSION | CONTAINER-RUNTIME | | |
| node-p8s68 | Ready | <none> | 25h | v1.33.1 | 10.135.0.72 | 139.59.128.36 | Debian |
| GNU/Linux 12 (bookworm) | | | | 6.1.0-39-amd64 | containerd://1.6.33 | | |
| node-p8s6c | Ready | <none> | 25h | v1.33.1 | 10.135.0.73 | 64.226.125.3 | Debian |
| GNU/Linux 12 (bookworm) | | | | 6.1.0-39-amd64 | containerd://1.6.33 | | |
| node-p8s6u | Ready | <none> | 25h | v1.33.1 | 10.135.0.74 | 159.223.24.231 | Debian |
| GNU/Linux 12 (bookworm) | | | | 6.1.0-39-amd64 | containerd://1.6.33 | | |

```
## im Client Externe NodeIP und NodePort verwenden
curl http://164.92.193.245:30280
```

Example II : Service with NodePort (long version)

```
## you will get port opened on every node in the range 30000+
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-nginx
spec:
  selector:
    matchLabels:
      web: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        web: my-nginx
    spec:
      containers:
        - name: cont-nginx
          image: nginx
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
  labels:
    run: svc-my-nginx
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
  selector:
    web: my-nginx
```

Example III: Service mit LoadBalancer (ExternalIP)

```
nano service.yml
## in Zeile type:
## NodePort ersetzt durch LoadBalancer

kubectl apply -f .
kubectl get svc svc-nginx
kubectl describe svc svc-nginx
kubectl get svc svc-nginx -w
## spätestens nach 5 Minuten bekommen wir eine externe ip
## z.B. 41.32.44.45

curl http://41.32.44.45
```

Example getting a specific ip from loadbalancer (if supported)

```

apiVersion: v1
kind: Service
metadata:
  name: svc-nginx2
spec:
  type: LoadBalancer
  # this line to get a specific ip if supported
  loadBalancerIP: 10.34.12.34
  ports:
    - port: 80
      protocol: TCP
  selector:
    web: my-nginx

```

Ref.

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

ConfigMap Example

Schritt 1: configmap vorbereiten

```

cd
mkdir -p manifests
cd manifests
mkdir configmaptests
cd configmaptests
nano 01-configmap.yml

```

```

## 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-configmap
data:
  # als Wertepaare
  database: mongodb
  database_uri: mongodb://localhost:27017
  testdata: |
    run=true
    file=/hello/you

```

```

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

```

Schritt 2: Beispiel als Datei

```

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
```

Schritt 3: Beispiel. ConfigMap als env-variablen

```
nano 03-pod-mit-env.yml
```

```
## 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 Example MariaDB

Schritt 1: configmap

```
cd
mkdir -p manifests
cd manifests
mkdir cftest
cd cftest
nano 01-configmap.yml
```

```
## 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: mariadb-configmap
data:
  # als Wertepaare
  MARIADB_ROOT_PASSWORD: 11abc432
  TEST_CASE: "47"
```

```
kubectl apply -f .
kubectl describe cm mariadb-configmap
kubectl get cm
kubectl get cm mariadb-configmap -o yaml
```

Schritt 2: Deployment

```
nano 02-deploy.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deployment
spec:
  selector:
    matchLabels:
      app: mariadb
  replicas: 1
```

```

template:
  metadata:
    labels:
      app: mariadb
  spec:
    containers:
      - name: mariadb-cont
        image: mariadb:10.11
        envFrom:
          - configMapRef:
              name: mariadb-configmap

```

```

kubectl apply -f .
kubectl get pods
kubectl exec -it deploy/mariadb-deployment -- bash

```

```

env
env | grep ROOT
env | grep TEST
exit

```

Schritt 3: Service für mariadb

```
nano 03-service.yml
```

```

apiVersion: v1
kind: Service
metadata:
  name: mariadb
spec:
  type: ClusterIP
  ports:
    - port: 3306
      protocol: TCP
  selector:
    app: mariadb

```

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

Schritt 4: Client aufsetzen

```
nano 04-client.yml
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-client
spec:
  selector:
    matchLabels:
      app: ubuntu
  replicas: 1 # tells deployment to run 2 pods matching the template
  template: # create pods using pod definition in this template
    metadata:
      labels:
        app: ubuntu
    spec:
      containers:
        - name: service
          image: ubuntu
          command: [ "/bin/sh" , "-c", "tail -f /dev/null" ]
          envFrom:
            - configMapRef:
                name: mariadb-configmap

```

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

```

## im client
kubectl exec -it deploy/mariadb-client -- bash
apt update; apt install -y mariadb-client iutils-ping

```

Schritt 5: mysql-zugang von aussen erstellen

```
kubectl exec -it deploy/mariadb-deployment -- bash
```

```
mysql -uroot -p$MARIADB_ROOT_PASSWORD

## innerhalb von mysql
create user ext@'%' identified by '11abc432';
grant all on .* to ext@'%';
```

Schritt 6: mysql von client aus testen

```
kubectl exec -it deploy/mariadb-client -- bash

mysql -uext -p$MARIADB_ROOT_PASSWORD -h mariadb

show databases;
```

Important Sidenode

- If configmap changes, deployment does not know
- So kubectl apply -f deploy.yml will not have any effect
- to fix, use stakater/reloader: <https://github.com/stakater/Reloader>

Secrets Example MariaDB

Schritt 1: secret

```
cd
mkdir -p manifests
cd manifests
mkdir secrettest
cd secrettest

kubectl create secret generic mariadb-secret --from-literal=MARIADB_ROOT_PASSWORD=11abc432 --dry-run=client -o yaml > 01-secrets.yaml

kubectl apply -f .
kubectl get secrets
kubectl get secrets mariadb-secret -o yaml
```

Schritt 2: Deployment

```
nano 02-deploy.yml

##deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deployment
spec:
  selector:
    matchLabels:
      app: mariadb
  replicas: 1
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      containers:
        - name: mariadb-cont
          image: mariadb:latest
          envFrom:
            - secretRef:
                name: mariadb-secret

kubectl apply -f .
```

Testing

```
## Führt den Befehl env in einem Pod des Deployments aus
kubectl exec deployment/mariadb-deployment -- env
## eigentlich macht er das:
## kubectl exec mariadb-deployment-c6df6f959-q6swp -- env
```

Important Sidenode

- If configmap changes, deployment does not know

- So kubectl apply -f deploy.yml will not have any effect
- to fix, use stakater/reloader: <https://github.com/stakater/Relo>

Connect to external database

Prerequisites

- MariaDB - Server is running on digitalocean in same network as dokos (kubernetes) - cluster (10.135.0.x)
- DNS-Entry for mariadb-server.t3isp.de -> pointing to private ip: 10.135.0.9

Variante 1:

Schritt 1: Service erstellen

```
cd
mkdir -p manifests
cd manifests
mkdir 05-external-db
cd 05-external-db
nano 01-external-db.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: dbexternal
spec:
  type: ExternalName
  externalName: mariadb-server.t3isp.de
```

```
kubectl apply -f 01-external-db.yml
```

Schritt 2: configmap anlegen oder ergänzen

```
## Ergänzen
## unter data zwei weitere Zeile
### 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: mariadb-configmap
data:
  # als Wertepaare
  MARIADB_ROOT_PASSWORD: 11abc432
  DB_USER: ext
  DB_PASS: 11dortmund22
```

```
kubectl apply -f 01-configmap.yml
```

```
## client deployment gelöscht
kubectl delete -f 04-client.yml
kubectl apply -f 04-client.yml
kubectl exec -it deploy/mariadb-client -- bash
```

```
## Im client
apt update; apt install -y mariadb-client iutils-ping
```

Schritt 3: Service testen

```
kubectl exec -it deploy/mariadb-client -- bash

## im container verbinden mit mysql
mysql -u$DB_USER -p$DB_PASS -h dbexternal

## im verbundenen MySQL-Client
show databases;
```

Variante 2:

```
cd
mkdir -p manifests
cd manifests
mkdir 05-external-db
cd 05-external-db
nano 02-external-endpoint.yml
```

Kubernetes Ingress (Grundlagen)

Hintergrund Ingress

Ref. / Dokumentation

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

Kubernetes Ingress (Nginx - deprecated)

Ingress Controller auf Digitalocean (doks) mit helm installieren

Basics

- Das Verfahren funktioniert auch so auf anderen Plattformen, wenn helm verwendet wird und noch kein IngressController vorhanden
- Ist kein IngressController vorhanden, werden die Ingress-Objekte zwar angelegt, es funktioniert aber nicht.

Prerequisites

- kubectl muss eingerichtet sein
- helm

Walkthrough Simple (Setup Ingress Controller)

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm upgrade --install nginx-ingress ingress-nginx/ingress-nginx --namespace ingress --create-namespace --version 4.13.2

## See when the external ip comes available
kubectl -n ingress get pods
kubectl -n ingress get svc

## 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-inginx

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

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

## vi values.yml
controller:
  publishService:
    enabled: true

## 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 -f values.yml

## 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-inginx

## Now setup wildcard - domain for training purpose
*.app1.t3isp.de A 157.245.20.222
```

Documentation for default ingress nginx

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

Beispiel Ingress

Prerequisites

```
## Ingress Controller muss aktiviert sein
microk8s enable ingress
```

Walkthrough

```
mkdir apple-banana-ingress

## apple.yml
## vi apple.yml
kind: Pod
apiVersion: v1
metadata:
  name: apple-app
  labels:
    app: apple
spec:
  containers:
    - name: apple-app
      image: hashicorp/http-echo
      args:
        - "-text=apple"
---
kind: Service
apiVersion: v1
metadata:
  name: apple-service
spec:
  selector:
    app: apple
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5678 # Default port for image
```

```
kubectl apply -f apple.yml
```

```
## banana
## vi banana.yml
kind: Pod
apiVersion: v1
metadata:
  name: banana-app
  labels:
    app: banana
spec:
  containers:
    - name: banana-app
      image: hashicorp/http-echo
      args:
        - "-text=banana"
---
kind: Service
apiVersion: v1
metadata:
  name: banana-service
spec:
  selector:
    app: banana
  ports:
    - port: 80
      targetPort: 5678 # Default port for image
```

```
kubectl apply -f banana.yml
```

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  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:
  ingressClassName: nginx
  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

```

Beispiel mit Hostnamen

Step 1: Walkthrough

```

cd
cd manifests
mkdir abi
cd abi
nano apple.yml

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

```

```
kubectl apply -f apple.yml
```

```
nano banana.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-<euer-name>"

---

```

```

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

```

```
kubectl apply -f banana.yml
```

Step 2: Testing connection by podIP and Service

```

kubectl get svc
kubectl get pods -o wide
kubectl run podtest --rm -it --image busybox

```

```
/ # wget -O - http://<pod-ip>:5678
/ # wget -O - http://<cluster-ip>
```

Step 3: Walkthrough

```
nano ingress.yml
```

```

## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:

```

```

    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: "<euername>.lab1.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

```

Reference

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

Find the problem

Schritt 1: api-version ändern

```

## Welche Landkarten gibt es ?
kubectl api-versions
## Auf welcher Landkarte ist Ingress jetzt
kubectl explain ingress

```

```

GROUP:      networking.k8s.io
KIND:       Ingress
VERSION:    v1

```

```

## ingress ändern ingress.yml
## von
## apiVersion: extensions/v1beta1
## in
apiVersion: networking.k8s.io/v1

```

Schritt 2: Fehler Eigenschaften beheben

```

Error from server (BadRequest): error when creating "ingress.yml": Ingress in version
 "v1" cannot be handled as a Ingress: strict decoding error: unknown field "spec.rules[0].http.paths[0].backend.serviceName", unknown field "spec.rules[0].http.paths[0].backend.servicePort", unknown field "spec.rules[0].http.paths[1].backend.serviceName", unknown field "spec.rules[0].http.paths[1].backend.servicePort"

```

```

## Problem serviceName beheben
## Was gibt es stattdessen
## -> es gibt service aber keine serviceName
kubectl explain ingress.spec.rules.http.paths.backend
## -> es gibt service.name
kubectl explain ingress.spec.rules.http.paths.backend.

```

```

## Korrektur 2x in ingress.yaml: inkl. servicePort (neu: service.port.number)
## vorher
## backend:
##   serviceName: apple-service
##   servicePort: 80
## jetzt:
##   service:
##     name: apple-service
##     port:
##       number: 80

```

```

kubectl apply -f .

```

Schritt 3: pathType ergänzen

```
* spec.rules[0].http.paths[0].pathType: Required value: pathType must be specified
* spec.rules[0].http.paths[1].pathType: Required value: pathType must be specified
```

- Es wird festgelegt wie der Pfad ausgewertet

```
## Wir müssen pathType auf der 1. Unterebene von paths einfügen
## Entweder exact oder prefix
```

```
http:
  paths:
    - path: /apple
      pathType: Exact
      backend:
        service:
```

```
kubectl apply -f .
```

Schritt 4: Testen aufrufen

```
curl http://<euername>.lab1.t3isp.de/apple
curl http://<euername>.lab1.t3isp.de/apple/ # Sollte nicht funktioniert
curl http://<euername>.lab1.t3isp.de/banana
curl http://<euername>.lab1.t3isp.de/banana/
curl http://<euername>.lab1.t3isp.de/banana/something
```

```
Das kann man auch so im Browser eingeben
```

Solution

```
nano ingress.yml
```

```
## 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:
  ingressClassName: nginx
  rules:
    - host: "<euername>.lab1.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
```

```
kubectl apply -f .
kubectl get ingress example-ingress
## mit describe herausfinden, ob er die services gefundet
kubectl describe ingress example-ingress
```

Beispiel Deployment mit Ingress und Hostnamen

Step 1: Walkthrough

```
cd
cd manifests
mkdir abi
cd abi
```

```
nano apple-deploy.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apple-app
  labels:
    app: apple
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apple
  template:
    metadata:
      labels:
        app: apple
    spec:
      containers:
        - name: apple-app
          image: hashicorp/http-echo
          args:
            - "-text=apple-<euer-name>"
```

```
nano apple-svc.yaml
```

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

```
kubectl apply -f .
```

```
nano banana-deploy.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: banana-app
  labels:
    app: banana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: banana
  template:
    metadata:
      labels:
        app: banana
    spec:
      containers:
        - name: apple-app
          image: hashicorp/http-echo
          args:
            - "-text=banana-<euer-name>"
```

```
nano banana-svc.yaml
```

```
kind: Service
apiVersion: v1
metadata:
  name: banana-service
spec:
  type: ClusterIP
  selector:
    app: banana
```

```

ports:
- port: 80
  targetPort: 5678 # Default port for image

kubectl apply -f .

```

Step 2: Testing connection by podIP and Service

```

kubectl get svc
kubectl get pods -o wide
kubectl run podtest --rm -it --image busybox

/ # wget -O - http://<pod-ip>:5678
/ # wget -O - http://<cluster-ip>

```

Step 3: Walkthrough

```

nano ingress.yml

## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: "<euernname>.lab.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

```

Reference

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

Step 4: Find the problem

Fix 4.1: Fehler: no matches kind "Ingress" in version "extensions/v1beta1"

```

## Gibt es diese Landkarte überhaupt
kubectl api-versions
## auf welcher Landkarte/Gruppe befindet sich Ingress jetzt
kubectl explain ingress
## -> jetzt auf networking.k8s.io/v1

nano ingress.yaml

## auf apiVersion: extensions/v1beta1
## wird -> networking.k8s.io/v1

kubectl apply -f .

```

Fix 4.2: Bad Request unkown field ServiceName / ServicePort

```

## was geht für die Property backend
kubectl explain ingress.spec.rules.http.paths.backend
## und was geht für service
kubectl explain ingress.spec.rules.http.paths.backend.service

nano ingress.yml

```

```
## Wir ersetzen
## serviceName: apple-service
## durch:
## service:
##   name: apple-service

## das gleiche für banana

kubectl apply -f .
```

Fix 4.3. BadRequest unknown field servicePort

```
## was geht für die Property backend
kubectl explain ingress.spec.rules.http.paths.backend
## und was geht für service
kubectl explain ingress.spec.rules.http.paths.backend.service.port
## number
kubectl explain ingress.spec.rules.http.paths.backend.service.port

## neue Variante sieht so aus
backend:
  service:
    name: apple-service
    port:
      number: 80
## das gleich für banana-service

kubectl apply -f .
```

Fix 4.4. pathType must be specified

```
## Was macht das ?
kubectl explain ingress.spec.rules.http.paths.pathType

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

kubectl apply -f .
kubectl get ingress example-ingress
```

Step 5: Testing

```
## mit describe herausfinden, ob er die services gefundet
kubectl describe ingress example-ingress

## Im Browser auf:
## hier euer Name
http://jochen.lab.t3isp.de/apple
http://jochen.lab.t3isp.de/apple/
http://jochen.lab.t3isp.de/apple/foo
http://jochen.lab.t3isp.de/banana
## geht nicht
http://jochen.lab.t3isp.de/banana/nix
```

Achtung: Ingress mit Helm - annotations

Welcher wird verwendet, angeben:

```
Damit das Ingress Objekt welcher Controller verwendet werden soll, muss dieser angegeben werden:

kubernetes.io/ingress.class: nginx
```

```

Als ganzes Object:
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - http:
    paths:
      - path: /apple
        backend:
          serviceName: apple-service
          servicePort: 80
      - path: /banana
        backend:
          serviceName: banana-service
          servicePort: 80

```

Ref:

- <https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nginx-ingress-on-digitalocean-kubernetes-using-helm>

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"
  name: destination-home
  namespace: my-namespace
spec:
  rules:
  - http:
    paths:
      - backend:
          service:
            name: http-svc
            port:
              number: 80
            path: /source
            pathType: ImplementationSpecific

## eine node mit ip-adresse aufrufen
curl -I http://41.12.45.21/source
HTTP/1.1 308
Permanent Redirect

```

Umbauen zu google ;o)

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

Refs:

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

Kubernetes Ingress (Traefik)

Install Traefik-IngressController

```
helm repo add traefik https://traefik.github.io/charts

helm upgrade -n ingress --install traefik traefik/traefik --version 37.4.0 --create-namespace --skip-crds --reset-values

## Use special crds helm chart instead, because it does not deploy crds for gateway-api by default
## We get an error on digitalocean dokis
helm -n ingress upgrade --install traefik-crds traefik/traefik-crds --version 1.12.0 --reset-values
```

Ingress mit traefik

Step 1: Walkthrough

```
cd
mkdir -p manifests
cd manifests
mkdir abi
cd abi
```

```
nano apple-deploy.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apple-app
  labels:
    app: apple
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apple
  template:
    metadata:
      labels:
        app: apple
    spec:
      containers:
        - name: apple-app
          image: hashicorp/http-echo
          args:
            - "-text=apple-<euer-name>"
```

```
nano apple-svc.yaml
```

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

```
kubectl apply -f .
```

```
nano banana-deploy.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: banana-app
  labels:
    app: banana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: banana
  template:
    metadata:
      labels:
```

```

    app: banana
spec:
  containers:
    - name: apple-app
      image: hashicorp/http-echo
      args:
        - "-text=banana-<euer-name>"
```

```
nano banana-svc.yaml
```

```

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

```
kubectl apply -f .
```

Step 2: Testing connection by podIP and Service

```

kubectl get svc
kubectl get pods -o wide
kubectl run podtest --rm -it --image busybox

/ # wget -O - http://<pod-ip>:5678
/ # wget -O - http://<cluster-ip>
```

Step 3: Walkthrough

```

nano ingress.yml

## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
spec:
  ingressClassName: traefik
  rules:
    - host: "<euernname>.app.do.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
```

Reference

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

Step 4: Find the problem

Fix 4.1: Fehler: no matches kind "Ingress" in version "extensions/v1beta1"

```

## Gibt es diese Landkarte überhaupt
kubectl api-versions
## auf welcher Landkarte/Gruppe befindet sich Ingress jetzt
kubectl explain ingress | head
## -> jetzt auf networking.k8s.io/v1
```

```
nano ingress.yml
```

```
## auf apiVersion: extensions/v1beta1
## wird -> networking.k8s.io/v1
```

```
kubectl apply -f .
```

Fix 4.2: Bad Request unkown field ServiceName / ServicePort

```
## was geht für die Property backend
kubectl explain ingress.spec.rules.http.paths.backend
## und was geht für service
kubectl explain ingress.spec.rules.http.paths.backend.service
```

```
nano ingress.yml
```

```
## Wir ersetzen
## serviceName: apple-service
## durch:
## service:
##   name: apple-service
## das gleiche für banana
```

```
kubectl apply -f .
```

Fix 4.3. BadRequest unknown field servicePort

```
## was geht für die Property backend
kubectl explain ingress.spec.rules.http.paths.backend
## und was geht für service
kubectl explain ingress.spec.rules.http.paths.backend.service.port
## number
kubectl explain ingress.spec.rules.http.paths.backend.service.port
```

```
## neue Variante sieht so aus
backend:
  service:
    name: apple-service
    port:
      number: 80
## das gleich für banana-service
```

```
kubectl apply -f .
```

Fix 4.4. pathType must be specified

```
## Was macht das ?
kubectl explain ingress.spec.rules.http.paths.pathType
```

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

```
kubectl apply -f .
kubectl get ingress example-ingress
```

Step 5: Testing

```
## mit describe herausfinden, ob er die services gefundet
kubectl describe ingress example-ingress
```

```
## Im Browser auf:
## hier euer Name
```

```

http://jochen.app.do.t3isp.de/apple
http://jochen.app.do.t3isp.de/apple/
http://jochen.app.do.t3isp.de/apple/foo
http://jochen.app.do.t3isp.de/banana
## geht nicht
http://jochen.app.do.t3isp.de/banana/nix

```

ingress mit traefik, letsencrypt und cert-manager

Schritt 1: cert-manager installieren

```

helm repo add jetstack https://charts.jetstack.io
helm upgrade --install cert-manager jetstack/cert-manager \
--namespace cert-manager --create-namespace \
--version v1.19.2 \
--set crds.enabled=true \
--reset-values

```

- Ref: <https://artifacthub.io/packages/helm/cert-manager/cert-manager>

Schritt 2: Create ClusterIssuer (gets certificates from Letsencrypt)

```

cd
mkdir -p manifests/cert-manager
cd manifests/cert-manager
nano cluster-issuer.yaml

```

```

## cluster-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    # Email-Adresse ändern - example.com ist nicht erlaubt
    email: your-email@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: traefik

```

```

kubectl apply -f .
## Should be True
kubectl get clusterissuer

```

Schritt 3: Ingress-Objekt mit TLS erstellen

```

nano example-ingress.yaml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  ingressClassName: traefik
  tls:
    - hosts:
        - <dein-name>.app.do.t3isp.de
      secretName: example-tls

  rules:
    - host: "<dein-name>.app.do.t3isp.de"
      http:
        paths:
          - path: /apple
            pathType: Prefix
            backend:
              service:
                name: apple-service
                port:
                  number: 80

```

```

- path: /banana
  pathType: Exact
  backend:
    service:
      name: banana-service
      port:
        number: 80

kubectl apply -f .

• Interessent, der cert-manager erstellt kurz ein Ingress - Objekt

cm-acme-http-solver-vf2hm <none> jochen.app.do.t3isp.de 174.138.100.89 80 10s
example-ingress traefik jochen.app.do.t3isp.de 174.138.100.89 80, 443 16m
tln1@client:~/manifests/abi$ kubectl get ingress
NAME          CLASS      HOSTS           ADDRESS      PORTS      AGE
example-ingress traefik   jochen.app.do.t3isp.de 174.138.100.89 80, 443 17m
tln1@client:~/manifests/abi$ kubectl get certificate
NAME      READY  SECRET      AGE
example-tls True   example-tls  85s

```

Schritt 4: Herausfinden, ob Zertifikate erstellt werden

```

kubectl describe certificate example-tls
kubectl get cert
## Certificate Request
kubectl get cr
## da ist das Zertifikat drin
kubectl get secret example-tls

```

Schritt 5: Testen

Ref:

- <https://hbayraktar.medium.com/installing-cert-manager-and-nginx-ingress-with-lets-encrypt-on-kubernetes-fe0dff4b1924>

Kubernetes Praxis (Stateful Sets)

Hintergrund statefulsets

Why ?

- stable network identities (always the same name across restarts) in contrast to deployments

```

Server: 10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name: web-0.nginx
Address 1: 10.244.1.6

nslookup web-1.nginx
Server: 10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name: web-1.nginx
Address 1: 10.244.2

```

The Pods' ordinals, hostnames, SRV records, and A record names have not changed, but the IP addresses associated with the Pods may have changed.

Features

- Scaling Up: Ordered creation on scaling (web 2 till ready then web-3 till ready and so on)

StatefulSet controller created each Pod sequentially with respect to its ordinal index,

and it waited for each Pod's predecessor to be Running and Ready

before launching the subsequent Pod

- Scaling Down: last created pod is torn down firstly, till finished, then the one before

The controller deleted one Pod at a time, in reverse order with respect to its ordinal index, and it waited for each to be completely shutdown before deleting the next.

- VolumeClaimTemplate (In addition if the pod is scaled the copies will have their own storage)
 - Plus: When you delete it, it gets recreated and claims the same persistentVolumeClaim

```
volumeClaimTemplates:
- metadata:
  name: www
spec:
  accessModes: [ "ReadWriteOnce" ]
  resources:
  requests:
    storage: 1Gi
```

- Update Strategy: RollingUpdate / OnDelete
- Feature: Staging an Update with Partitions
 - <https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/#staging-an-update>
- Feature: Rolling out a canary
 - <https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/#rolling-out-a-canary>

Reference

- <https://kubernetes.io/docs/concepts/workloads/controllers/statefulsets/>

Example stateful set

Schritt 1:

```
cd
mkdir -p manifests
cd manifests
mkdir sts
cd sts
```

```
nano 01-svc.yml
```

```
## vi 01-svc.yml
## Headless Service - no ClusterIP
## Just used for name resolution of pods
## web-0.nginx
## web-1.nginx
## nslookup web-0.nginx
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

```
nano 02-sts.yml
```

```
## vi 02-sts.yml
apiVersion: apps/v1
kind: StatefulSet
metadata:
## name des statefulset wird nachher für den dns-namen verwendet
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
    - name: nginx
      image: registry.k8s.io/nginx-slim:0.8
      ports:
      - containerPort: 80
        name: web-nginx
```

```
kubectl apply -f .
```

Schritt 2: Auflösung Namen.

```
kubectl run --rm -it podtester --image=busybox
```

```
## In der shell
## web ist der name des statefulsets
ping web-0.nginx
ping web-1.nginx
exit
```

```
## web-0 / web-1
kubectl get pods -o wide
kubectl get sts web
kubectl delete sts web
kubectl apply -f .
kubectl run --rm -it podtest --image=busybox

ping web-0.nginx

kubectl describe svc nginx
```

Referenz

- <https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/>

Kubernetes Secrets und Encrypting von z.B. Credentials

Kubernetes secrets Typen

Welche Arten von Secrets gibt es ?

| Built-in Type | Usage |
|-------------------------------------|---------------------------------------|
| Opaque | arbitrary user-defined data |
| kubernetes.io/service-account-token | ServiceAccount token |
| kubernetes.io/dockercfg | serialized ~/.dockercfg file |
| kubernetes.io/dockerconfigjson | serialized ~/.docker/config.json file |
| kubernetes.io/basic-auth | credentials for basic authentication |
| kubernetes.io/ssh-auth | credentials for SSH authentication |
| kubernetes.io/tls | data for a TLS client or server |
| bootstrap.kubernetes.io/token | bootstrap token data |

- Ref: <https://kubernetes.io/docs/concepts/configuration/secret/#secret-types>

Sealed Secrets - bitnami

2 Komponenten

- Sealed Secrets besteht aus 2 Teilen
 - kubeseal, um z.B. die Passwörter zu verschlüsseln
 - Dem Operator (ein Controller), der das Entschlüsseln übernimmt

Schritt 1: Walkthrough - Client Installation (als root)

```
curl -OL "https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.33.1/kubeseal-0.33.1-linux-amd64.tar.gz"
tar -xvzf kubeseal-0.33.1-linux-amd64.tar.gz kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

Schritt 2: Walkthrough - Server Installation mit kubectl client

```
helm repo add bitnami-labs https://bitnami-labs.github.io/sealed-secrets/
helm upgrade --install sealed-secrets --namespace kube-system bitnami-labs/sealed-secrets --version 2.17.9 --reset-values
```

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

Übung ist hier zu finden:

Beispiel mit kubeseal arbeiten

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/>

Exercise Sealed Secret mariadb

Prerequisites: MariaDB secrets done

[MariaDB Secret](#)

Based on mariadb secrets exercise

```
cd
cd manifests/secrettest

## Cleanup
kubectl delete -f 02-deploy.yml
kubectl delete -f 01-secrets.yml
## rm
rm 01-secrets.yml

## Öffentlichen Schlüssel zum Signieren holen
kubeseal --fetch-cert --controller-namespace=kube-system --controller-name=sealed-secrets > pub-sealed-secrets.pem
cat pub-sealed-secrets.pem

## Secret - config erstellen mit dry-run, wird nicht auf Server angewendet (nicht an Kube-Api-Server geschickt)
kubectl create secret generic mariadb-secret --from-literal=MARIADB_ROOT_PASSWORD=11abc432 --dry-run=client -o yaml > 01-top-secret.yaml
cat 01-top-secret.yaml

kubeseal --format=yaml --cert=pub-sealed-secrets.pem < 01-top-secret.yaml > 01-top-secret-sealed.yaml
cat 01-top-secret-sealed.yaml

## Ausgangsfile von dry-run löschen
rm 01-top-secret.yaml

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

kubectl apply -f .

## Kurz danach erstellt der Controller aus dem sealed secret das secret
kubectl get secret

kubectl get sealedsecrets
kubectl get secret mariadb-secret -o yaml

kubectl exec -it deploy/mariadb-deployment -- env | grep ROOT
kubectl delete -f 01-top-secret-sealed.yaml
kubectl get secrets
kubectl get sealedsecrets
```

registry mit secret auth

- <https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>

Kubernetes API-Objekte (Teil 2)

Kubernetes Praxis

Befehle in pod ausführen - Übung

```
kubectl run my-nginx --image=nginx:1.23

kubectl exec my-nginx -- ls -la
kubectl exec -it my-nginx -- bash
kubectl exec -it my-nginx -- sh

## in der shell
cat /etc/os-release
cd /var/log/nginx
ls -la
exit
```

```
## Logs ausgeben  
kubectl logs my-nginx
```

Welche Pods mit Namen gehören zu einem Service

```
kubectl get svc svc-nginx -o yaml  
kubectl get pods -l web=my-nginx
```

Security

ServiceLinks nicht in env in Pod einbinden

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: mariadb-deployment  
spec:  
  selector:  
    matchLabels:  
      app: mariadb  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        app: mariadb  
    spec:  
      ## Das ist hier --_>  
      enableServiceLinks: false  
      containers:  
      - name: mariadb-cont  
        image: mariadb:10.11  
        envFrom:  
        - configMapRef:  
          name: mariadb-configmap
```

Helm (Kubernetes Paketmanager)

Helm - Was kann Helm

- **Installieren und Deinstallieren** von Anwendungen in Kubernetes (`helm install` / `helm uninstall`)
- **Upgraden** von bestehenden Installationen (`helm upgrade`)
- **Rollbacks** durchführen, falls etwas schief läuft (`helm rollback`)
- **Anpassen** von Anwendungen durch Konfigurationswerte (`values.yaml`)
- **Veröffentlichen** eigener Charts (z.B. in einem Helm-Repository)

Helm Grundlagen

Wo kann ich Helm-Charts suchen ?

- Im Telefonbuch von helm <https://artifacthub.io/>

Komponenten

Chart

- beinhaltet Beschreibung und Komponenten

Chart - Bereitstellungsformen

- url
- .tgz (Abkürzung tar.gz) - Format
- oder Verzeichnis

```
Wenn wir ein Chart installieren, wird eine Release erstellen  
(parallel: image -> container, analog: chart -> release)
```

Installation

Was brauchen wir ?

- helm client muss installiert sein

Und sonst so ?

```
## Beispiel ubuntu  
## snap install --classic helm  
  
## Cluster auf das ich zugreifen kann und im Client -> helm und kubectl  
## 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
```

Helm Warum ?

Ein Paket für alle Komponenten
Einfaches Installieren, Updaten und deinstallieren
Konfigurations-Values-Files übergeben zum Konfigurieren
Feststehende Struktur

Was kann helm ?

- **Installieren und Deinstallieren** von Anwendungen in Kubernetes (`helm install` / `helm uninstall`)
- **Upgraden** von bestehenden Installationen (`helm upgrade`)
- **Rollbacks** durchführen, falls etwas schief läuft (`helm rollback`)
- **Anpassen** von Anwendungen durch Konfigurationswerte (`values.yaml`)
- **Veröffentlichen** eigener Charts (z.B. in einem Helm-Repository)

Helm Example

Prerequisites

- helm needs a config-file (kubeconfig) to know how to connect and credentials in there
- Good: helm (as well as kubectl) 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

Simple Walkthrough (Example 0: Step 1)

```
## Repo hinzufügen
helm repo add bitnami https://charts.bitnami.com/bitnami
## geachte Informationen aktualisieren
helm repo update

helm search repo bitnami
## helm install release-name bitnami/mysql
```

Simple Walkthrough (Example 0: Step 2: for learning - pull)

```
helm pull bitnami/mysql
tar xvfz mysql*
```

Simple Walkthrough (Example 0: Step 3: install)

```
helm install my-mysql bitnami/mysql
## Chart runterziehen ohne installieren
## helm pull bitnami/mysql

## Release anzeigen zu lassen
helm list

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

## weitere release installieren
## 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
```

```
## Show how the template would look like being sent to kube-api-server
helm template bitnami/mysql
```

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
```

Example 3: Install wordpress

Example 3.1: Setting values with --set

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-wordpress \
  --set wordpressUsername=admin \
  --set wordpressPassword=password \
  --set mariadb.auth.rootPassword=secretpassword \
  bitnami/wordpress
```

Example 3.2: Setting values with values.yml file

```
cd
mkdir -p manifests
cd manifests
mkdir helm-wordpress
cd helm-wordpress
nano values.yml

## values.yml
wordpressUsername: admin
wordpressPassword: password
mariadb:
  auth:
    rootPassword: secretpassword

## helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-wordpress -f values.yml bitnami/wordpress
```

Referenced

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

Installation, Upgrade, Uninstall helm-Chart exercise - simple (mariadb-cloudpirates)

Schritt 1: install mariadb von cloudpirates

```
## Mini-Step 1: Testen
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.5.1 --dry-run
```

```
## Mini-Step 2: Installieren
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.5.1
```

```
## Geht das denn auch ?
kubectl get pods
```

Schritt 2: Umschauen

```
kubectl get pods
helm status my-mariadb
helm list
## alle helm charts anzeigen, die im gesamten Cluster installiert wurden
helm list -A
helm history my-mariadb
```

Schritt 3: Umschauen get

```
## Wo speichert er Information, die er später mit helm get abruft
kubectl get secrets
```

```
helm get values my-mariadb
helm get manifest my-mariadb
## Zeige alle Kinds an
helm get manifest my-mariadb | grep -i -A 4 kind
## Can I see all values use -> YES
## Look for COMPUTED VALUES in get all ->
helm get all my-mariadb
```

```
## Hack COMPUTED VALUES anzeigen lassen
## Welche Werte (values) hat er zur Installation verwendet
helm get all my-mariadb | grep -i computed -A 200
```

Schritt 4: Exercise: Upgrade to new version

Schritt 4.1 Default values (auf terminal) ausfindig machen

```
## Recherchiere wie die Werte gesetzt werden (artifacthub.io) oder verwende die folgenden Befehle:
helm show values oci://registry-1.docker.io/cloudpirates/mariadb
helm show values oci://registry-1.docker.io/cloudpirates/mariadb | less
```

Schritt 4.2 Upgrade und resources ändern

```
cd
mkdir -p mariadb-values
cd mariadb-values
mkdir prod
cd prod
```

```
nano values.yaml
```

```
resources:
  limits:
    memory: 300Mi
  requests:
    memory: 300Mi
    cpu: 100m
```

```
cd ..
```

```
## Testen
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.5.3 --dry-run -f
prod/values.yaml
```

```
## Real Upgrade
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.5.3 -f
prod/values.yaml
```

```
kubectl get pods
kubectl describe pods my-mariadb-0
helm list
helm history my-mariadb
helm get values my-mariadb
```

Schritt 4.3 Weiteres Update der Chart - Version (auf Version 0.9.0)

```
## Testen
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.9.0 --dry-run -f prod/values.yaml

## Real Upgrade
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.9.0 -f prod/values.yaml

## Schlägt fehle, weil mit dem apply bestimmte Felder nicht überschrieben dürfen, die geändert wurden im Template
```

Lösung

- Deinstallieren (pvc bleibt erhalten auch beim Deinstallieren -> so macht das helm)
- Und wieder installieren in der neuen Version

```
## Frage, ist das pvc noch ?
kubectl get pvc
## Ja !
```

| NAME | STATUS | VOLUME | CAPACITY |
|-------------------|------------------|--|----------|
| ACCESS MODES | STORAGECLASS | VOLUMEATTRIBUTESCLASS | AGE |
| data-my-mariadb-0 | Bound | pvc-575fd652-e0da-46ed-b917-a2c28629e164 | 8Gi |
| RWO | do-block-storage | <unset> | 66m |

```
helm uninstall my-mariadb
kubectl get pvc
## auch nach der Deinstallation ist der pvc noch da
## Super !!

## Real Upgrade
helm upgrade --install my-mariadb oci://registry-1.docker.io/cloudpirates/mariadb --reset-values --version 0.9.0 -f
prod/values.yaml

kubectl get pods
```

Tipp: values aus alter revision anzeigen

```
## Beispiel:
helm get values my-mariadb --revision 1
```

Uninstall

```
helm uninstall my-mariadb
## namespace wird nicht gelöscht
## handisch löschen
kubectl delete ns app-<namenskuerzel>
## crd's werden auch nicht gelöscht
```

Problem: OutOfMemory (OOM-Killer) if container passes limit in memory

- if memory of container is bigger than limit an OOM-Killer will be triggered
- How to fix. Use memory limit in the application too !
 - <https://techcommunity.microsoft.com/blog/appsonazureblog/unleashing-javascript-applications-a-guide-to-boosting-memory-limits-in-node-js/4080857>

Helm Exercise with nginx

Part 1: Chart erstellen

```
cd
mkdir -p charts
cd charts
## mit helm neues Chart erstellen
helm create beispiel-chart
```

Part 2: chart installieren

```
helm upgrade --install my-nginx beispiel-chart
```

Part 3: funktioniert es ?

```
kubectl get pods  
helm list
```

Part 4: Spezielle Konfiguration

Part 4.1: Analyse

```
## Können wir die replicas und den type server ändern  
## Entweder variante 1 ins Chart  
less beispiel-chart/values.yaml  
## mit helm bordmitteln  
helm show values beispiel-chart | less
```

Part 4.2. values.yaml (eigene Konfiguration) erstellen und anwenden

```
cd  
mkdir helm-values  
cd helm-values  
mkdir beispiel-chart  
cd beispiel-chart  
nano values.yaml  
  
## in der Datei values.yaml  
replicaCount: 2  
service:  
  type: NodePort  
  
cd  
cd charts  
helm upgrade --install my-nginx beispiel-chart --reset-values -f ../helm-values/beispiel-chart/values.yaml  
kubectl get pods  
## neue Revision  
helm list  
## hier NodePort auslesen  
kubectl get svc my-nginx-beispiel-chart  
kubectl get nodes -o wide  
  
## Testen  
curl http://<ip-aus-nodes>:<nodePort>  
## z.B.  
curl http://159.223.24.231:32465
```

Part 4.3 Explore

```
helm list  
helm list -A (über alle namespaces hinweg)  
  
## Zeige mir alles von der installierten Release  
helm get all my-nginx  
helm get values my-nginx  
helm get manifest my-nginx  
  
## chart von online  
## für unser chart  
cd  
cd charts  
helm show values beispiel-chart # latest version
```

Part 5 Uninstall nginx

```
## Achtung keine Deinstallation von CRD's, keine Deinstallation von PVC (Persistent Volume Claims), RBAC  
helm uninstall my-nginx  
  
## Überprüfung, ob Deinstallation erfolgt ist:  
helm list
```

Helm Spickzettel

Hilfe

```
helm help  
helm help <command>  
helm help upgrade
```

Alle helm-releases anzeigen

```
## im eigenen Namespace  
helm list  
## in allen Namespaces  
helm list -A  
## für einen speziellen  
helm -n kube-system list
```

Helm - Chart installieren

```
## Empfehlung mit namespace  
## Repo hinzufügen für Client  
helm repo add bitnami https://charts.bitnami.com/bitnami  
helm install my-nginx bitnami/nginx --version 19.0.1 --create-namespace --namespace=app-  
<namenskuerzel>
```

Helm - Suche

```
## welche Repos sind konfiguriert  
helm repo list  
helm search repo bitnami  
helm search hub
```

Helm - template

```
## Rendern des Templates  
helm repo add bitnami https://charts.bitnami.com/bitnami  
helm template my-nginx bitnami/nginx  
helm template bitnami/nginx
```

Helm Charts erstellen und analysieren

Eigenes Helm-Chart erstellen

Chart erstellen

```
cd  
mkdir my-charts  
cd my-charts  
  
helm create my-app
```

Install helm - chart

```
## Variante 1:  
helm -n my-app-  
<namenskuerzel> install meine-app my-app --create-namespace  
  
## Variante 2:  
cd my-app  
helm -n my-app-  
<namenskuerzel> install meine-app . --create-namespace  
  
kubectl -n my-app-  
<namenskuerzel> get all  
kubectl -n my-app-  
<namenskuerzel> get pods
```

Chart zur Analyse runterladen und entpacken

```
cd  
mkdir -p charts-download  
cd charts-download  
  
helm pull oci://registry-1.docker.io/cloudpirates/mariadb  
  
## Lädt die letzte version herunter  
helm pull oci://registry-1.docker.io/cloudpirates/mariadb  
  
## Lädt bestimmte chart-version runter  
## helm pull oci://registry-1.docker.io/cloudpirates/mariadb --version 0.9.0  
## evtl. entpacken wenn gewünscht  
## tar xvf mariadb-12.1.6.tgz  
  
## Schnelle Variante  
helm pull oci://registry-1.docker.io/cloudpirates/mariadb --version 0.9.0 --untar
```

Helm - Fehleranalyse

Beispiel Cloudpirates - helm chart nginx

Test (mit aktuell letzter Version 0.1.10)

```
helm upgrade --install my-nginx oci://registry-1.docker.io/cloudpirates/nginx --reset-values --version 0.1.10

### wie sehen die logs aus
kubectl logs deployment/my-nginx

Permission denied port 80
```

Lauffähig mit (leider aktuell so nicht in der Doku)

```
cd
mkdir helm-values/nginx
cd helm-values/nginx
nano values.yaml

containerPorts:
- name: http
  containerPort: 8080
  protocol: TCP

serverConfig: |
  server {
    listen 0.0.0.0:8080;
    root /usr/share/nginx/html;
    index index.html index.htm;

    location / {
      try_files $uri $uri/ /index.html;
    }
  }
livenessProbe:
  type: httpGet
  path: /
readinessProbe:
  type: httpGet
  path: /


helm upgrade --install my-nginx oci://registry-1.docker.io/cloudpirates/nginx --version 0.1.10 --reset-values -f values.yaml
```

```
kubectl exec -it deploy/my-nginx -- sh

## in der shell
id
```

```
/ $ id
uid=101(nginx) gid=101(nginx) groups=101(nginx)
/ $ ^C
```

```
## pod läuft nicht unter root
## Deshalb funktioniert ein öffnen des Ports 80 beim Starten nicht
```

Helpful plugins

Use shortnames for kubectl - commands

- <https://gist.github.com/doevelopper/ff4a9a211e74f8a2d44eb4afb21f0a38>

Kubernetes Debugging

Probleme über Logs identifiziert - z.B. non-root image

Schritt 1:

```
cd
mkdir -p manifests
cd manifests
mkdir -p unpriv
cd unpriv
```

```

nano pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-unprivileged
spec:
  securityContext:
    runAsUser: 1000 # Container läuft mit UID 1000 statt root
  containers:
    - name: nginx
      image: nginx:1.25
      ports:
        - containerPort: 80

kubectl apply -f .
kubectl get pods

```

Schritt 2: Debuggen

```

## CrashLoopBackoff
kubectl get pods nginx-unprivileged
kubectl describe pods nginx-unprivileged

## permission denied identifiziert
kubectl logs nginx-unprivileged

```

Schritt 3: Lösung anderes image nehmen

```

## in pod.yaml
## Zeile image: nginx:1.25
## in -> image: bitnami/nginx:1.25
## ändern

kubectl apply -f .
kubectl get pods

```

Weiter lernen

Lernumgebung

- <https://killercoda.com/>

Kubernetes Doku - Bestimmte Tasks lernen

- <https://kubernetes.io/docs/tasks/configure-pod-container/assign-memory-resource/>

Kubernetes Videos mit Hands On

- <https://www.youtube.com/watch?v=16fgzklcF7Y>

Kubernetes Storage (CSI)

Überblick Persistant Volumes (CSI)

Überblick

Warum CSI ?

- Each vendor can create his own driver for his storage

Vorteile ?

I. Automatically create storage when required.
II. Make storage available to containers wherever they're scheduled.
III. Automatically delete the storage when no longer needed.

Wie war es vorher ?

Vendor needed to wait till his code was checked in in tree of kubernetes (in-tree)

Unterschied static vs. dynamisch

The main difference relies on the moment when you want to configure storage. For instance, if you need to pre-populate data in a volume, you choose static provisioning. Whereas, if you need to create volumes on demand, you go for dynamic provisioning.

Komponenten

Treiber

- Für jede Storage Class (Storage Provider) muss es einen Treiber geben

Storage Class

Liste der Treiber mit Features (CSI)

- <https://kubernetes-csi.github.io/docs/drivers.html>

Übung Persistant Storage

- Step 1 + 2 : nur Trainer
- ab Step 3: Trainees

Step 1: Do the same with helm - chart

```
helm repo add csi-driver-nfs https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/master/charts
helm upgrade --install csi-driver-nfs csi-driver-nfs/csi-driver-nfs --namespace kube-system --version v4.12.1 --reset-values
```

Step 2: Storage Class

```
cd
mkdir -p manifests
cd manifests
mkdir csi-storage
cd csi-storage
nano 01-storageclass.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-csi
provisioner: nfs.csi.k8s.io
parameters:
  server: 10.135.0.70
  share: /var/nfs
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

```
kubectl apply -f .
```

Step 3: Persistent Volume Claim

```
cd
mkdir -p manifests
cd manifests
mkdir csi
cd csi
nano 02-pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-dynamic
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
  storageClassName: nfs-csi
```

```
kubectl apply -f .
kubectl get pvc
##
kubectl get pv
```

Step 4: Pod

```
nano 03-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-nfs
spec:
  containers:
```

```

- image: nginx:1.23
  name: nginx-nfs
  command:
    - "/bin/bash"
    - "-c"
    - set -e; while true; do echo $(date) >> /mnt/nfs/outfile; sleep 1; done
  volumeMounts:
    - name: persistent-storage
      mountPath: "/mnt/nfs"
      readOnly: false
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: pvc-nfs-dynamic

```

```
kubectl apply -f .
kubectl get pods
```

Step 5: Testing

```

kubectl exec -it nginx-nfs -- bash

cd /mnt/nfs
ls -la
## outfile
tail -f /mnt/nfs/outfile

CTRL+C
exit

```

Step 6: Destroy

```

kubectl delete -f 03-pod.yaml

### Verify in nfs - trainer !!

```

Step 7: Recreate

```

kubectl apply -f 03-pod.yaml

kubectl exec -it nginx-nfs -- bash

## is old data here ?
head /mnt/nfs/outfile
##
tail -f /mnt/nfs/outfile

CTRL + C
exit

```

Step 8: Cleanup

```
kubectl delete -f .
```

Reference:

- <https://rudimartinsen.com/2024/01/09/nfs-csi-driver-kubernetes/>

Beispiel mariadb

- How to persistently use mariadb with a storage class / driver nfs.csi.

Step 1: Treiber installieren

- <https://github.com/kubernetes-csi/csi-driver-nfs/blob/master/docs/install-csi-driver-v4.6.0.md>

```
curl -skSL https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/v4.6.0/deploy/install-driver.sh | bash -s v4.6.0 --
```

Step 2: Storage Class

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-csi
provisioner: nfs.csi.k8s.io

```

```
parameters:
  server: 10.135.0.18
  share: /var/nfs
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:
  - nfsvers=3
```

Step 3: PVC, Configmap, Deployment

```
mkdir -p manifests
cd manifests
mkdir mariadb-csi
cd mariadb-csi
```

```
nano 01-pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-dynamic-mariadb
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
  storageClassName: nfs-csi
```

```
kubectl apply -f .
```

```
nano 02-configmap.yaml
```

```
### 02-configmap.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: mariadb-configmap
data:
  # als Wertepaare
  MARIADB_ROOT_PASSWORD: 11abc432
```

```
nano 03-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deployment
spec:
  selector:
    matchLabels:
      app: mariadb
  replicas: 1
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      containers:
        - name: mariadb-cont
          image: mariadb:10.11
          envFrom:
            - configMapRef:
                name: mariadb-configmap
          volumeMounts:
            - name: persistent-storage
              mountPath: "/var/lib/mysql"
              readOnly: false
      volumes:
        - name: persistent-storage
          persistentVolumeClaim:
            claimName: pvc-nfs-dynamic-mariadb
```

```
kubectl apply -f .
```

```
kubectl describe po mariadb-deployment-<euer-pod>
```

Kubernetes Installation

k3s installation

Exercise

```
Schritt 1: windows store ubuntu installieren und ausführen

Siehe:
https://docs.k3s.io/quick-start

## in den root benutzer wechseln
sudo su -
## passwort eingeben

curl -sfL https://get.k3s.io | sh -

systemctl stop k3s
## k3s automatischer start beim booten ausschalten
systemctl disable k3s
systemctl start k3s

## Verwenden
kubectl cluster-info
kubectl get nodes
```

Kubernetes Monitoring

Prometheus Monitoring Server (Overview)

What does it do ?

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

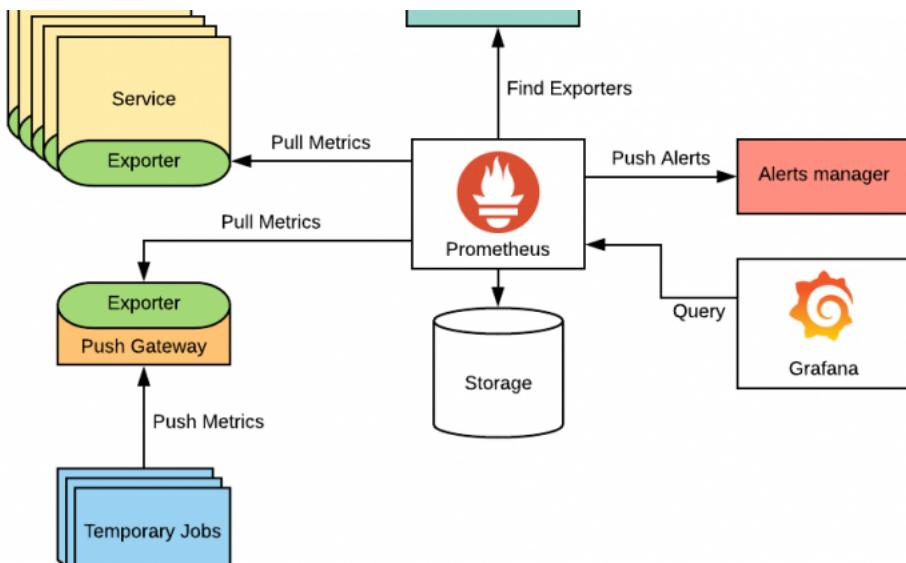
Technical

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

What are time series ?

- A time series is a sequence of data points that occur in successive order over some period of time.
- Beispiel:
 - Du willst die täglichen Schlusspreise für eine Aktie für ein Jahr dokumentieren
 - Damit willst Du weitere Analysen machen
 - Du würdest das Paar Datum/Preis dann in der Datumsreihenfolge sortieren und so ausgeben
 - Dies wäre eine "time series"

Komponenten von Prometheus



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

Prometheus Server

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

Grafana ?

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

Prometheus / Grafana Stack installieren

- using the kube-prometheus-stack (recommended !: includes important metrics)

Step 1: Prepare values-file

```

cd
mkdir -p manifests
cd manifests
mkdir -p monitoring
cd monitoring

vi values.yml

fullnameOverride: prometheus

alertmanager:
  fullnameOverride: alertmanager

grafana:
  fullnameOverride: grafana

kube-state-metrics:
  fullnameOverride: kube-state-metrics

prometheus-node-exporter:
  fullnameOverride: node-exporter

```

Step 2: Install with helm

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus prometheus-community/kube-prometheus-stack -f values.yaml --namespace monitoring --create-namespace --
version 61.3.1
```

Step 3: Connect to prometheus from the outside world

Step 3.1: Start proxy to connect (to on Linux Client)

```
## this is shown in the helm information
helm -n monitoring get notes prometheus

## Get pod that runs prometheus
kubectl -n monitoring get service
kubectl -n monitoring port-forward svc/prometheus-prometheus 9090 &
```

Step 3.2: Start a tunnel in (from) your local-system to the server

```
ssh -L 9090:localhost:9090 tln1@164.92.129.7
```

Step 3.3: Open prometheus in your local browser

```
## in browser
http://localhost:9090
```

Step 4: Connect to the grafana from the outside world

Step 4.1: Start proxy to connect

```
## Do the port forwarding
## Adjust your pods here
kubectl -n monitoring get pods | grep grafana
kubectl -n monitoring port-forward grafana-56b45d8bd9-bp899 3000 &
```

Step 4.2: Start a tunnel in (from) your local-system to the server

```
ssh -L 3000:localhost:3000 tln1@164.92.129.7
```

References:

- <https://github.com/prometheus-community/helm-charts/blob/main/charts/kube-prometheus-stack/README.md>
- <https://artifacthub.io/packages/helm/prometheus-community/prometheus>

Kubernetes QoS / HealthChecks / Live / Readiness

Quality of Service - evict pods

Die Class wird auf Basis der Limits und Requests der Container vergeben

```
Request: Definiert wieviel ein Container mindestens braucht (CPU,memory)
Limit: Definiert, was ein Container maximal braucht.

in spec.containers.resources
kubectl explain pod.spec.containers.resources
```

Art der Typen:

- Guaranteed
- Burstable
- BestEffort

Wie werden die Pods evicted

- Das wird in der folgenden Reihenfolge gemacht: Zu erst alle BestEffort, dann burstable und zum Schluss Guaranteed

Guaranteed

```
Type: Guaranteed:
https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/#create-a-pod-that-gets-assigned-a-qos-class-of-
guaranteed

set when limit equals request
(request: das braucht er,
limit: das braucht er maximal)

Garantied ist die höchste Stufe und diese werden bei fehlenden Ressourcen
als letztes "evicted"
```

Guaranteed Exercise

```

cd
mkdir -p manifests
cd manifests
mkdir qos
cd qos
nano 01-pod.yaml

apiVersion: v1

kind: Pod
metadata:
  name: qos-demo
spec:
  containers:
    - name: qos-demo-ctr
      image: nginx
      resources:
        limits:
          memory: "200Mi"
          cpu: "700m"
        requests:
          memory: "200Mi"
          cpu: "700m"

kubectl apply -f .
kubectl describe po qos-demo

```

Risiko Guaranteed

- Limit CPU: Diese wird maximal zur Verfügung gestellt
- Limit Memory: Wenn die Anwendung das Limit überschreitet, greift der OOM-Killer (Out of Memory Killer)
- Wenn Limit Memory: Dann auch dafür sorgen, dass das laufende Programme selbst auch eine Speichergrenze
 - Java-Programm ohne Speichergrenze oder zu hoher Speichergrenze

Burstable

- At least one Container in the Pod has a memory or CPU request or limit

```

cd
mkdir -p manifests
cd manifests
mkdir burstable
cd burstable

nano 01-pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: qos-burstable
spec:
  containers:
    - name: qos-demo-2-ctr
      image: nginx
      resources:
        limits:
          memory: "200Mi"
        requests:
          memory: "100Mi"

kubectl apply -f .
kubectl describe po qos-burstable

```

BestEffort

- gar keine Limits und Requests gesetzt (bitte nicht machen)

LiveNess/Readiness - Probe / HealthChecks

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 seconds 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
nano 01-pod-liveness-command.yaml

```

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/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 .
kubectl describe -l test=liveness pods | grep -A 40 Events
sleep 30
kubectl describe -l test=liveness pods | grep -A 40 Events
sleep 5
kubectl describe -l test=liveness pods | grep -A 40 Events

```

```

## cleanup
kubectl delete -f .

```

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([]byte(fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]byte("ok"))
    }
})

```

```

## Step 1: Pod - manifest
## vi 02-pod-liveness-http.yaml
## 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/>

Taints / Tolerations

Taints

Taints schliessen auf einer Node alle Pods aus, die nicht bestimmte tolerations haben:

Möglichkeiten:

- o Sie werden nicht gescheduled - NoSchedule
- o Sie werden nicht executed - NoExecute
- o Sie werden möglichst nicht gescheduled. - PreferNoSchedule

Tolerations

Tolerations werden auf Pod-Ebene vergeben:
tolerations:

Ein Pod kann (wenn es auf einem Node taints gibt), nur
gescheduled bzw. ausgeführt werden, wenn er die
Labels hat, die auch als
Taints auf dem Node vergeben sind.

Walkthrough

Step 1: Cordon the other nodes - scheduling will not be possible there

```

## Cordon nodes n11 and n111
## You will see a taint here
kubectl cordon n11
kubectl cordon n111
kubectl describe n111 | grep -i taint

```

Step 2: Set taint on first node

```
kubectl taint nodes n1 gpu=true:NoSchedule
```

Step 3

```

cd
mkdir -p manifests
cd manifests
mkdir tainttest
cd tainttest
nano 01-no-tolerations.yml

```

```

##vi 01-no-tolerations.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-no-tol
  labels:
    env: test-env
spec:
  containers:

```

```

- name: nginx
  image: nginx:1.21

kubectl apply -f .
kubectl get po nginx-test-no-tol
kubectl get describe nginx-test-no-tol

```

Step 4:

```

## vi 02-nginx-test-wrong-tol.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-wrong-tol
  labels:
    env: test-env
spec:
  containers:
  - name: nginx
    image: nginx:latest
  tolerations:
  - key: "cpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"

kubectl apply -f .
kubectl get po nginx-test-wrong-tol
kubectl describe po nginx-test-wrong-tol

```

Step 5:

```

## vi 03-good-tolerations.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-good-tol
  labels:
    env: test-env
spec:
  containers:
  - name: nginx
    image: nginx:latest
  tolerations:
  - key: "gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"

kubectl apply -f .
kubectl get po nginx-test-good-tol
kubectl describe po nginx-test-good-tol

```

Taints rausnehmen

```
kubectl taint nodes n1 gpu:true:NoSchedule-
```

uncordon other nodes

```
kubectl uncordon n11
kubectl uncordon n111
```

References

- [Doku Kubernetes Taints and Tolerations](#)
- <https://blog.kubecost.com/blog/kubernetes-taints/>

Installation mit microk8s

Schritt 1: auf 3 Maschinen mit Ubuntu 24.04LTS

Walkthrough

```

sudo snap install microk8s --classic
## Important enable dns // otherwise not dns lookup is possible
microk8s enable dns

```

```
microk8s status

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

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

Working with snaps

```
snap info microk8s
```

Ref:

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

Schritt 2: cluster - node2 + node3 einbinden - master ist node 1

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.
```

Add Node only as Worker-Node

```
microk8s join 10.135.0.15:25000/5857843e774c2ebe368e14e8b95bdf80/9bf3ceb70a58 --worker
Contacting cluster at 10.135.0.15

root@n41:~# microk8s status
This MicroK8s deployment is acting as a node in a cluster.
Please use the master node.
```

Ref:

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

Schritt 3: Remote Verbindung einrichten

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.
```

Add Node only as Worker-Node

```
microk8s join 10.135.0.15:25000/5857843e774c2ebe368e14e8b95bdf80/9bf3ceb70a58 --worker
Contacting cluster at 10.135.0.15

root@n41:~# microk8s status
This MicroK8s deployment is acting as a node in a cluster.
Please use the master node.
```

Ref:

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

Installation mit kubeadm

Schritt für Schritt mit kubeadm

Version

- Ubuntu 20.04 LTS
- 1 control plane und 3 worker nodes

Done for you (for 4 Servers)

- Servers are setup:
 - ssh-running
 - kubeadm, kubelet, kubectl installed
 - containerd - runtime installed
- Installed on all nodes (with cloud-init)

```
#!/bin/bash

groupadd sshadmin
USERS="mysupersecretuser"
SUDO_USER="mysupersecretuser"
PASS="yoursupersecretpass"
for USER in $USERS
do
    echo "Adding user $USER"
    useradd -s /bin/bash --create-home $USER
    usermod -aG sshadmin $USER
    echo "$USER:$PASS" | chpasswd
done

## We can sudo with $SUDO_USER
usermod -aG sudo $SUDO_USER

## 20.04 and 22.04 this will be in the subfolder
if [ -f /etc/ssh/sshd_config.d/50-cloud-init.conf ]
then
    sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config.d/50-cloud-init.conf
fi

### both is needed
sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config

usermod -aG sshadmin root

## TBD - Delete AllowUsers Entries with sed
## otherwise we cannot login by group

echo "AllowGroups sshadmin" >> /etc/ssh/sshd_config
systemctl reload sshd

## Now let us do some generic setup
echo "Installing kubeadm kubelet kubectl"

#### A lot of stuff needs to be done here
#### https://www.linuxtechies.com/install-kubernetes-on-ubuntu-22-04/

## 1. no swap please
swapoff -a
sudo sed -i '/ swap / s/^(\.*\)$/#\1/g' /etc/fstab

## 2. Loading necessary modules
echo "overlay" >> /etc/modules-load.d/containerd.conf
echo "br_netfilter" >> /etc/modules-load.d/containerd.conf
modprobe overlay
modprobe br_netfilter

## 3. necessary kernel settings
echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.d/kubernetes.conf
sysctl --system

## 4. Update the meta-information
apt-get -y update

## 5. Installing container runtime
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.add-apt-repository
"deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" apt-get install -y containerd.io
```

```

## 6. Configure containerd
containerd config default > /etc/containerd/config.toml
sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
systemctl restart containerd
systemctl enable containerd

## 7. Add Kubernetes Repository for Kubernetes
mkdir -m 755 /etc/apt/keyrings
apt-get install -y apt-transport-https ca-certificates curl gpg
curl -fsSL https://pkgs.k8s.io/core:/stable:/$K8S_VERSION/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/$K8S_VERSI
# 8. Install kubectl kubeadm kubectl
apt-get -y update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold -y kubelet kubeadm kubectl

## 9. Install helm
snap install helm --classic

## Installing nfs-common
apt-get -y install nfs-common

```

Prerequisites

- 4 Servers setup and reachable through ssh.
- user: 11trainingdo
- pass: PLEASE ask your instructor

```

## Important - Servers are not reachable through
## Domain !! Only IP.
controlplane.tln<nr>.t3isp.de
worker1.tln<nr>.do.t3isp.de
worker2.tln<nr>.do.t3isp.de
worker3.tln<nr>.do.t3isp.de

```

Step 1: Setup controlnode (login through ssh)

```

## This CIDR is the recommendation for calico
## Other CNI's might be different
CLUSTER_CIDR="192.168.0.0/16"

kubeadm init --pod-network-cidr=$CLUSTER_CIDR && \
mkdir -p /root/.kube && \
cp -i /etc/kubernetes/admin.conf /root/.kube/config && \
chown $(id -u):$(id -g) /root/.kube/config && \
cp -i /root/.kube/config /tmp/config.kubeadm && \
chmod o+r /tmp/config.kubeadm

## Copy output of join (needed for workers)
## e.g.
kubeadm join 159.89.99.35:6443 --token rpylp0.rdp PHPzbavdyx3llz \
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```

Step 2: Setup worker1 - node (login through ssh)

```

## use join command from Step 1:
kubeadm join 159.89.99.35:6443 --token rpylp0.rdp PHPzbavdyx3llz \
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```

Step 3: Setup worker2 - node (login through ssh)

```

## use join command from Step 1:
kubeadm join 159.89.99.35:6443 --token rpylp0.rdp PHPzbavdyx3llz \
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```

Step 4: Setup worker3 - node (login through ssh)

```

## use join command from Step 1:
kubeadm join 159.89.99.35:6443 --token rpylp0.rdp PHPzbavdyx3llz \
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```

Step 5: CNI-Setup (calico) on controlnode (login through ssh)

```
kubectl get nodes
```

```

## Output
root@controlplane:~# kubectl get nodes
NAME        STATUS   ROLES      AGE     VERSION
controlplane NotReady control-plane 6m27s v1.28.6
worker1      NotReady <none>    3m18s v1.28.6
worker2      NotReady <none>    2m10s v1.28.6
worker3      NotReady <none>    60s    v1.28.6

## Installing calico CNI
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.29.2/manifests/tigera-operator.yaml
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.29.2/manifests/custom-resources.yaml
kubectl get ns
kubectl -n calico-system get all
kubectl -n calico-system get pods -o wide -w

## After if all pods are up and running -> CTRL + C

kubectl -n calico-system get pods -o wide
## all nodes should be ready now
kubectl get nodes -o wide

## Output
root@controlplane:~# kubectl get nodes
NAME        STATUS   ROLES      AGE     VERSION
controlplane Ready    control-plane 14m    v1.28.6
worker1      Ready    <none>    11m    v1.28.6
worker2      Ready    <none>    10m    v1.28.6
worker3      Ready    <none>    9m9s   v1.28.6

```

Do it with ansible

- <https://spacelift.io/blog/ansible-kubernetes>

Tipps & Tricks

Pods bleiben im terminate-mode stehen

Ein Pod hängt in Terminierung meist aus folgenden Gründen:

Häufigste Ursachen:

1. Graceful Shutdown Timeout

- Container reagiert nicht auf SIGTERM
- Nach `terminationGracePeriodSeconds` (default 30s) wird SIGKILL gesendet
- Check: `kubectl describe pod <pod>` → Events

2. Finalizers blockieren

```
kubectl get pod <pod> -o yaml | grep finalizers -A 5
```

Manuelle Entfernung (Notfall):

```
kubectl patch pod <pod> -p '{"metadata":{"finalizers":null}}'
```

3. PreStop Hook hängt

- Zählt zur Grace Period
- Hook-Fehler blockiert Terminierung

4. Volume Unmount Problem

- Node kann Volume nicht freigeben
- Oft bei NFS/CSI-Storage

5. Node NotReady

- Kubelet antwortet nicht
- Pod bleibt in Terminating bis Node zurück oder nach ~5min force-deleted

Quick Fix:

```

## Force delete (wenn nichts anderes hilft)
kubectl delete pod <pod> --grace-period=0 --force

```

Debug-Befehle:

```
kubectl get pod <pod> -o yaml  
kubectl describe pod <pod>  
kubectl logs <pod> --previous
```

Was sind finalizer

- Finalizers sind Strings im metadata.finalizers-Array eines Pods, die verhindern, dass Kubernetes das Objekt vollständig löscht, bis alle Finalizer entfernt wurden.

```
Bei kubectl delete pod setzt Kubernetes nur metadata.deletionTimestamp  
Pod geht in Status Terminating  
Controller/Operator mit zuständigem Finalizer führt Cleanup aus  
Controller entfernt seinen Finalizer aus dem Array  
Erst wenn Array leer → vollständige Löschung
```

Probleme mit Volumes

VolumeAttachment-Objekt (separates K8s-Objekt):

```
apiVersion: storage.k8s.io/v1  
kind: VolumeAttachment  
metadata:  
  finalizers:  
    - external-attacher/csi-driver-name # <-- Hier ist der Finalizer  
spec:  
  attacher: csi-driver  
  nodeName: worker-1  
  source:  
    persistentVolumeName: pvc-xyz
```

Pod-Objekt hat normalerweise **keine Volume-Finalizer**:

```
apiVersion: v1  
kind: Pod  
metadata:  
  finalizers: [] # Meist leer oder mit anderen Finalizern
```

Warum Pods trotzdem bei Volumes hängen:

- Kubelet wartet auf erfolgreiches Volume-Unmount (nicht Finalizer-basiert)
- VolumeAttachment-Finalizer verhindert Detach vom Node
- Node nicht erreichbar → Kubelet kann nicht ummounten
- CSI-Driver-Probleme → external-attacher kann Finalizer nicht clearn

Typisches Problem-Szenario:

```
## Pod bleibt Terminating  
kubectl get pod -o yaml  
## deletionTimestamp gesetzt, aber keine Volume-Finalizer  
  
## VolumeAttachment existiert noch  
kubectl get volumeattachment  
## Hat Finalizer vom CSI-Driver
```

Force-Delete hilft hier nicht beim Volume-Problem, sondern man muss das VolumeAttachment-Objekt oder den Node-Status addressieren.

Podman

Podman vs. Docker

Was ist Podman?

Podman (kurz für: *Pod Manager*) ist eine Open-Source-Container-Engine, die als Alternative zu Docker entwickelt wurde.

Warum Podman?

Hier die wichtigsten Gründe, warum Podman gerne verwendet wird:

| Vorteil | Erklärung |
|-------------------|--|
| Daemonless | Podman benötigt keinen zentralen Hintergrunddienst (Daemon), sondern läuft als normaler Benutzerprozess. Dadurch weniger Angriffsfläche und flexibler. |
| Rootless Support | Container können ohne Root-Rechte gestartet werden, was Sicherheitsvorteile bringt. |
| Docker-kompatibel | Podman kann Dockerfiles bauen und verwendet dieselben Container Images. Auch <code>podman</code> CLI ist zu <code>docker</code> CLI fast identisch (alias <code>docker=podman</code> geht oft problemlos). |

| | |
|--|--|
| Pods-Unterstützung | Inspirierte von Kubernetes: Mehrere Container in einem gemeinsamen Netzwerk-Namespace (Pod). Praktisch für lokale Kubernetes-ähnliche Setups. |
| Bessere Integration für Systemd | Einfaches Erstellen von Systemd-Units aus Containern (<code>podman generate systemd</code>). Ideal für Serverdienste ohne externes Orchestration-Tool. |
| Kein Root-Daemon | Keine ständigen Root-Rechte nötig, weniger Sicherheitsrisiko durch kompromittierte Daemons. |
| Red Hat / Fedora / CentOS bevorzugen Podman | Dort wird Podman inzwischen oft als Standardlösung ausgeliefert. |

Typische Einsatzbereiche

- Entwicklung: wie Docker
- Serverbetrieb: Container als Systemdienste
- Kubernetes-nahe Testing (Pods)
- Rootless Deployment auf Servern ohne Container-Daemon

Beispiel: Container starten

```
podman run -d -p 8080:80 nginx
```

Fast wie bei Docker.

Podman vs Docker kurz gesagt:

| | Docker | Podman |
|---------------------|------------|-------------------|
| Daemon | ja | nein |
| Rootless | begrenzt | ja |
| Kubernetes-nah | weniger | starker |
| Systemd-Integration | wenig | stark |
| Kompatibilität | verbreitet | Docker-kompatibel |

ServiceMesh

Why a ServiceMesh ?

What is a service mesh ?

A service mesh is an infrastructure layer that gives applications capabilities like zero-trust security, observability, and advanced traffic management, without code changes.

Advantages / Features

- Observability & monitoring
- Traffic management
- Resilience & Reliability
- Security
- Service Discovery

Observability & monitoring

- Service mesh offers:
 - valuable insights into the communication between services
 - effective monitoring to help in troubleshooting application errors.

Traffic management

- Service mesh offers:
 - intelligent request distribution
 - load balancing,
 - support for canary deployments.
 - These capabilities enhance resource utilization and enable efficient traffic management

Resilience & Reliability

- By handling retries, timeouts, and failures,
 - service mesh contributes to the overall stability and resilience of services
 - reducing the impact of potential disruptions.

Security

- Service mesh enforces security policies, and handles authentication, authorization, and encryption
 - ensuring secure communication between services and eventually, strengthening the overall security posture of the application.

Service Discovery

- With service discovery features, service mesh can simplify the process of locating and routing services dynamically
- adapting to system changes seamlessly. This enables easier management and interaction between services.

Overall benefits

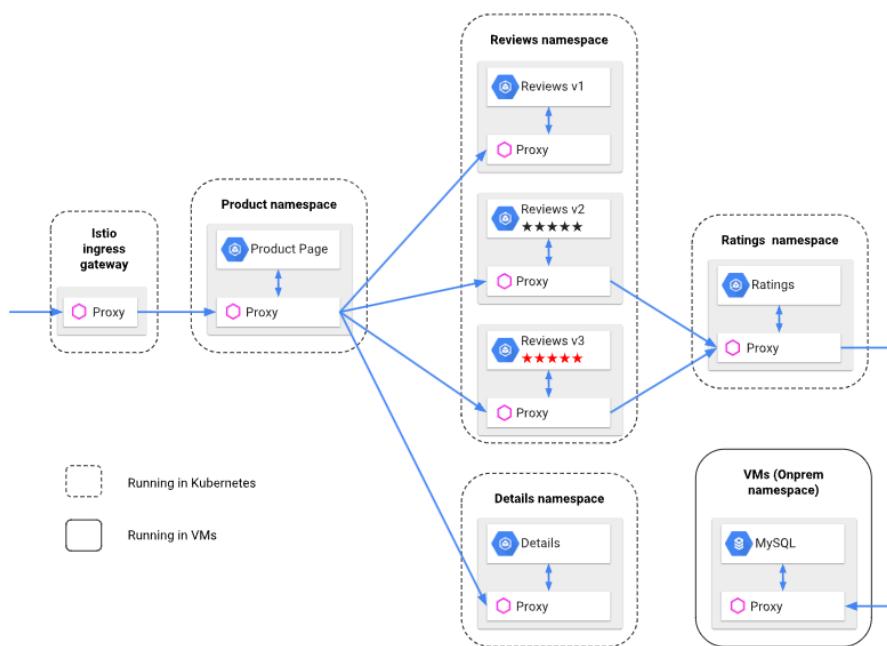
Microservices communication:

Adopting a service mesh can simplify the implementation of a microservices architecture by abstracting away infrastructure complexities.

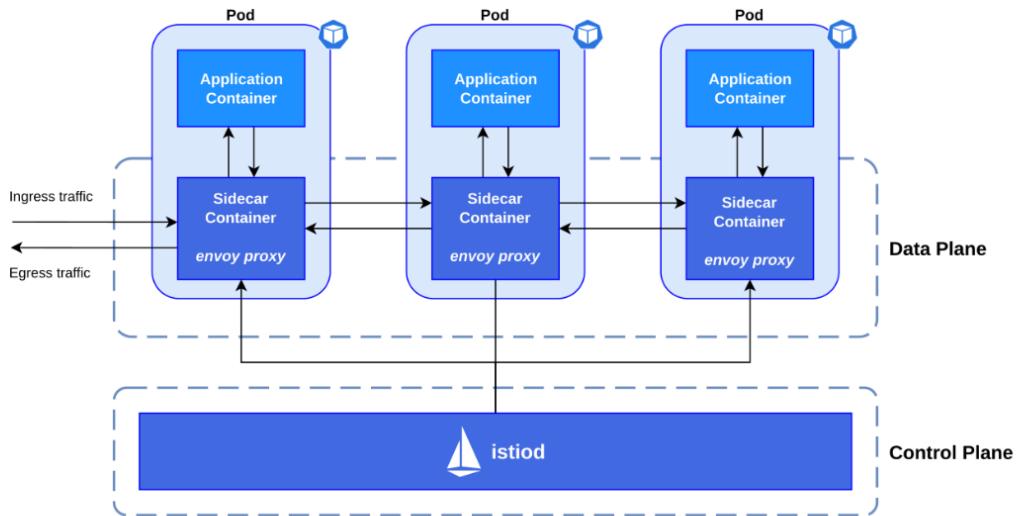
It provides a standardized approach to manage and orchestrate communication within the microservices ecosystem.

How does a ServiceMesh work? (example istio)

Overview



Istio control plane and data plane



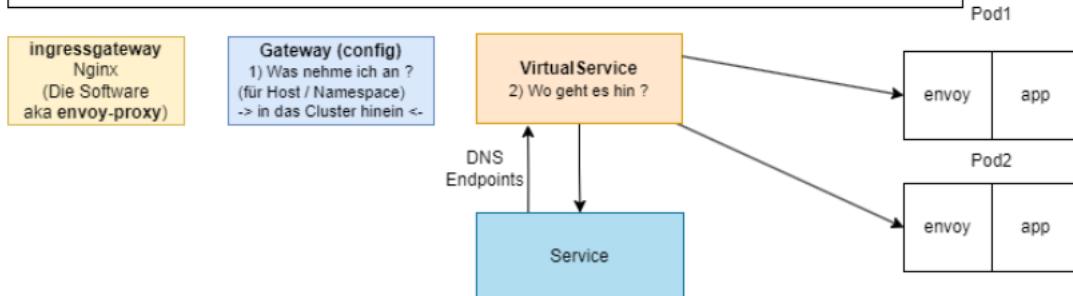
• Source: kubebyexample.com

Istio vs. ingress

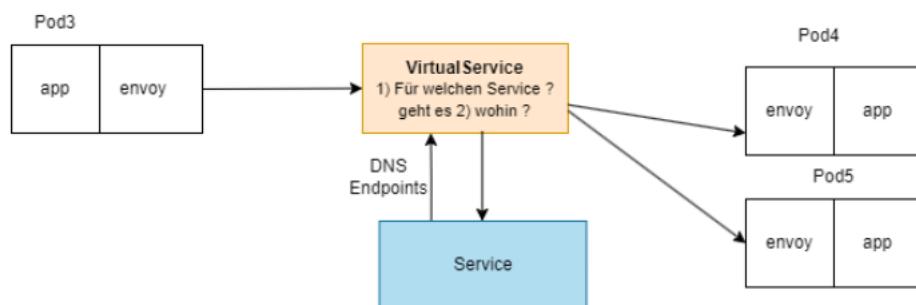
Klassisch Ingress (Kubernetes)



Istio (Traffic in das Cluster hinein)

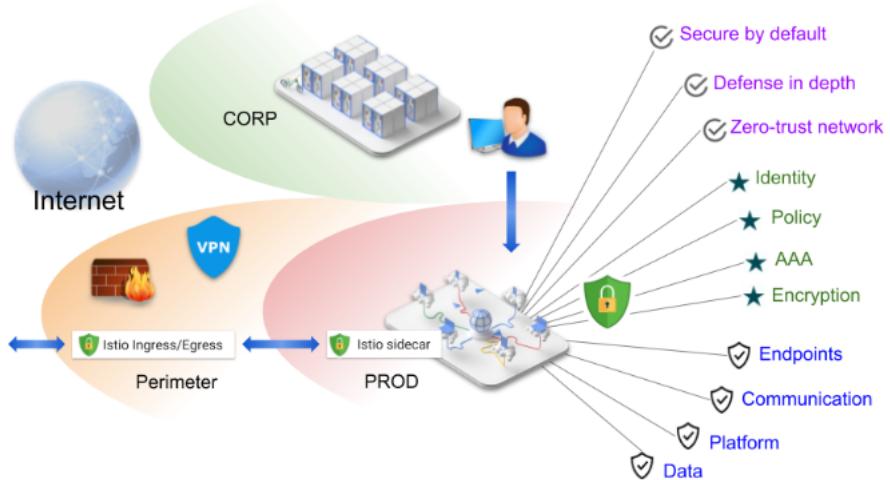


Istio (Innerhalb des Clusters)



istio security features

Overview



Security overview

Security needs of microservices

- To defend against man-in-the-middle attacks, they need traffic encryption.
- To provide flexible service access control, they need mutual TLS and fine-grained access policies.
- To determine who did what at what time, they need auditing tools.

Implementation of security

The Istio security features provide strong identity, powerful policy, transparent TLS encryption, and authentication, authorization and audit (AAA) tools to protect your services and data. The goals of Istio security are:

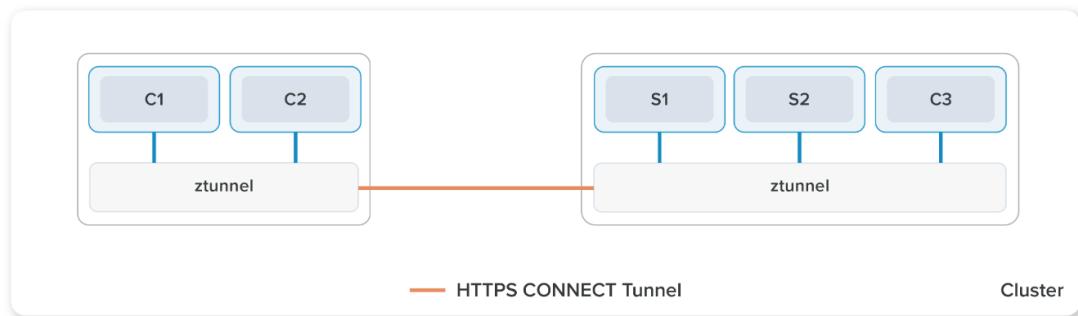
- Security by default: no changes needed to application code and infrastructure
- Defense in depth: integrate with existing security systems to provide multiple layers of defense
- Zero-trust network: build security solutions on distrusted networks

istio-service mesh - ambient mode

Light: Only Layer 4 per node (ztunnel)

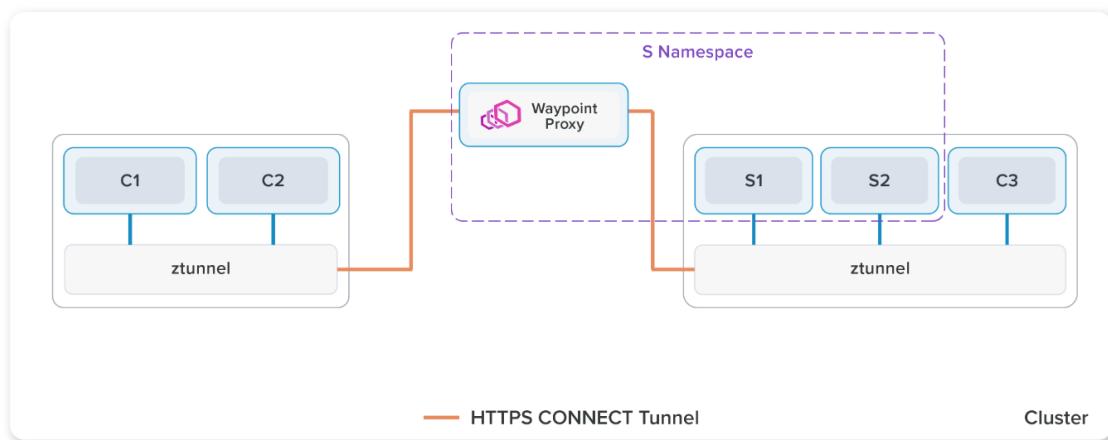
- No sidecar (envoy-proxy) per Pod, but one ztunnel agent per Node (Layer 4)
- Enables security features (mtls, traffic encryption)

Like so:



Full fledged: Layer 4 (ztunnel) per Node & Layer 7 per Namespace

- One waypoint - proxy is rolled out per Namespace, which connects to the ztunnel agents



When additional features are needed, ambient mesh deploys waypoint proxies, which ztunnels connect through for policy enforcement

Features in "fully-fledged" - ambient - mode

| Application deployment use case | Ambient mode configuration |
|---|------------------------------|
| Zero Trust networking via mutual-TLS, encrypted and tunneled data transport of client application traffic, L4 authorization, L4 telemetry | ztunnel only (default) |
| As above, plus advanced Istio traffic management features (including L7 authorization, telemetry and VirtualService routing) | ztunnel and waypoint proxies |

Advantages:

- Less overhead
- One can start step-by-step moving towards a mesh (Layer 4 firstly and if wanted Layer 7 for specific namespaces)
- Old pattern: sidecar and new pattern: ambient can live side by side.

Performance comparison - baseline,sidecar,ambient

- <https://livewyver.io/blog/2024/06/06/comparison-of-service-meshes-part-two/>
- <https://github.com/livewyver-ops/poc-servicemesh2024/blob/main/docs/test-report-istio.md>

Kubernetes Ingress

Ingress HA-Proxy Sticky Session

It's easy to setup session stickyness

- <https://www.haproxy.com/documentation/kubernetes-ingress/ingress-tutorials/load-balancing/>

Nginx Ingress Session Stickyness

Yes, session stickiness (affinity) via cookie-based stickiness does work with the open-source NGINX Ingress Controller.

Here's what you need to know to get it working:

✓ How to Enable Sticky Sessions with Open Source NGINX Ingress

The open-source NGINX Ingress Controller supports sticky sessions using **annotations** and **cookies**.

Example Ingress YAML:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/session-cookie-name: "route"
    nginx.ingress.kubernetes.io/session-cookie-hash: "sha1"
spec:
  ingressClassName: nginx
  rules:
    - host: myapp.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-app-service
                port:
                  number: 80
```

Explanation of the Annotations

- `nginx.ingress.kubernetes.io/affinity: "cookie"`
→ Enables cookie-based affinity.
- `nginx.ingress.kubernetes.io/session-cookie-name: "route"`
→ Names the session cookie (optional, default is `INGRESSCOOKIE`).
- `nginx.ingress.kubernetes.io/session-cookie-hash: "sha1"`
→ Defines how the cookie value is hashed. Can be `md5`, `sha1`, `index`, etc.

You can also optionally define:

```
nginx.ingress.kubernetes.io/session-cookie-path: "/"
nginx.ingress.kubernetes.io/session-cookie-expire: "172800"  # in seconds
nginx.ingress.kubernetes.io/session-cookie-max-age: "172800"
```

Important Notes

- This works **only with the open-source NGINX ingress controller**.
(The annotations won't apply to controllers like Traefik, HAProxy, or cloud-specific ones like AWS ALB.)
- You must ensure that **all replicas of the app can handle the traffic** if they receive the same cookie.
- This stickiness is **not persistent across restarts** unless the session cookie is preserved client-side (and the pod labels/service don't change unpredictably).

Bonus Tip: Enable Ingress Logging (Optional but Useful)

To troubleshoot or verify stickiness:

```
controller:
  config:
    enable-access-log: "true"
    log-format-upstream: "$request_id $remote_addr to $upstream_addr via $cookie_route"
```

Want a ready-made Helm values file or `kubectl` manifest for this?

Flow Overview: Cookie-Based Stickiness in NGINX Ingress

```
[Client Browser]
  " 
[NGINX Ingress Controller (Ingress Pod)]
  "
[Kubernetes Service (ClusterIP)]
  "
[App Pod A / B / C (Behind the Service)]
```

What Actually Happens (Step-by-Step)

1. Client makes a request

- Let's say the client hits `myapp.example.com`.

2. Ingress Controller (NGINX) receives the request

- NGINX is exposed via a LoadBalancer, NodePort, or IngressClass.
- It parses the Ingress resource and applies sticky session rules based on annotations.

3. First Request: No Cookie

- No session cookie is present, so:
 - NGINX picks a backend pod **randomly** via the Kubernetes Service.
 - It sets a sticky cookie on the response, e.g.:

```
Set-Cookie: route=backend1; Path=/; HttpOnly
```

4. Subsequent Requests: Cookie Present

- On later requests, the client sends back:

```
Cookie: route=backend1
```

- NGINX uses this cookie value to route to the **same backend pod**.

So Where Does the Kubernetes Service Come In?

The Kubernetes Service is used **internally by NGINX to proxy requests to pods**.

NGINX configuration looks like this (simplified):

```
upstream myapp-service {
    sticky cookie route;
    server 10.0.1.2:8080; # Pod A
    server 10.0.1.3:8080; # Pod B
}
```

- These IPs are discovered via the **Kubernetes Service** using Endpoints or EndpointSlices.
- NGINX tracks these pods and their IPs automatically (via a sync controller loop).

✓ Who Makes Routing Decisions?

- NGINX Ingress Controller makes the routing decision **based on the cookie value**, not Kubernetes.
- Kubernetes Service is just a **source of backend pod IPs**, not the router in this case.

[https mit ingressController und Letsencrypt](https://mit.ingressController.und.Letsencrypt)

Schritt 1: cert-manager installieren

```
helm repo add jetstack https://charts.jetstack.io
helm install cert-manager jetstack/cert-manager \
--namespace cert-manager --create-namespace \
--version v1.19.1 \
--set installCRDs=true
```

- Ref: <https://artifacthub.io/packages/helm/cert-manager/cert-manager>

Schritt 2: Create ClusterIssuer (gets certificates from Letsencrypt)

```
cd
mkdir -p manifests/cert-manager
cd manifests/cert-manager
nano cluster-issuer.yaml
```

```
## cluster-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
```

```

spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: your-email@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: nginx

kubectl apply -f .
## Should be True
kubectl get clusterissuer

```

Schritt 3: Ingress-Objekt mit TLS erstellen

```

## tls-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - test.devopslearnwith.us
      secretName: example-tls
  rules:
    - host: test.devopslearnwith.us
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-service
              port:
                number: 80

```

Schritt 4: Herauszufinden, ob Zertifikate erstellt werden

```

kubectl describe certificate example-tls
kubectl get cert

```

Schritt 5: Testen

Ref:

- <https://hbayraktar.medium.com/installing-cert-manager-and-nginx-ingress-with-lets-encrypt-on-kubernetes-fe0dff4b1924>

Kubernetes Pod Termination

LifeCycle Termination

- <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-termination>

preStopHook

- <https://www.datree.io/resources/kubernetes-guide-graceful-shutdown-with-lifecycle-prestop-hook>

How to wait till a pod gets terminated

When stopping (terminating) a pod in Kubernetes, there **are hooks available**, but they're limited to the **termination lifecycle**. If you're thinking of stopping a pod *without* killing it and still triggering hooks, that's trickier (more on that below). Here's a full breakdown of what's available:

✓ Kubernetes Hooks for Pod Termination

1. preStop Hook

Executed **before the container is terminated** (but *after* the SIGTERM signal is sent).

Example:

```

lifecycle:
  preStop:

```

```
exec:  
  command: ["/bin/sh", "-c", "echo pre-stop hook running; sleep 10"]
```

- You can use this to delay shutdown or do cleanup.
- Note: Kubernetes still sends `SIGTERM` as normal; `preStop` doesn't block that, but the container stays up for the duration of the hook.

2. `terminationGracePeriodSeconds`

Specifies how long Kubernetes will wait before force-killing the container with `SIGKILL`.

```
spec:  
  terminationGracePeriodSeconds: 30
```

- Combine this with `preStop` to allow your container to finish in-flight work or deregister from a service.

If you tell me **what kind of "stop" you're envisioning**, I can help tailor a hook-like workaround. For example:

- "I want to stop background work when CPU is high"
- "Pause processing until a Kafka topic is ready"— totally different tactics.

Kubernetes Security

Best practices security pods

5. Security / Best practice pods

5.1. Pods

1) Use Readiness / Liveness check

Not we really security, but to have a stable system

2) Use Non-Root Images

(is not allowed in OpenShift anyways)

3) SecurityContext: Restrict the Features in the pod/container as much as possible

Essentially covered by Default SCC's:

<https://docs.openshift.com/container-platform/4.18/authentication/managing-security-context-constraints.html>

Essentially use the v2 versions.

Question will Always be: Do I really Need this for this post
(e.g. HostNetwork). Is there are better/safer way to achieve this

Best practices in general

6. Security (other stuff)

6.1. Be sure upgrade your system and use the newest versions (OS / OpenShift)

6.2. Setup Firewall rules, for the cluster components. (OpenShift) -

https://docs.openshift.com/container-platform/4.16/installing/install_config/configuring-firewall.html

6.3. Do not install any components, that you do not Need (with helm)

6.4. Always download Images instead of using them locally.

I think it also has to do with auth. When set to always, the pod will pull the image from the registry, hence it has to do auth and have valid credentials to actually get the image.

If the image is already in the node, and let's say permission has been removed to access that image for that node in the registry, a pod could still be created since the image is already there.

-> Wie sicherstellen, dass das gesetzt ist ?

OPA Gateway

6.5. Scan all your Images before using them

6.5.1. In development

6.5.2. CI / CD Pipeline

6.5.3 Registry (when uploading them)

6.6. Restrict ssh Access

(no ssh-access to cluster nodes please !)

6.7. Use NetworkPolicies

```
https://docs.openshift.com/container-platform/4.12/networking/network\_policy/about-network-policy.html
-> BUT: Use the specific Network Policies of your CNI
```

```
### Images in kubernetes von privatem Repo verwenden
```

```
* Zugriff auf registries mit authentifizierung
```

```
### Exercise
```

```
mkdir -p manifests cd manifests mkdir private-repo cd private-repo
```

```
kubectl create secret docker-registry regcred --docker-server=registry.do.t3isp.de
--docker-username=11trainingdo --docker-password= --dry-run=client -o yaml > 01-secret.yaml
```

```
kubectl create secret generic mariadb-secret --from-literal=MARIADB_ROOT_PASSWORD=11abc432 --dry-run=client -o yaml > 02-secret.yaml
```

```
nano 02-pod.yaml
```

```
apiVersion: v1 kind: Pod metadata: name: private-reg spec: containers:
```

- name: private-reg-container image: registry.do.t3isp.de/mariadb:11.4.5 envFrom:
 - secretRef: name: mariadb-secret

```
imagePullSecrets:
```

- name: regcred

```
kubectl apply -f . kubectl get pods -o wide private-reg kubectl describe pods private-reg
```

Kubernetes Monitoring/Security

Überwachung, ob Images veraltet sind, direkt in Kubernetes

- Can also update images (i would always go towards gitlab ci/cd doing this)
- Kann z.B. über slack benachrichtigen

Setup (Achtung ungetestet)

```
## Korrekte Struktur für aktuelle Keel Version
helmProvider:
  enabled: true

## Korrekte Notification-Konfiguration
notification:
  slack:
    enabled: true
    token: "xoxb-YOUR-TOKEN"
    channel: "#updates"

## Korrekte Approval-Konfiguration
approvals:
  enabled: true

## Korrekte Trigger-Konfiguration
triggers:
  poll:
    enabled: true
  pubsub:
    enabled: false
```

```
helm repo add keel https://charts.keel.sh
helm repo update
```

```
## 2. Installation mit der values.yaml
helm install keel keel/keel \
--namespace keel \
```

```
--create-namespace \
-f values.yaml
```

Helm (IDE - Support)

Kubernetes-Plugin IntelliJ

- <https://www.jetbrains.com/help/idea/kubernetes.html>

IntelliJ - Helm Support Through Kubernetes Plugin

- <https://blog.jetbrains.com/idea/2018/10/intellij-idea-2018-3-helm-support/>

Helm - Charts entwickeln

Unser erstes Helm Chart erstellen

Chart erstellen

```
cd
mkdir my-charts
cd my-charts
```

```
helm create my-app
```

Install helm - chart

```
## Variante 1:
helm -n my-app-<namenskuerzel> install meine-app my-app --create-namespace
```

```
## Variante 2:
cd my-app
helm -n my-app-<namenskuerzel> install meine-app . --create-namespace
```

```
kubectl -n my-app-<namenskuerzel> get all
kubectl -n my-app-<namenskuerzel> get pods
```

Wie starte ich am besten - Übung

Exercise

```
cd
mkdir -p my-charts
cd my-charts
helm create simple-chart
```

```
## Alles Weg was wir nicht brauchen
cd simple-chart
rm values.yaml
cd templates
rm -f *.yaml
rm -fR tests
echo "Ausgabe nach Install" > NOTES.txt
```

```
nano deploy.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8 # tells deployment to run 8 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.26
          ports:
            - containerPort: 80
```

```

## aus dem templates ordner raus
cd ..
## aus dem chart raus
cd ..

## Installieren
helm -n my-simple-app-<namenskuerzel> upgrade --install my-simple-app simple-chart --create-namespace
kubectl -n my-simple-app-<namenskuerzel> get all

```

Exercise Phase 2: Um Replicas erweitern

```

cd simple-chart
nano values.yaml

deployment:
  replicas: 5

cd templates
nano deploy.yaml

## aus der Zeile:
## replicas: 9
## wird ->
replicas: {{ .Values.deployment.replicas }}

# Gehen aus dem Chart raus
cd ..
cd ..
helm template simple-chart
helm -n my-simple-app-<namenskuerzel> upgrade --install my-simple-app simple-chart --create-namespace
kubectl -n my-simple-app-<namenskuerzel> get pods

nano simple-app-values.yaml

deployment:
  replicas: 2

helm -n my-simple-app-<namenskuerzel> upgrade --install my-simple-app simple-chart -f simple-app-values.yaml
kubectl -n my-simple-app-<namenskuerzel> get pods

```

Helm und Kustomize kombinieren

Helm und Kustomize kombinieren

Übersicht

Die Kombination von Helm und Kustomize bietet die Flexibilität von Kustomize mit der Paketierung und Versionierung von Helm. Dies ist besonders nützlich für komplexe Deployments, die environment-spezifische Anpassungen benötigen.

Helm Post-Rendering mit Kustomize

Grundlegendes Konzept

Helm kann nach dem Template-Rendering einen Post-Renderer aufrufen. Hier kann Kustomize die gerenderten Manifeste weiter anpassen.

Workflow

1. Helm rendert Templates basierend auf Values
2. Kustomize modifiziert die gerenderten Manifeste
3. Finale Manifeste werden deployed

Übung

Schritt 1: Arbeitsverzeichnis erstellen

```

## Erstelle Arbeitsverzeichnis
cd
mkdir helm-kustomize-demo
cd helm-kustomize-demo

```

Schritt 2: Helm Chart erstellen

```

## Erstelle ein neues Helm Chart
helm create my-chart

```

Schritt 3: Kustomize-Verzeichnis erstellen

```
## Erstelle Kustomize-Verzeichnis  
mkdir kustomize  
cd kustomize
```

Schritt 4: Post-Renderer Script erstellen

```
## Erstelle das Post-Renderer Script  
cat > kustomize-post-renderer.sh << 'EOF'  
#!/bin/bash  
## Wechsle ins kustomize Verzeichnis  
cd "$(dirname "$0")"  
## Speichere Helm Output als base.yaml  
cat <&0 > base.yaml  
## Führe kustomize build aus  
kubectl kustomize .  
EOF  
  
## Script ausführbar machen  
chmod +x kustomize-post-renderer.sh
```

Schritt 5: Patches-Verzeichnis erstellen

```
## Erstelle patches Verzeichnis  
mkdir -p patches
```

Schritt 6: Deployment Patch erstellen

```
## Erstelle deployment-patch.yaml  
cat > patches/deployment-patch.yaml << 'EOF'  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: my-app-my-chart  
spec:  
  template:  
    spec:  
      containers:  
      - name: my-chart  
        resources:  
          requests:  
            memory: "80Mi"  
            cpu: "300m"  
          limits:  
            memory: "80Mi"  
            cpu: "300m"  
EOF
```

Schritt 7: Kustomization.yaml erstellen

```
## Erstelle kustomization.yaml  
cat > kustomization.yaml << 'EOF'  
apiVersion: kustomize.config.k8s.io/v1beta1  
kind: Kustomization  
  
resources:  
- base.yaml  
  
patches:  
- path: patches/deployment-patch.yaml  
  
images:  
- name: nginx  
  newTag: "1.21"  
EOF
```

Schritt 8: Zurück ins Hauptverzeichnis

```
## Gehe zurück ins Hauptverzeichnis  
cd ..
```

Schritt 9: Verzeichnisstruktur prüfen

```
## Prüfe die Verzeichnisstruktur
tree .
## Ergebnis sollte sein:
## .
## └── kustomize/
##     ├── kustomization.yaml
##     ├── kustomize-post-renderer.sh
##     └── patches/
##         └── deployment-patch.yaml
## └── my-chart/
##     ├── Chart.yaml
##     ├── charts/
##     ├── templates/
##     └── values.yaml
```

Schritt 10: Deployment testen

```
## Teste das Setup mit dry-run
helm upgrade --install -n my-kapp-<namenskuerzel> my-app ./my-chart --post-renderer ./kustomize/kustomize-post-renderer.sh --
dry-run --debug --create-namespace
```

Schritt 11: Deployment ausführen

```
## Führe das Deployment aus
helm upgrade --install -n my-kapp-<namenskuerzel> my-app ./my-chart --post-renderer ./kustomize/kustomize-post-renderer.sh --
--create-namespace

helm -n my-kapp-<namenskuerzel> list
helm -n my-kapp-<namenskuerzel> get manifest my-app
```

Schritt 12: Deployment prüfen

```
## Prüfe das Deployment
kubectl -n my-kapp-<namenskuerzel> get pods
kubectl -n my-kapp-<namenskuerzel> describe deployment my-app-my-chart
```

Environment-spezifische Anpassungen

Ordnerstruktur

```
helm-kustomize/
├── chart/
│   ├── Chart.yaml
│   ├── values.yaml
│   └── templates/
└── environments/
    ├── dev/
    │   ├── kustomization.yaml
    │   └── patches/
    ├── staging/
    │   ├── kustomization.yaml
    │   └── patches/
    └── prod/
        ├── kustomization.yaml
        └── patches/
└── scripts/
    └── deploy.sh
```

Development Environment

```
## environments/dev/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base.yaml

patches:
- path: patches/dev-resources.yaml

replicas:
```

```
- name: my-app
  count: 1

commonLabels:
  environment: dev
```

Production Environment

```
## environments/prod/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base.yaml

patches:
- path: patches/prod-resources.yaml
- path: patches/prod-security.yaml

replicas:
- name: my-app
  count: 3

commonLabels:
  environment: prod
```

Best Practices

2. Testing

```
## Dry-run für Testing
helm template my-app ./chart --values values-dev.yaml | \
kubectl kustomize environments/dev | \
kubectl apply --dry-run=client -f -
```

LoadBalancer on Premise (metallb)

MetalLB

General

- Supports bgp and arp
- Divided into controller, speaker

Installation Ways

- helm
- manifests

Step 1: install metallb

```
## Just to show some basics
## Page from metallb says that digitalocean is not really supported well
## So we will not install the speaker .

helm repo add metallb https://metallb.github.io/metallb

## Eventually disabling speaker
## vi values.yml

helm install metallb metallb/metallb --namespace=metallb-system --create-namespace --version 0.14.8
```

Step 2: addresspool und Propagation-type (config)

```
cd
mkdir -p manifests
cd manifests
mkdir lb
cd lb
nano 01-addresspool.yml
```

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
```

```
spec:  
  addresses:  
    # we will use our external ip here  
    - 134.209.231.154-134.209.231.154  
    # both notations are possible  
    - 157.230.113.124/32
```

```
kubectl apply -f .
```

```
nano 02-advertisement.yml
```

```
apiVersion: metallb.io/vbeta1  
kind: L2Advertisement  
metadata:  
  name: example  
  namespace: metallb-system
```

```
kubectl apply -f .
```

Schritt 4: Test do i get an external ip

```
nano 03-deploy.yml
```

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

```
nano 04-service.yml
```

```
apiVersion: v1  
kind: Service  
metadata:  
  name: svc-nginx  
  labels:  
    svc: nginx  
spec:  
  type: LoadBalancer  
  ports:  
  - port: 80  
    protocol: TCP  
  selector:  
    run: web-nginx
```

```
kubectl apply -f .  
kubectl get pods  
kubectl get svc
```

```
## auf dem client  
curl http://<ip aus get svc>
```

```
kubectl delete -f 03-deploy.yml 04-service.yml
```

Schritt 5: Referenz:

- <https://metallb.io/installation/#installation-with-helm>

Helm mit gitlab ci/cd

Helm mit gitlab ci/cd ausrollen

Step 1: Create gitlab - repo and pipeline

1. Create new repo on gitlab
2. Click on pipeline Editor and creat .gitlab-ci.yml with Button

Step 2: Push your helm chart files to repo

- Now looks like this

training-helm-chart-kubernetes-gitlab-ci-cd

| Name | Last commit |
|----------------|----------------------------|
| charts/my-app | initial release |
| config | Add new file |
| .gitlab-ci.yml | Update .gitlab-ci.yml file |

Step 3: Add your KUBECONFIG as Variable (type: File) to Variables

- https://gitlab.com/jmetzger/training-helm-chart-kubernetes-gitlab-ci-cd/-/settings/ci_cd#js-cicd-variables-settings

ochen Metzger / Training Helm Chart Kubernetes Gitlab Ci Cd / CI/CD Settings

Maintainer
 Developer

Save changes

Project variables

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature protects variable values, but is not a guaranteed method to prevent malicious users from accessing variables. [How can I make my variables secure?](#)

| CI/CD Variables | Value | Enviro |
|-------------------|-------|----------|
| KUBECONFIG_SECRET | ***** | All (de) |

Pipeline trigger tokens
Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates the user who triggered the pipeline.

Deploy freezes
Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deploy freezes added here. [Learn more](#). Specify deploy freezes using [cron syntax](#).

Job token permissions

Edit variable

Type: File

Environments: All (default)

Visibility: Visible
Can be seen in job logs.

Masked
Masked in job logs but value can be revealed in CI/CD settings. Requires values to meet [regular expressions requirements](#).

Masked and hidden
Masked in job logs, and can never be revealed in the CI/CD settings after the variable is saved.

Flags:

Protect variable
Export variable to pipelines running on protected branches and tags only.

Expand variable reference
\$ will be treated as the start of a reference to another variable.

Description (optional): ACCESS TO KUBERNETES

The description of the variable's value or usage.

Step 4: Create a pipeline for deployment

```

stages:          # List of stages for jobs, and their order of execution
- deploy

variables:
APP_NAME: my-first-app

deploy:
stage: deploy
image:
name: alpine/helm:3.2.1
## Important to unset entrypoint
entrypoint: []
script:
- ls -la
- cd; mkdir .kube; cd .kube; cat $KUBECONFIG_SECRET > config; ls -la;
- cd $CI_PROJECT_DIR; helm upgrade ${APP_NAME} ./charts/my-app --install --namespace ${APP_NAME} --create-namespace -f
./config/values.yaml
rules:
- if: $CI_COMMIT_BRANCH == 'master'
when: always

```

Reference: Example Project (Public)

- <https://gitlab.com/metzger/training-helm-chart-kubernetes-gitlab-ci-cd>

Kubernetes Verlässlichkeit erreichen

Keine 2 pods auf gleichem Node - PodAntiAffinity

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchLabels:
                  app: my-app
                topologyKey: kubernetes.io/hostname
            containers:
              - name: my-app
                image: nginx:latest
                ports:
                  - containerPort: 80

```

Metrics-Server / Große Cluster

Metrics-Server mit helm installieren

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.

Walkthrough

```

helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm -n kube-system upgrade --install metrics-server metrics-server/metrics-server --version 3.13.0

```

```
## 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>

Speichernutzung und CPU berechnen für Anwendungen

- <https://learnk8s.io/kubernetes-node-size>

Kubernetes -> High Availability Cluster (multi-data center)

High Availability multiple data-centers

What needs to be there ?

- etcd in einer ungeraden Zahl an Rechenzentrum in ungerade Zahl, d.h. z.B. 1 etcd in 3 Rechenzentren = 3 etcd (ungerade)
 - etcd kann auch in einem Rechenzentrum in der Cloud sein, in der nur etcd läuft und keine anderen Komponenten (Schiedsrichter)
- Control Plane in jedem Rechenzentrum, was verwendet wird (nicht im "Schiedsrichter - Rechenzentrum")
- HA für ControlPlane (entweder HA Proxy oder kube-vip (ist dann LoadBalancer im Cluster kubernetes-native))

Ausblick für Kubernetes Applikationen selbst

- Pods ausgerollt über (Deployment etc.) dürfen auch nicht nur in einem Rechenzentrum (pod anti affinity)
- IngressController in jedem Rechenzentrum (einfachste Variante : DaemonSet)

PodAntiAffinity für Hochverfügbarkeit

PodAffinity

Exercise

```
## Deployment 1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-a
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app-a
  template:
    metadata:
      labels:
        app: app-a
    spec:
      containers:
        - name: app-a
          image: nginx
```

```
## Deployment 2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-b
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app-b
  template:
    metadata:
      labels:
        app: app-b
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - app-a
            topologyKey: "kubernetes.io/hostname"
```

```

containers:
- name: app-b
  image: nginx

## Kubernetes -> etcd

### etcd - cleaning of events

## Who cleans up events in etc, kubernetes api server ?

### Overview
No, the **Kubernetes API server** does not directly clean up events in etcd**.

#### Who Cleans Up Events in etcd?
- **The Kubernetes controller manager** is responsible for cleaning up expired events, **not the API server**.
- The **event garbage collection** process runs periodically to remove events whose **TTL (Time-to-Live)** has expired.
- The **default TTL is 1 hour**, but this can be configured using the `--event-ttl` flag in the API server settings.

#### How Does Event Cleanup Work?
1. **Event is Created** → Stored in `etcd`.
2. **TTL Countdown Begins** → Typically 1 hour unless configured otherwise.
3. **Event Expires** → Becomes eligible for deletion.
4. **Garbage Collection (Controller Manager)** → Detects expired events and removes them from `etcd`.

#### Key Takeaway:
- The API server **stores and serves** event data from `etcd`, but it **does not handle cleanup**.
- The **controller manager** is responsible for **event garbage collection** after TTL expiration.

### etcd in multi-data-center setup

### Wieviele ?

* Ungerade Zahl an etcd - Nodes (3,5,7 max. 7)
* Ausreichend in der Regel 3
* Wenn man möchte, dass 2 ausfallen können dann 5

### Besonderheiten bei der Nutzung der etc.

* Schnelle Platte, idealerweise ssd .
* Begrenzung des Key->Values Stores auf 2,1 GB (standardmäßig)

### Besonderheiten multi-data-center Setup

* Ursprünglich nicht dafür entwickelt.
* Sowohl ungerade Zahl an etcd-Nodes als auch ungerade Zahl an Rechenzentren.
* Ideal ist ein RTT (round trip time) von 10 ms / (maximal 100ms / 1.5 => ca. 66,6 ms)

### etcd

* Tuning: https://etcd.io/docs/v3.4/tuning/
* Maintenance: https://etcd.io/docs/v3.5/op-guide/maintenance/

## Kubernetes Storage

### Praxis. Beispiel (Dev/Ops)

### 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)
```

knodel1

```
/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) (Version 1)
```

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
```

```
### Setup PersistentVolume and PersistentVolumeClaim in cluster
```

```
#### Schritt 1:
```

```
cd cd manifests mkdir -p nfs; cd nfs nano 01-pv.yml
```

```
apiVersion: v1 kind: PersistentVolume metadata:
```

any PV name

```
name: pv-nfs-tln labels: volume: nfs-data-volume-tln 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/nginx server: 10.135.0.7 readOnly: false storageClassName: ""
```

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

```
#### Schritt 2:
```

```
nano 02-pvc.yml
```

vi 02-pvc.yml

now we want to claim space

```
apiVersion: v1 kind: PersistentVolumeClaim metadata: name: pv-nfs-claim-tln spec: storageClassName: "" volumeName: pv-nfs-tln accessModes:
```

- ReadWriteMany resources: requests: storage: 1Gi

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

```
#### Schritt 3:
```

```
nano 03-deploy.yml
```

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<nr>
```

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

```
nano 04-service.yml
```

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
```

```
#### Schritt 4
```

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
```

get external ip

```
kubectl get nodes -o wide
```

now try to connect

```
kubectl get svc
```

connect with ip and port

```
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
```

curl http://

exit

oder alternative von extern (Browser) auf Client

```
http://30154 (Node Port) - ext-ip -> kubectl get nodes -o wide
```

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://

exit

```
#### Schritt 5
```

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://: # port -> > 30000

exit

```
## Kubernetes Netzwerk

### Kubernetes Netzwerke Übersicht

### Show us

![pod to pod across nodes] (https://www.inovex.de/wp-content/uploads/2020/05/Pod-to-Pod-Networking.png)

### Die Magie des Pause Containers

![Overview Kubernetes Networking] (https://www.inovex.de/wp-content/uploads/2020/05/Container-to-Container-Networking_3_neu-400x412.png)

### CNI

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

### Docker - Container oder andere

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

### Welche gibt es ?

* Flannel
* Canal
* Calico
* Cilium

### Flannel

#### Overlay - Netzwerk

* virtuelles Netzwerk was sich oben darüber und eigentlich auf Netzwerkebene nicht existiert
* VXLAN

#### Vorteile

* Guter einfacher Einstieg
* reduziert auf eine Binary flanneld

#### Nachteile

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

```

### Canal

#### General
* Auch ein Overlay - Netzwerk
* Unterstützt auch policies

### Calico

#### Generell
* klassische Netzwerk (BGP)

#### Vorteile gegenüber Flannel
* Policy über Kubernetes Object (NetworkPolicies)

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

#### Referenz
* https://projectcalico.docs.tigera.io/security/calico-network-policy

### Cilium

#### Generell

### 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.

```

### DNS - Resolution - Services

### Exercise

kubectl run podtest --rm -ti --image busybox

### Example with svc-nginx

```

in sh

```
wget -O http://svc-nginx wget -O http://svc-nginx.jochen wget -O http://svc-nginx.jochen.svc wget -O http://svc-nginx.jochen.svc.cluster.local
```

```
### How to find the FQDN (Full qualified domain name)
```

in busybox (clusterIP)

Schritt 1: Service-IP ausfindig machen

```
wget -O http://svc-nginx
```

z.B. 10.109.24.227

Schritt 2: nslookup mit dieser Service-IP

```
nslookup 10.109.24.227
```

Ausgabe

name = svc-nginx.jochen.svc.cluster.local

```
### Kubernetes Firewall / Cilium Calico
```

```
### Um was geht es ?  
* Wir wollen Firewall-Regeln mit Kubernetes machen (NetworkPolicy)  
* Firewall in Kubernetes -> Network Policies
```

```
### Gruppe mit eigenem cluster
```

= nix z.B. policy-demo => policy-demo

```
### Gruppe mit einem einzigen Cluster
```

= Teilnehmernummer
z.B. policy-demo => policy-demo1

```
### Walkthrough
```

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 ist
```

```
cd cd manifests mkdir network cd network nano 01-policy.yml
```

Deny Regel

```
kind: NetworkPolicy apiVersion: networking.k8s.io/v1 metadata: name: default-deny namespace: policy-demo spec: podSelector: matchLabels: {}
```

```
kubectl apply -f 01-policy.yml
```

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

kein Zugriff möglich

```
wget -O - nginx
```

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

```
cd cd manifests cd network nano 02-allow.yml
```

Schritt 3:

02-allow.yml

```
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
```

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

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  
* https://kubernetes.io/docs/concepts/services-networking/network-policies/  
* https://docs.cilium.io/en/latest/security/policy/language/#http  
  
### Sammlung istio/mesh  
  
### Schaubild  
  
![istio Schaubild](https://istio.io/latest/docs/examples/virtual-machines/vm-bookinfo.svg)  
  
### 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
```

```
### istio tls  
* https://istio.io/latest/docs/ops/configuration/traffic-management/tls-configuration/
```

```
### istio - the next generation without sidecar
* https://istio.io/latest/blog/2022/introducing-ambient-mesh/
## Kubernetes NetworkPolicy (Firewall)
### Kubernetes Network Policy Beispiel

### Schritt 1: Deployment und Service erstellen
```

KURZ=jm kubectl create ns policy-demo-\$KURZ

```
cd mkdir -p manifests cd manifests mkdir -p np cd np
```

nano 01-deployment.yaml

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 1 template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.23 ports: - containerPort: 80
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

nano 02-service.yaml

```
apiVersion: v1 kind: Service metadata: name: nginx spec: type: ClusterIP # Default Wert ports:
  • port: 80 protocol: TCP selector: app: nginx
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

```
### Schritt 2: Zugriff testen ohne Regeln
```

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

```
kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox
```

innerhalb der shell

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

Optional: Pod anzeigen in 2. ssh-session zu jump-host

```
kubectl -n policy-demo-$KURZ get pods --show-labels
```

```
### Schritt 3: Policy festlegen, dass kein Zugriff erlaubt ist.
```

nano 03-default-deny.yaml

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

in diesem namespace: policy-demo-\$KURZ

```
kind: NetworkPolicy apiVersion: networking.k8s.io/v1 metadata: name: default-deny spec: podSelector: matchLabels: {}
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

```
### Schritt 3.5: Verbindung mit deny all Regeln testen
```

```
kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox
```

innerhalb der shell

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

```
### Schritt 4: Zugriff erlauben von pods mit dem Label run=access (alle mit run gestarteten pods mit namen access haben dieses label per default)
```

nano 04-access-nginx.yaml

```
apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: access-nginx spec: podSelector: matchLabels: app: nginx ingress: - from: - podSelector: matchLabels: run: access
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

```
### Schritt 5: Testen (zugriff sollte funktionieren)
```

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-$KURZ access --rm -ti --image busybox
```

innerhalb der shell

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

```
### Schritt 6: Pod mit label run=no-access - da sollte es nicht gehen
```

```
kubectl run --namespace=policy-demo-$KURZ no-access --rm -ti --image busybox
```

in der shell

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

```
### Schritt 7: Aufräumen
```

```
kubectl delete ns policy-demo-$KURZ
```

```
### Ref:
```

```
* https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic
```

```
## Kubernetes Autoscaling
```

```
### Kubernetes Autoscaling
```

```
### Overview
```

```
![image] (https://github.com/user-attachments/assets/5b0f80d9-9f17-4c8a-896b-2ae1bb7506d7)
```

```
### Example: newest version with autoscaling/v2 used to be hpa/v1
```

```
#### Prerequisites
```

```
* Metrics-Server needs to be running
```

Test with

```
kubectl top pods
```

Install with helm chart

```
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm upgrade --install metrics-server metrics-server/metrics-server --version 3.13.0 --create-namespace --namespace=metrics-server --reset-values
```

after that at will be available in kube-system namespace as pod

```
kubectl -n metrics-server get pods
```

```
#### Step 1: deploy app
```

```
cd mkdir -p manifests cd manifests mkdir hpa cd hpa nano 01-deploy.yaml
```

```
apiVersion: apps/v1 kind: Deployment metadata: name: hello spec: replicas: 3 selector: matchLabels: app: hello template: metadata: labels: app: hello spec: containers: - name: hello image: k8s.gcr.io/hpa-example resources: requests: cpu: 100m
```

```
kind: Service apiVersion: v1 metadata: name: hello spec: selector: app: hello ports: - port: 80 targetPort: 80
```

```
apiVersion: autoscaling/v2 kind: HorizontalPodAutoscaler metadata: name: hello spec: scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: hello minReplicas: 2 maxReplicas: 20 metrics:
```

- type: Resource resource: name: cpu target: type: Utilization averageUtilization: 80

```
kubectl apply -f .
```

```
### Step 2: Load Generator
```

```
nano 02-loadgenerator.yaml
```

```
apiVersion: apps/v1 kind: Deployment metadata: name: load-generator labels: app: load-generator spec: replicas: 100 selector: matchLabels: app: load-generator template: metadata: name: load-generator labels: app: load-generator spec: containers: - name: load-generator image: busybox command: - /bin/sh - -c - "while true; do wget -q -O- http://hello; done"
```

```
kubectl apply -f .
```

```
### Step 3: Zurücklehnen und geniessen
```

```
watch kubectl get pods -l app=hello
```

2.Session aufmachen und ..

```
watch kubectl get nodes
```

```
### Downscaling
```

```
* Downscaling will happen after 5 minutes o
```

Adjust down to 1 minute

```
apiVersion: autoscaling/v2 kind: HorizontalPodAutoscaler metadata: name: hello spec:
```

change to 60 secs here

```
behavior: scaleDown: stabilizationWindowSeconds: 60
```

end of behaviour change

```
scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: hello minReplicas: 2 maxReplicas: 20 metrics:
```

- type: Resource resource: name: cpu target: type: Utilization averageUtilization: 80

```
For scaling down the stabilization window is 300 seconds (or the value of the --horizontal-pod-autoscaler-downscale-stabilization flag if provided)
```

```
### Reference
* https://docs.digitalocean.com/tutorials/cluster-autoscaling-ca-hpa/
* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/#autoscaling-on-more-specific-metrics
* https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054
## Autoscaling
### Example:
```

```
apiVersion: autoscaling/v1 kind: HorizontalPodAutoscaler metadata: name: busybox-1 spec: scaleTargetRef: kind: Deployment name: busybox-1 minReplicas: 3 maxReplicas: 4 targetCPUUtilizationPercentage: 80
```

```
### Reference
* https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054
## Kubernetes Secrets / ConfigMap
### Configmap Example 1

### Schritt 1: configmap vorbereiten
```

```
cd mkdir -p manifests cd manifests mkdir configmap tests cd configmap tests nano 01-configmap.yml
```

01-configmap.yml

```
kind: ConfigMap apiVersion: v1 metadata: name: example-configmap data:
```

als Wertepaare

```
database: mongodb database_uri: mongodb://localhost:27017 testdata: | run=true file=/hello/you
```

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

```
### Schritt 2: Beispiel als Datei
```

```
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

```
### Schritt 3: Beispiel. ConfigMap als env-variablen
```

```
nano 03-pod-mit-env.yml
```

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
```

```
### Secrets Example 1
```

```
### Übung 1 - ENV Variablen aus Secrets setzen
```

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 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
```

```
### Änderung in ConfigMap erkennen und anwenden
* https://github.com/stakater/Reloader

## Kubernetes RBAC (Role based access control)

### RBAC Übung kubectl

### 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 und secret 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 1.5: Secret erstellen
* From Kubernetes 1.25 tokens are not created automatically when creating a service account (sa)
* You have to create them manually with annotation attached
* https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token
```

vi secret.yml

```
apiVersion: v1 kind: Secret type: kubernetes.io/service-account-token metadata: name: trainingtoken namespace: default annotations: kubernetes.io/service-account.name: training
```

```
kubectl apply -f .
```

```
#### Mini-Schritt 2: ClusterRole 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", "create"]

```
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 (ab Kubernetes-Version 1.25.)
```

```
#### Mini-Schritt 1: kubeconfig setzen
```

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

extract name of the token from here

```
TOKEN= kubectl get secret trainingtoken<nr> -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
```

```
#### Mini-Schritt 3: Zurück zum alten Default-Context
```

```
kubectl config get-contexts
```

```
CURRENT NAME CLUSTER AUTHINFO NAMESPACE microk8s microk8s-cluster admin2
```

- training-ctx microk8s-cluster training2

```
kubectl config use-context microk8s
```

```

### 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

### Ref: Create Service Account Token
* https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

## Kubernetes Operator Konzept

### Ueberblick

### Overview

```

o Possibility to extend functionality (new resource/object)
o Mainly to add new controllers to automate things
o Operator will control states
o Makes it easier to configure things. e.g.
a crd prometheus could create a prometheus server, which consists of different building blocks (Deployment, Service a.s.o)

```
### How to see CRD's (customresourcedefinitions)
```

```
kubectl get crd
```

Cilium, if present on the system

```
kubectl api-resources | grep cil
```

```

## Kubernetes Deployment Strategies

### 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

```
### Praxis-Übung A/B Deployment
```

```
### Walkthrough
```

```
cd cd manifests mkdir ab cd ab
```

vi 01-cm-version1.yml

```
apiVersion: v1 kind: ConfigMap metadata: name: nginx-version-1 data: index.html: |
```

Welcome to Version 1

Hi! This is a configmap Index file Version 1

```
vi 02-deployment-v1.yml
```

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy-v1 spec: selector: matchLabels: version: v1 replicas: 2 template: metadata: labels: app: nginx version: v1 spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 volumeMounts: - name: nginx-index-file mountPath: /usr/share/nginx/html/ volumes: - name: nginx-index-file configMap: name: nginx-version-1
```

```
vi 03-cm-version2.yml
```

```
apiVersion: v1 kind: ConfigMap metadata: name: nginx-version-2 data: index.html: |
```

Welcome to Version 2

Hi! This is a configmap Index file Version 2

```
vi 04-deployment-v2.yml
```

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy-v2 spec: selector: matchLabels: version: v2 replicas: 2 template: metadata: labels: app: nginx version: v2 spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 volumeMounts: - name: nginx-index-file mountPath: /usr/share/nginx/html/ volumes: - name: nginx-index-file configMap: name: nginx-version-2
```

```
vi 05-svc.yml
```

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

- port: 80 protocol: TCP selector: app: nginx

```
kubectl apply -f .
```

get external ip

```
kubectl get nodes -o wide
```

get port

```
kubectl get svc my-nginx -o wide
```

test it with curl apply it multiple time (at least ten times)

```
curl :
```

```
## Kubernetes Monitoring  
  
### Prometheus / blackbox exporter  
  
### Prerequisites  
  
* prometheus setup with helm  
  
### Step 1: Setup
```

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts helm install my-prometheus-blackbox-exporter prometheus-community/prometheus-blackbox-exporter --version 8.17.0 --namespace monitoring --create-namespace
```

```
### Step 2: Find SVC  
kubectl -n monitoring get svc | grep blackbox
```

```
my-prometheus-blackbox-exporter ClusterIP 10.245.183.66 9115/TCP
```

```
### Step 3: Test with Curl  
kubectl run -it --rm curltest --image=curlimages/curl -- sh
```

Testen nach google in shell von curl

```
curl http://my-prometheus-blackbox-exporter.monitoring:9115/probe?target=google.com&module=http\_2xx
```

Looking for metric

```
probe_http_status_code 200
```

```
### Step 4: Test apple-service with Curl
```

From within curlimages/curl pod

```
curl http://my-prometheus-blackbox-exporter.monitoring:9115/probe?target=apple-service.app&module=http\_2xx
```

```
### Step 5: Scrape Config (We want to get all services being labeled example.io/should_be_probed = true
```

```
prometheus: prometheusSpec: additionalScrapeConfigs: - job_name: "blackbox-microservices" metrics_path: /probe params: module: [http_2xx] # Autodiscovery through kube-api-server # https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes\_sd\_config kubernetes_sd_configs: - role: service relabel_configs: # Example relabel to probe only some services that have "example.io/should_be_probed = true" annotation - source_labels: [__meta_kubernetes_service_annotation_example_io_should_be_probed] action: keep regex: true - source_labels: [address] target_label: __param_target - target_label: address replacement: my-prometheus-blackbox-exporter:9115 - source_labels: [__param_target] target_label: instance - action: labelmap regex: __meta_kubernetes_service_label(.+) - source_labels: [__meta_kubernetes_namespace] target_label: app - source_labels: [__meta_kubernetes_service_name] target_label: kubernetes_service_name
```

```
### Step 6: Test with relabeler
```

```
* https://relabeler.promlabs.com
```

```
### Step 7: Scrapeconfig einbauen
```

von kube-prometheus-grafana in values und upgraden

```
helm upgrade prometheus prometheus-community/kube-prometheus-stack -f values.yml --namespace monitoring --create-namespace --version 61.3.1
```

```
### Step 8: annotation in service einfügen
```

```
kind: Service apiVersion: v1 metadata: name: apple-service annotations: example.io/should_be_probed: "true"  
spec: selector: app: apple ports: - protocol: TCP port: 80 targetPort: 5678 # Default port for image
```

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

```
### Step 9: Look into Status -> Discovery Services and wait
```

```
* blackbox services should now appear under blackbox_microservices  
* and not being dropped
```

```
### Step 10: Unter http://64.227.125.201:30090/targets?search=gucken
```

```
* ... ob das funktioniert

### Step 11: Hauptseite (status code 200)

* Metrik angekommen `?
* http://64.227.125.201:30090/graph?
g0.expr=probe_http_status_code&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h

### Step 12: pod vom service stoppen

apiVersion: apps/v1 kind: Deployment metadata: name: apple-deployment spec: selector: matchLabels: app: apple replicas: 8 template: metadata: labels: app: apple spec: containers: - name: apple-app image: hashicorp/http-echo args: - "-text=apple"-
```

```
kubectl apply -f apple.yaml # (deployment)
```

```
### Step 13: status_code 0

* Metrik angekommen `?
* http://64.227.125.201:30090/graph?
g0.expr=probe_http_status_code&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h

### Kubernetes Metrics Server verwenden

### Schritt 1: Trainer installs metrics-server

helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/ helm upgrade --install metrics-server metrics-server/metrics-server --namespace=metrics --create-namespace
```

Check it pods are running

```
kubectl -n metrics get pods
```

```
### Schritt 2: Use it

kubectl run nginx-data --image=nginx:1.27
```

how much does it use ?

```
kubectl top pods nginx-data
```

```
![image] (https://github.com/user-attachments/assets/edc6a4f7-e0af-4904-8c97-c97d84e745cf)

## Tipps & Tricks

### Netzwerkverbindung zum Pod testen

### Situation

Managed Cluster und ich kann nicht auf einzelne Nodes per ssh zugreifen

### Was wollen wir testen (auf der Verbindungsebene) ?

![image](https://github.com/user-attachments/assets/937221ca-20ff-4b1f-926c-ceef5923f60)
```

der einfachste Weg

```
kubectl run podtest --rm -it --image busybox
```

Alternative

```
kubectl run podtest --rm -it --image busybox -- /bin/sh
```

```
### Example test connection
```

wget befehl zum Kopieren

```
ping -c4 10.244.0.99 wget -O - http://10.244.0.99
```

-O -> Output (grosses O (buchstabe))

```
kubectl run podtest --rm -ti --image busybox -- /bin/sh # wget -O - http://10.244.0.99 / # exit
```

```
### Debug Container neben Container erstellen
```

```
### Beispiel 1a: Walkthrough Debug Container
```

```
kubectl run ephemeral-demo --image=registry.k8s.io/pause:3.1 --restart=Never kubectl exec -it ephemeral-demo -- sh
```

```
kubectl debug -it ephemeral-demo --image=ubuntu --target=
```

```
### Beispiel 1b: Walkthrough Debug Container with apple-app
```

```
cd mkdir -p manifests cd manifests mkdir debugcontainer cd debugcontainer nano apple.yml
```

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

```
kubectl apply -f .
```

does not work

```
kubectl exec -it newapple-app -- bash kubectl exec -it newapple-app -- sh
```

```
kubectl debug -it newapple-app --image=ubuntu
```

Durch --target=apple-app sehe ich dann auch die Prozesse des anderen containers

```
kubectl debug -it newapple-app --image=ubuntu --target=apple-app
```

```
### Aufbauend auf 1b: copy des containers erstellen
```

```
kubectl debug newapple-app -it --image=busybox --share-processes --copy-to=newappleapp-debug
```

```
### Walkthrough Debug Node
```

```
kubectl get nodes kubectl debug node/mynode -it --image=ubuntu
```

```
### Reference
```

```
* https://kubernetes.io/docs/tasks/debug/debug-application/debug-running-pod/#ephemeral-container
```

```
### Debug Pod auf Node erstellen
```

node/

```
kubectl debug -it node/node-6icn1 --image=busybox
```

im pod

ip a cd /host ls -la

```
## Kubernetes Administration /Upgrades  
### Kubernetes Administration / Upgrades
```

I. Schritt 1 (Optional aber zu empfehlen): Testsystem mit neuer Version aufsetzen (z.B. mit kind oder direkt in der Cloud)

II. Schritt 2: Manifeste auf den Stand bringen, dass sie mit den neuen Api's funktionieren, sprich ApiVersion anheben.

III. Control Plane upgraden.

Achtung: In dieser Zeit steht die API nicht zur Verfügung. Die Workloads im Cluster funktionieren nach wievor.

IV. Nodes upgraden wie folgt in 2 Varianten:

Variante 1: Rolling update

Jede Node wird gedrainet und die der Workload auf einer neuen Node hochgezogen.

Variante 2: Surge Update

Es werden eine Reihe von weiteren Nodes bereitgestellt, die bereits mit der neuen Version laufen.

Alle Workloads werden auf den neuen Nodes hochgezogen und wenn diese dort laufen, wird auf diese Nodes umgeschwicht.

<https://medium.com/google-cloud/zero-downtime-gke-cluster-node-version-upgrade-and-spec-update-dad917e25b53>

```
### Terminierung von Container vermeiden  
* https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/
```

preStop - Hook

Prozess läuft wie folgt:

Timeout before runterskalierung erfolgt ? Was ist, wenn er noch rechnet ? (task läuft, der nicht beendet werden soll)

Timeout: 30 sec. preStop

This is the process.

a. State of pod is set to terminate b. preStop hook is executed, either exec or http after success. c. Terminate - Signal is sent to pod/container d. Wait 30 secs. e. Kill - Signal is set, if container did stop yet.

```
### Praktische Umsetzung RBAC anhand eines Beispiels (Ops)  
### Enable RBAC in microk8s
```

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

microk8s enable rbac

```
### Wichtig:
```

Jeder verwendet seine eigene teilnehmer-nr z.B. training1 training2 usw. ;o)

```
### 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 # entsprechend eintragen 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- # für teilnehmer - nr eintragen 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- # durch teilnehmer nr ersetzen subjects:
```

- kind: ServiceAccount name: training # nr durch teilnehmer - nr ersetzen 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 # nr durch teilnehmer - nr ersetzen
```

```
## 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 # durch teilnehmer - nr ersetzen
```

extract name of the token from here

```
TOKEN_NAME= kubectl -n default get serviceaccount training<nr> -o jsonpath='{.secrets[0].name}' # nr durch teilnehmer ersetzen
```

```
TOKEN= kubectl -n default get secret $TOKEN_NAME -o jsonpath='{.data.token}' | base64 --decode echo $TOKEN kubectl config set-credentials training --token=$TOKEN # durch teilnehmer - nr ersetzen 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>

```
## Documentation (Use Cases)
```

```
### Case Studies Kubernetes
```

```

* https://kubernetes.io/case-studies/
### Use Cases
* https://codilime.com/blog/harnessing-the-power-of-kubernetes-7-use-cases/
## Interna von Kubernetes
### OCI, Container, Images Standards

### Grundlagen
* Container und Images sind nicht docker-spezifisch, sondern folgen der OCI Spezifikation (Open Container Initiative)
* D.h. die "Bausteine" Image, Container, Registry sind standards
* Ich brauche kein Docker, um Images zu bauen, es gibt Alternativen:
  * z.B. buildah
* kubelet -> redet mit CRI (Container Runtime Interface) -> Redet mit Container Runtime z.B. containerd (Docker), CRI-O (Redhat)
  * [CRI] (https://kubernetes.io/docs/concepts/architecture/cri/)

### Hintergründe
* Container Runtime (CRI-O, containerd)
* [OCI image (Spezifikation)](https://github.com/opencontainers/image-spec)
* OCI container (Spezifikation)
* [Sehr gute Lernreihe zu dem Thema Container (Artikel)](https://iximiuz.com/en/posts/not-every-container-has-an-operating-system-inside/)

## Andere Systeme / Verschiedenes
### Kubernetes vs. Cloudfoundry

```

cloudfoundry hat als kleinsten Baustein, die application. Der Entwickler entwickelt diese und pushed diese dann. Dadurch wird der gesamte Prozess angetriggert (Es wird IMHO ein build pack verwendet) und das image wird gebaut.

Meiner Meinung nach verwendet es auch OCI für die Images (not sure)

Als Deployment platform für cloudfoundry kann auch kubernetes verwendet werden

Kubernetes setzt beim image an, das ist der kleinste Baustein. Kubernetes selbst ist nicht in der Lage images zu bauen.

Um diesen Prozess muss sich der Entwickler kümmern oder es wird eine Pipeline bereitgestellt, die das ermöglicht.

Kubernetes skaliert nicht out of the box, zumindest nicht so integriert wie das bei Cloudfoundry möglich ist.

Die Multi-Tenant möglichkeit geht nicht, wie ich das in Cloudfoundry verstehe out of the box.

Datenbanken sind bei Kubernetes nicht ausserhalb, sondern Teil von Kubernetes (bei Cloudfoundry ausserhalb)

Eine Verknüpfung der application mit der Datenbank erfolgt nicht automatisch

Quintessenz: Wenn ich Kubernetes verwende, muss ich mich um den Prozess "Von der Applikation zum Deployment/Image/Container" selbst kümmern, bspw. in dem ich eine Pipeline in gitlab baue

```

### Kubernetes Alternativen

```

docker-compose

Vorteile:

 Einfach zu lernen

Nachteile:

 Nur auf einem Host rudimentäre Features (kein loadbalancing)

Mittel der Wahl als Einstieg

docker swarm

Zitat Linux Magazin: Swarm ist das Gegenangebot zu Kubernetes für alle Admins, die gut mit den Docker-Konventionen leben können und den Umgang mit den Standard-Docker-APIs gewöhnt sind. Sie haben bei Swarm weniger zu lernen als bei Kubernetes.

Vorteile:

 Bereits in Docker integriert (gleiche Kommandos) Einfacher zu lernen

Nachteile:

||||| Kleinere Community Kleineres Feature-Set als Kubernetes (Opinion): Bei vielen Containern wird es unhandlich

openshift 4 (Redhat)

- Verwendet als runtime: CRI-O (Redhat)

Vorteile:

Container laufen nicht als root (by default) Viele Prozesse bereits mitgedacht als Tools ?? Applikation deployen ??

In OpenShift 4 - Kubernetes als Unterbau

Nachteile:

||||| o Lizenzgebühren (Redhat) o kleinere Userbase

mesos

Mesos ist ein Apache-Projekt, in das Mesospheres Marathon und DC/OS eingeflossen sind. Letzteres ist ein Container-Betriebssystem. Mesos ist kein Orchestrator im eigentlichen Sinne. Vielmehr könnte man die Lösung als verteiltes Betriebssystem bezeichnen, das eine gemeinsame Abstraktionsschicht der Ressourcen, auf denen es läuft, bereitstellt.

Vorteile:

Nachteile:

Rancher

Graphical frontend, build on containers to deploy multiple kubernetes clusters

Hyperscalers vs. Kubernetes on Premise

Neutral:

o Erweiterungen spezifisch für die Cloud-Platform o Spezielle Kommandozeilen - Tools

Vorteile:

o Kostenabrechnung nach Bedarf (Up- / Downscaling) o Storage-Lösung (Clusterbasierte) beim CloudProvider. o Backup mitgedacht. o Leichter Upgrades zu machen o wenig Operations-Aufwand durch feststehende Prozesse und Tools

Nachteile:

o Gefahr des Vendor Logins o Kosten-Explosion o Erst_initialisierung: Aneignen von Spezial-Wissen für den jeweiligen Cloud-Provider (Lernkurve und Invest)

Gibt es eine Abstraktionsschicht, die für alle Cloud-Anbieter verwenden kann.

Lokal Kubernetes verwenden
Kubernetes in ubuntu installieren z.B. innerhalb virtualbox
Walkthrough

sudo snap install microk8s --classic

Important enable dns // otherwise not dns lookup is possible

microk8s enable dns microk8s status

Execute kubectl commands like so

microk8s kubectl microk8s kubectl cluster-info

Make it easier with an alias

echo "alias kubectl='microk8s kubectl'" >> ~/.bashrc source ~/.bashrc kubectl

Working with snaps

```
snap info microk8s
```

```
### Ref:  
* https://microk8s.io/docs/setting-snap-channel  
### minikube
```

```
### Decide for an hypervisor
```

e.g. hyperv

```
* https://minikube.sigs.k8s.io/docs/drivers/hyperv/  
### Install minikube  
* https://minikube.sigs.k8s.io/docs/start/  
### rancher for desktop  
* https://github.com/rancher-sandbox/rancher-desktop/releases/tag/v1.9.1
```

```
## Microservices
```

```
### Microservices vs. Monolith
```

```
### Schaubild
```

! [Monolithisch vs. Microservices] (https://d1.awsstatic.com/Developer%20Marketing/containers/monolith_1-monolith-microservices.70b547e30e30b013051d58a93a6e35e77408a2a8.png)

Quelle: AWS Amazon

```
### Monolithische Architektur
```

- * Alle Prozesse eng miteinander verbunden.
- * Alles ist ein einziger Service
- * Skalierung:
 - * Gesamte Architektur muss skaliert werden bei Spitzen

```
### Herausforderung: Monolithische Architektur
```

- * Verbesserung und Hinzufügen neuer Funktionen wird mit zunehmender Codebasis zunehmend komplexer
- * Nachteil: Schwer zu experimentieren
- * Nachteil: Hinderlich für die Umsetzung neuer Ideen/Konzepte

```
### Vorteile: Monolithische Architektur
```

- * Gut geeignet für kleinere Konzepte und Teams
- * Gut geeignet, wenn Projekt nicht stark wächst.
- * Gut geeignet wenn Projekt durch ein kleines Team entwickelt wird.
- * Guter Ausgangspunkt für ein kleineres Projekt
- * Mit einer MicroService - Architektur zu starten, kann hinderlich sein.

```
### Microservices
```

- * Jede Anwendung wird in Form von eigenständigen Komponenten erstellt.
- * Jeder Anwendungsprozess wird als Service ausgeführt
- * Services kommunizieren über schlanke API's miteinander
- * Entwicklung in Hinblick auf Unternehmensfunktionen
- * Jeder Service erfüllt eine bestimmte Funktion.
- * Sie werden unabhängig voneinander ausgeführt, daher kann:
 - * Jeder Service aktualisiert
 - * bereitgestellt
 - * skaliert werden

```
### Eigenschaften von microservices
```

- * Eigenständigkeit
- * Spezialisierung

```
### Vorteil: Microservices
```

- * Agilität
 - * kleines Team sind jeweils für einen Service verantwortlich
 - * können schnell und eigenverantwortlich arbeiten
 - * Entwicklungszyklus wird verkürzt.

- * Flexible Skalierung
 - * Jeder Service kann unabhängig skaliert werden.

- * Einfache Bereitstellung
 - * kontinuierliche Integration und Bereitstellung
 - * einfach:
 - * neue Konzepte auszuprobieren und zurückzunehmen, wenn etwas nicht funktioniert.

- * Technologische Flexibilität
 - * Die Teams haben die Freiheit, das beste Tool zur Lösung ihrer spezifischen Probleme auszuwählen.
 - * Infolgedessen können Teams, die Microservices entwickeln, das beste Tool für die jeweilige Aufgabe wählen.

- * Wiederverwendbarer Code
 - * Die Aufteilung der Software in kleine, klar definierte Module ermöglicht es Teams, Funktionen für verschiedene Zwecke zu nutzen.
 - * Ein Service/Funktion als Baustein

- * Resilienz
 - * Gut geplant/designed -> erhöht die Ausfallsicherheit
 - * Monolithisch: Eine Komponente fällt aus, kann zum Ausfall der gesamten Anwendung führen.
 - * Microservice: kompletter Ausfall wird vermieden, nur einzelnen Funktionalitäten sind betroffen

Nachteile: Microservices

- * Höhere Komplexität
- * Bei schlechter / nicht automatischer Dokumentation kann man bei einer größeren Anzahl von Microservices den Überblick der Zusammenarbeit verlieren
 - * Aufwand: Architektur von Monolithisch nach Microservices IST Aufwand !
 - * Aufwand Wartung und Monitoring (Kubernetes)
 - * Erhöhte Knowledge bzgl. Debugging.
 - * Fallback-Aufwand (wenn ein Service nicht funktioniert, muss die Anwendung weiter arbeiten können, ohne das andere Service nicht funktionieren)
 - * Erhöhte Anforderung an Konzeption (bzgl. Performance und Stabilität)
 - * Wichtiges Augenmerk (Netzwerk-Performance)

Nachteile: Microservices in Kubernetes

- * andere Anforderungen an Backups und Monitoring

Monolith schneiden/aufteilen

Wie kann ich schneiden (NOT's) ?

- * Code-Größe
- * Technische Schnitt
- * Amazon: 2 Pizzas, wieviele können sich davon, wie gross kann man team
- * Microserver wegschmeissen und er müsste in wenigen Tagen oder mehreren Wochen wieder herstellen

Wie kann ich schneiden (GUT) ?

- * DDD (Domain Driven Design) - Welche Aufgaben gibt es innerhalb des sogenannten Bounded Context in meiner Domäne
- * Domäne: Bibliothek
- * In der Bibliothek
 - * Leihen
 - * Suche

Bounded Context

! [Bounded Context] (<https://martinfowler.com/bliki/images/boundedContext/sketch.png>)

Zwei Merkmale mit den wir arbeiten

- * Kohäsion (innerer Zusammenhalt des Fachbereichs) - innerhalb eines Services
- * Bindung (lose Bindung) - zwischen den Services
- * Jeder Service soll unabhängig sein

Was heißt unabhängiger Service

1. Er muss funktionieren, auch wenn ein anderes Service nicht läuft (keine Abhängigkeit)
2. Er darf nicht DIREKT auf die Daten eines anderen Services zugreifen (maximal über Schnittstelle)
3. Jeder hat Service, ist völlig autark und seine eigene BusinessLogik und seine eigene Datenbank

```
### Regeln für das Design von Services
```

```
#### Regel 1:
```

Es sollte eine große Kohäsion innerhalb des Services sein. (Bindung). Alles sollte möglichst benötigt werden.

(Ist eine schwache Kohäsion innerhalb des Services, sind Funktionen dort, die eigentlich in einen anderen Service gehören)

```
#### Regel 2: lose Bindung (zwischen Services)
```

Es sollte eine lose Bindung zu anderen Services geben. (Ist die Bindung zu gross, sind entweder die Services zu klein konzipiert oder Funktionen sind an der falschen Stelle implementiert)

zu klein: zu viele Abfragen anderer Service

```
#### Regel 3: unabhängigkeit
```

...

Jeder Service muss eigenständig sein und seine eigene Datenbank haben.

...

```
### Datenbanken
```

```
#### Herangehensweise
```

...

heisst auch:

o Kein großes allmächtiges Datenmodell, sondern viele kleine
(nicht alles in jedem kleinen Datenmodell, sondern nur, was im jeweiligen
Bounded Context benötigt wird)

...

```
#### Eine Datenbank pro Service (eigenständig / abgespeckt)
```

```
#### Warum ?
```

...

Axiom: Eine eigenständige Datenbank pro Service. Warum ?
(Service will NEVER reach into another services database)

...

```
#### Punkt 1 : Jeder Service soll unabhängig laufen können
```

...

We want each service to run independently of other services

- o no DB for everything (If DB goes down our service goes down)
- o it easier to scale (if one service needs more capacity)
- o more resilient. If one service goes down, our service will still work.

...

```
#### Punkt 2: Datenbank schemata könnten sich unerwartet ändern
```

...

- o We (Service A) use data from Service B, directly retrieving it from the db.
- o We (Service) want property name: Lisa
- o Team of Service B changes this property to: firstName
AND do not inform us.
(This breaks our service !!) . OUR SERV

...

```
#### Punkt 3: Freiheit der Datenbankwahl
```

...

3.4.3 Some services might function more efficiently with different types
of DB's (sql vs. nosql)

...

```
## Beispiel - Bounded
```

...

Der Bounded Context definiert den Einsatzbereich eines Domänenmodells.

...

...

Es umfasst die Geschäftslogik für eine bestimmte Fachlichkeit. Als Beispiel beschreibt ein Domänenmodell die Buchung von S-Bahn-Fahrkarten und ein weiteres die Suche nach S-Bahn-Verbindungen.

...

...

Da die beiden Fachlichkeiten wenig miteinander zu tun haben, sind es zwei getrennte Modelle. Für die Fahrkarten sind die Tarife relevant und für die Verbindung die Zeit, das Fahrziel und der Startpunkt der Reise.

...

...

oder z.B. die Domäne: Bibliothek

Bibliothek

- * Leihen (bounded context 1)
- * Suche (bounded context 2)

...

Strategic Patterns – wid monolith praktisch umbauen

Pattern: Strangler Fig Application

- * Technik zum Umschreiben von Systemen

Wie umleitung, z.B.

- * http proxy
- * oder s.u. branch by extraction
- * An- und Abschalten mit Feature Toggle
- * Über message broker

http - proxy - Schritte

1. Schritt: Proxy einfügen
2. Schritt: Funktionalität migrieren
3. Schritt: Aufrufe umleiten

Message broker

- * Monolith reagiert auf bestimmte Messages bzw. ignoriert bestimmte messages
- * monolith bekommt bestimmte Nachrichten garnicht
- * service reagiert auf bestimmte Nachrichten

Pattern: Parallel Run

- * Service und Teil im Monolith wird parallel ausgeführt
- * Und es wird überprüft, ob das Ergebnis in beiden Systemen das gleiche ist (z.B. per batch job)

Pattern: Decorating Collaborator

- * Ansteuerung als nachgelagerten Prozess über einen Proxy

Pattern Branch by Abstraction

- * Beispiel Notification

Schritt 1: Abstraction der zu ersetzenen Funktionalität erstellen

Schritt 2: Ändern Sie die Clients der bestehenden Funktionalität so, dass sie die neue Abstraktion verwenden

Schritt 3: Neue Implementierung der Abstraktion

...

Erstellen Sie eine neue Implementierung der Abstraktion mit der überarbeiteten Funktionalität.

In unserem Fall wird diese neue Implementierung unser neuen Mikroservice aufrufen

...

Schritt 4: Abstraktion anpassen -> neue Implementierung

```

```
Abstraktion anpassen, dass sie unsere neue Implementierung verwendet
```

#### Schritt 5: Abstraktion aufräumen und alte Implementierung entfernen

### Literatur von Monolith zu Microservices

* https://www.amazon.de/Vom-Monolithen-Microservices-bestehende-umzugestalten/dp/3960091400/

## Extras

### Install minikube on wsl2

### Eventually update wsl

```
We need the newest version of wsl as of 09.2022
because systemd was included there
in powershell
wsl --shutdown
wsl --update
wsl
```

### Walkthrough (Step 1) - in wsl

```
as root in wsl
sudo su -
echo "[boot]" >> /etc/wsl.conf
echo "systemd=true" >> /etc/wsl.conf
```

### Walkthrough (Step 2) - restart wsl

```
in powershell
wsl --shutdown
takes a little bit longer now
wsl
```

### Walkthrough (step 3) - Setup minikube

```
as unprivileged user, e.g. yourname
sudo apt-get install -y \
 apt-transport-https \
 ca-certificates \
 curl \
 software-properties-common

key for rep
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \
 "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
 $(lsb_release -cs) \
 stable"

sudo apt-get update -y
sudo apt-get install -y docker-ce

sudo usermod -aG docker $USER && newgrp docker
sudo apt install -y conntrack

Download the latest Minikube
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

Make it executable
chmod +x ./minikube

Move it to your user's executable PATH
sudo mv ./minikube /usr/local/bin/

```

```

##Set the driver version to Docker
minikube config set driver docker

install minikube
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
and start it
minikube start

find out system pods
kubectl get pods -A

Note: kubernetes works within docker now
you can figure this out by
docker container ls
Now exec into the container you see: e.g acer
docker exec -it acer bash
within the container (docker runs within the container as well)
docker container ls
```

### Reference
* No need to install systemd mentioned here.
* https://www.virtualizationhowto.com/2021/11/install-minikube-in-wsl-2-with-kubectl-and-helm/

### kustomize - gute Struktur für größere Projekte

### Structure
![image](https://github.com/jmetzger/training-kubernetes-einfuehrung/assets/1933318/33d725f3-b910-4f27-9235-c6c5d3e0030a)

* Source: https://www.reddit.com/r/kubernetes/comments/sd50hk/kustomize\_with\_multiple\_deployments\_how\_to\_keep/

### kustomize with helm
* https://fabianlee.org/2022/04/18/kubernetes-kustomize-with-helm-charts/

## Documentation

### References
* https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/deployment-v1/#DeploymentSpec

### Tasks Documentation - Good one !
* https://kubernetes.io/docs/tasks/

## AWS

### ECS (managed containers) vs. Kubernetes

Perfekt - bei **wenigen Containern ohne Skalierungsbedarf** und wenn du **ausschließlich in AWS arbeitest**, ist **Amazon ECS mit Fargate** in der Regel die beste Wahl.

#### Warum ECS mit Fargate passt:
* Du brauchst **keine Cluster-Infrastruktur verwalten** (Fargate = serverless).
* **Automatisches Provisioning** der Ressourcen.
* Du zahlst nur für das, was du nutzt (CPU/RAM).
* **Einfaches Deployment** via AWS CLI, CDK oder Console.
* Ideal für kleine oder mittlere Workloads mit stabiler Last.

#### Beispielhafte Einsatzfälle:
* Kleiner Webservice (z. B. Flask, Express, Spring Boot)
* Cronjobs oder Hintergrundprozesse
* API-Gateways oder Backend-Komponenten

#### Wann **doch Kubernetes (EKS)** in Betracht kommt:
* Du hast **bereits Know-how oder Tools auf K8s-Basis** (z. B. Helm, ArgoCD).
* Bestimmte Komponenten nutzen (Ingress, Gateway API, SideCar) - helm
* Operatoren nutzen (z.B. mariadb)
```

```

* Du planst **zukünftig Komplexität oder Wachstum** (z. B. mehrere Teams, Multi-Tenants, CI/CD-Integration).
* Du willst dich **nicht an AWS binden**.

---

**Fazit:**

> Für dein Szenario: **Amazon ECS mit Fargate** - einfach, günstig, minimaler Wartungsaufwand.

## Documentation for Settings right resources/limits

### Goldilocks

* https://www.fairwinds.com/blog/introducing-goldilocks-a-tool-for-recommending-resource-requests

## Kubernetes - Überblick

### Allgemeine Einführung in Container (Dev/Ops)

### Architektur

![Docker Architecture - copyright geekflare](https://geekflare.com/wp-content/uploads/2019/09/docker-architecture-609x270.png)

### Was sind Docker Images

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

### Was sind Docker Container ?

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

### Weil :

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

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

Container vs. VM

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

Dockerfile

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

Einfaches Beispiel eines Dockerfiles

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

```
## beispiel
## cd beispiel
## ls
## Dockerfile
```

```

```

docker build .
docker push
```
### Komplexeres Beispiel eines Dockerfiles
* https://github.com/StefanScherer/whoami/blob/main/Dockerfile

### Microservices (Warum ? Wie ?) (Devs/Ops)

### Was soll das ?
```
Ein mini-dienst, soll das minimale leisten, d.h. nur das wofür er da ist.

-> z.B. Webserver
oder Datenbank-Server
oder Dienst, der nur reports erstellt
```

### Wie erfolgt die Zusammenarbeit
```
Orchestrierung (im Rahmen der Orchestrierung über vorgefertigte Schnittstellen, d.h. auch feststehende Benamung)
- Label
```
```
Vorteile
```
```
Leichtere Updates von Microservices, weil sie nur einen kleinere Funktionalität
```

```
Nachteile
```
```
* Komplexität
 * z.B. in Bezug auf Debugging
 * Logging / Backups
```
```
Wann macht Kubernetes Sinn, wann nicht?
```
```
Wann nicht sinnvoll ?
```
```
* Anwendung, die ich nicht in Container "verpackt" habe
* Spielt der Dienstleister mit (Wartungsvertrag)
* Kosten / Nutzenverhältnis (Umstellen von Container zu teuer)
* Anwendung lässt sich nicht skalieren
 * z.B. Bottleneck Datenbank
 * Mehr Container bringen nicht mehr (des gleichen Typs)
```
```
Wo spielt Kubernetes seine Stärken aus ?
```
```
* Skalieren von Anwendungen.
* bessere Hochverfügbarkeit out-of-the-box
* Heilen von Systemen (neu starten von Containern)
* Automatische Überwachung (mit deklarativem Management) - ich beschreibe, was ich will
* Neue Versionen auszurollen (Canary Deployment, Blue/Green Deployment)
```
```
Mögliche Nachteile
```
```
* Steigert die Komplexität.
* Debugging wird u.U. schwieriger
* Mit Kubernetes erkaufe ich mir auch, die Notwendigkeit.
 * Über adequate Backup-Lösungen nachzudenken (Moving Target, Kubernetes Aware Backups)
 * Bereitsstellung von Monitoring
 * Bereitsstellung Observability (Log-Aggregationslösung, Tracing)
```
```
Klassische Anwendungsfällen (wo Kubernetes von Vorteil)
```
```
* Webbasierte Anwendungen (z.B. auch API's bzw. Web)

```

- \* Ausser Problematik: Session StickyNess

### Aufbau Allgemein

### Schaubild  
![image] (<https://github.com/user-attachments/assets/f4de7c54-33a8-46e5-916c-1119575b1aed>)

### Komponenten / Grundbegriffe

#### Control Plane (Master)

#### Aufgaben

- \* Der Control Plane (Master) koordiniert den Cluster
- \* Der Control Plane (Master) koordiniert alle Aktivitäten in Ihrem Cluster
- \* Planen von Anwendungen
- \* Verwalten des gewünschten Status der Anwendungen
- \* Skalieren von Anwendungen
- \* Rollout neuer Updates.

#### Komponenten des Masters

##### etcd

- \* Verwalten der Konfiguration und des Status des Clusters (key/value - pairs)

##### kube-controller-manager

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

##### kube-api-server

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

##### kube-scheduler

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

### Nodes

- \* Worker Nodes (Knoten) sind die Arbeiter (Maschinen), die Anwendungen ausführen
- \* Ref: <https://kubernetes.io/de/docs/concepts/architecture/nodes/>

### Pod/Pods

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

### Node (Minion) - components

### General

- \* On the nodes we will rollout the applications

### kubelet

...

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

### Kube-proxy

```

* Läuft auf jedem Node
* = Netzwerk-Proxy für die Kubernetes-Netzwerk-Services.
* Kube-proxy verwaltet die Netzwerkkommunikation der Services innerhalb des Clusters

Referenzen

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

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

![Aufbau] (/images/aufbau-komponente-kubernetes.png)

Welches System ? (minikube, micro8ks etc.)

Überblick der Systeme

General

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

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

Kubeadm

General

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

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 (control plane)

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

Disadvantages

* Nicht so flexible wie kubeadm
* z.B. freie Wahl des CNI - Providers (z.B Calico)
* nicht so flexibel bei speziell config (z.B.andere IP-Ranges)

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 (wsl oder virtuelle Maschine)

* sehr schlank.
* lokal installierbar (eine node, ca 5 minuten)
* ein einziges binary
* https://docs.k3s.io/quick-start

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$4xSfDFQjKWFaoGkZF3LFaxM4hgl3d6ATbf2kEu9zMOfwLxkyMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/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:aaaff: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 einer 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:$6$HeLUW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo
```

Kubernetes - microk8s (Installation und Management)

Installation Ubuntu - snap

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

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

## Make it easier with an alias
echo "alias kubectl='microk8s kubectl'" >> ~/.bashrc
source ~/.bashrc
kubectl
```
```
#### Working with snaps
snap info microk8s
```
```
#### Ref:
* https://microk8s.io/docs/setting-snap-channel

### 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.
```
` ```

Add Node only as Worker-Node

```
` ```

microk8s join 10.135.0.15:25000/5857843e774c2ebe368e14e8b95bdf80/9bf3ceb70a58 --worker
Contacting cluster at 10.135.0.15

root@n41:~# microk8s status
This MicroK8s deployment is acting as a node in a cluster.
Please use the master node.
```
` ```

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

```
` ```

## node 1 - aktivieren
microk8s enable registry

```
` ```

Creating an image mit docker

```
` ```

```

```

## node 1 / nicht client
snap install docker

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

docker build -t localhost:32000/myubuntu .
docker images
docker push localhost:32000/myubuntu
````

Installation Kuberenetes Dashboard

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

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops) - Spickzettel

Hilfe

```
## Hilfe zu befehl
kubectl help config
## Hilfe nächste Ebene
kubectl config set-context --help
```

Allgemein

```
## Zeige Informationen über das Cluster
kubectl cluster-info

## Welche Ressourcen / Objekte gibt es, z.B. Pod
kubectl api-resources
kubectl api-resources | grep namespaces

## Hilfe zu object und eigenschaften bekommen
kubectl explain pod
kubectl explain pod.metadata
kubectl explain pod.metadata.name
```

```
## namespaces

```
``` 
kubectl get ns
kubectl get namespaces

## namespace wechseln, z.B. nach Ingress
kubectl config set-context --current --namespace=ingress
## jetzt werden alle Objekte im Namespace Ingress angezeigt
kubectl get all,configmaps

## wieder zurückwechseln.
## der standardmäßige Namespace ist 'default'
kubectl config set-context --current --namespace=default
```

```
## 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
```

```
Recursive Löschen
cd ~/manifests
multiple subfolders subfolders present
kubectl delete -f . -R
```

```
```
```

```
### Ausgabeformate / Spezielle Informationen
```

```
```
```

```
Ausgabe kann in verschiedenen Formaten erfolgen
kubectl get pods -o wide # weitere informationen
im json format
kubectl get pods -o json
```

```
gilt natürlich auch für andere kommandos
kubectl get deploy -o json
kubectl get deploy -o yaml
```

```
Label anzeigen
kubectl get deploy --show-labels
```

```
```
```

```
### 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
```

```
Pods in allen namespaces anzeigen
kubectl get pods -A
```

```
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
Löscht alle Pods im eigenen Namespace bzw. Default
kubectl delete pods --all
```

```
Kommando in pod ausführen
kubectl exec -it nginx -- bash
```

```
```
```

```
### Alle Objekte anzeigen
```

```
```
```

```

Nur die wichtigsten Objekte werden mit all angezeigt
kubectl get all
Dies, kann ich wie folgt um weitere ergänzen
kubectl get all,configmaps

Über alle Namespaces hinweg
kubectl get all -A
```

### Logs
```
kubectl logs <container>
kubectl logs <deployment>
e.g.
kubectl logs -n namespace8 deploy/nginx
with timestamp
kubectl logs --timestamps -n namespace8 deploy/nginx
continuously show output
kubectl logs -f <pod>
letzten x Zeilen anschauen aus log anschauen
kubectl logs --tail=5 <your pod>
```

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

### kubectl example with run

#### Example (that does work)
```
Synopsis (most simplistic example
kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
example
kubectl run nginx --image=nginx:1.23

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

#### Example (that does not work)
```
kubectl run testpod --image=foo2
ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
Weitere status - info
kubectl describe pods testpod
```

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

### kubectl/manifest/pod

#### Walkthrough
```
cd
mkdir -p manifests
cd manifests/
mkdir -p 01-web
cd 01-web
nano nginx-static.yml
```

```
vi nginx-static.yml

apiVersion: v1
kind: Pod
metadata:

```

```

name: nginx-static-web
labels:
 webserver: nginx
spec:
 containers:
 - name: web
 image: nginx:1.23
```
```
kubectl apply -f nginx-static.yml
```
```
kubectl get pod/nginx-static-web -o wide
kubectl describe pod nginx-static-web
show config
kubectl get pod/nginx-static-web -o yaml
```
```
Aufräumen
```
```
kubectl delete -f .
```
```
kubectl/manifest/replicaset

Walkthrough Erstellen
```
```
cd
mkdir -p manifests
cd manifests
mkdir 02-rs
cd 02-rs
nano rs.yml
```
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: nginx-replica-set
spec:
 replicas: 5
 selector:
 matchLabels:
 tier: frontend
 template:
 metadata:
 name: template-nginx-replica-set
 labels:
 tier: frontend
 spec:
 containers:
 - name: nginx
 image: nginx:1.23
 ports:
 - containerPort: 80
```
```
kubectl apply -f .
kubectl get all
kubectl get pods -l tier=frontend
kubectl get pods --show-labels
name anpassen
kubectl describe pod/nginx-replica-set-lpkbs
```
```
Pod löschen, was passiert

```

```

```
## kubectl delete po nginx-r<TAB>
## einfach einen pod raussuchen und löschen
## z.B.
kubectl delete po nginx-replica-set-xg8jp
```

```
## gucken, welches sind die neuesten ?
kubectl get pods
```

Walthrough Skalieren

```
nano rs.yml
```

```
## Ändern
## replicas: 5
## -> ändern in
## replicas: 8
```

```
kubectl apply -f .
kubectl get pods
```

Aufräumen des replicaset

```
kubectl delete -f .
```

kubectl/manifest/deployments

Prepare

```
cd
mkdir -p manifests
cd manifests
mkdir 03-deploy
cd 03-deploy
nano nginx-deployment.yml
```

```
## vi nginx-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8 # tells deployment to run 8 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginxinc/nginx-unprivileged:1.28
          ports:
            - containerPort: 8080
```

```
kubectl apply -f .
```

```

```

Explore
```
kubectl get all
```

Optional: Change image - Version
```
nano nginx-deployment.yml
```

Version 1: (optical nicer)
```
## Ändern des images von nginxinc/nginx-unprivileged:1.28 -> auf 1.29
## danach
kubectl apply -f . && watch kubectl get pods
```

Version 2:
```
## Ändern des images von nginxinc/nginx-unprivileged:1.28 -> auf 1.29
## danach
kubectl apply -f .
kubectl get all
kubectl get pods -w
```

kubectl/manifest/service

Warum Services ?

- Wenn in einem Deployment bei einem Wechsel des images neue Pods erstellt werden, erhalten diese eine neue IP-Adresse
- Nachteil: Man müsste diese dann in allen Applikation ständig ändern, die auf die Pods zugreifen.
- Lösung: Wir schalten einen Service davor !

Hintergrund IP-Wechsel
![image](https://github.com/user-attachments/assets/26c16134-1f2a-4b42-8cca-355099d08604)
```
* Image-Version wurde jetzt in Deployment geändert, Ergebnis:
![image](https://github.com/user-attachments/assets/fb5a665b-98a7-445b-8ec7-27f12c2267e1)

```

```

    web: my-nginx
replicas: 2
template:
  metadata:
    labels:
      web: my-nginx
spec:
  containers:
    - name: cont-nginx
      image: nginx
      ports:
        - containerPort: 80
```
```
nano service.yml
```

```
```
apiVersion: v1
kind: Service
metadata:
 name: svc-nginx
spec:
 type: ClusterIP
 ports:
 - port: 80
 protocol: TCP
 selector:
 web: my-nginx
```
```
```
kubectl apply -f .
## wie ist die ClusterIP ?
kubectl get all
kubectl get svc svc-nginx
## Find endpoints / did svc find pods ?
kubectl describe svc svc-nginx
```
```
##### Schritt 3: Deployment löschen
```
```
kubectl delete -f deploy.yml
## Keine endpunkte mehr
kubectl describe svc svc-nginx
```
```
##### Schritt 4: Deployment wieder erstellen
```
```
kubectl apply -f .
## Endpunkte wieder da
kubectl describe svc svc-nginx
```
```
##### Example II : Short version (NodePort)
```
```
## Wo sind wir ?
## cd; cd manifests/04-service
```
```
```
nano service.yml
in Zeile type:
ClusterIP ersetzt durch NodePort

kubectl apply -f .
NodePort ab 30.000 ausfindig machen
kubectl get svc
```


```

```

```
kubectl get nodes -o wide
```


```
im Client Externe NodeIP und NodePort verwenden
curl http://164.92.193.245:30280
```

### Example II : Service with NodePort (long version)

```
you will get port opened on every node in the range 30000+
apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-nginx
spec:
 selector:
 matchLabels:
 web: my-nginx
 replicas: 2
 template:
 metadata:
 labels:
 web: my-nginx
 spec:
 containers:
 - name: cont-nginx
 image: nginx
 ports:
 - containerPort: 80
```
---  

apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
  labels:
    run: svc-my-nginx
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
  selector:
    web: my-nginx
```
```

### Example III: Service mit LoadBalancer (ExternalIP)

```
nano service.yml
in Zeile type:
NodePort ersetzt durch LoadBalancer

kubectl apply -f .
kubectl get svc svc-nginxx
kubectl describe svc svc-nginxx
kubectl get svc svc-nginxx -w
spätestens nach 5 Minuten bekommen wir eine externe ip
z.B. 41.32.44.45

curl http://41.32.44.45
```

### Example getting a specific ip from loadbalancer (if supported)

```
apiVersion: v1
kind: Service
metadata:
 name: svc-nginxx2
spec:

```

```

type: LoadBalancer
this line to get a specific ip if supported
loadBalancerIP: 10.34.12.34
ports:
- port: 80
 protocol: TCP
selector:
 web: my-nginx
```

### Ref.

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

### Hintergrund Ingress

### 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

### Prerequisites

```
Ingress Controller muss aktiviert sein
microk8s enable ingress
```

### Walkthrough

```
mkdir apple-banana-ingress

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

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

```
kubectl apply -f apple.yml
```
```

```

```

banana
vi banana.yml
kind: Pod
apiVersion: v1
metadata:
 name: banana-app
 labels:
 app: banana
spec:
 containers:
 - name: banana-app
 image: hashicorp/http-echo
 args:
 - "-text=banana"
```
```
kind: Service
apiVersion: v1
metadata:
 name: banana-service
spec:
 selector:
 app: banana
 ports:
 - port: 80
 targetPort: 5678 # Default port for image
```
```
```
```
kubectl apply -f banana.yml
```
```
```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
          - path: /apple
            backend:
              serviceName: apple-service
              servicePort: 80
          - path: /banana
            backend:
              serviceName: banana-service
              servicePort: 80
```
```
```
```
## ingress
kubectl apply -f ingress.yml
kubectl get ing
```
```
### Reference
* https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html
```
Find the problem
```
```
Hints
```
## 1. Which resources does our version of kubectl support
## Can we find Ingress as "Kind" here.
kubectl api-resources
```
```
## 2. Let's see, how the configuration works
kubectl explain --api-version=networking.k8s.io/v1 ingress.spec.rules.http.paths.backend.service

```

```

## now we can adjust our config
```

Solution

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

Beispiel mit Hostnamen

Step 1: Walkthrough

```
cd
cd manifests
mkdir abi
cd abi
nano apple.yml
```

```
## 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-<euer-name>"
```

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

```

```

```
kubectl apply -f apple.yml
```

```
nano banana.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-<euernname>"

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

```
kubectl apply -f banana.yml
```

Step 2: Testing connection by podIP and Service

```
kubectl get svc
kubectl get pods -o wide
kubectl run podtest --rm -it --image busybox
```

```
/ # wget -O - http://<pod-ip>:5678
/ # wget -O - http://<cluster-ip>
```

Step 3: Walkthrough

```
nano ingress.yml
```

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - host: "<euernname>.lab1.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
```
### Reference
* https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html
```
Find the problem

Schritt 1: api-version ändern
```
## Welche Landkarten gibt es ?
kubectl api-versions
## Auf welcher Landkarte ist Ingress jetzt
kubectl explain ingress
```
![image](https://github.com/user-attachments/assets/15111f3d-f82e-4b79-99c4-2670e4332524)

Ingress ändern ingress.yml
von
apiVersion: extensions/v1beta1
in
apiVersion: networking.k8s.io/v1
```
#### Schritt 2: Fehler Eigenschaften beheben
![image](https://github.com/user-attachments/assets/c2f7760e-2853-4f24-b6a1-20bafa779024)


## Problem serviceName beheben
## Was gibt es stattdessen
## -> es gibt service aber keine serviceName
kubectl explain ingress.spec.rules.http.paths.backend
## -> es gibt service.name
kubectl explain ingress.spec.rules.http.paths.backend.
```
```
## Korrektur 2x in ingress.yaml: Inkl. servicePort (neu: service.port.number)
## vorher
## backend:
##   serviceName: apple-service
##   servicePort: 80
## jetzt:
## service:
##   name: apple-service
##   port:
##     number: 80
```
```
kubectl apply -f .
```
Schritt 3: pathType ergänzen
![image](https://github.com/user-attachments/assets/8da36c28-7737-49ba-a9a0-994a21fd02fb)

* Es wird festgelegt wie der Pfad ausgewertet
```

```

```

## Wir müssen pathType auf der 1. Unterebene von paths einfügen
## Entweder exact oder prefix
```
![image](https://github.com/user-attachments/assets/615884d5-2335-4dc1-99fd-cc79a224a8b6)

```

```

Als ganzes Object:
Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: example-ingress
 annotations:
 ingress.kubernetes.io/rewrite-target: /
 kubernetes.io/ingress.class: nginx
spec:
 rules:
 - http:
 paths:
 - path: /apple
 backend:
 serviceName: apple-service
 servicePort: 80
 - path: /banana
 backend:
 serviceName: banana-service
 servicePort: 80
```
### Ref:
* https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nginx-ingress-on-digitalocean-kubernetes-using-helm

### 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"
 name: destination-home
 namespace: my-namespace
spec:
 rules:
 - http:
 paths:
 - backend:
 service:
 name: http-svc
 port:
 number: 80
 path: /source
 pathType: ImplementationSpecific
```
```
eine node mit ip-adresse aufrufen
curl -I http://41.12.45.21/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


### Schritt 1: configmap vorbereiten
```
cd
mkdir -p manifests
cd manifests
mkdir configmaptests
cd configmaptests
nano 01-configmap.yml
```

```
01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
 name: example-configmap
data:
 # als Wertepaare
 database: mongodb
 database_uri: mongodb://localhost:27017
 testdata: |
 run=true
 file=/hello/you
```

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

### Schritt 2: Beispiel als Datei
```
```
```
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
```

### Schritt 3: Beispiel. ConfigMap als env-variablen

```
nano 03-pod-mit-env.yml
```

```
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 - ENV - Variablen für den Container setzen

ENV - Variablen - Übung

Übung 1 - einfach ENV-Variablen direkt setzen

```
## mkdir envtests
## cd envtest
## vi 01-simple.yml
apiVersion: v1
kind: Pod
metadata:
  name: print-envs
spec:
  containers:
    - name: env-print-demo
      image: nginx
      env:
        - name: APP_VERSION
          value: 1.21.1
        - name: APP_FEATURES
```

```

```
value: "backend,stats,reports"
```
```
kubectl apply -f 01-simple.yml
kubectl get pods
kubectl exec -it print-envs -- bash
env | grep APP
```
```
Übung 2 - ENV-Variablen von Feldern setzen (aus System)

```
## erstmal falsch
## und noch ein 2. versteckter Fehler
## vi 02-feldref.yml
apiVersion: v1
kind: Pod
metadata:
  name: print-envs-fields
spec:
  containers:
    - name: env-ref-demo
      image: nginx
      env:
        - name: APP_VERSION
          value: 1.21.1
        - name: APP_FEATURES
          value: "backend,stats,reports"
        - name: APP_POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: APP_POD_STATUS
          valueFrom:
            fieldRef:
              fieldPath: status.phase
```
```
kubectl apply -f 02-feldref.yml
## Fehler, weil es das Objekt schon gibt und es so nicht geupdatet werden kann
## Einfach zum Löschen verwenden
kubectl delete -f 02-feldref.yml
## Nochmal anlegen.
## Wieder fehler s.u.
kubectl apply -f 02-feldres.yml
```
```
## Fehler
* spec.containers[0].env[3].valueFrom.fieldRef.fieldPath: Unsupported value: "status.phase": supported values: "metadata.name", "metadata.namespace", "metadata.uid", "spec.nodeName", "spec.serviceAccountName", "status.hostIP", "status.podIP", "status.podIPs"
```
```
## letztes Feld korrigiert
apiVersion: v1
kind: Pod
metadata:
  name: print-envs-fields
spec:
  containers:
    - name: env-ref-demo
      image: nginx
      env:
        - name: APP_VERSION
          value: 1.21.1
        - name: APP_FEATURES
          value: "backend,stats,reports"
        - name: APP_POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
```

```

- name: APP_POD_NODE
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
```
```
kubectl apply -f 02-feldref.yml
kubectl exec -it print-envs -- bash
## env | grep APP
```
```
### Beispiel mit labels, die ich gesetzt habe:

```
vi 02-feldref.yml
apiVersion: v1
kind: Pod
metadata:
 name: print-envs-fields
 labels:
 app: foo
spec:
 containers:
 - name: env-ref-demo
 image: nginx
 env:
 - name: APP_VERSION
 value: 1.21.1
 - name: APP_FEATURES
 value: "backend,stats,reports"
 - name: APP_POD_IP
 valueFrom:
 fieldRef:
 fieldPath: status.podIP
 - name: LABEL_APP
 valueFrom:
 fieldRef:
 fieldPath: metadata.labels['app']
```
```
Übung 3 - ENV Variablen aus configMaps setzen.

```
## Step 1: ConfigMap
## 03-matchmaker-config.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  labels:
    app: matchmaker
data:
  MYSQL_DB: matchmaker
  MYSQL_USER: user_matchmaker
  MYSQL_DATA_DIR: /var/lib/mysql
```
```
## Step 2: applying map
kubectl apply -f 03-matchmaker-config.yml
## Das ist der Trostpreis !!
kubectl get configmap app-config
kubectl get configmap app-config -o yaml
```
```
## Step 3: setup another pod to use it in addition
## vi 04-matchmaker-app.yml
apiVersion: v1
kind: Pod
metadata:
  name: print-envs-multi
spec:
  containers:
    - name: env-ref-demo
      image: nginx

```

```

env:
- name: APP_VERSION
  value: 1.21.1
- name: APP_FEATURES
  value: "backend,stats,reports"
- name: APP_POD_IP
  valueFrom:
    fieldRef:
      fieldPath: status.podIP
- name: APP_POD_NODE
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
  envFrom:
- configMapRef:
  name: app-config

```
```
```
```
```

kubectl apply -f 04-matchmaker-app.yml
kubectl exec -it print-envs-multi -- bash
env | grep -e MYSQL -e APP_
```

### Übung 4 - ENV Variablen aus Secrets setzen
```
```
## 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 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_FEATURES
      value: "backend,stats,reports"
    - name: APP_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: APP_POD_NODE
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: APP_PASSWORD
      valueFrom:
        secretKeyRef:

```

```

        name: mysecret
        key: APP_PASSWORD
- name: APP_EMAIL
  valueFrom:
    secretKeyRef:
      name: mysecret
      key: APP_EMAIL

  envFrom:
  - configMapRef:
    name: app-config

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

## Kubernetes - Arbeiten mit einer lokalen Registry (microk8s)

### microk8s lokale Registry

### Installation

```
```
## node 1 - aktivieren
microk8s enable registry

```
```

### Creating an image mit docker

```
```
## node 1 / nicht client
snap install docker

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

docker build -t localhost:32000/myubuntu .
docker images
docker push localhost:32000/myubuntu
```
```

## Kubernetes Praxis Scaling/Rolling Updates/Wartung

### Wartung mit drain / uncordon (Ops)

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

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

## Wenn fertig bin:
kubectl uncordon n17

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

```

```

```
Ausblick AutoScaling (Ops)

Overview

![image](https://github.com/user-attachments/assets/5b0f80d9-9f17-4c8a-896b-2ae1bb7506d7)

Example: newest version with autoscaling/v2 used to be hpa/v1

Prerequisites

* Metrics-Server needs to be running

```
## Test with
kubectl top pods
```

```
## Install with helm chart
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm upgrade --install metrics-server metrics-server/metrics-server --version 3.13.0 --create-namespace --namespace=metrics-server --reset-values
```

```
## after that at will be available in kube-system namespace as pod
kubectl -n metrics-server get pods
```

Step 1: deploy app

```
cd
mkdir -p manifests
cd manifests
mkdir hpa
cd hpa
nano 01-deploy.yaml
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello
        image: k8s.gcr.io/hpa-example
        resources:
          requests:
            cpu: 100m
```

```
---
kind: Service
apiVersion: v1
metadata:
  name: hello
spec:
  selector:
    app: hello
  ports:
  - port: 80
```

```

```

targetPort: 80

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: hello
spec:
 scaleTargetRef:
 apiVersion: apps/v1
 kind: Deployment
 name: hello
 minReplicas: 2
 maxReplicas: 20
 metrics:
 - type: Resource
 resource:
 name: cpu
 target:
 type: Utilization
 averageUtilization: 80
```
```
kubectl apply -f .
```
### Step 2: Load Generator
```
nano 02-loadgenerator.yml
```
```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: load-generator
 labels:
 app: load-generator
spec:
 replicas: 100
 selector:
 matchLabels:
 app: load-generator
 template:
 metadata:
 name: load-generator
 labels:
 app: load-generator
 spec:
 containers:
 - name: load-generator
 image: busybox
 command:
 - /bin/sh
 - -c
 - "while true; do wget -q -O- http://hello; done"
```
```
kubectl apply -f .
```
### Step 3: Zurücklehnen und geniessen
```
watch kubectl get pods -l app=hello
```
```
2.Session aufmachen und ..
watch kubectl get nodes
```
### Downscaling

```

```

* Downscaling will happen after 5 minutes o

```
Adjust down to 1 minute
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: hello
spec:
 # change to 60 secs here
 behavior:
 scaleDown:
 stabilizationWindowSeconds: 60
 # end of behaviour change
 scaleTargetRef:
 apiVersion: apps/v1
 kind: Deployment
 name: hello
 minReplicas: 2
 maxReplicas: 20
 metrics:
 - type: Resource
 resource:
 name: cpu
 target:
 type: Utilization
 averageUtilization: 80

```
```
For scaling down the stabilization window is 300 seconds (or the value of the --horizontal-pod-autoscaler-downscale-stabilization flag if provided)
```

### Reference


- https://docs.digitalocean.com/tutorials/cluster-autoscaling-ca-hpa/
- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/#autoscaling-on-more-specific-metrics
- https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054


## Autoscaling
### Example:
```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: busybox-1
spec:
 scaleTargetRef:
 kind: Deployment
 name: busybox-1
 minReplicas: 3
 maxReplicas: 4
 targetCPUUtilizationPercentage: 80

```
```
Reference

- https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054

Kubernetes Storage
Praxis. Beispiel (Dev/Ops)

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) (Version 1)

```
#### 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
```

Setup PersistentVolume and PersistentVolumeClaim in cluster

Schritt 1:

```
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: 10.135.0.7
  readOnly: false
  storageClassName: ""
```

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

```

```

```
Schritt 2:

```
nano 02-pvc.yml
```

```
## vi 02-pvc.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-pvc.yml
```

Schritt 3:

```
nano 03-deploy.yml
```

```
## 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<nr>
```

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

```

```

```
nano 04-service.yml
```

```
## 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
```

Schritt 4

```
## 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

## get external ip
kubectl get nodes -o wide

## 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

### oder alternative von extern (Browser) auf Client
http://<ext-ip>:30154 (Node Port) - ext-ip -> kubectl get nodes -o wide

## 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
```

Schritt 5

```
## 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>:<port> # port -> > 30000
## exit
```

Kubernetes Networking

Überblick
```

```

Show us

![pod to pod across nodes](https://www.inovex.de/wp-content/uploads/2020/05/Pod-to-Pod-Networking.png)

Die Magie des Pause Containers

![Overview Kubernetes Networking](https://www.inovex.de/wp-content/uploads/2020/05/Container-to-Container-Networking_3_neu-400x412.png)

CNI

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

Docker - Container oder andere

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

Welche gibt es ?

- Flannel
- Canal
- Calico
- Cilium

Flannel

Overlay - Netzwerk

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

Vorteile

- Guter einfacher Einstieg
- reduziert auf eine Binary flanneld

Nachteile

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

Canal

General

- Auch ein Overlay - Netzwerk
- Unterstützt auch policies

Calico

Generell

- klassische Netzwerk (BGP)

Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

Vorteile

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

Referenz

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

Cilium

Generell

mikrok8s 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.
```

Beispiel NetworkPolicies

Um was geht es ?

* Wir wollen Firewall-Regeln mit Kubernetes machen (NetworkPolicy)
* Firewall in Kubernetes -> Network Policies

Gruppe mit eigenem cluster

<tln> = nix
z.B.
policy-demo<tln> => policy-demo
```

#### Gruppe mit einem einzigen Cluster

<tln> = Teilnehmernummer
z.B.
policy-demo<tln> => policy-demo1
```

Walkthrough

Schritt 1:
kubectl create ns policy-demo<tln>
kubectl create deployment --namespace=policy-demo<tln> nginx --image=nginx
kubectl expose --namespace=policy-demo<tln> deployment nginx --port=80
lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo<tln> access --rm -ti --image busybox -- /bin/sh
```
```
innerhalb der shell
wget -q nginx -O -
```

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

```
cd
cd manifests
mkdir network
cd network
nano 01-policy.yml
```

```
Deny Regel
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: default-deny
 namespace: policy-demo<tln>
spec:
 podSelector:
 matchLabels: {}
```

```

```

```
kubectl apply -f 01-policy.yml
```

```
lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo<tln> access --rm -ti --image busybox -- /bin/sh
```

```
innerhalb der shell
kein Zugriff möglich
wget -O - nginx
```

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

```
cd
cd manifests
cd network
nano 02-allow.yml
```

```
Schritt 3:
02-allow.yml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: access-nginx
 namespace: policy-demo<tln>
spec:
 podSelector:
 matchLabels:
 app: nginx
 ingress:
 - from:
 - podSelector:
 matchLabels:
 run: access
```

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

```
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<tln> access --rm -ti --image busybox -- /bin/sh
```

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

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

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

```
kubectl delete ns policy-demo<tln>
```

### Ref:

```

```

* https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic
* https://kubernetes.io/docs/concepts/services-networking/network-policies/
* https://docs.cilium.io/en/latest/security/policy/language/#http

## Kubernetes Paketmanagement (Helm)

### Warum ? (Dev/Ops)

```
Ein Paket für alle Komponenten
Einfaches Installieren, Updaten und deinstallieren
Konfigurations-Values-Files übergeben zum Konfigurieren
Feststehende Struktur
```

### Was kann helm ?

- **Installieren** und **Deinstallieren** von Anwendungen in Kubernetes (`helm install / helm uninstall`)
- **Upgraden** von bestehenden Installationen (`helm upgrade`)
- **Rollbacks** durchführen, falls etwas schief läuft (`helm rollback`)
- **Anpassen** von Anwendungen durch Konfigurationswerte (`values.yaml`)
- **Veröffentlichen** eigener Charts (z. B. in einem Helm-Repository)

### Grundlagen / Aufbau / Verwendung (Dev/Ops)

### Wo kann ich Helm-Charts suchen ?

* Im Telefonbuch von helm [https://artifacthub.io/](https://artifacthub.io)

### Komponenten

#### Chart

* beeinhaltet Beschreibung und Komponenten

#### Chart - Bereitstellungsformen

* url
* .tgz (Abkürzung tar.gz) - Format
* oder Verzeichnis

```
Wenn wir ein Chart installieren, wird eine Release erstellen
(parallel: image -> container, analog: chart -> release)
```

#### Installation

#### Was brauchen wir ?

* helm client muss installiert sein

#### Und sonst so ?

```
Beispiel ubuntu
snap install --classic helm

Cluster auf das ich zugreifen kann und im Client -> helm und kubectl
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

```

#### Praktisches Beispiel bitnami/mysql (Dev/Ops)

#### Prerequisites

* helm needs a config-file (kubeconfig) to know how to connect and credentials in there

```

```

* Good: helm (as well as kubectl) 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

### Simple Walkthrough (Example 0: Step 1)

```
Repo hinzufügen
helm repo add bitnami https://charts.bitnami.com/bitnami
gezeichnete Informationen aktualisieren
helm repo update

helm search repo bitnami
helm install release-name bitnami/mysql
```

### Simple Walkthrough (Example 0: Step 2: for learning - pull)

```
helm pull bitnami/mysql
tar xvfz mysql*
```

```
Simple Walkthrough (Example 0: Step 3: install)

```
helm install my-mysql bitnami/mysql
## Chart runterziehen ohne installieren
## helm pull bitnami/mysql

## Release anzeigen zu lassen
helm list

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

## weitere release installieren
## 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

## Show how the template would look like being sent to kube-api-server
helm template bitnami/mysql
```

```
### 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
```
` `` `

### Example 3: Install wordpress

### Example 3.1: Setting values with --set

```
` `` `

helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-wordpress \
--set wordpressUsername=admin \
--set wordpressPassword=password \
--set mariadb.auth.rootPassword=secretpassword \
 bitnami/wordpress
```
` `` `

### Example 3.2: Setting values with values.yml file

```
` `` `

cd
mkdir -p manifests
cd manifests
mkdir helm-wordpress
cd helm-wordpress
nano values.yml
```
` `` `

```
` `` `

values.yml
wordpressUsername: admin
wordpressPassword: password
mariadb:
 auth:
 rootPassword: secretpassword
```
` `` `

```
` `` `

helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-wordpress -f values.yml bitnami/wordpress
```
` `` `

### Referenced

```

```

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

## Kustomize

### Beispiel ConfigMap - Generator

### Walkthrough

```
External source of truth
Create a application.properties file
vi application.properties
USER=letterman
ORG=it

No use the generator
the name need to be kustomization.yaml
```

```
kustomization.yaml
configMapGenerator:
- name: example-configmap-1
 files:
 - application.properties
```

```
See the output
kubectl kustomize .

run and apply it
kubectl apply -k .
configmap/example-configmap-1-k4dmb9cmb created
```

```
Ref.

* https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/
```

### Beispiel Overlay und Patching

### Konzept Overlay

* Base + Overlay = Gepatchtes manifest
* Sachen patchen.
* Die werden drübergelegt.

### Example 1: Walkthrough

```
cd
mkdir -p manifests
cd manifests
mkdir kexample
cd kexample
```

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

```

```

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

```
Step 3: Create the service - file
vi base/service.yaml
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
kubectl kustomize ./base
```

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

```
Step 5: create overlay (patch files)
vi overlays/prod/service-ports.yaml
kind: Service
apiVersion: v1
metadata:
 #Name der zu patchenden Ressource
 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
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)

```
#####
DEPRECATED ---- use below version
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 1:
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../base
patches:
- path: service-patch.yaml
 target:
 kind: Service
 name: service-app
 version: v1
```

```
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

```
kustomize overlays/prod
```

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

Resources

Where ?
* Used in base

```
## base/kustomization.yaml
## which resources to use
## e.g
resources:
- my-manifest.yml
```

Which ?
* URL
* filename
* Repo (git)

Example:
```
## kustomization.yaml
resources:
## a repo with a root level kustomization.yaml
- github.com/Liujingfang1/mysql
## a repo with a root level kustomization.yaml on branch test
- github.com/Liujingfang1/mysql?ref=test
## a subdirectory in a repo on branch repoUrl2
- github.com/Liujingfang1/kustomize/examples/helloWorld?ref=repoUrl2
## a subdirectory in a repo on commit `7050a45134e9848fca214ad7e7007e96e5042c03`
- github.com/Liujingfang1/kustomize/examples/helloWorld?ref=7050a45134e9848fca214ad7e7007e96e5042c03
```

Kubernetes Rechteverwaltung (RBAC)

Wie aktivieren?
```
## Generell

```
Es muss das flat --authorization-mode=RBAC für den Start des Kube-Api-Server gesetzt werden

Dies ist bei jedem Installationssystem etwas anders (microk8s, Rancher etc.)

```
## Wie ist es bei microk8s

```
Auf einem der Node:
```
microk8s enable rbac
ausführen
```
Wenn ich ein HA-Cluster (control-planes) eingerichtet habe, ist dies auch auf den anderen Nodes (Control-Planes) aktiv.
```

```

```

### Praktische Umsetzung anhand eines Beispiels (Ops)

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

### Wichtig:
```
Jeder verwendet seine eigene teilnehmer-nr z.B.
training1
training2
usw. ;o)
```

### 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<nr> # <nr> entsprechend eintragen
 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-<nr> # für <nr> teilnehmer - nr eintragen
rules:
- apiGroups: [""]
 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<nr>
 namespace: default
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
```

```

```

name: pods-clusterrole-<nr> # <nr> durch teilnehmer nr ersetzen
subjects:
- kind: ServiceAccount
  name: training<nr> # nr durch teilnehmer - nr ersetzen
  namespace: default

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

```
```
#### Mini-Schritt 4: Testen (klappt der Zugang)

```
```
#### 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<nr> # <nr> durch teilnehmer - nr ersetzen

## extract name of the token from here
TOKEN_NAME=`kubectl -n default get serviceaccount training<nr> -o jsonpath='{.secrets[0].name}'` # nr durch teilnehmer <nr> ersetzen

TOKEN=`kubectl -n default get secret $TOKEN_NAME -o jsonpath='{.data.token}' | base64 --decode`  

echo $TOKEN
kubectl config set-credentials training<nr> --token=$TOKEN # <nr> druch teilnehmer - nr ersetzen
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 Backups

#### Kubernetes Backup

#### Background
* Belongs to veeam (one of the major companies for backup software)

#### What does Kubernetes Native Backup mean ?
* It is tight into the control plane, so it knows about the objects
* Uses the api to find out about Kubernetes

#### Setup a storage class (Where to store backup)
* https://docs.kasten.io/latest/install/storage.html#direct-provider-integration

#### Inject backup into a namespace to be used by app
* https://docs.kasten.io/latest/install/generic.html#using-sidecars

#### Restore:
```
```
Restore is done on the K10 - Interface
```
```

```

```

### Creating MYSQL - Backup / Restore with Kasten

* TODO: maybe move this to a seperate page
* https://blog.kasten.io/kubernetes-backup-and-restore-for-mysql

### Ref:

* https://www.kasten.io
* [Installation DigitalOcean] (https://docs.kasten.io/install/digitalocean/digitalocean.html)
* [Installation Kubernetes (Other distributions)] (https://docs.kasten.io/install/other/other.html#prerequisites)

### Kasten.io overview

* https://docs.kasten.io/latest/usage/overview.html

## Kubernetes Monitoring

### Debugging von Ingress

### 1. Schritt Pods finden, die als Ingress Controller fungieren

```
-A alle namespaces
kubectl get pods -A | grep -i ingress
jetzt sollten die pods zu sehen
Dann logs der Pods anschauen und gucken, ob Anfrage kommt
Hier steht auch drin, wo sie hin geht (zu welcher PodIP)
microk8s -> namespace ingress
Frage: HTTP_STATUS_CODE welcher ? z.B. 404
kubectl logs -n ingress <controller-ingress-pod>
```

```
Dann den Pod herausfinden, wo die Anfrage hinging
anhand der IP
kubectl get pods -o wide

Den entsprechenden pod abfragen bzgl. der Logs
kubectl logs <pod-name-mit-ziel-ip>
```

```
Ebenen des Loggings

* container-level logging
* node-level logging
* Cluster-Ebene (cluster-wide logging)
```

### Working with kubectl logs

### Logs

```
kubectl logs <container>
kubectl logs <deployment>
e.g.
kubectl logs -n namespace8 deploy/nginx
with timestamp
kubectl logs --timestamp -n namespace8 deploy/nginx
continuously show output
kubectl logs -f <container>
```

### Built-In Monitoring tools - kubectl top pods/nodes

```

```

### 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

```
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm -n kube-system upgrade --install metrics-server metrics-server/metrics-server --version 3.13.0
```

```
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

### Protokollieren mit Elasticsearch und Fluentd (Devs/Ops)

### Installieren

```
microk8s enable fluentd

Zum anzeigen von kibana
kubectl port-forward -n kube-system service/kibana-logging 8181:5601
in anderer Session Verbindung aufbauen mit ssh und port forwarding
ssh -L 8181:127.0.0.1:8181 11trainingdo@167.172.184.80

Im browser
http://localhost:8181 aufrufen
```

### Konfigurieren

```
Discover:
Innerhalb von kibana -> index erstellen
auch nochmal in Grafiken beschreiben (screenshots von kibana)
https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes
```

### Long Installation step-by-step - Digitalocean

* https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes

### Setting up metrics-server - microk8s

### 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.
```

Walkthrough

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

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

```

```
### Kubernetes

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

## Kubernetes Security

### Grundlagen und Beispiel (Praktisch)

### 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
```

## Grundlagen Security

### Geschichte


- Namespaces sind die Grundlage für Container
- LXC - Container



### Grundlagen


- letztendlich nur ein oder mehreren laufenden Prozesse im Linux - Systeme



### Seit: 1.2.22 Pod Security Admission


- 1.2.22 - Alpha - D.h. ist noch nicht aktiviert und muss als Feature Gate aktiviert (Kind)
- 1.2.23 - Beta -> d.h. aktiviert



### Vorgefertigte Regelwerke


- privileged - keinerlei Einschränkungen
- baseline - einige Einschränkungen
- restricted - sehr streng



### Praktisches Beispiel für Version ab 1.2.23 - Problemstellung
```
Schritt 1: Namespace anlegen

mkdir manifests/security
cd manifests/security
vi 01-ns.yml

apiVersion: v1
kind: Namespace
metadata:
 name: test-ns<tln>
 labels:
 pod-security.kubernetes.io/enforce: baseline
 pod-security.kubernetes.io/audit: restricted
 pod-security.kubernetes.io/warn: restricted
```
```
Schritt 2: Testen mit nginx - pod
vi 02-nginx.yml

apiVersion: v1
kind: Pod
metadata:
 name: nginx
 namespace: test-ns<tln>
spec:
 containers:
 - image: nginx
 name: nginx
 ports:
 - containerPort: 80
```
```
a lot of warnings will come up
kubectl apply -f 02-nginx.yml
```
```
Schritt 3:
Anpassen der Sicherheitseinstellung (Phase1) im Container
```

```

```

## vi 02-nginx.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns<tln>
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
```
```
```
```
kubectl delete -f 02-nginx.yml
kubectl apply -f 02_pod.yml
kubectl -n test-ns<tln> get pods
```
```
```
```
## Schritt 4:
## Weitere Anpassung runAsNonRoot
## vi 02-nginx.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns12
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
      runAsNonRoot: true
```
```
```
```
## pod kann erstellt werden, wird aber nicht gestartet
kubectl delete -f 02_pod.yml
kubectl apply -f 02_pod.yml
kubectl -n test-ns<tln> get pods
kubectl -n test-ns<tln> describe pods nginx
```
```
```
Praktisches Beispiel für Version ab 1.2.23 -Lösung - Container als NICHT-Root laufen lassen

* Wir müssen ein image, dass auch als NICHT-Root kaufen kann
* ... oder selbst eines bauen (;o)
o bei nginx ist das bitnami/nginx

```
## vi 03-nginx-bitnami.yml
apiVersion: v1
kind: Pod
metadata:
  name: bitnami-nginx
  namespace: test-ns12
spec:
  containers:
    - image: bitnami/nginx
      name: bitnami-nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
      runAsNonRoot: true

```

```

```
```

## und er läuft als nicht root
kubectl apply -f 03_pod-bitnami.yml
kubectl -n test-ns<tin> get pods
```

Kubernetes GUI

Rancher

Was ist Rancher ?

* Eine GUI für Kubernetes
* Neben dem Kubernetes Cluster, gibt es den Rancher-Server eine Web-Oberfläche zum Verwalten des Cluster und dafür Anwendungen auszurollen
* Verwendet k3s als Kubernetes-Distribution (https://rancher.com/docs/k3s/latest/en/architecture/)

Reference

* Nette kurze Beschreibung
* https://www.dev-insider.de/container-orchestrierung-mit-rancher-a-886962/
* Hintergründe:
* https://rancher.com/why-rancher

Kubernetes Dashboard

Setup / Walkthrough

Step 1: Enable Dashboard

```
## Auf Node 1:
microk8s enable dashboard

## Wenn rbac aktiviert ist, einen Nutzer mit Berechtigung einrichten
microk8s status | grep -i rbac
```

Step 2: Create a user and bind it to a specific role

```
## Wir verwenden die Rolle cluster-admin, die standardmäßig alles darf
kubectl -n kube-system get ClusterRole cluster-admin -o yaml

## Wir erstellen einen System-Account (quasi ein Nutzer): admin-user
mkdir manifests/dashboard
cd manifests/dashboard
```

```
## vi dashboard-admin-user.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
```

```
## Apply'en
kubectl apply -f dashboard-admin-user.yaml
```

```
## Jetzt erfolgt die Zuordnung des Users zur Rolle
## adminuser-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```

```

name: cluster-admin
subjects:
- kind: ServiceAccount
 name: admin-user
 namespace: kube-system
```
```
Und anwenden
kubectl apply -f adminuser-rolebinding.yaml
```
```
Damit wir zugreifen können, brauchen wir jetzt den Token für den Service - Account
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')
Diesen kopieren wir in das Clipboard und brauche ihn dann demnächst zum Anmelden
```
```
* Tricky to find a good solution because of different namespace
* Ref: https://www.linkedin.com/pulse/9-steps-enable-kubernetes-dashboard-microk8s-hendri-t/

Step 3: Verbindung aufbauen
```
## Auf Client proxy starten
kubectl proxy

## Wenn Client, nicht Dein eigener Rechner ist, dann einen Tunnel von Deinem eigenen Rechner zum Client aufbauen
ssh -L localhost:8001:127.0.0.1:8001 tln1@138.68.92.49

## In Deinem Browser auf Deinem Rechner folgende URL öffnen
http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/
## Jetzt kannst Du Dich einloggen - verwende das Token von oben, dass Du ins clipboard kopiert hast.
```
```
## Kubernetes CI/CD (Optional)

## Tipps & Tricks

### Ubuntu client aufsetzen
```
Now let us do some generic setup
echo "Installing kubectl"
snap install --classic kubectl

echo "Installing helm"
snap install --classic helm

apt-get update
apt-get install -y bash-completion
source /usr/share/bash-completion/bash_completion
is it installed properly
type _init_completion

activate for all users
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

Activate syntax - stuff for vim
Tested on Ubuntu
echo "hi CursorColumn ctermNONE ctermfg=lightred ctermbg=white" >> /etc/vim/vimrc.local
echo "autocmd FileType y?ml setlocal ts=2 sts=2 sw=2 ai number expandtab cursorline cursorcolumn" >> /etc/vim/vimrc.local

Activate Syntax highlighting for nano
cd /usr/local/bin
git clone https://github.com/serialhex/nano-highlight.git
Now set it generically in /etc/nanorc to work for all
echo 'include "/usr/local/bin/nano-highlight/yaml.nanorc"' >> /etc/nanorc
```
```
bash-completion
```
## Walkthrough

```

```

```
apt install bash-completion
source /usr/share/bash-completion/bash_completion
is it installed properly
type _init_completion

activate for all users
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

verifizieren - neue login shell
su -

zum Testen
kubectl g<TAB>
kubectl get
```
### Alternative für k als alias für kubectl

```
source <(kubectl completion bash)
complete -F __start_kubectl k
```
### Reference
* https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/

### Alias in Linux kubectl get -o wide

```
cd
echo "alias kgw='kubectl get -o wide'" >> .bashrc
for it to take immediately effect or relogin
bash
kgw pods
```
### vim einrückung für yaml-dateien

### Ubuntu (im Unterverzeichnis /etc/vim - systemweit)

```
hi CursorColumn cterm=None ctermbg=lightred ctermfg=white
autocmd FileType yml 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)

```
### kubectl spickzettel

### Hilfe

```
Hilfe zu befehl
kubectl help config
Hilfe nächste Ebene
kubectl config set-context --help
```
### Allgemein

```
Zeige Informationen über das Cluster
kubectl cluster-info
```

```

```

## Welche Ressourcen / Objekte gibt es, z.B. Pod
kubectl api-resources
kubectl api-resources | grep namespaces

## Hilfe zu object und eigenschaften bekommen
kubectl explain pod
kubectl explain pod.metadata
kubectl explain pod.metadata.name

```
```

### namespaces

```
```

kubectl get ns
kubectl get namespaces

## namespace wechseln, z.B. nach Ingress
kubectl config set-context --current --namespace=ingress
## jetzt werden alle Objekte im Namespace Ingress angezeigt
kubectl get all,configmaps

## wieder zurückwechseln.
## der standardmäßige Namespace ist 'default'
kubectl config set-context --current --namespace=default

```
```

### 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

## Recursive Löschen
cd ~/manifests
## multiple subfolders subfolders present
kubectl delete -f . -R

```
```

### Ausgabeformate / Spezielle Informationen

```
```

## Ausgabe kann in verschiedenen Formaten erfolgen
kubectl get pods -o wide # weitere informationen
## im json format
kubectl get pods -o json

## gilt natürlich auch für andere commandos
kubectl get deploy -o json
kubectl get deploy -o yaml

## Label anzeigen
kubectl get deploy --show-labels

```
```

### 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

## Pods in allen namespaces anzeigen
kubectl get pods -A

## 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
## Löscht alle Pods im eigenen Namespace bzw. Default
kubectl delete pods --all

## Kommando in pod ausführen
kubectl exec -it nginx -- bash

```
```

### Alle Objekte anzeigen
```
```

## Nur die wichtigsten Objekte werden mit all angezeigt
kubectl get all
## Dies, kann ich wie folgt um weitere ergänzen
kubectl get all,configmaps

## Über alle Namespaces hinweg
kubectl get all -A
```
```

### Logs
```
```

kubectl logs <container>
kubectl logs <deployment>
## e.g.
## kubectl logs -n namespace8 deploy/nginx
## with timestamp
kubectl logs --timestamps -n namespace8 deploy/nginx
## continuously show output
kubectl logs -f <pod>
## letzten x Zeilen anschauen aus log anschauen
kubectl logs --tail=5 <your pod>
```
```

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

### Alte manifests migrieren

### 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-converter

### X-Forward-Header-For setzen in Ingress

```
```

## Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: apache-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/configuration-snippet: |
      more_set_headers "X-Forwarded-For $http_x_forwarded_for";
spec:
  rules:
  - http:
    paths:
    - path: /project
      pathType: Prefix
      backend:
        service:
          name: svc-apache
          port:
            number: 80
```
```

### Refs:
* https://stackoverflow.com/questions/62337379/how-to-append-nginx-ip-to-x-forwarded-for-in-kubernetes-nginx-ingress-controller
* https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#configuration-snippet

## Übungen

### Übung Tag 3

```
```

2) Übung

a) Deployed ein apache-server

-> hub.docker.com -> httpd
DocumentRoot (Pfad der Dokumente)
/usr/local/apache2/htdocs

b) Volume einhängen
/var/nfs/tln<x>/apache/
Im Container einhängen wie unter a) genannt ... apache2/htdocs usw.

-> Testen

C) Service bereitstellen ohne NodePort
(ClusterIP)

-> Testen

D) Ingress-Config bereitstellen

/project

```

```

ACHTUNG: Struktur auf dem WebServer so angelegt sein muss
wie auf nfs, (was den Unterordner betrifft)

-> Testen

```
` ```

übung Tag 4

```
` ```

Verwendet das nachfolgende Deployment und
baut MYSQL_ROOT_PASSWORD so um, dass
es aus secret kommt, welches aus einem
sealed secret erstellt wird.

Stellt einen Service svc-mysql bereit, der auf einem
NodePort lauscht.

```
` ```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - image: mysql:8.0
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: password
          ports:
            - containerPort: 3306
              name: mysql
```
` ```

Fragen

Q and A

```
### Wieviele Replicaset beim Deployment zurückbehalt / Löschen von Replicaset

```
` ```

kubectl explain deployment.spec.revisionHistoryLimit

```
apiVersion: apps/v1
kind: Deployment
## ...
spec:
  # ...
  revisionHistoryLimit: 0 # Default to 10 if not specified
  # ...

```
` ```

```
### Wo dokumentieren, z.B. aus welchem Repo / git

```
` ```

Labels can be used to select objects and to find collections of objects that satisfy certain conditions. In contrast, annotations
are not used to identify and select objects.

```
* https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/
* https://kubernetes.io/docs/reference/labels-annotations-taints/
```

```

```
Wie groß werden die Logs der einzelnen Pods maximal ?
...
10 mb. max
Wird im kubelet konfiguriert.
containerMaxLogSize
...

Kuberentes und Ansible

Warum ?

- Hilft mir mein Cluster auszurollen (Infrastruktur)
- Verwalten der gesamten Applikation (manifeste etc.) über Ansible

Für Infrastruktur

- Hervorragende Lösung. Erleichtert die Deployment-Zeit.
- Möglichst schlank und einfach mit Module halten,
- z.B. https://docs.ansible.com/ansible/latest/collections/community/aws/aws_eks_cluster_module.html

Empfehlungen Applikation

- Eigenes Repos mit manifesten (losgelöst von ansible playbooks)
- Vorteil: Entwickler und andere Teams können das gleiche Repo verwenden
- Kein starkes Solution-LockIn.
- Denkbar: Das dann ansible darauf zugreift.

Fragen Applikation

- Zu klären: Wie läuft der LifeCycle.
- Wie werden neue Versionen ausgerollt ? -> Deployment - Prozess

Empfehlung Image

- Bereitstellen über Registry (nicht repo ansible)
- Binaries gehören nicht in repos (git kann das nicht so gut)

Alternativ bzw. Ergänzung

- Terraform

Documentation

Kubernetes mit VisualStudio Code

- https://code.visualstudio.com/docs/azure/kubernetes

Kube Api Ressources - Versionierungsschema

Wie ist die deprecation policy ?

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

Was ist wann deprecated ?

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

Reference:

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

Kubernetes Labels and Selector

- https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Documentation - Sources

controller manager

- https://github.com/kubernetes/kubernetes/tree/release-1.29/cmd/kube-controller-manager/app/options

```