

# Kubernetes, Helm, Prometheus, Gitlab

## 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 - Überblick
  - [Warum Kubernetes, was macht Kubernetes](#)
  - [Aufbau Allgemein](#)
  - [Wann macht Kubernetes Sinn, wann nicht?](#)
  - [Aufbau mit helm, OpenShift, Rancher\(RKE\), microk8s](#)
  - [Welches System ? \(minikube, micro8ks etc.\)](#)
  - [Installation - Welche Komponenten from scratch](#)
3. kubectl installieren und einrichten
  - [Ubuntu client aufsetzen](#)
  - [bash-completion](#)
  - [kubectl einrichten mit namespace](#)
  - [kubectl cheatsheet kubernetes](#)
4. Kubernetes Praxis API-Objekte
  - [Das Tool kubectl \(Devs/Ops\) - Spickzettel](#)
  - [kubectl example with run](#)
  - [Bauen einer Applikation mit Resource Objekten](#)
  - [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 \(Devs/Ops\)](#)
  - [IngressController \(Devs/Ops\)](#)
  - [Hintergrund Ingress](#)
  - [Ingress Controller auf Digitalocean \(doks\) mit helm installieren](#)
  - [Documentation for default ingress nginx](#)
  - [Beispiel Ingress](#)
  - [Beispiel mit Hostnamen](#)
  - [Achtung: Ingress mit Helm - annotations](#)
  - [Permanente Weiterleitung mit Ingress](#)
  - [ConfigMap Example](#)
  - [ConfigMap Example MariaDB](#)
5. Kubernetes - Jobs
  - [Documentation Jobs + kubectl cp](#)
6. Kubernetes Administration / Tipps & Tricks
  - [Hängenden Namespace löschen - Stuck](#)
7. LoadBalancer on Premise (metallb)
  - [Metallb](#)
8. Kubernetes Storage (CSI)
  - [Überblick Persistant Volumes \(CSI\)](#)
  - [Übung Persistant Storage](#)
  - [Beispiel mariadb](#)
9. Helm (Kubernetes Paketmanager)
  - [Warum ? \(Dev/Ops\)](#)
  - [Grundlagen / Aufbau / Verwendung \(Dev/Ops\)](#)
  - [Praktisches Beispiel bitnami/mysql \(Dev/Ops\)](#)
10. Kubernetes Secrets / ConfigMap
  - [Secret with mariadb](#)
  - [Secrets Example 1](#)
  - [Änderung in ConfigMap erkennen und anwenden](#)

11. Kubernetes - gitlab ci-cd - Projekt (secrets,kubectl,environments)

- [01 Kubernetes Secrets in gitlab ci/cd](#)
- [02 Access Kubernetes](#)
- [03 Working with environments](#)
- [04 Deploy manifests to cluster](#)
- [05 Execute jobs multiple times - one job definition](#)

12. gitlab ci/cd - Überblick

- [gitlab Architektur](#)
- [Overview/Pipelines](#)

13. gitlab ci/cd - Praxis I

- [Using the test - template](#)
- [Examples running stages](#)
- [Predefined Vars](#)
- [Variablen definieren](#)
- [Example Defining and using artifacts](#)

14. gitlab ci/cd (Artifacts)

- [Pass artifacts between jobs in same stage](#)

15. gitlab ci/cd (Praxis II)

- [Mehrzeile Kommandos in gitlab ci-cd ausführen](#)

16. gitlab ci/cd - Tipps & Kniffe

- [Warum before\\_script ?](#)
- [GIT STRATEGY usw.](#)

17. gitlab-ci/cd - Workflows

- [Workflows + only.start\\_by\\_starting.pipeline](#)
- [Templates for branch and merge request workflow](#)

18. gitlab-ci/cd - Variables

- [Variablen in Pipelines Web-Dialog anzeigen](#)

19. gitlab - ci/cd - Pipelines strukturieren / Templates

- [Includes mit untertemplates](#)
- [Parent/Child Pipeline](#)
- [Multiproject Pipeline / Downstream](#)
- [Vorgefertigte Templates verwenden](#)
- [Arbeiten mit extend und anchor - Dinge wiederverwenden](#)

20. gitlab - wann laufen jobs ?

- [Job nur händisch über Pipelines starten](#)
- [Auch weiterlaufen, wenn Job fehlschlägt](#)

21. gitlab ci/cd - docker

- [Docker image automatisiert bauen - gitlab registry](#)
- [Docker image automatisiert bauen - docker hub registry](#)

22. gitlab ci/cd - Documentation

- [gitlab ci/cd predefined variables](#)
- [.gitlab-ci.yml Reference](#)
- [Referenz: global -> workflow](#)
- [Referenz: global -> default](#)

23. gitlab ci/cd - Documentation - Includes

- [includes](#)
- [includes -> rules](#)
- [includes -> rules -> variables](#)
- [includes -> templates -> override-configuration](#)
- [includes -> defaults](#)

24. gitlab ci/cd - Documentation - Services

- [Documentation services](#)

25. Kubernetes Monitoring

- [Prometheus Monitoring Server \(Overview\)](#)
- [Prometheus mit helm und grafana aufsetzen](#)
- [Prometheus helm-chart, setup ingress](#)
- [Prometheus gui-walkthrough](#)
- [Prometheus / Übung blackbox exporter](#)
- [Prometheus / Übung ServiceMonitor für neue App aufsetzen](#)

## 26. Tipps & Tricks

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

## 27. Weiter lernen

- [Lernumgebung](#)
- [Bestimmte Tasks lernen](#)
- [Udemy Online Training](#)
- [Kubernetes Videos mit Hands On](#)

## 28. Documentation

- [References](#)

## Backlog (gitlab ci/cd)

### 1. gitlab ci/cd (Praxis I)

- [Variablen überschreiben/leeren](#)
- [Rules](#)

### 2. gitlab ci/cd (Praxis II)

- [Kommandos auf Zielsystem mit ssh ausführen \(auch multiline\)](#)

### 3. gitlab ci/cd docker compose

- [Docker compose local testen](#)
- [Docker compose über ssh](#)
- [Docker compose classic über scp](#)

## Backlog (last day)

### 1. Kubernetes Deployment Strategies

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

### 2. Kubernetes QoS / HealthChecks / Live / Readiness

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

## Backlog II

### 1. Kubernetes - Überblick

- [Allgemeine Einführung in Container \(Dev/Ops\)](#)
- [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 Kuberentes Dashboard](#)

### 3. Kubernetes Praxis API-Objekte

- [Connect to external database](#)
- [Hintergrund statefulsets](#)
- [Example stateful set](#)

### 4. Kubernetes - ENV - Variablen für den Container setzen

- [ENV - Variablen - Übung](#)

### 5. Kubernetes Secrets und Encrypting von z.B. Credentials

- [Kubernetes secrets Typen](#)
- [Sealed Secrets - bitnami](#)

### 6. Kubernetes - Arbeiten mit einer lokalen Registry (microk8s)

- [microk8s lokale Registry](#)

### 7. Kubernetes Praxis Scaling/Rolling Updates/Wartung

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

#### 8. Helm (IDE - Support)

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

#### 9. Kubernetes Storage

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

#### 10. Kubernetes Netzwerk

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

#### 11. Kubernetes NetworkPolicy (Firewall)

- [Kubernetes Network Policy Beispiel](#)

#### 12. Kubernetes Autoscaling

- [Kubernetes Autoscaling](#)

#### 13. Kubernetes Storage

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

#### 14. Kubernetes Networking

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

#### 15. Kustomize

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

#### 16. Kubernetes Rechteverwaltung (RBAC)

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

#### 17. Kubernetes Backups

- [Kubernetes Backup](#)

#### 18. 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](#)

#### 19. Kubernetes Security

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

#### 20. Kubernetes GUI

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

#### 21. Kubernetes CI/CD (Optional)

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

#### 22. Tipps & Tricks

- [Alias in Linux kubectl get -o yaml](#)
- [vim einrückung für yaml-dateien](#)
- [kubectl spickzettel](#)
- [alte manifests migrieren](#)
- [X-Forward-Header-Forward setzen in Ingress](#)

## 23. Übungen

- [Übung Tag 3](#)
- [Übung Tag 4](#)

## 24. Fragen

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

## 25. Interna von Kubernetes

- [OCI Container Images Standards](#)

## 26. Kubernetes Administration /Upgrades

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

## 27. Andere Systeme / Verschiedenes

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

## 28. Lokal Kubernetes verwenden

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

## 29. Microservices

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

## 30. Extras

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

## 31. Documentation

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

## 32. Documentation (Use Cases)

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

## 33. Documentation (Komponenten)

- [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
- 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 kleihalten

- 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 - 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

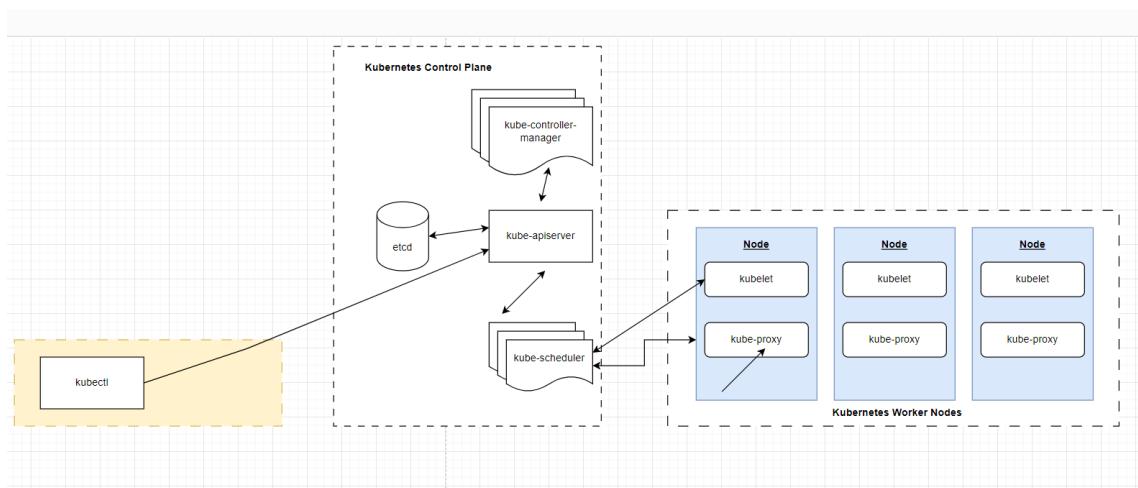
- Gegenüber Virtualisierung von Hardware - x-fache bessere Auslastung
- Google als Ausgangspunkt (Borg)
- Software 2014 als OpenSource zur Verfügung gestellt
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbstheilend

### Wozu dient Kubernetes

- Orchestrierung von Containern
- am gebräuchlichsten aktuell Docker -Images

### Aufbau Allgemein

#### Schaubild



### Komponenten / Grundbegriffe

#### Master (Control Plane)

##### Aufgaben

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

##### Komponenten des Masters

###### ETCD

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

###### KUBE-CONTROLLER-MANAGER

- Zuständig für die Überwachung der Stati im Cluster mit Hilfe von endless 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 innerhalb oder außerhalb Ihres Clusters.

#### **Referenzen**

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

#### **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 Pods)
- Automatische Überwachung mit deklarativen Management) - ich beschreibe, was ich will
- Neue Versionen zu auszurollen (Canary Deployment, Blue/Green Deployment)

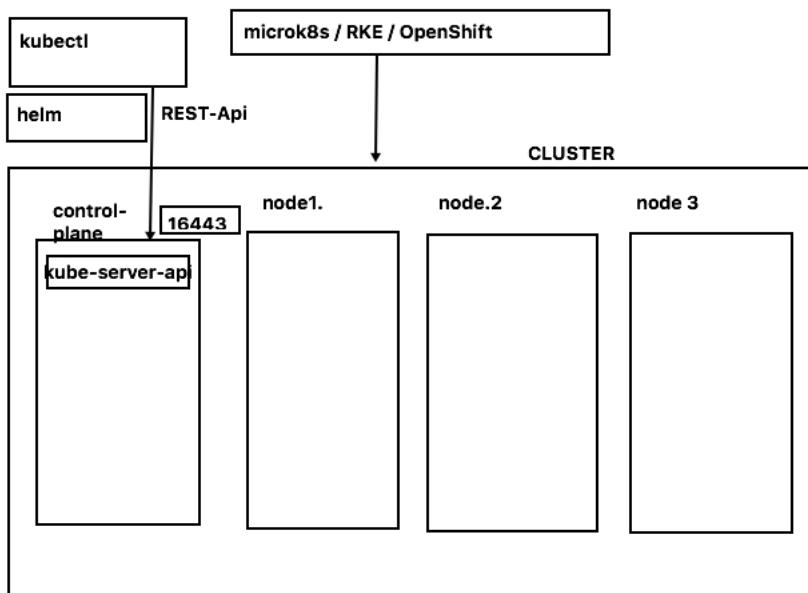
#### **Mögliche Nachteile**

- 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 Daten Log-Aggregierungslösung

#### **Klassische Anwendungsfällen**

- Webbasierte Anwendungen (z.B. auch API's bzw. Web)

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



Welches System ? (minikube, microk8s etc.)

## Überblick der Systeme

### General

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

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

### Kubeadm

#### General

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

#### Disadvantages

- Plugins sind oftmals etwas schwierig zu aktivieren

### microk8s

#### General

- Created by Canonical (Ubuntu)
- Runs on Linux
- Runs only as snap
- In the meantime it is also available for Windows/Mac
- HA-Cluster

#### Production-Ready ?

- Short answer: YES

Quote canonical (2020) :

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

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

#### Advantages

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

### minikube

### Disadvantages

- Not usable / intended for production

### Advantages

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

### k3s

#### kind (Kubernetes-In-Docker)

##### General

- Runs in docker container

##### For Production ?

Having a footprint, where kubernetes runs within docker  
and the applications 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:fe4:cbce/64
## private ip
eth1 UP 10.135.0.3/16 fe80::8081:aaff:feaa:780/64

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

## Funktioniert microk8s
microk8s status
```

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

```
## Was macht das ?
## 1. Basisnutzer (11trainingdo) - keine Voraussetzung für microk8s
## 2. Installation von microk8s
##.>>>> microk8s installiert <<<<<
## - snap install --classic microk8s
## >>>> Zuordnung zur Gruppe microk8s - notwendig für bestimmte plugins (z.B. helm)
## usermod -a -G microk8s root
## >>>> Setzen des .kube - Verzeichnisses auf den Nutzer microk8s -> nicht zwingend erforderlich
## chown -r -R microk8s ~/.kube
## >>>> REQUIRED .. DNS aktivieren, wichtig für Namensauflösungen innerhalb der PODS
## >>>> sonst funktioniert das nicht !!!
## microk8s enable dns
## >>>> kubectl alias gesetzt, damit man nicht immer microk8s kubectl eingeben muss
## - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

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

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
```

```

- echo " " >> /etc/ssh/sshd_config
- echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
- echo "AllowUsers root" >> /etc/ssh/sshd_config
- systemctl reload sshd
- sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.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:fea4:cbce/64
## private ip
eth1    UP          10.135.0.5/16 fe80::8081:aaff:feaa:780/64

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

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

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

## Mini-Schritt 2:
## Auf dem Client 1 (diese Maschine) kubeconfig an die richtige Stelle bringen
## Standardmäßig der Client nach 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
## 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
## mikrok8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 3:
## Server 1: token besorgen für node 3
## mikrok8s add-node

## Mini-Schritt 4:
## Server 3: Den Node hinzufügen durch den JOIN-Befehl
## mikrok8s 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$4xSfDFQjKWFaOgkZF3LFAXM4hg13d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo

```

#### kubectl installieren und einrichten

##### 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 cterm=None ctermfg=lightred ctermbg=white" >> /etc/vim/vimrc.local
echo "autocmd FileType yaml 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/include-optional-kubectl-configs-bash-linux/>

### **kubectl einrichten mit namespace**

#### **config einrichten**

```

cd
mkdir .kube
cd .kube
cp -a /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 jochen
kubectl get ns
kubectl config set-context --current --namespace jochen

```

### **kubectl cheatsheet kubernetes**

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

## **Kubernetes Praxis API-Objekte**

### **Das Tool kubectl (Devs/Ops) - Spickzettel**

#### **Allgemein**

```

## Zeige Informationen über das Cluster
kubectl cluster-info

## Welche api-resources gibt es ?
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

## 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

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

```

#### Example (that does not work)

```

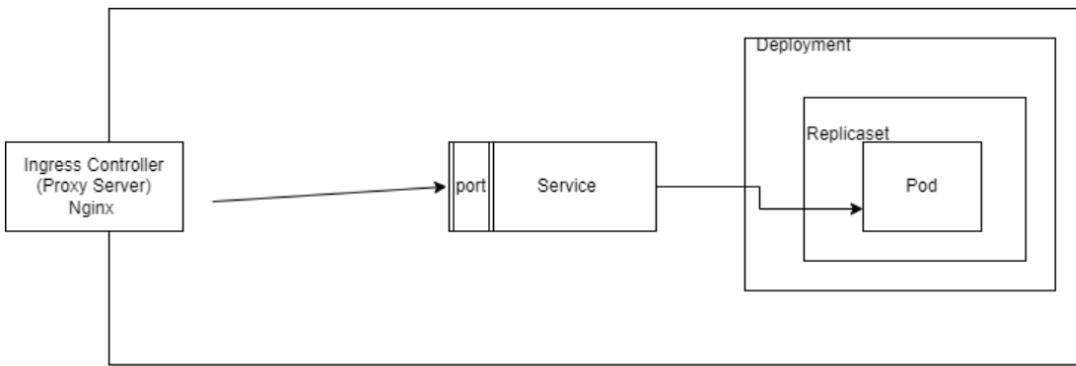
kubectl run testpod --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
## Weitere status - info
kubectl describe pods testpod

```

### Ref:

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

### Bauen einer Applikation mit Resource Objekten



### kubectl/manifest/pod

#### Walkthrough

```

cd
mkdir -p manifests
cd manifests/
mkdir 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 describe pod nginx-static-web
## show config
kubectl get pod/nginx-static-web -o yaml
kubectl get pod/nginx-static-web -o wide

```

### kubectl/manifest/replicaset

```

cd
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/manifest/deployments

### Prepare

```

cd
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: nginx:1.22
          ports:
            - containerPort: 80

kubectl apply -f .

```

### New Version

```

nano nginx-deployment.yml

## Ändern des images von nginx:1.22 in nginx:1.23
## 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
```

#### Beispiel: Eigenen Pod starten mit busybox

```

kubectl run podtest --rm -it --image busybox -- /bin/sh
## und es geht noch einfacher
kubectl run podtest --rm -it --image busybox

```

#### Example test connection

```

## wget befehl zum Kopieren
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

### Example I : Service with ClusterIP

```
cd  
mkdir -p manifests  
cd manifests  
mkdir 04-service  
cd 04-service  
  
nano deploy.yml  
  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: web-nginxx  
spec:  
  selector:  
    matchLabels:  
      web: my-nginxx  
  replicas: 2  
  template:  
    metadata:  
      labels:  
        web: my-nginxx  
    spec:  
      containers:  
      - name: cont-nginxx  
        image: nginx  
        ports:  
        - containerPort: 80
```

```
nano service.yml
```

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

```
kubectl apply -f .
```

### Example II : Short version

```
nano svc.yml  
## in Zeile type:  
## ClusterIP ersetzt durch NodePort  
  
kubectl apply -f .  
kubectl get svc  
kubectl get nodes -o wide  
## in Client  
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-nginxx  
spec:  
  selector:  
    matchLabels:  
      run: my-nginxx  
  replicas: 2
```

```

template:
  metadata:
    labels:
      run: my-nginx
  spec:
    containers:
      - name: cont-nginx
        image: nginx
        ports:
          - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
  selector:
    run: 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>

## 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

### Walkthrough (Setup Ingress Controller) - Service is published by default

```

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

## 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

```

### Walkthrough (Setup Ingress Controller) (old-version: values are being set)

```

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 yaml > nginx-ingress-ingress-nginx-controller

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

## Now setup wildcard - domain for training purpose
*.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:  
  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:  
  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: "<euename>.app1.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

```
## 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:  
  - host: "app12.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

```

#### 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>
- 

### 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
```

```
kubectl apply -f .
kubectl get cm
kubectl get cm mariadb-configmap -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:10.11
        envFrom:
          - configMapRef:
              name: mariadb-configmap
```

```
kubectl apply -f .
```

#### 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>

## Kubernetes - Jobs

### Documentation Jobs + kubectl cp

## Kubernetes Administration / Tipps & Tricks

### Hängenden Namespace löschen - Stuck

```
kubectl get namespace "metallb-system" -o json | tr -d "\n" | sed "s/\"finalizers\": \[\[^\]]\+\]\]/\"finalizers\": []/" | kubectl replace --raw /api/v1/namespaces/metallb-system/finalize -f -
```

## LoadBalancer on Premise (metallb)

### MetalLB

#### Installation

- Refs: <https://metallb.universe.tf/installation/>

#### Step 1: Installation:

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.9/config/manifests/metallb-native.yaml
```

#### Step 2: Konfiguration

```
mkdir -p manifests  
cd manifests  
mkdir metallb  
vi 01-pool.yaml
```

```
apiVersion: metallb.io/v1beta1  
kind: IPAddressPool  
metadata:  
  name: first-pool  
  namespace: metallb-system  
spec:  
  addresses:  
  - 192.168.1.240-192.168.1.250
```

```
vi 02-l2.yaml
```

```
## now we need to propagate  
apiVersion: metallb.io/v1beta1  
kind: L2Advertisement  
metadata:  
  name: example  
  namespace: metallb-system
```

### References

- <https://microk8s.io/docs/addon-metallb>
- <https://metallb.universe.tf/>
- Calico Issues: <https://metallb.universe.tf/configuration/calico/>

### Documentation

- [Set IP to specific interface and node](#)

## 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

## Übung Persistant Storage

### Step 1a: Treiber installieren (manifests)

- <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 --
```

### Alternative: Step 1b: Do the same with helm - chart

```
helm repo add csi-driver-nfs https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/master/charts  
helm install csi-driver-nfs csi-driver-nfs/csi-driver-nfs --namespace kube-system --version v4.6.0
```

### Step 2: Storage Class

```
cd  
mkdir manifests  
cd manifests  
mkdir csi  
cd csi  
nano 01-storageclass.yml
```

```
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: Retain  
volumeBindingMode: Immediate
```

### Step 3: Persistent Volume Claim

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

### Step 4: Pod

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

```

containers:
- image: nginx:1.23
  name: nginx-nfs
  command:
    - "/bin/bash"
    - "-c"
    - set -eux pipefail; 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

```

#### 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

```

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

```

```

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

```

```

##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
        volumeMounts:
          - name: persistent-storage
            mountPath: "/var/lib/mysql"
            readOnly: false
    volumes:
      - name: persistent-storage
        persistentVolumeClaim:
          claimName: pvc-nfs-dynamic-mariadb

```

## Helm (Kubernetes Paketmanager)

### Warum ? (Dev/Ops)

Ein Paket für alle Komponenten  
Einfaches Installieren, Updaten und deinstallieren  
Feststehende Struktur

### Grundlagen / Aufbau / Verwendung (Dev/Ops)

#### Wo ?

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

#### Komponenten

Chart - beeinhaltet Beschreibung und Komponenten  
tar.gz - Format  
oder Verzeichnis  
  
Wenn wir ein Chart ausführen wird eine Release erstellen  
(parallel: image -> container, analog: chart -> release)

#### Installation

```
## 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

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

#### 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 xvzf 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.yaml file

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

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

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

#### Referenced

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

### Kubernetes Secrets / ConfigMap

#### Secret with mariadb

##### Schritt 1: secret

```
cd
mkdir -p manifests
cd manifests
mkdir cf-test
cd cf-test
nano 01-secret.yaml

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

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

##### Schritt 2: Deployment

```
nano 02-deploy.yaml

##deploy.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:
```

```

- secretRef:
  name: mariadb-secret

kubectl apply -f .

```

### 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 - gitlab ci-cd - Projekt (secrets,kubectl,environments)

#### 01 Kubernetes Secrets in gitlab ci/cd

#### 02 Access Kubernetes

##### Step 1: Set KUBECONFIG\_FILE -> Settings -> CI/CD -> Variables

- Use your credentials on client from ~/.kube/config

**Variables**

Variables store information that you can use in job scripts. Each project can define more.

Variables can be accidentally exposed in a job log, or maliciously sent to a third party. This feature can help reduce the risk of accidentally exposing variable values, but is not a replacement for secure communication between users.

Variables can have several attributes. [Learn more.](#)

- Visibility:** Set the visibility level for the value. Can be visible, masked, or masked and hidden.
- Flags**
  - Protected:** Only exposed to protected branches or protected tags.
  - Expanded:** Variables with \$ will be treated as the start of a reference to another variable.

CI/CD Variables </> 3		
Key ↑	Value	Environment
KUBECONFIG_FILE	*****	All
TEST_CONTENT	*****	All
TEST_URL	*****	All

**Pipeline trigger tokens**

Trigger a pipeline for a branch or tag by generating a trigger token and using it with your user's project access and permissions. [Learn more.](#)

## Step 2: Adjust in Pipeline editor:

```

default:
  image: alpine

stages:
  - build

.prep:
  before_script:
    - apk add kubectl envsubst
    - env | grep ^TEST
    - for i in $(env | grep ^TEST); do MYVAR=$(echo $i | cut -d '=' -f 1); if [ ${MYVAR:0:4} == "TEST" ]; then echo $MYVAR >> /tmp/VARLIST; fi; done
    - cat /tmp/VARLIST
    - for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_${MYVAR}="$(eval echo \"\$${MYVAR}\" | base64)"' >> /tmp/MYENV; done
    - source /tmp/MYENV
    - cat /tmp/MYENV
    - env | grep ^SECRET

build-stage:
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info

    # Now we get a file
    # - cat manifests/mysecret.yaml | envsubst > /tmp/mysecret.yaml
    # - cat /tmp/mysecret.yaml

```

## 03 Working with environments

### Schritt 1: Environment anlegen

- environments für production und development anlegen

The screenshot shows two views of the GitLab interface. The top view displays the 'Environments' page with two environments listed: 'development' and 'production'. The bottom view shows the 'New environment' creation dialog.

**Top View: Environments Page**

- Header: Merge requests, Pipelines, Pipeline editor, Commits, Repository
- Search bar: Q Search or go to...
- Project sidebar: Merge requests (0), Pipelines, Pipeline editor, Commits, Repository, Manage, Plan, Code, Build, Secure, Deploy, Operate.
- Environment List:
  - development (Active): Stop button
  - production (Stopped): Stop button, more options menu
- Buttons: Clean up environments, Enable review apps, New environment

**Bottom View: New environment Dialog**

- Header: Q Search or go to...
- Project sidebar: Merge requests (0), Pipelines, Pipeline editor, Commits, Repository, Manage, Plan, Code, Build, Secure, Deploy, Operate, Environments (selected).
- Section: **New environment**
- Section: **Environments**
  - Text: Environments allow you to track deployments of your application. [More information.](#)
- Form fields:
  - Name: integration (with help icon)
  - Description: Preview rich text editor (with toolbar icons: B, I, S, etc.)  
Write a description or drag your files here...  
Switch to rich text editing
  - External URL: Input field
  - GitLab agent: Select agent dropdown  
Select an agent with Kubernetes access to the project or group. [How do I grant Kubernetes access?](#)
- Buttons: Save, Cancel

#### Schritt 2: Variablen für ein Environment festlegen

- Variable; TEST\_URL für production und development anlegen

The screenshot shows the GitLab CI/CD Settings interface. On the left, there's a sidebar with project navigation. The main area displays 'CI/CD Variables' with three entries: KUBECONFIG\_FILE (File), TEST\_CONTENT (Masked), and TEST\_URL. A 'Pipeline trigger tokens' section follows, and then a 'Deploy freezes' section with a configuration snippet.

**CI/CD Variables**

Key ↑	Value	Environments
KUBECONFIG_FILE	*****	All
TEST_CONTENT	*****	All
TEST_URL	*****	All

**Pipeline trigger tokens**

Trigger a pipeline for a branch or tag by generating a trigger token and using it with your user's project access and permissions. Learn more.

**Deploy freezes**

Add a freeze period to prevent unintended releases during a period of time for a deployment jobs in `.gitlab-ci.yml` according to the deploy freezes added here. Using cron syntax.

### Schritt 3: Pipeline editor

```

default:
  image: alpine

stages:
  - build

.prep:
  before_script:
    - apk add kubectl envsubst
    - env | grep ^TEST
    - for i in $(env | grep ^TEST); do MYVAR=$(echo $i | cut -d '=' -f 1); if [ ${MYVAR:0:4} == "TEST" ]; then echo $MYVAR >> /tmp/VARLIST; fi; done
    - cat /tmp/VARLIST
    - for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'="$(eval echo \"\$${MYVAR}\" | base64)'"' >> /tmp/MYENV; done
    #- for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'="$(eval echo \"\$${MYVAR}\")'"' >> /tmp/MYENV; done
    - source /tmp/MYENV
    - cat /tmp/MYENV
    - env | grep ^SECRET

build-stage-production:
  environment: production
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info

build-stage-development:
  environment: development
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info

```

### 04 Deploy manifests to cluster

#### Adjust pipeline with pipeline editor

```

default:
  image: alpine

stages:
  - build

.prep:
  before_script:
    - apk add kubectl envsubst
    - env | grep ^TEST
    - for i in $(env | grep ^TEST); do MYVAR=$(echo $i | cut -d '=' -f 1); if [ ${MYVAR:0:4} == "TEST" ]; then echo $MYVAR >> /tmp/VARLIST; fi; done
    - cat /tmp/VARLIST
    - for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'='$(eval echo \"\$\$MYVAR\" | base64)'"' >> /tmp/MYENV; done
    #- for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'='$(eval echo \"\$\$MYVAR\"))'"' >> /tmp/MYENV; done
    - source /tmp/MYENV
    - cat /tmp/MYENV
    - env | grep ^SECRET

build-stage-production:
  environment: production
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info
    - cd manifests; kubectl apply --dry-run=client -o yaml -R -f . | envsubst > /tmp/all-manifests.yml
    - cat /tmp/all-manifests.yml
    - kubectl apply -f /tmp/all-manifests.yml
    - kubectl get all

build-stage-development:
  environment: development
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info
    - cd manifests; kubectl apply --dry-run=client -o yaml -R -f . | envsubst > /tmp/all-manifests.yml
    - kubectl apply -f /tmp/all-manifests.yml
    - kubectl get all

```

## 05 Execute jobs multiple times - one job definition

### Pipeline Editor (entsprechend ändern)

```

default:
  image: alpine

stages:
  - build

.prep:
  before_script:
    - apk add kubectl envsubst
    - env | grep ^TEST
    - for i in $(env | grep ^TEST); do MYVAR=$(echo $i | cut -d '=' -f 1); if [ ${MYVAR:0:4} == "TEST" ]; then echo $MYVAR >> /tmp/VARLIST; fi; done
    - cat /tmp/VARLIST
    - for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'='$(eval echo \"\$\$MYVAR\" | base64)'"' >> /tmp/MYENV; done
    #- for MYVAR in $(cat /tmp/VARLIST); do echo 'export SECRET_'"$MYVAR"'='$(eval echo \"\$\$MYVAR\"))'"' >> /tmp/MYENV; done
    - source /tmp/MYENV
    - cat /tmp/MYENV
    - env | grep ^SECRET

build-stage-production:
  environment: $ENVIRONMENT
  stage: build
  extends: .prep
  script:
    - export KUBECONFIG=$KUBECONFIG_FILE
    - kubectl cluster-info
    - cd manifests; kubectl apply --dry-run=client -o yaml -R -f . | envsubst > /tmp/all-manifests.yml
    - cat /tmp/all-manifests.yml
    - kubectl apply -f /tmp/all-manifests.yml
    - kubectl get all
parallel:

```

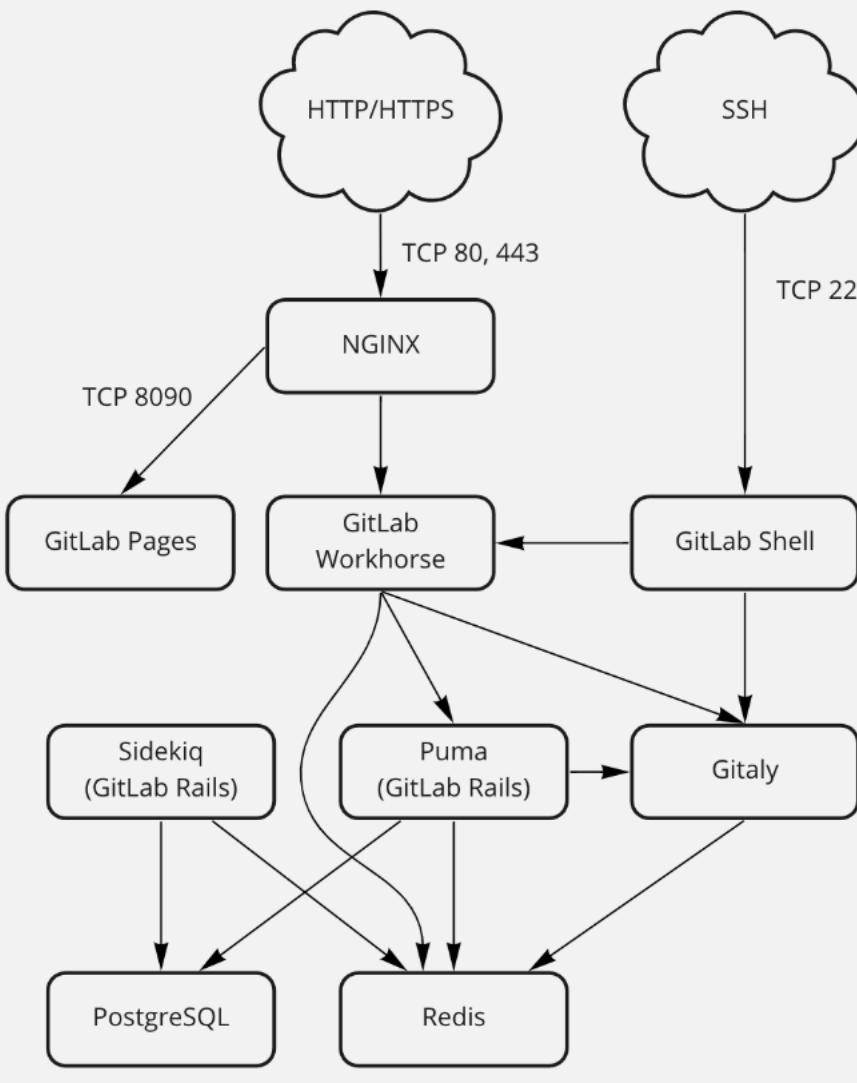
```
matrix:  
- ENVIRONMENT: ['production','development']
```

## gitlab ci/cd - Überblick

### gitlab Architektur

#### Overview

## GitLab high-level architecture



### Components

```
GitLab Workhorse  
=====  
-> smart reverse proxy  
GitLab Workhorse is a smart reverse proxy for GitLab. It handles "large" HTTP requests such as file downloads, file uploads, Git push/pull and Git archive downloads.  
  
GitLab Shell  
=====  
GitLab Shell handles Git SSH sessions for GitLab and modifies the list of authorized keys. GitLab Shell is not a Unix shell nor a replacement for Bash or Zsh.  
GitLab supports Git LFS authentication through SSH.  
--> Alternative notwendig statt openssh
```

When you access the GitLab server over SSH, GitLab Shell then:

1. Limits you to predefined Git commands (git push, git pull, git fetch).
2. Calls the GitLab Rails API to check if you are authorized, and what Gitaly server your repository is on.
3. Copies data back and forth between the SSH client and the Gitaly server.

```
Sidekiq (GitLab Rails -> gitlab rails console)
```

```
=====
```

```
Simple, efficient background processing for Ruby.
```

```
Sidekiq uses threads to handle many jobs at the same time in the same process. It does not require Rails but will integrate tightly with Rails to make background processing dead simple.
```

```
Puma (Gitlab Rails -> gitlab rails console)
```

```
=====
```

```
Puma is a fast, multi-threaded, and highly concurrent HTTP 1.1 server for Ruby applications. It runs the core Rails application that provides the user-facing features of GitLab.
```

```
Gitaly
```

```
=====
```

```
Dein Repo liegt auf einem bestimmten gitaly - Server
```

```
Gitaly provides high-level RPC access to Git repositories. It is used by GitLab to read and write Git data.
```

```
Gitaly is present in every GitLab installation and coordinates Git repository storage and retrieval. Gitaly can be:
```

```
A background service operating on a single instance Linux package installation (all of GitLab on one machine). Separated onto its own instance and configured in a full cluster configuration, depending on scaling and availability requirements.
```

## Overview/Pipelines

### Pipelines

- The foundation of ci/cd are the pipelines
- You can either have preconfigured pipelines (using Auto DevOps)
- Or you can
  - Adjust them yourself (from Auto Devops, templates)
  - Create one from scratch
- Pipelines are either defined by Auto Devops or:
  - By .gitlab-ci.yml - file in the root-level - folder of your project
- There is also an editor under CI/CD -> Editor

### Type of pipelines: Basic Pipeline

- Image: [https://docs.gitlab.com/ee/ci/pipelines/pipeline\\_architectures.html#basic-pipelines](https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#basic-pipelines)
- (each stage runs concurrently)
- Default behaviour

```
## Example:
stages:
  - build
  - test
  - deploy

image: alpine

build_a:
  stage: build
  script:
    - echo "This job builds something."

build_b:
  stage: build
  script:
    - echo "This job builds something else."

test_a:
  stage: test
  script:
    - echo "This job tests something. It will only run when all jobs in the"
    - echo "build stage are complete."

test_b:
  stage: test
  script:
    - echo "This job tests something else. It will only run when all jobs in the"
    - echo "build stage are complete too. It will start at about the same time as test_a."
```

```

deploy_a:
  stage: deploy
  script:
    - echo "This job deploys something. It will only run when all jobs in the"
    - echo "test stage complete."
  needs: []

deploy_b:
  stage: deploy
  script:
    - echo "This job deploys something else. It will only run when all jobs in the"
    - echo "test stage complete. It will start at about the same time as deploy_a."
  needs: []

```

#### Type of pipelines: DAG (Directed Acyclic Graph) Pipelines

- Image:
- Deploy\_a can run, although build\_b>test\_b is not even ready
- Because gitlab knows the dependencies by keyword: needs:

```

## Example:
stages:
- build
- test
- deploy

image: alpine

build_a:
  stage: build
  script:
    - echo "This job builds something quickly."

build_b:
  stage: build
  script:
    - sleep 20
    - echo "This job builds something else slowly."

test_a:
  stage: test
  needs: [build_a]
  script:
    - echo "This test job will start as soon as build_a finishes."
    - echo "It will not wait for build_b, or other jobs in the build stage, to finish."

test_b:
  stage: test
  needs: [build_b]
  script:
    - echo "This test job will start as soon as build_b finishes."
    - echo "It will not wait for other jobs in the build stage to finish."

deploy_a:
  stage: deploy
  needs: [test_a]
  script:
    - echo "Since build_a and test_a run quickly, this deploy job can run much earlier."
    - echo "It does not need to wait for build_b or test_b."

deploy_b:
  stage: deploy
  needs: [test_b]
  script:
    - echo "Since build_b and test_b run slowly, this deploy job will run much later."

```

#### Type of pipelines: Child- / Parent - Pipelines

- [https://docs.gitlab.com/ee/ci/pipelines/pipeline\\_architectures.html#child--parent-pipelines](https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#child--parent-pipelines)
- in Example: two types of things that could be built independently.
  - Combines child and DAG in this case
  - Trigger is used to start the child - pipeline
- Include:
  - not to repeat yourself + eventually as template (using . - prefix)
- Rules:
  - are like conditions

```

## Example
## File 1: .gitlab-ci.yml
stages:
- triggers

trigger_a:
stage: triggers
trigger:
  include: a/.gitlab-ci.yml
rules:
- changes:
  - a/*

trigger_b:
stage: triggers
trigger:
  include: b/.gitlab-ci.yml
rules:
- changes:
  - b/*

## File 2: a/.gitlab-ci.yml
stages:
- build
- test
- deploy

image: alpine

build_a:
stage: build
script:
- echo "This job builds something."

test_a:
stage: test
needs: [build_a]
script:
- echo "This job tests something."

deploy_a:
stage: deploy
needs: [test_a]
script:
- echo "This job deploys something."

## File 3: a/.gitlab-ci.yml
stages:
- build
- test
- deploy

image: alpine

build_b:
stage: build
script:
- echo "This job builds something else."

test_b:
stage: test
needs: [build_b]
script:
- echo "This job tests something else."

deploy_b:
stage: deploy
needs: [test_b]
script:
- echo "This job deploys something else."

```

#### Type of pipelines: Ref:

- [https://docs.gitlab.com/ee/ci/pipelines/pipeline\\_architectures.html](https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html)

#### Stages

- Stages run one after each other

- They default to build, test, deploy (if you do not define any)
- If you want to have less, you have to define which
- Reference:

## Jobs

- Jobs define what to do within the stages
- Normally jobs are run concurrently in each stage
- Reference:

## gitlab ci/cd - Praxis I

### Using the test - template

#### Example Walkthrough

```
## Schritt 1: Neues Repo aufsetzen

## Setup a new repo
## Setting:
## o Public, dann bekommen wir mehr Rechenzeit
## o No deployment planned
## o No SAST
## o Add README.md

## Using naming convention
## Name it however you want, but have you tln - nr inside
## e.g.
## test-artifacts-tln1

## Schritt 2: Ein Standard-Template als Ausgangsbasis holen
## Get default ci-Template
Build -> Pipeline - Editor

## Es erscheint der Editor mit einem Test-Template

ix speichern und committen.

## Jetzt wird es in der Pipeline ausgeführt.
```

### Examples running stages

#### Running default stages

- build, test, deploy are stages set by default

```
## No stages defined, so build, test and deploy are run

build-job:      # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."

unit-test-job:  # This job runs in the test stage.
  stage: test   # It only starts when the job in the build stage completes successfully.
  script:
    - echo "Running unit tests... This will take about 60 seconds."
    - sleep 1
    - echo "Code coverage is 90%"

deploy-job:     # This job runs in the deploy stage.
  stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."
```

#### only run some

```
## einfaches stages - keyword ergänzen und die stages die man haben will
stages:
  - build
  - deploy
```

```

build-job:      # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."

## unit-test-job wurde gelöscht

deploy-job:     # This job runs in the deploy stage.
  stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."

```

- Danach sich die Pipelines anschauen (CI/CD -> Pipeline)

## Predefined Vars

### Example to show them

```

stages:
  - build

show_env:
  stage: build
  script:
    - env
    - pwd

```

## Reference

- [https://docs.gitlab.com/ee/ci/variables/predefined\\_variables.html](https://docs.gitlab.com/ee/ci/variables/predefined_variables.html)

## Variablen definieren

### Möglichkeit 1: TopLevel (Im Project)

```

## Settings -> CI/CD -> Variables
TEST_URL -> http://www.meinseite.de # masked
TEST_CONTENT -> als File-Type setzen mit Inhalt "tolle Sache"

```

## Beispiele:

```

## gitlab-ci.yml
variables:
  TEST_URL: http://schulung.t3isp.de # globalen Scope - in der gesamten Pipeline
                                         # Überschreibt NICHT -> ... Settings -> CI/CD -> Variables
  TEST_VERSION: "1.0" # global
  TEST_ENV: Prod # global
  TEST_VAR: "overwrite?"

stages:
  - build
  - test

show_env:
  stage: build

variables:
  TEST_JOB: lowrunner # variable mit lokalem Scope - nur in Job
  TEST_ENV: "Das überschreibt -> Prod aus globalem Scope"
  TEST_URL: http://www.test.de # Auch das überschreibt NICHT -> ... Settings -> CI/CD -> Variables

script:
  - echo $TEST_VAR
  - echo $TEST_MASKED
  - echo $TEST_URL
  - echo $TEST_URL > /tmp/urltest.txt
  - cat /tmp/urltest.txt
  - echo $TEST_CONTENT # tolle Sache
  - cat $TEST_CONTENT
  - echo $TEST_VERSION
  - echo $TEST_ENV
  - echo $TEST_JOB

test_env:

```

```

stage: test

script:
- echo $TEST_URL
- echo $TEST_CONTENT # tolle Sache
- cat $TEST_CONTENT
- echo $TEST_VERSION
- echo $TEST_ENV
- echo $TEST_JOB

```

#### Reihenfolge, welche Variablen welche überschreien (Ebenene)

- <https://docs.gitlab.com/ee/ci/variables/#cicd-variable-precedence>

#### Example Defining and using artifacts

##### What is it ?

Jobs can output an archive of files and directories. This output is known as a job artifact.  
You can download job artifacts by using the GitLab UI or the API.

#### Example 1: Creating an artifact

```

## .gitlab-ci.yml

stages:
- build

create_txt:
  stage: build
  script:
    - echo "hello" > ergebnis.txt
  artifacts:
    paths:
      - ergebnis.txt

```

#### Example 2: creating artifacts with wildcards and different name

```

## .gitlab-ci.yml
stages:
- build
create_txt:
  stage: build
  script:
    - mkdir -p path/my-xyz
    - echo "hello" > path/my-xyz/ergebnis.txt
    - mkdir -p path/some-xyz
    - echo "some" > path/some-xyz/testtext.txt
  artifacts:
    name: meine-daten
    paths:
      - path/*xyz/*

```

#### Example 3: Artifakte und Name aus Variable vergeben

- If your branch-name contains forward slashes
  - (for example feature/my-feature)
  - it's advised to use \$CI\_COMMIT\_REF\_SLUG instead of \$CI\_COMMIT\_REF\_NAME
    - for proper naming of the artifact.

```

## .gitlab-ci.yml
stages:
- build
create_txt:
  stage: build
  script:
    - mkdir -p path/my-xyz
    - echo "hello" > path/my-xyz/ergebnis.txt
    - mkdir -p path/some-xyz
    - echo "some" > path/some-xyz/testtext.txt

```

```

artifacts:
  name: "$CI_JOB_NAME-$CI_COMMIT_REF_NAME"
  paths:
    - path/*xyz/*

```

#### **Example 4: Alle files in einem Verzeichnis recursive**

```

## .gitlab-ci.yml
stages:
  - build
create_txt:
  stage: build
  script:
    - mkdir -p path/my-xyz
    - echo "toplevel" > path/you-got-it.txt
    - echo "hello" > path/my-xyz/ergebnis.txt
    - mkdir -p path/some-xyz
    - echo "some" > path/some-xyz/testtext.txt
  artifacts:
    paths:
      - path/

```

#### **Example 5: Artifakte und Bedingungen**

```

## nur artifact erstellen, wenn ein commit-tag gesetzt ist.
## Gibt es kein commit-tag ist diese Variable NICHT GESETZT.

### .gitlab-ci.yml
stages:
  - build

output_something:
  stage: build
  script:
    - echo "just writing something"
    - env
    - echo "CI_COMMIT_TAG:..$CI_COMMIT_TAG.."

create_txt:
  stage: build
  script:
    - mkdir -p path/my-xyz
    - echo "toplevel" > path/you-got-it.txt
    - echo "hello" > path/my-xyz/ergebnis.txt
    - mkdir -p path/some-xyz
    - echo "some" > path/some-xyz/testtext.txt
    - env
    - echo "TAG ? $CI_COMMIT_TAG"
  artifacts:
    paths:
      - path/
  rules:
    - if: $CI_COMMIT_TAG

```

- Test 1: committen und Pipeline beobachten
- Test 2: Tag über repository > Tags erstellen und nochmal Pipeline beobachten

#### **Passing 1: Passing artifacts between stages (enabled by default)**

```

default:
  image: ubuntu:22.04

## stages are set to build, test, deploy by default
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - echo "in building..." >> ./control.txt
  artifacts:

```

```

paths:
- control.txt
expire_in: 1 week

my_unit_test:
stage: test
script:
- ls
- cat control.txt
- echo "now in unit testing ..." >> ./control.txt
artifacts:
paths:
- control.txt
expire_in: 1 week

deploy:
stage: deploy
script:
- ls
- cat control.txt

```

#### **Passing 2: artifacts between stages (enabled by default) - only writing it in stage: build**

```

## only change in stage: build
image: ubuntu:20.04

## stages are set to build, test, deploy by default

build:
stage: build
script:
- echo "in building..." >> ./control.txt
artifacts:
paths:
- control.txt
expire_in: 1 week

my_unit_test:
stage: test
script:
- cat control.txt

deploy:
stage: deploy
script:
- ls
- cat control.txt

```

#### **Passing 3: artifacts (+committing test - stage)**

- You can decide in which state you need the artifacts

```

## only change in stage: build
image: ubuntu:20.04

## stages are set to build, test, deploy by default

build:
stage: build
script:
- echo "in building..." >> ./control.txt
artifacts:
paths:
- control.txt
expire_in: 1 week

my_unit_test:
stage: test
dependencies: []
script:
- ls -la
- echo "no control.txt here"
- ls -la

deploy:

```

```

stage: deploy
script:
  - ls
  - cat control.txt

```

### Using the gitlab - artifacts api

#### API - Reference:

- [https://docs.gitlab.com/ee/api/job\\_artifacts.html](https://docs.gitlab.com/ee/api/job_artifacts.html)

#### Reference:

- [https://docs.gitlab.com/ee/ci/pipelines/job\\_artifacts.html](https://docs.gitlab.com/ee/ci/pipelines/job_artifacts.html)

## gitlab ci/cd (Artifacts)

### Pass artifacts between jobs in same stage

- Important: needs keyword, so that they are NOT executed in parallel

```

stages:           # List of stages for jobs, and their order of execution
- build

build:
  stage: build
  script:
    - echo "in building..." >> ./control.txt
  artifacts:
    paths:
      - control.txt
    expire_in: 1 week

build-new:
  stage: build
  needs: ["build"]
  script:
    - ls -la

```

## gitlab ci/cd (Praxis II)

### Mehrzeile Kommandos in gitlab ci-cd ausführen

#### Step 1:

- Create new repo

#### Step 2: create good.sh next to README.md

- Create file good.sh in repo root-level

```

#!/bin/bash

echo "good things start now"
ls -la
date

```

#### Step 3: create gitlab-ci.yml with Pipeline Editor

```

stages:
- build

workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "web"

build-stage:
  stage: build
  variables:
    CMD: |
      echo hello-you;
      ls -la;
  script:
    - echo "execute script from git repo"
    - bash -s < good.sh
    - echo -n $CMD
    - echo "eval the command"
    - bash -c "$CMD"
    - |-

```

```

bash -s << HEREDOC
  echo hello
  ls -la
HEREDOC
- |
  tr a-z A-Z << END_TEXT
  one two three
  four five six
END_TEXT
- |
  bash -s << HEREDOC
  echo hello
  ls -la
HEREDOC
- |-
  tr a-z A-Z << END_TEXT
  one two three
  four five six
END_TEXT
- >
  echo "First command line
is split over two lines."
echo "Second command line."

```

### Run Pipeline (need to trigger manually)

#### Reference

##### Reference:

- <https://docs.gitlab.com/ee/ci/yaml/script.html#split-long-commands>
- <https://stackoverflow.com/questions/3790454/how-do-i-break-a-string-in-yaml-over-multiple-lines/21699210#21699210>

### gitlab ci/cd - Tipps & Kniffe

#### Warum before\_script ?

```

## Wir können einen hidden job definieren, der dann beim Job verwendet wird.
## Kommandos von before_script und script überschreiben sich gegenseitig

.install_dependencies:
  before_script:
    - pip install --upgrade pip
    - pip install -r requirements.txt

my_test_job:
  extends: .install_dependencies
  script:
    - pytest

```

#### GIT\_STRATEGY usw.

Es gibt tatsächlich ein GIT\_DEPTH.  
Zwei Probleme gibt es beim automatischen Clonen.  
I. Es wird nur der aktuelle branch gezogen.  
II. Die Tiefe steht standardmäßig auf 20.  
Das könnten man hochsetzen

<https://docs.gitlab.com/ee/ci/pipelines/settings.html#limit-the-number-of-changes-fetched-during-clone>

Besser wäre wahrscheinlich  
GIT\_STRATEGY none  
(bei den Variablen), dann greift nur das 2.)

Ausgabe, obwohl 2 Branches  
git branch -vr  
origin/main 5932a8c Update project1.gitlab-ci.yml

[https://docs.gitlab.com/ee/ci/runners/configure\\_runners.html#git-fetch-extra-flags](https://docs.gitlab.com/ee/ci/runners/configure_runners.html#git-fetch-extra-flags)

variables:  
GIT\_FETCH\_EXTRA\_FLAGS: --all  
script:

```

- ls -al cache/
---> RESULT: ated fresh repository.
fatal: fetch --all does not make sense with refsspecs

So all does not work here, because we fetch a specific branch with refspec

```

### Workflows + only start by starting pipeline

#### What for ?

- Configure how pipelines behaves

#### Only start pipeline by starting it with pipeline run (manually)

```

## only: web geht hier nicht, aber das steht eigentlich für:
## '$CI_PIPELINE_SOURCE == "web"'
stages:
- build

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'

build-stage:
  stage: build
  script:
    - echo "hello build"

```

#### More information about possible values for \$CI\_PIPELINE\_SOURCE

- [https://docs.gitlab.com/ee/ci/jobs/job\\_control.html#common-if-clauses-for-rules](https://docs.gitlab.com/ee/ci/jobs/job_control.html#common-if-clauses-for-rules)

#### Templates for branch and merge request workflow

#### gitlab-ci/cd - Variables

##### Variablen in Pipelines Web-Dialog anzeigen

#### gitlab - ci/cd - Pipelines strukturieren / Templates

##### Includes mit untertemplates

##### Prerequisites

```

1x main .gitlab-ci.yml
1x project1/project1.gitlab-ci.yml
1x project2/project2.gitlab-ci.yml

```

#### Step 1a: gitlab-ci.yml (simple)

```

stages:          # List of stages for jobs, and their order of execution
- build

include:
- project1/project1.gitlab-ci.yml
- project2/project2.gitlab-ci.yml

```

#### Step 1b: gitlab-ci.yml (start with pipeline start and variable setting)

```

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'

stages:          # List of stages for jobs, and their order of execution
- build

include:
- local: project1/project1.gitlab-ci.yml
  rules:
    - if: $BUILD_PROJECT1 == "true"

- local: project2/project2.gitlab-ci.yml
  rules:
    - if: $BUILD_PROJECT2 == "true"

```

```

dummy-build:
  stage: build
  script:
    - echo "dummy build"
  rules:
    - if: $BUILD_PROJECT1 != "true" && $BUILD_PROJECT2 != "true"

```

#### Step 2: project1/project1.gitlab-ci.yml

```

stages:
  - build

project1.build-job:
  stage: build
  script:
    - echo "in project1 .. building"

```

#### Step 3: project2/project2.gitlab-ci.yml

```

stages:
  - build

project2.build-job:
  stage: build
  script:
    - echo "in project2 .. building"

```

### Parent/Child Pipeline

#### Variante 1: gitlab-ci.yml (no subfolders)

```

project1:
  trigger:
    include: project1/project1.gitlab-ci.yml
    strategy: depend
  rules:
    - changes: [project1/*]
project2:
  trigger:
    include: project2/project2.gitlab-ci.yml
    strategy: depend
  rules:
    - changes: [project2/*]

```

#### Variante 2: gitlab-ci.yml (with subfolders)

```

project1:
  trigger:
    include: project1/project1.gitlab-ci.yml
    strategy: depend
  rules:
    - changes: [project1/**/*]
project2:
  trigger:
    include: project2/project2.gitlab-ci.yml
    strategy: depend
  rules:
    - changes: [project2/**/*]

```

#### Variante 3: gitlab-ci.yml (with subfolders ....)

- Not able to be started on run pipeline (manually)
- But, when it is triggered on changes

```

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
      when: never
    - when: always

project1:
  trigger:
    include: project1/project1.gitlab-ci.yml
    strategy: depend
  rules:

```

```

    - changes: [project1/**/*]
project2:
  trigger:
    include: project2/project2.gitlab-ci.yml
    strategy: depend
  rules:
    - changes: [project2/**/*]

```

#### **project1/project1.gitlab-ci.yml**

```

stages:
  - build

project1.build-job:
  stage: build
  script:
    - echo "in project1 .. building"
    - echo $CI_PIPELINE_SOURCE

```

#### **project2/project2.gitlab-ci.yml**

```

stages:
  - build

project2.build-job:
  stage: build
  script:
    - echo "in project2 .. building"
    - echo $CI_PIPELINE_SOURCE

```

#### **Alternative mit anderen stages in child**

```

stages:
  - project1-build
  - project1-test
  - project1-deploy

project1.build-job:
  stage: project1-build
  script:
    - echo "in project1 .. building"
    - echo $CI_PIPELINE_SOURCE

project1.test-job:
  stage: project1-test
  script:
    - echo "in project1 .. test"
    - echo $CI_PIPELINE_SOURCE

project1.deploy-job:
  stage: project1-deploy
  script:
    - echo "in project1 .. deploy"
    - echo $CI_PIPELINE_SOURCE

```

#### **Refs:**

- [https://docs.gitlab.com/ee/ci/pipelines/downstream\\_pipelines.html](https://docs.gitlab.com/ee/ci/pipelines/downstream_pipelines.html)

#### **Multiproject Pipeline / Downstream**

##### **Practical Example (Variant 1)**

###### **Trigger - job in .gitlab-ci.yml**

```

trigger_job:
  trigger:
    project: training.tn1/jochentest-multi1 # project/repo sonst geht es nicht (muss komplett angegeben werden)
    strategy: depend

```

###### **New repo -> training.tn1/jochentest-multi1**

```

.gitlab-ci.yml

## This is how my other project looks like
workflow:

```

```

rules:
- if: '$CI_PIPELINE_SOURCE == "web"'
- if: '$CI_PIPELINE_SOURCE == "pipeline"' # ein projekt gestart innerhalb multiproject - pipeline

default:
image: alpine

stages:          # List of stages for jobs, and their order of execution
- build

build-job:      # This job runs in the build stage, which runs first.
stage: build
script:
- echo "Started"
- echo "Show us the pipeline source $CI_PIPELINE_SOURCE"

```

#### Practical Example (Variant 2): Deploy after all Build triggers are done

```

stages:
- build
- deploy

project1:
stage: build
trigger:
  include: project1/project1.gitlab-ci.yml
  strategy: depend
# rules:
#   - changes: [project1/**/*]
project2:
stage: build
trigger:
  include: project2/project2.gitlab-ci.yml
  strategy: depend
rules:
- changes: [project2/**/*]

trigger_job:
stage: build
trigger:
  project: training.tn11/jochentest-multi2 # project/repo sonst geht es nicht (muss komplett angegeben werden)
  strategy: depend

deploy_job:
stage: deploy
image: alpine
script:
- echo "i am good to go"
- sleep 30

```

#### Reference

- [https://docs.gitlab.com/ee/ci/pipelines/downstream\\_pipelines.html?tab=Multi-project+pipeline](https://docs.gitlab.com/ee/ci/pipelines/downstream_pipelines.html?tab=Multi-project+pipeline)

#### Vorgefertigte Templates verwenden

##### Step 1: Browser Template in Pipeline Editor (Top-Bottom) to find the one you want

##### Step 2: Include template in your gitlab-ci.yml - config

```

workflow:
rules:
- if: '$CI_PIPELINE_SOURCE == "web"'

stages:          # List of stages for jobs, and their order of execution
- deploy
- test

include:
template: Jobs/Test.gitlab-ci.yml

run-deploy:
stage: deploy
script:
- echo "deploy started"

```

## Arbeiten mit extend anchor - Dinge wiederverwenden

### Hinweis:

- Dinge, die wiederverwendet werden sollen, müssen vorher definiert sein, in der Datei
- d.h. .base vor myjob
- .default\_scripts bzw &default\_scripts vor Verwendung als "default\_scripts"

### gitlab-ci.yml

```
.base:
variables:
  TEST_CASE: "true"
  VERSION: "1.0"

.default_scripts: &default_scripts
  - echo "from _default_scripts"
  - echo "next default step"

myjob:
variables:
  TEST_CASE: 'bad'
extends: .base
script:
  - *default_scripts
  - echo "in MYJOB"
  - ls -la
  - echo $TEST_CASE
  - echo $VERSION
```

## gitlab - wann laufen jobs ?

### Job nur händisch über Pipelines starten

```
## gitlab-ci.yml
stages:          # List of stages for jobs, and their order of execution
- build
- test

build-job:       # This job runs in the build stage, which runs first.
stage: build
only:
- web
image: maven
script:
- echo "Compiling the code..."
- echo "Compile complete."

unit-test-job:   # This job runs in the test stage.
stage: test      # It only starts when the job in the build stage completes successfully.
only:
- web
script:
- echo "Running unit tests... This will take about 60 seconds."
- sleep 60
- echo "Code coverage is 90%"
```

### Auch weiterlaufen, wenn Job fehlschlägt

### Walkthrough

```
stages:
- build
- deploy

default:
image: alpine

build_job1:
stage: build
allow_failure: true
script:
- xls

build_job2:
```

```

stage: build
script:
  - ls -la
  - sleep 120

build_job3:
  stage: build
  script:
    - echo "ich bin job3"

deploy_job:
  stage: deploy
  script:
    - echo "i am good to go"
    - sleep 30

```

## gitlab ci/cd - docker

### Docker image automatisiert bauen - gitlab registry

#### good.sh

```

##!/bin/bash
date

```

#### Dockerfile - RootLevel

```

FROM ubuntu:22.04
##
RUN apt-get update && \
    apt-get install -y openssh-client && \
    rm -rf /var/lib/apt/lists/*
COPY good.sh /usr/local/bin/better.sh

```

#### Variante 1: .gitlab-ci.yml (Version with docker-dind (docker-in-docker))

```

stages:          # List of stages for jobs, and their order of execution
- build

build-image:     # This job runs in the build stage, which runs first.
  stage: build
  image: docker:20.10.10
  services:
    - docker:20.10.10-dind
  script:
    - echo "user:\"$CI_REGISTRY_USER"
    - echo "pass:\"$CI_REGISTRY_PASSWORD
    - echo "registry:\"$CI_REGISTRY
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $CI_REGISTRY --password-stdin
    - docker build -t $CI_REGISTRY_IMAGE .
    - docker images
    - docker push $CI_REGISTRY_IMAGE
    - echo "BUILD for $CI_REGISTRY_IMAGE done"

```

#### Variante 2: kaniko (rootless from google)

```

build:
  stage: build
  image:
    name: gcr.io/kaniko-project/executor:v1.9.0-debug
    entrypoint: []
  script:
    - /kaniko/executor
    --context "${CI_PROJECT_DIR}"
    --dockerfile "${CI_PROJECT_DIR}/Dockerfile"
    --destination "${CI_REGISTRY_IMAGE}:${CI_COMMIT_TAG}"
  rules:
    - if: $CI_COMMIT_TAG

```

### Docker image automatisiert bauen - docker hub registry

#### Docker Hub (gitlab-ci.yml)

```

variables:
  DOCKER_REGISTRY_USER: $DOCKER_REGISTRY_USER

```

```

DOCKER_REGISTRY_PASSWORD: $DOCKER_REGISTRY_PASSWORD
DOCKER_REGISTRY_IMAGE: dockertrainereu/jochen1:latest

stages:          # List of stages for jobs, and their order of execution
- build

build-image:     # This job runs in the build stage, which runs first.
stage: build
image: docker:20.10.10
services:
- docker:20.10.10-dind
script:
- echo "user:\"$DOCKER_REGISTRY_USER"
- echo "pass:\"$DOCKER_REGISTRY_PASSWORD
- echo $DOCKER_REGISTRY_PASSWORD | docker login -u $DOCKER_REGISTRY_USER --password-stdin
- docker build -t $DOCKER_REGISTRY_IMAGE .
- docker push $DOCKER_REGISTRY_IMAGE
- echo "BUILD for $DOCKER_REGISTRY_IMAGE done"

```

## gitlab ci/cd - Documentation

### gitlab ci/cd predefined variables

- [https://docs.gitlab.com/ee/ci/variables/predefined\\_variables.html](https://docs.gitlab.com/ee/ci/variables/predefined_variables.html)

### .gitlab-ci.yml Reference

- <https://docs.gitlab.com/ee/ci/yaml/>

### Referenz: global -> workflow

- <https://docs.gitlab.com/ee/ci/yaml/#workflow>

### Referenz: global -> default

- <https://docs.gitlab.com/ee/ci/yaml/#default>

## gitlab ci/cd - Documentation - Includes

### includes

- <https://docs.gitlab.com/ee/ci/yaml/includes.html>

### includes -> rules

- <https://docs.gitlab.com/ee/ci/yaml/includes.html#use-rules-with-include>

### includes -> rules -> variables

- <https://docs.gitlab.com/ee/ci/yaml/#rulesvariables>

### includes -> templates -> override-configuration

- <https://docs.gitlab.com/ee/ci/yaml/includes.html#override-included-configuration-values>

### includes -> defaults

- <https://docs.gitlab.com/ee/ci/yaml/includes.html#use-default-configuration-from-an-included-configuration-file>

## gitlab ci/cd - Documentation - Services

### Documentation services

- <https://docs.gitlab.com/ee/ci/services/>

## 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

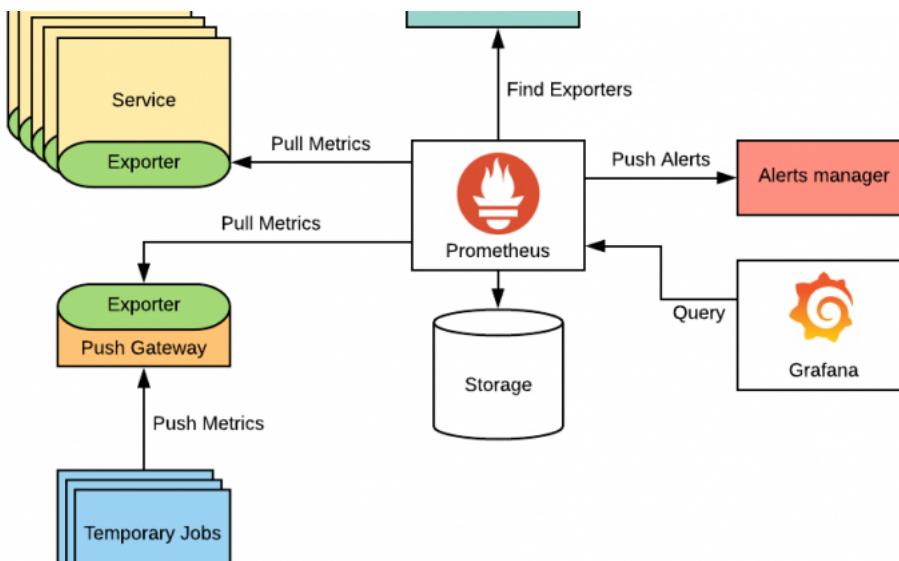
- Prometheus has a TDB (Time Series Database) and is good at storing time series with data
- Prometheus includes a local on-disk time series database, but also optionally integrates with remote storage systems.
- Prometheus's local time series database stores data in a custom, highly efficient format 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), wo meine Daten herkommen

#### Prometheus mit helm und grafana aufsetzen

- using the kube-prometheus-stack (recommended !)
- <https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack>

#### Step 1: Prepare values-file

```
cd
mkdir -p manifests
cd manifests
mkdir 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 65.1.0
```

### Step 2.1 Check ob alles läuft

```
kubectl -n monitoring get all
```

## Step 3: Connect to prometheus from the outside world

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

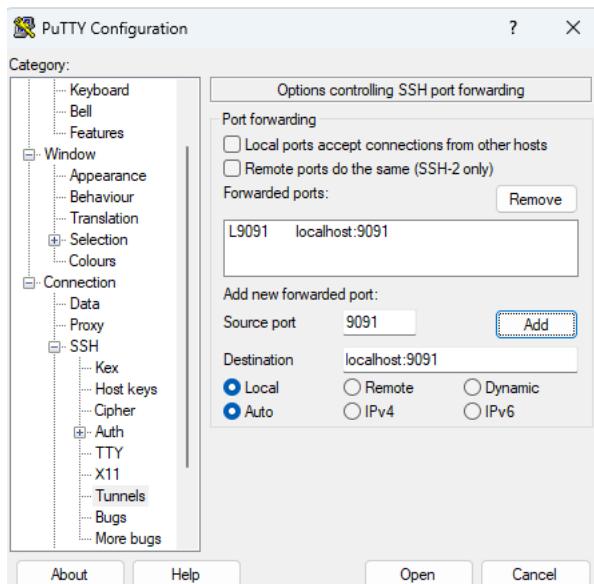
```
## this is shown in the helm information  
helm -n monitoring get notes prometheus  
  
## Get pod that runs prometheus  
kubectl -n monitoring get service  
kubectl -n monitoring port-forward svc/prometheus-prometheus 9091:http-web &
```

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

- each tln using her or his tln-nr. e.g. 2 -> 9092

```
## ip is the ip of the client machine (jump host)  
ssh -L 9091:localhost:9091 tln1@164.92.129.7
```

- Under Connection -> SSH -> Tunnels



### Step 3.3: Open prometheus in your local browser

```
## in browser with tln - nr, e.g. tln2 -> 9092  
http://localhost:9091
```

## 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

```
## 164.. -> ip of client machine (jump-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>

#### Prometheus helm-chart, setup ingress

##### Step 1: Prerequisites: Install Ingress Controller

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

kubectl -n ingress get pods
kubectl -n ingress get svc -w
```

##### Step 2: Add ip to dns

```
grafana.tln1 A 167.45.23.21
```

##### Step 3: Adjust values.yml

```
fullnameOverride: prometheus

alertmanager:
  fullnameOverride: alertmanager

grafana:
  fullnameOverride: grafana
  ingress:
    enabled: true
    ingressClassName: nginx
    # must be set, otherwise it is not working
    hosts:
      - grafana.tln1.t3isp.de

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

prometheus-node-exporter:
  fullnameOverride: node-exporter
```

##### Step 4: Run upgrade or install of installation

- Important: Please be careful about different versions (59.1 -> 65.1. have breaking changes)
- So it is more safe to delete the crds firstly or evaluate which crds are present

```
helm -n monitoring list
helm upgrade prometheus prometheus-community/kube-prometheus-stack -f values.yml --install --namespace monitoring --create-namespace --version 65.1.0
```

##### Step 5: Extract username and password of grafana

```
##admin
kubectl -n monitoring get secrets grafana -o jsonpath='{.data.admin-user}' | base64 -d
##prom-operator
kubectl -n monitoring get secrets grafana -o jsonpath='{.data.admin-password}' | base64 -d
```

##### Step 6: Now connect to grafana dashboard

```
## in browser
http://grafana.tln1.t3isp.de
## enter the credentials from Step 5
```

#### Prometheus-gui-walkthrough

##### Step 1: Prerequisites: Port-Forward svc/prometheus-prometheus 9090

- Eventually you will need to open a ssh-tunnel

##### Step 2: Now we explore:

###### 2.1. Config

- <http://localhost:9090/config> (Status -> Config)
- This config will get automatically created by the operator and objects (e.g. ServiceMonitor)

## 2.2. TSDB-Status

- Shows the status of the TSDB (Time Series Database)
- <http://localhost:9090/tsdb-status>
- nothing fancy involved here

## 2.3. Command-line flags

- With which params was prometheus started
- Only a bit helpful, because we do not use these flags in most cases
- <http://localhost:9090/flags>

## 2.4. Alerting Rules

- Defines, when alerts trigger
- All these are predefined, and this is a great library here:
  - <https://runbooks.prometheus-operator.dev/>
  - When an alert is fired, you will see this reference
- <http://localhost:9090/rules>

## 2.5. Targets (real targets being scraped)

- IMPORTANT
- These are the targets that have been discovered based on
  - ServiceMonitor
  - configuration (ServiceMonitor will always be shown in config-file) after being parsed

## 2.6. Service Discovery

- Prometheus includes a mechanism of service discovery
  - which essentially queries the kube-api-server to find out about the services
- This is also used by the ServiceMonitor, but you can also configure it manually

## Prometheus / Übung 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 9.0.1 --namespace monitoring --create-namespace
```

### Step 2: Find SVC

```
kubectl -n monitoring get svc | grep blackbox
my-prometheus-blackbox-exporter   ClusterIP  10.245.183.66  <none>        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

```
kubectl run -it --rm curltest --image=curlimages/curl -- sh
## From within curlimages/curl pod
curl http://my-prometheus-blackbox-exporter.monitoring:9115/probe?target=apple-service.default&module=http_2xx
```

### Step 5: Scrape Config (We want to get all services being annotated with example.io/should\_be\_probed = true

```
adjust values.yaml for helm chart

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 ugraden
helm upgrade prometheus prometheus-community/kube-prometheus-stack -f values.yml --install --namespace monitoring --create-namespace --version 65.1.0

```

#### 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
```

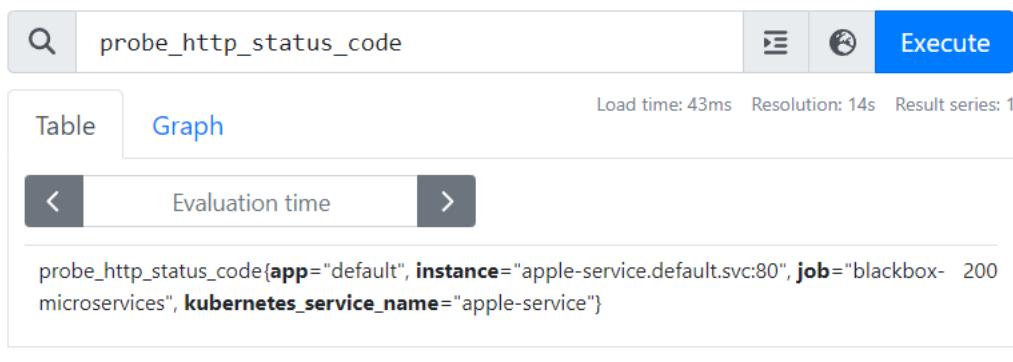
#### Step 9: Look into Status -> Discovery Services and wait

- blackbox services should now appear under blackbox\_microservices
- and not being dropped

#### Step 10: Unter Prometheus gucken

- In targets (z.B. <http://localhost:9091/targets?search=>)
- .. 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](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

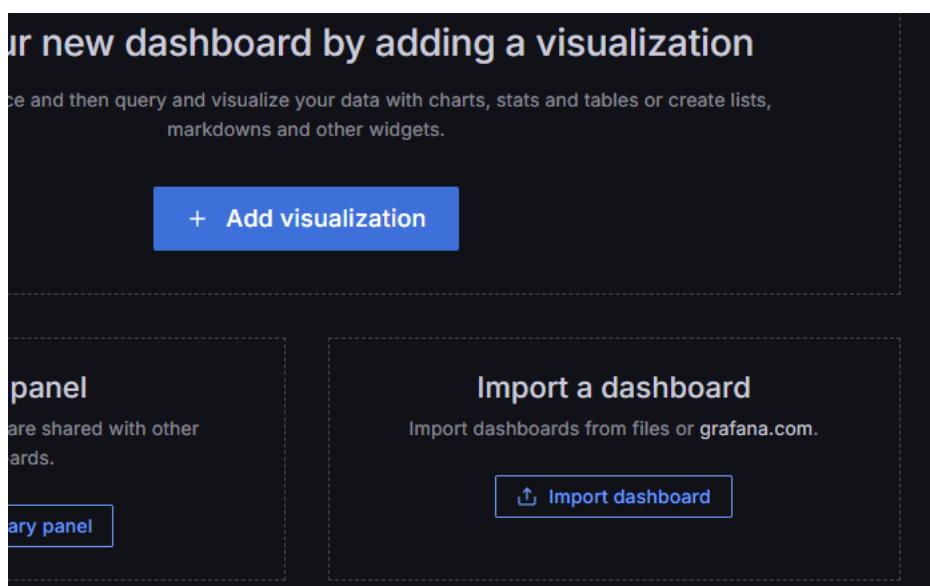
```
kubectl delete pod apple-app
```

#### 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](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 14: add Blackbox Exporter Dashboard

- <https://grafana.com/grafana/dashboards/14928-prometheus-blackbox-exporter/>
- Copy ID



Upload dashboard JSON file

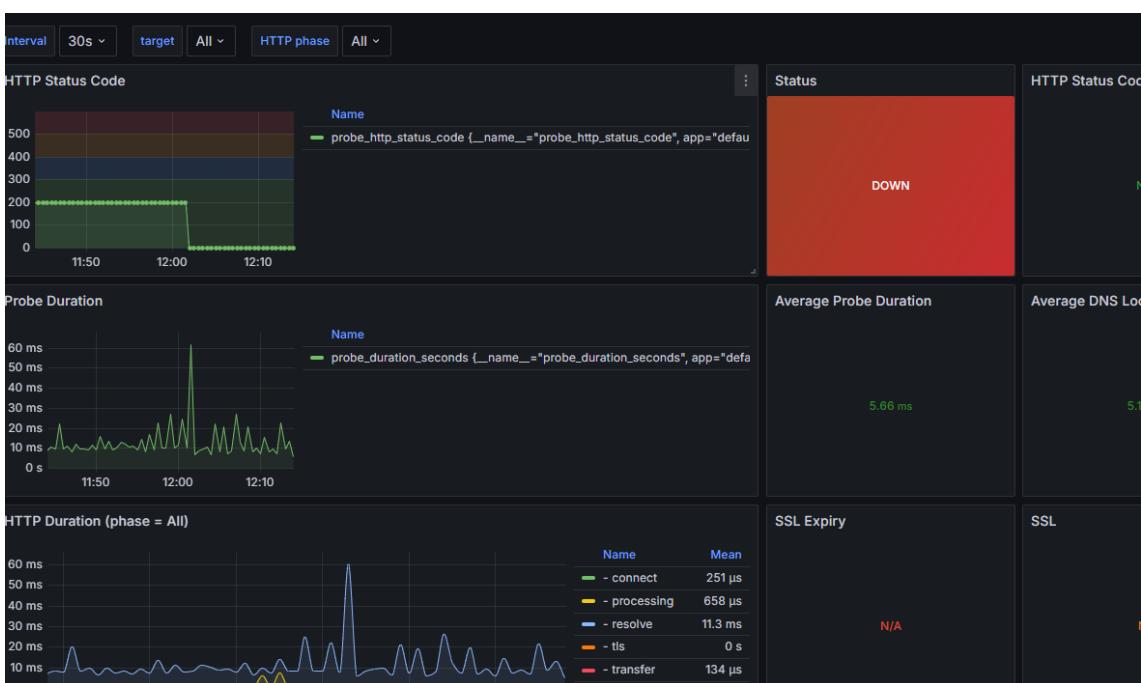
Drag and drop here or click to browse  
Accepted file types: .json, .txt

Find and import dashboards for common applications at [grafana.com/dashboards](https://grafana.com/dashboards)

14928 Load

Import via dashboard JSON model

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_0HnEoN4z",
  "panels": [...]
  ...
}
```



### Prometheus / Übung ServiceMonitor für neue App aufsetzen

#### ServiceMonitor: Step 1: Create Test app

```
mkdir testapp
nano main.js

const express = require("express");
const prometheus = require("prom-client");

const app = express();

const counter = new prometheus.Counter({
  name: "TotalRequests",
  help: "Total requests made to the service",
  labelNames: ["uri"]
});

const counterMiddleware = (req, res, next) => {
  if (req.path === "/prometheus") {
    counter.inc({uri: req.path});
  }
}
```

```

    next();
};

app.use(counterMiddleware);

app.get("/prometheus", (req, res) => {
  res.set("Content-Type", prometheus.register.contentType);
  prometheus.register.metrics().then(metrics => res.send(metrics));
});

app.get("*", (req, res) => res.send("Spacelift Prometheus demo app"));

app.listen(80, () => console.log("Ready"))

npm install express prom-client

```

#### ServiceMonitor: Step 2: Create Dockerfile, build and upload

```

nano Dockerfile

FROM node:18
WORKDIR /app

COPY *.json ./
RUN npm ci

COPY *.js ./

ENTRYPOINT ["node", "main.js"]

## user dockertrainereu
## pass: ask trainer
docker login
docker build -t dockertrainereu/testapp:latest .
docker push -t dockertrainereu/testapp:latest

```

#### ServiceMonitor: Step 3: Create Deployment / Service

```

cd
mkdir -p manifests
cd manifests
mkdir testapp
cd testapp

nano deploy.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: testapp-prometheus-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testapp-prometheus-demo
  template:
    metadata:
      labels:
        app: testapp-prometheus-demo
    spec:
      containers:
        - name: app
          image: dockertrainereu/testapp:latest
        ports:
          - name: http
            containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: testapp-prometheus-demo
  labels:
    app: testapp-prometheus-demo
spec:

```

```

selector:
  app: testapp-prometheus-demo
ports:
  - name: http
    port: 80

kubectl apply -f .

```

#### ServiceMonitor: Step 4: Connect to service

```

kubectl run -it --rm podtest --image busybox

## Do this 5 x
wget -O - http://testapp-prometheus-demo
wget -O - http://testapp-prometheus-demo/prometheus

```

#### ServiceMonitor: Step 5: Rollout ServiceMonitor

```

nano testapp-service-monitor.yaml

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: testapp-prometheus-servicemonitor
  labels:

  ## According to:
  ## prometheus - server object
  ## kubectl -n monitoring get prometheus -o yaml | grep -A 4 serviceMonitorSelector
  ##
  ## serviceMonitorSelector:
  ##   matchLabels:
  ##     release: prometheus

  release: prometheus
spec:
  endpoints:
    - port: http
      path: /prometheus
      interval: 15s
  selector:
    matchLabels:
      app: testapp-prometheus-demo

kubectl apply -f .

```

#### ServiceMonitor: Step 6: Check in the logs, if it was detected

```

kubectl -n monitoring logs prometheus-prometheus-prometheus-0 | grep -i testapp

```

#### ServiceMonitor: Step 7: Check if config was changed in prometheus

```

## port-forwarding and evtl. ssh-tunnel needs to be set
http://localhost:9090

http://localhost:9090/config # Menu: Status -> Con![image](https://github.com/user-attachments/assets/09e00e98-0ce2-4eec-9564-5379ce8de08c)
figuration

## Find the corresponding block -> job
## This got automatically created, by having the object and detecting it.
scrape_configs:
- job_name: serviceMonitor/default/testapp-prometheus-servicemonitor/0

```

#### ServiceMonitor: Step 8: Check in Service Discovery (Raw Data)

- It shows, it has detected the endpoints of the service
- It shows the labels it has found
- All labels starting with \_\_ will not be written to database
  - We will need to rewrite them

## serviceMonitor/default/testapp-prometheus-servicemonitor/0

[show less](#)

### Discovered Labels

```
_address_="10.244.0.138:80"
meta_kubernetes_endpoint_address_target_kind="Pod"
meta_kubernetes_endpoint_address_target_name="testapp-prometheus-demo-655ffc6b45-48fsb"
meta_kubernetes_endpoint_node_name="node-w66m0"
meta_kubernetes_endpoint_port_name="http"
meta_kubernetes_endpoint_port_protocol="TCP"
meta_kubernetes_endpoint_ready="true"
meta_kubernetes_endpoints_annotation_endpoints_kubernetes_io_last_change_trigger_time="2024-10-07T16:25:17Z"
meta_kubernetes_endpoints_annotationpresent_endpoints_kubernetes_io_last_change_trigger_time="true"
meta_kubernetes_endpoints_label_app="testapp-prometheus-demo"
meta_kubernetes_endpoints_labelpresent_app="true"
meta_kubernetes_endpoints_name="testapp-prometheus-demo"
meta_kubernetes_namespace="default"
meta_kubernetes_pod_container_image="dokertrainereu/testapp:latest"
meta_kubernetes_pod_container_name="app"
```

### ServiceMonitor: Step 9: Check in Targets (Rewritten Data)

- <http://localhost:9090/targets?search=> (Menu -> Status -> Targets)
- Shows the relabeled information (That is the way it is save in db and can be used, e.g. by grafana)
- (If you unfold it, also the discovered labels)

## serviceMonitor/default/testapp-prometheus-servicemonitor/0 (1/1 up)

[show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://10.244.0.138/prometheus">http://10.244.0.138/prometheus</a>	UP	<code>container="app" endpoint="http"</code> <code>instance="10.244.0.138:80"</code> <code>job="testapp-prometheus-demo"</code> <code>namespace="default"</code> <code>pod="testapp-prometheus-demo-655ffc6b45-48fsb"</code> <code>service="testapp-prometheus-demo"</code>	4.338s ago	5.949ms	

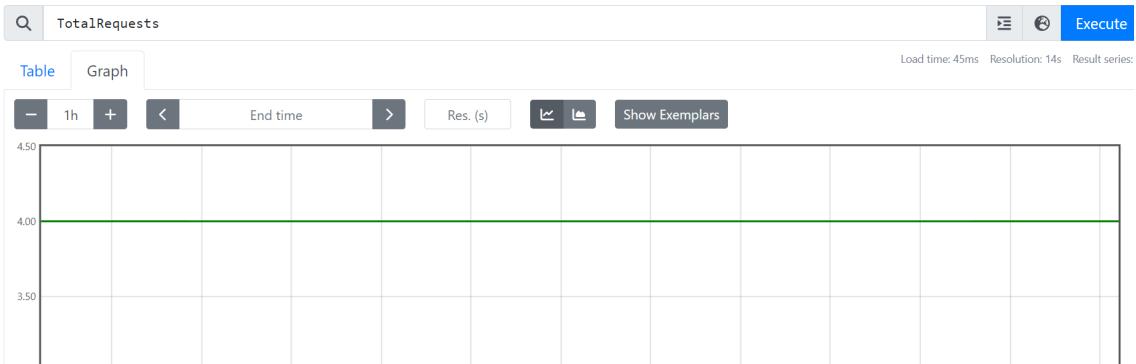
### ServiceMonitor: Step 10: Show the data

- Go to main page: <http://localhost:9090>
- Search for: Total Requests, press Execute
- Click on it: You will see the metrics collected

The screenshot shows the Grafana interface with a search bar containing 'Tot'. Below the search bar, a table titled 'Table' lists various metrics. The 'Total Requests' metric is highlighted in blue. A tooltip for this metric states: 'Total requests made to the service'. The table includes other metrics like 'node\_memory\_MemTotal\_bytes', 'node\_memory\_HugePages\_Total', etc.

Table	Series	Type
No d	TotalRequests	counter
	node_memory_MemTotal_bytes	gauge
	node_memory_HugePages_Total	gauge
	node_memory_SwapTotal_bytes	gauge
	node_memory_VmallocTotal_bytes	gauge
	node_intr_total	counter
	node_forks_total	counter
	prober_probe_total	counter
	etcd_requests_total	counter
	coredns_panics_total	counter

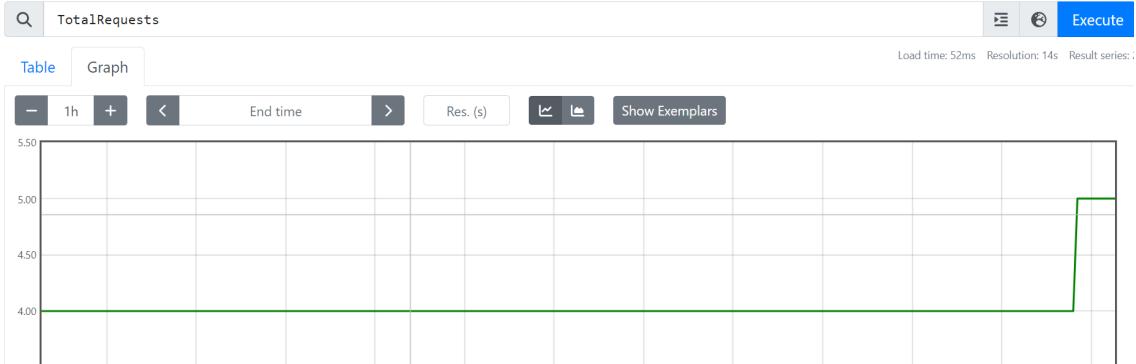
- Now click on graph -> you will see the information as graph



- Now rerun a call to the pod like earlier:

```
kubectl run -it --rm podtest --image busybox
## Do this 5 x
wget -O - http://testapp-prometheus-demo
wget -O - http://testapp-prometheus-demo/prometheus
```

- Try again after 1 minute (Main Page -> Total Requests -> Table
  - You will see the last, increased value
- Also Try: Main Page -> Total Requests -> Graph
  - You will find a spike at the end of the Graph



#### Service Monitor: Step 10: Add to grafana

```
Open grafana:
e.g.
http://grafana.tln1.t3isp.de
user: admin
pass: prom-operator
```

```
Click on: Dashboards -> New Dashboard
```

Nicht sicher grafana.tln1.t3isp.de/dashboards

The screenshot shows the Grafana interface for managing dashboards. On the left, there's a sidebar with links to Home, Starred, Dashboards (which is selected), Explore, Alerting, Connections, and Administration. The main area is titled 'Dashboards' and contains a search bar and a filter section for 'Filter by tag' and 'Starred'. A table lists existing dashboards with columns for 'Name' and 'Tags'. Some tags are color-coded: 'alertmanager-mixin' (purple), 'coredns dns' (green), 'etcd-mixin' (red), 'kubernetes-mixin' (blue), and 'kubernetes-mixin' (blue). A blue 'New' button is located in the top right corner.

Click on: Add Visualization

Select Data Source: Prometheus

Home > Dashboards > New dashboard > Edit panel

This screenshot shows the 'Edit panel' screen for creating a new dashboard. The top navigation bar includes 'Home', 'Starred', 'Dashboards' (selected), 'Explore', 'Alerting', 'Connections', and 'Administration'. The main area is titled 'Select data source' and features a search bar. Below it, a dropdown menu is open, showing 'Prometheus' (selected) and 'default' as options. Other options listed are '-- Mixed --', '-- Dashboard --', and '-- Grafana --'. There are also links for 'Use multiple data sources' and 'Reuse query results from other visualizations'.

1. Search for your TotalRequests metric in the "Select metric" dropdown within the query builder at the bottom of the screen
2. Click apply

This screenshot shows the 'Metrics explorer' interface within the Grafana dashboard editor. It displays a list of metrics under the heading 'Metrics explorer' with a 'Metrics explorer' button and an 'Open' link. The metrics listed are 'TotalRequests', 'node\_memory\_HugePages\_Total', 'node\_memory\_MemTotal\_bytes', 'node\_memory\_SwapTotal\_bytes', and 'node\_memory\_VmallocTotal\_bytes'. Below the list is a 'Run queries' button and tabs for 'Builder' and 'Code'. At the bottom of the screen, there is a query builder with fields for 'Tot', 'Q', 'Select label', '=', 'Select value', and a '+' button. A 'Hit enter to add' placeholder is visible above the builder. A 'Operations' button is located at the bottom left of the dashboard editor area.

1. Click: Run Queries to see how it goes
2. Click: Apply (top right on page)
3. Click on the save item, and give the dashboard a name, e.g. Service Testapp  
(now you can find it under dashboards -> scroll down.

#### Service Monitor: Step 11: Add it to the home screen

Star the dashboard  
(Being in the dashboard, press the start at the top left of the screen)



Set it for your user:  
(under: Home Dashboard)

Home > admin > Profile

**Profile**

Name: Name

Email: admin@localhost

Username: admin

**Save**

**Preferences**

Interface theme: Default

Home Dashboard: Default dashboard

Profile

Notification history

Change password

Sign out

Or under Organization (then it is for the whole organization)

```
## Alternative would be setting it for Teams
```

## Reference

- <https://spacelift.io/blog/prometheus-operator#how-to-set-up-servicemonitor-and-metrics-sources>

## Tipps & Tricks

### Netzwerkverbindung zum Pod testen

#### Situation

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

#### Beispiel: Eigenen Pod starten mit busybox

```
kubectl run podtest --rm -it --image busybox -- /bin/sh
## und es geht noch einfacher
kubectl run podtest --rm -it --image busybox
```

#### Example test connection

```
## wget befehl zum Kopieren
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=<container-name>
```

##### 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>

#### Weiter lernen

##### Lernumgebung

- <https://killercoda.com/>

##### Bestimmte Tasks lernen

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

##### Udemy Online Training

- <https://www.udemy.com/course/certified-kubernetes-security-specialist/>

##### Kubernetes Videos mit Hands On

- <https://www.youtube.com/watch?v=16fqzklcF7Y>

#### Documentation

##### References

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

#### gitlab ci/cd (Praxis I)

##### Variablen überschreiben/leeren

##### gitlab-ci.yml

```

default:
  image: alpine

variables:
  VAR_GLOBAL: "meine globale var"

```

```

.base:
  script: test
  variables:
    VAR1: base var 1

.job-test3:
  extends: .base
  variables: {} ## globale variable sollte danach eigentlich leer sein !!
  script:
    - echo $VAR1
    - echo "global->\"$VAR_GLOBAL"

.job-test4:
  extends: .base
  variables: null
  script:
    - echo $VAR1

```

## Rules

### CI\_PIPELINE\_SOURCE

- <https://gitlab.com/training.tn1/jochen-siegen1/-/jobs/4867098253>

### Ref:

- [https://docs.gitlab.com/ee/ci/jobs/job\\_control.html#specify-when-jobs-run-with-rules](https://docs.gitlab.com/ee/ci/jobs/job_control.html#specify-when-jobs-run-with-rules)

## gitlab ci/cd (Praxis II)

### Kommandos auf Zielsystem mit ssh ausführen (auch multiline)

#### Preparation on Server

##### Step 1: Create public/private key

```

## on destinationn server
ssh-keygen
cd .ssh
ls -la

```

##### Step 2: Add id\_rsa.pub /Public key to authorized\_keys

```
cat id_rsa.pub >> authorized_keys
```

##### Step 3: Add id\_rsa (private key) to GITLAB ci/cd -> Settings -> CI/CD as new Variable

```

cat id_rsa
## copy content and add as content to new variable SERVER_SSH_KEY
## DO not set variable as protected

```

##### create good.sh in root-folder of repo (git)

```

#!/bin/bash

echo "good things start now"
ls -la
date

```

## Create gitlab-ci.yml

```

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'

stages:          # List of stages for jobs, and their order of execution
  - deploy

deploy-job:

  stage: deploy
  image: ubuntu

  variables:
    CMD: |
      echo hello-you;
      ls -la;

```

```

before_script:
- apt -y update
- apt install -y openssh-client
- eval $(ssh-agent -s)
- echo "$SERVER_SSH_KEY" | tr -d '\r' | ssh-add -
- ls -la
- mkdir -p ~/.ssh
- chmod 700 ~/.ssh
- ssh-keyscan $SERVER_IP >> ~/.ssh/known_hosts
- chmod 644 ~/.ssh/known_hosts
##- echo $SERVER_SSH_KEY

script:
- echo 'V1 - commands in line'
#####
# For ssh to exit on error, start your commands with first command set -e
# - This will exit the executed on error with return - code > 0
# - AND will throw an error from ssh and in pipeline
#####
- ssh root@$SERVER_IP -C "set -e; ls -la; cd $SERVER_WEBDIR; ls -la;"
- echo 'V2 - content of Variable $CMD'
- ssh root@$SERVER_IP -C $CMD
- echo 'V3 - script locally - executed remotely'
- ssh root@$SERVER_IP < good.sh
- echo 'V4 - script in heredoc'
- |
  ssh root@$SERVER_IP bash -s << HEREDOC
  echo "hello V4"
  ls -la
HEREDOC
- echo 'V5 - copy script and execute'
- scp good.sh root@$SERVER_IP:/usr/local/bin/better.sh
- ssh root@$SERVER_IP -C "chmod u+x /usr/local/bin/better.sh; better.sh"

```

## gitlab ci/cd docker compose

### Docker compose local testen

#### Evolutions-Phase 1: Testen eines docker compose file lokal auf unserem Zielsystem

```

## public/private key muss eingerichtet sein
ssh root@<ziel-ip>

```

### Docker installieren

```

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

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

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

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

```

```

### Läuft der Dienst (dockerd)
systemctl status docker

```

```

mkdir cms
cd cms
nano docker-compose.yaml

```

```

services:
  db:
    # We use a mariadb image which supports both amd64 & arm64 architecture
    image: mariadb:10.6.4-focal
    # If you really want to use MySQL, uncomment the following line
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'

```

```

volumes:
  - db_data:/var/lib/mysql
restart: always
environment:
  - MYSQL_ROOT_PASSWORD=somewordpress
  - MYSQL_DATABASE=wordpress
  - MYSQL_USER=wordpress
  - MYSQL_PASSWORD=wordpress
expose:
  - 3306
  - 33060
wordpress:
  image: wordpress:latest
  volumes:
    - wp_data:/var/www/html
  ports:
    - 80:80
  restart: always
  environment:
    - WORDPRESS_DB_HOST=db
    - WORDPRESS_DB_USER=wordpress
    - WORDPRESS_DB_PASSWORD=wordpress
    - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:

```

```
docker compose up -d
```

### Docker compose über ssh

#### Evolutions-Phase 2: Anwenden eines Docker Compose files über ssh

Vorbereitung: Im Repo cms/docker-compose.yaml einfügen

```

services:
  db:
    # We use a mariadb image which supports both amd64 & arm64 architecture
    image: mariadb:10.6.4-focal
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:

```

#### Schritt 1: gitlab-ci.yaml

```

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
stages:          # List of stages for jobs, and their order of execution
  - deploy

```

```

deploy-job:
  stage: deploy
  image: ubuntu

  before_script:
    - apt-get -y update
    - apt-get install -y openssh-client ca-certificates curl gnupg lsb-release
    - mkdir -p /etc/apt/keyrings
    # We want the newest version from docker
    # version from ubuntu repo does not work (docker compose) - version too old
    - curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
    - echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
    - apt-get update -y
    - apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
    - eval $(ssh-agent -s)
    - echo "$SERVER_SSH_KEY" | tr -d '\r' | ssh-add -
    - ls -la
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan SERVER_IP >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
    # - echo $SERVER_SSH_KEY
    # eventually not needed
    #- echo $SERVER_SSH_KEY > ~/.ssh/id_rsa
    #- chmod 600 ~/.ssh/id_rsa

script:
  - echo 'Deploying wordpress'
  - cd cms
  - export DOCKER_HOST="ssh://root@$TOMCAT_SERVER_IP"
  - docker info
  - docker container ls
  - docker compose up -d

```

### Docker compose classic über scp

#### Evolutions-Phase 3:

##### Vorbereitung:

- SSH\_SERVER\_KEY (private key), SERVER\_IP als Variablen in Settings -> CI/CD angelegt werden.

#### Schritt 1: cms/docker-compose.yaml in gitlab anlegen

```

services:
  db:
    image: mariadb:10.6.4-focal
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
  volumes:
    db_data:
    wp_data:

```

#### Schritt 2: gitlab-ci.yaml

```

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'

stages:           # List of stages for jobs, and their order of execution
  - deploy

deploy-job:
  stage: deploy
  image: ubuntu

  before_script:
    - apt-get -y update
    - apt-get install -y openssh-client
    - eval $(ssh-agent -s)
    - echo "$SERVER_SSH_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan $SERVER_IP >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
    #- echo $SERVER_SSH_KEY
    # eventually not neededrsa

  script:
    - echo 'Deploying wordpress'

    - |
      ssh root@$SERVER_IP bash -s << HEREDOC
      cd
      mkdir -p cms
      HEREDOC
    - scp cms/docker-compose.yaml root@$SERVER_IP:~/cms/
    - ssh root@$SERVER_IP -C "cd; cd cms; docker compose up -d"

```

#### Schritt 3: Pipeline manuell über pipeline menü starten

### 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: |
    <html>
      <h1>Welcome to Version 1</h1>
      <br>
      <h1>Hi! This is a configmap Index file Version 1 </h1>
    </html>

## 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: |
    <html>
      <h1>Welcome to Version 2</h1>
      <br>
      <h1>Hi! This is a configmap Index file Version 2 </h1>
    </html>

## 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 <external-ip>:<node-port>

```

## 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

### 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"

apiVersion: v1

kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: nginx

```

```

resources:
  limits:
    memory: "200Mi"
    cpu: "700m"

  requests:
    memory: "200Mi"
    cpu: "700m"

```

## 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 01-pod-liveness-command.yaml
kubectl describe -l test=liveness pods
sleep 30
kubectl describe -l test=liveness pods
sleep 5
kubectl describe -l test=liveness pods

## cleanup
kubectl delete -f 01-pod-liveness-command.yaml

```

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

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

```

#### Reference:

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

#### 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/>

## Kubernetes - Überblick

### Allgemeine Einführung in Container (Dev/Ops)

#### Architektur



#### 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:

- 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 kleinen 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 Pods)
- Automatische Überwachung mit deklarativem Management) - ich beschreibe, was ich will
- Neue Versionen zu auszurollen (Canary Deployment, Blue/Green Deployment)

## Mögliche Nachteile

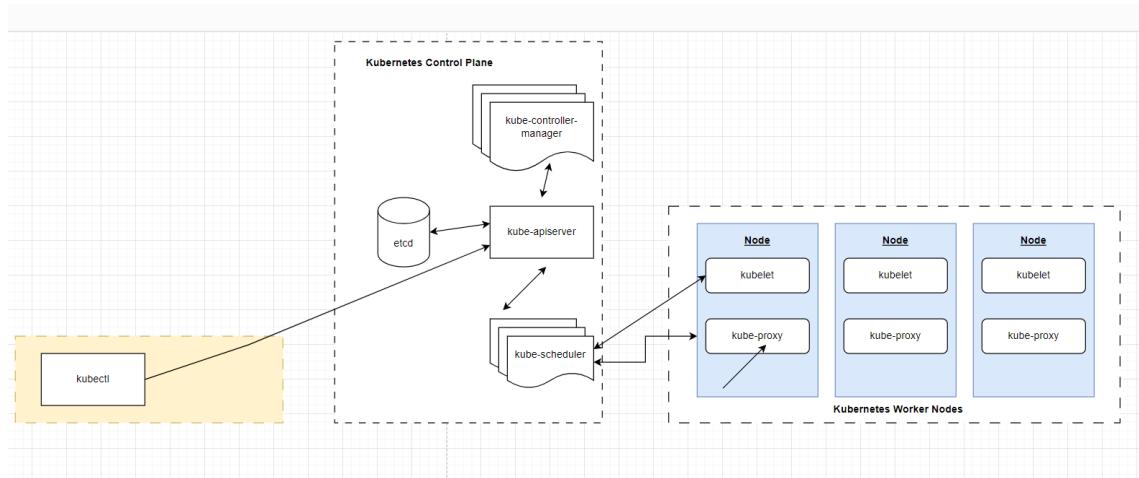
- Steigert die Komplexität.
- Debugging wird u.U. schwieriger
- Mit Kubernetes erkaufe ich mir auch, die Notwendigkeit.
  - Über adequate Backup-Lösungen nachzudenken (Moving Target, Kubernetes Aware Backups)
  - Bereitsstellung von Monitoring Daten Log-Aggregierungslösung

## Klassische Anwendungsfällen

- Webbasierte Anwendungen (z.B. auch API's bzw. Web)

## Aufbau Allgemein

### Schaubild



## Komponenten / Grundbegriffe

### Master (Control Plane)

#### Aufgaben

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

#### Komponenten des Masters

##### ETCD

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

##### KUBE-CONTROLLER-MANAGER

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

##### KUBE-API-SERVER

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

##### KUBE-SCHEDULER

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

#### Nodes

- 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.
```

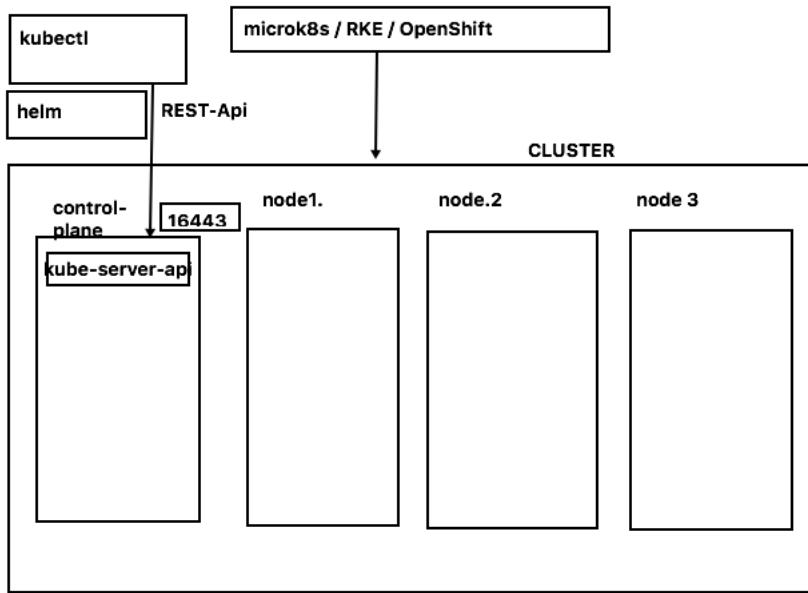
##### Kube-proxy

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

#### Referenzen

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

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



Welches System ? (minikube, microk8s etc.)

## Überblick der Systeme

### General

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

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

### Kubeadm

#### General

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

#### Disadvantages

- Plugins sind oftmals etwas schwierig zu aktivieren

### microk8s

#### General

- Created by Canonical (Ubuntu)
- Runs on Linux
- Runs only as snap
- In the meantime it is also available for Windows/Mac
- HA-Cluster

#### Production-Ready ?

- Short answer: YES

Quote canonical (2020) :

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

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

#### Advantages

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

### minikube

## Disadvantages

- Not usable / intended for production

## Advantages

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

## k3s

### kind (Kubernetes-In-Docker)

#### General

- Runs in docker container

#### For Production ?

Having a footprint, where kubernetes runs within docker  
and the applications 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:fe4:cbce/64
## private ip
eth1 UP 10.135.0.3/16 fe80::8081:aaff:feaa:780/64

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

## Funktioniert microk8s
microk8s status
```

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

```
## Was macht das ?
## 1. Basisnutzer (11trainingdo) - keine Voraussetzung für microk8s
## 2. Installation von microk8s
##.>>>> microk8s installiert <<<<<
## - snap install --classic microk8s
## >>>> Zuordnung zur Gruppe microk8s - notwendig für bestimmte plugins (z.B. helm)
## usermod -a -G microk8s root
## >>>> Setzen des .kube - Verzeichnisses auf den Nutzer microk8s -> nicht zwingend erforderlich
## chown -r -R microk8s ~/.kube
## >>>> REQUIRED .. DNS aktivieren, wichtig für Namensauflösungen innerhalb der PODS
## >>>> sonst funktioniert das nicht !!!
## microk8s enable dns
## >>>> kubectl alias gesetzt, damit man nicht immer microk8s kubectl eingeben muss
## - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

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

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
```

```

- echo " " >> /etc/ssh/sshd_config
- echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
- echo "AllowUsers root" >> /etc/ssh/sshd_config
- systemctl reload sshd
- sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.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:fea4:cbce/64
## private ip
eth1    UP          10.135.0.5/16 fe80::8081:aaff:feaa:780/64

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

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

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

## Mini-Schritt 2:
## Auf dem Client 1 (diese Maschine) kubeconfig an die richtige Stelle bringen
## Standardmäßig der Client nach 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
## 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
## mikrok8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 3:
## Server 1: token besorgen für node 3
## mikrok8s add-node

## Mini-Schritt 4:
## Server 3: Den Node hinzufügen durch den JOIN-Befehl
## mikrok8s 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$4xSfDFQjKWfAoGkZF3LFAXM4hg13d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo

```

### Kubernetes - mikrok8s (Installation und Management)

#### Installation Ubuntu - snap

#### Walkthrough

```

sudo snap install mikrok8s --classic
## Important enable dns // otherwise not dns lookup is possible
mikrok8s 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 10.128.63.86:25000/567a21bdxfc9a64738ef4b3286b2b8a69

```

## 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

### Connect to external database

#### Prerequisites

- MariADB - Server is running on digitalocean in same network as dok (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 iputils-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

```

### 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

## 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
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx

## 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

  kubectl apply -f .

```

##### Schritt 2: Auflösung Namen.

```

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

## web ist der name des statefulsets
ping web-0.nginx
ping web-1.nginx

kubectl delete sts web
kubectl apply -f .
kubectl run --rm -it podtest --image=busybox

ping web-0.nginx

```

## Referenz

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

## 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 geupdated 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
        - 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_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 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)

```

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

```

#### Schritt 2: Walkthrough - Server Installation mit kubectl client

```

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

```

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

```

kubeseal --fetch-cert

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

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

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

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

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

kubectl apply -f basic-auth-sealed.yaml

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

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

```

#### Hinweis: Ubuntu snaps

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

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

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

#### Ref:

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

### 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)

#### 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>

## 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/>

## 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

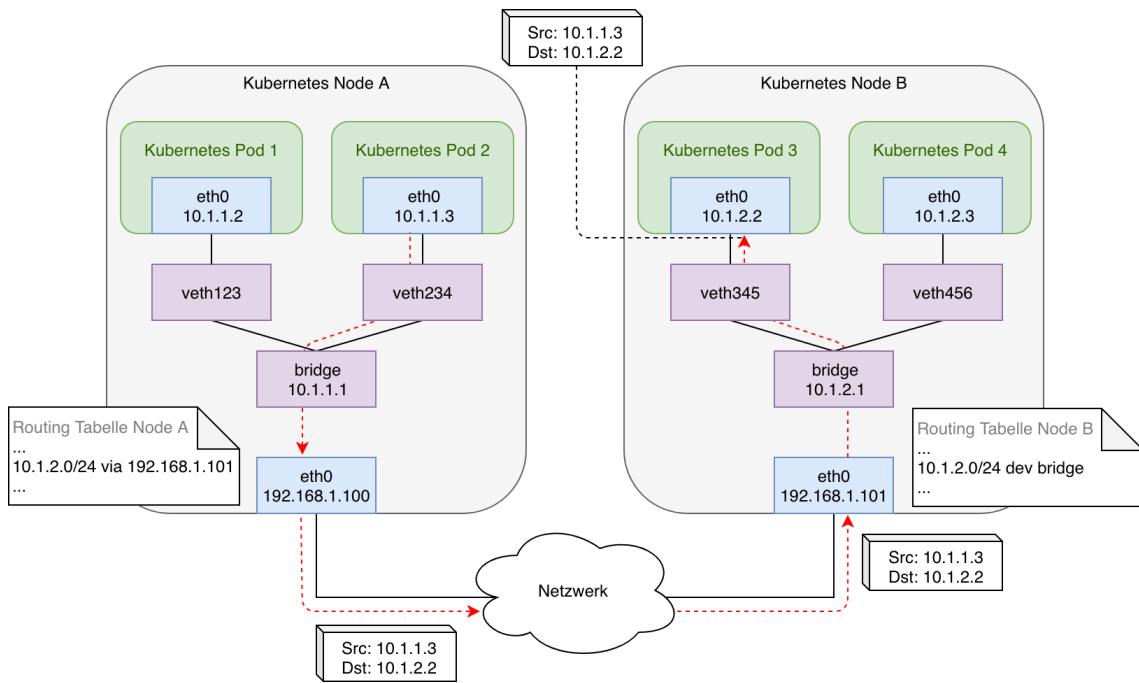
```
## now start deployment again
kubectl apply -f 03-deploy.yml

## and try connection again
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>:<port> # port -> > 30000
## exit
```

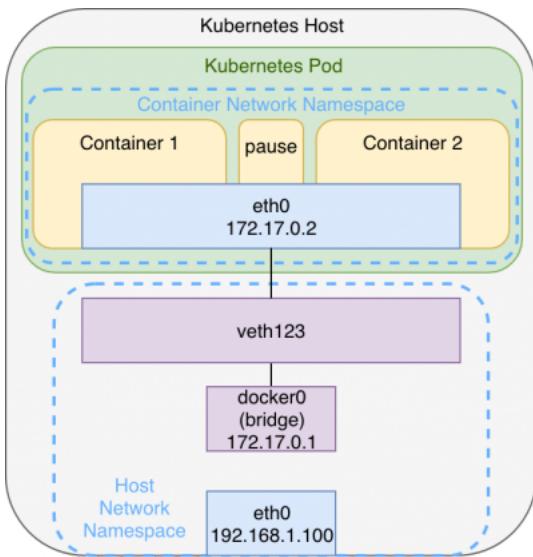
## Kubernetes Netzwerk

### Kubernetes Netzwerke Übersicht

Show us



### Die Magie des Pause Containers



## 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 runterfahren -> über CNI -> Netzwerk - IP wird released

## Welche gibt es ?

- Flannel
- Canal
- Calico
- Cilium

## Flannel

### Overlay - Netzwerk

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

### Vorteile

- Guter einfacher Einstieg
- reduziert auf eine Binary flanneld

### Nachteile

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

## Canal

### General

- Auch ein Overlay - Netzwerk
- Unterstützt auch policies

## Calico

### Generell

- klassische Netzwerk (BGP)

### Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

### Vorteile

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

### Referenz

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

## 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**

```
kubectl run podtest --rm -ti --image busybox -- /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget -O - http://apple-service.jochen
Connecting to apple-service.jochen (10.245.39.214:80)
writing to stdout
apple-tln1
-
          100%
|*****| 11  0:00:00
ETA
written to stdout
/ # wget -O - http://apple-service.jochen.svc.cluster.local
Connecting to apple-service.jochen.svc.cluster.local (10.245.39.214:80)
writing to stdout
apple-tln1
-
          100%
|*****| 11  0:00:00
ETA
written to stdout
/ # wget -O - http://apple-service
Connecting to apple-service (10.245.39.214:80)
writing to stdout
apple-tln1
-
          100%
|*****| 11  0:00:00
ETA
written to stdout
```

## **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**

```
<tln> = nix
z.B.
policy-demo<tln> => policy-demo
```

### **Gruppe mit einem einzigen Cluster**

```
<tln> = Teilnehmernummer
z.B.
policy-demo<tln> => policy-demo
```

### **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>

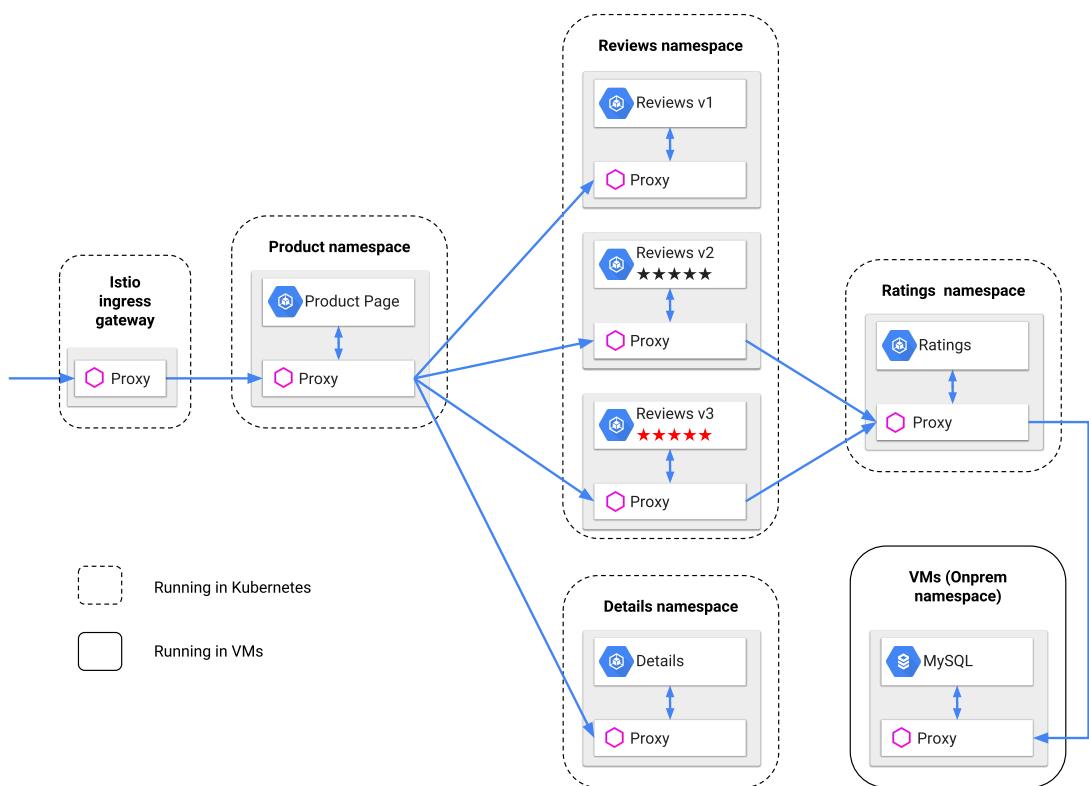
```

### 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>

### Sammlung istio/mesh

#### Schaubild



## 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.yml
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

#### 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<r>/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

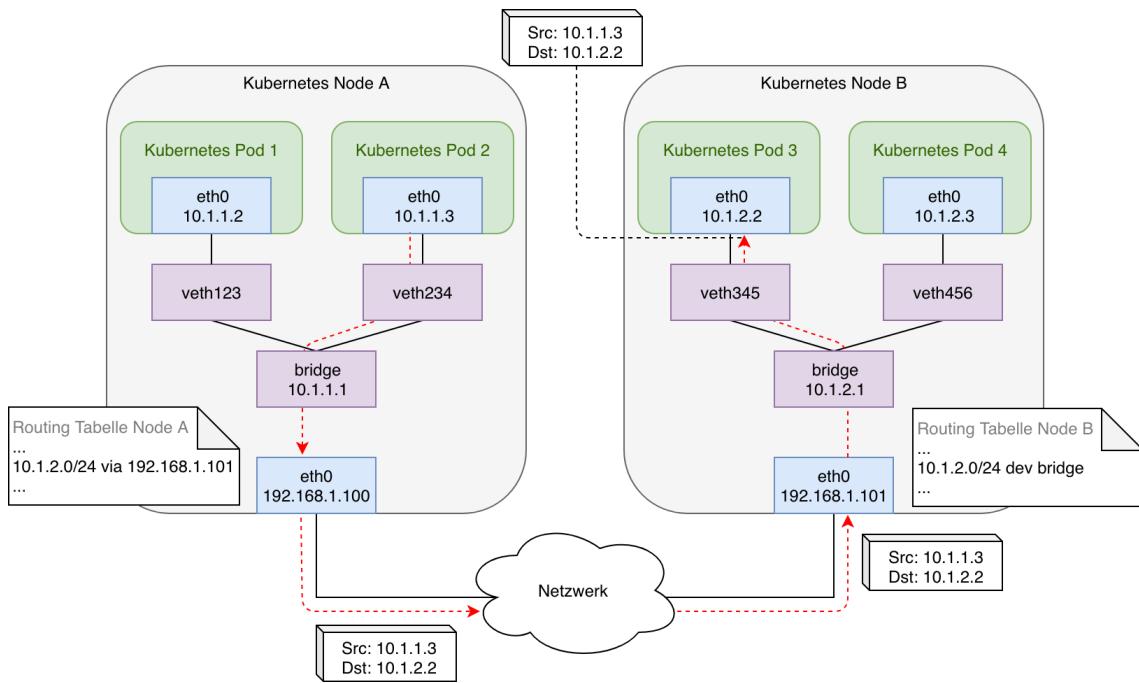
## and try connection again
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>:<port> # port -> > 30000
## exit

```

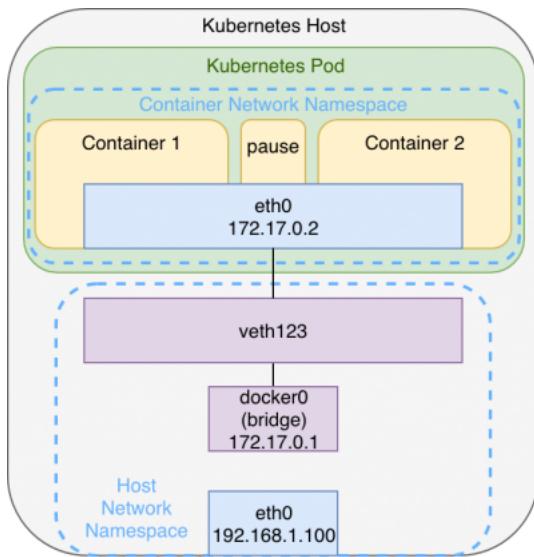
## Kubernetes Networking

### Überblick

#### Show us



#### Die Magie des Pause Containers



### 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 runterfahren -> über CNI -> Netzwerk - IP wird released

### Welche gibt es ?

- Flannel
- Canal
- Calico
- Cilium

### Flannel

#### Overlay - Netzwerk

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

#### Vorteile

- Guter einfacher Einstieg
- reduziert auf eine Binary flanneld

#### Nachteile

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

### Canal

#### General

- Auch ein Overlay - Netzwerk
- Unterstützt auch policies

### Calico

#### Generell

- klassische Netzwerk (BGP)

#### Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

#### Vorteile

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

#### Referenz

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

### 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.
```

## **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>

## 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-k4dmb9cbmb 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.yml
resources:
- service.yml

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

## See how it looks like
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 Porteneinstellungen 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
```

```
kubectl 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.yaml
```

##### 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 flag --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)**

```
kubectl auth can-i get pods -n default --as system:serviceaccount:default:training<nr> # 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<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> 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>

## 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 separate page
- <https://blog.kasten.io/kubernetes-backup-and-restore-for-mysql>

#### Ref:

- <https://www.kasten.io>
- [Installation DigitalOcean](#)
- [Installation Kubernetes \(Other distributions\)](#)

## 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>
```

#### 2. Schritt Pods analysieren, die Anfrage bekommen

```
## 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.

#### 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>

#### 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 - mikrok8s

#### 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

- privileges - 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.yaml
```

```
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
```

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

```
## 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<tn> get pods  
kubectl -n test-ns<tn> describe pods nginx
```

#### Praktisches Beispiel für Version ab 1.23 -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<tn> 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

#### 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 ctermfg=lightred ctermbg=white
autocmd FileType yaml 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

#### Allgemein

```

## Zeige Informationen über das Cluster
kubectl cluster-info

## Welche api-resources gibt es ?
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

## 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 separately on Client (or where you have your manifests)

#### Walkthrough

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert"
## Validate the checksum
curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert.sha256"
echo "$(<kubectl-convert.sha256) kubectl-convert" | sha256sum --check
## install
sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert

## Does it work
kubectl convert --help

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

### Reference

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

### 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
```

## Kubernetes 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/eks\\_cluster\\_module.html](https://docs.ansible.com/ansible/latest/collections/community/aws/eks_cluster_module.html)

### Empfehlungen Applikation

- Eigene Repos mit manifesteren (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

## 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
- kubellet -> redet mit CRI (Container Runtime Interface) -> Redet mit Container Runtime z.B. containerd (Docker), CRI-O (Redhat)
  - [CRI](#)

## Hintergründe

- Container Runtime (CRI-O, containerd)
- [OCI image \(Spezifikation\)](#)
- OCI container (Spezifikation)
- [Sehr gute Lernreihe zu dem Thema Container \(Artikel\)](#).

## 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 gedrained 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 umgeswichtzt.

<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<nr> # <nr> entsprechend eintragen  
  namespace: default
```

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

### 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-<nr> # für <nr> 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 ClusterRole 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)

```
kubectl auth can-i get pods -n default --as system:serviceaccount:default:training<nr> # 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 # <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> 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/contengaddingserviceacctoken.htm>
- <https://microk8s.io/docs/multi-user>
- <https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286>

## 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 applikation 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

## Hyperscaler 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 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/>

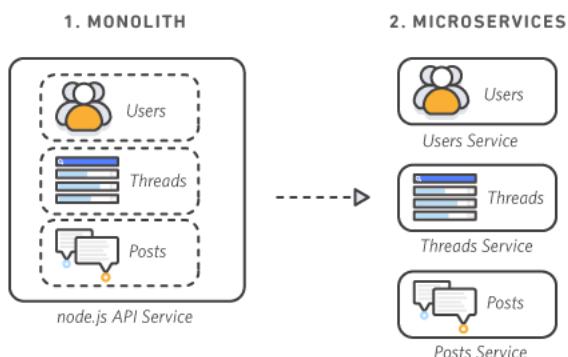
#### rancher for desktop

- <https://github.com/rancher-sandbox/rancher-desktop/releases/tag/v1.9.1>

## Microservices

### Microservices vs. Monolith

#### Schaubild



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 dass 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**

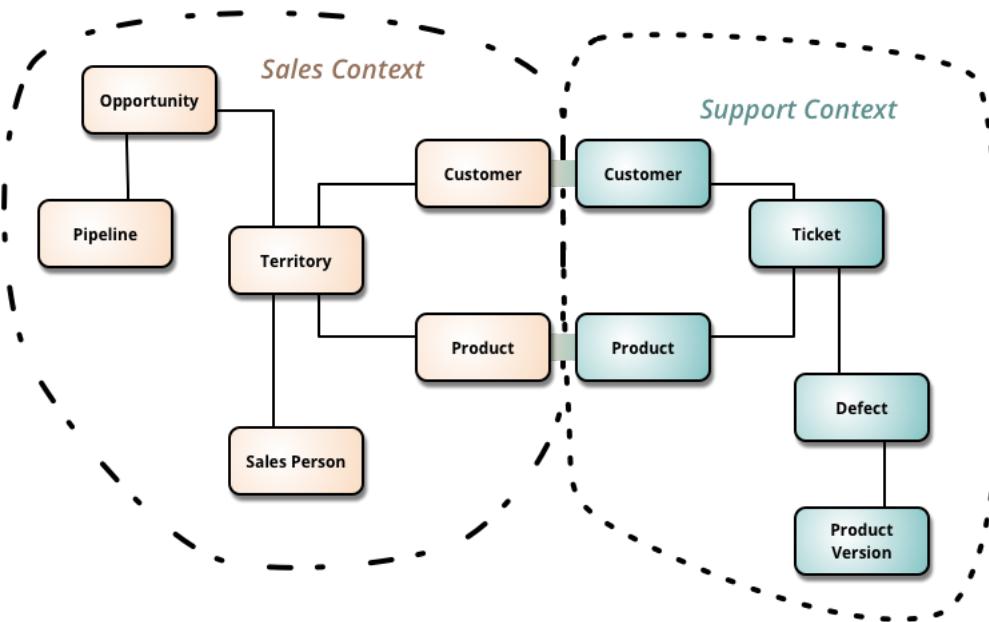
##### **Wie kann ich schneiden (NOT's) ?**

- 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



#### 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  
Leihe (bounded context 1)  
Suche (bounded context 2)

#### Strategic Patterns - von 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 eines nachgelagerten Prozesses 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 acce
docker exec -it acce 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

Zum Beenden des Vollbildmodus F11 drücken

```
tree apps/
apps/
├── README.md
├── app-proxy
│   ├── base
│   │   └── kustomization.yaml
│   └── overlays
│       └── aws-prod
│           ├── config.json
│           ├── ingress.yaml
│           └── kustomization.yaml
└── argocd-agent
    ├── base
    │   └── kustomization.yaml
    └── overlays
        └── aws-prod
            ├── config.json
            └── kustomization.yaml
aws-prod-apps
└── aws-prod
    └── config_dir.json
default-git-source
└── aws-prod
    └── config_dir.json
demoapp
├── base
│   └── kustomization.yaml
└── overlays
    └── aws-prod
        ├── config.json
        └── kustomization.yaml
events
├── base
│   └── kustomization.yaml
└── overlays
    └── aws-prod
        ├── config.json
        └── kustomization.yaml
```

- Source: [https://www.reddit.com/r/kubernetes/comments/sd50hk/kustomize\\_with\\_multiple\\_deployments\\_how\\_to\\_keep/](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

#### 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 (Use Cases)**

##### **Case Studies Kubernetes**

- <https://kubernetes.io/case-studies/>

##### **Use Cases**

- <https://codilime.com/blog/harnessing-the-power-of-kubernetes-7-use-cases/>

#### **Documentation (Komponenten)**

##### **controller manager**

- <https://github.com/kubernetes/kubernetes/tree/release-1.29/cmd/kube-controller-manager/app/options>