

# Microservices mit Docker und Kubernetes

## Agenda

### 1. Grundlagen

- [Was sind Microservices ?](#)
- [Grundkonzepte von Microservices](#)
- [Monolith vs. Microservices](#)
- [Praxisbeispiele](#)
- [Was ist devops](#)
- [API-Abfrage über REST-API](#)
- [Asynchrones Messaging](#)
- [Microservice and Database](#)

### 2. Grundwissen Microservices (Teil 2)

- [Brainstorming Domäne](#)
- [Datenbank - Patterns - Teil 1](#)
- [Datenbank - Patterns - Teil 2](#)
- [Strategische Patterns](#)
- [Tests](#)
- [Monolith schneiden microservices](#)
- [EventBus Implementierungen/Überblick](#)

### 3. Linux Tipps & Tricks

- [In den Root-Benutzer wechseln](#)

### 4. Docker-Grundlagen

- [Übersicht Architektur](#)
- [Was ist ein Container ?](#)
- [Was sind container images](#)
- [Container vs. Virtuelle Maschine](#)
- [Was ist ein Dockerfile](#)

### 5. Docker-Installation

- [BEST for Ubuntu : Install Docker from Docker Repo](#)

### 6. Docker-Praxis

- [Docker run mit nginx](#)
- [Die wichtigsten Befehle](#)
- [Aufräumen - container und images löschen](#)
- [Logs des Host-Systems zu den Containern auslesen](#)
- [Logs anschauen - docker logs - mit Beispiel nginx](#)
- [Nginx mit portfreigabe laufen lassen](#)

### 7. Example with Dockerfile

- [Ubuntu mit ping](#)
- [Slim multistage-build](#)

### 8. Docker Security

- [Docker Security](#)
- [Scanning docker image with docker scan/snyx\(Deprecated\)](#)

### 9. Docker Compose

- [Ist docker-compose installiert?](#)
- [Example with Wordpress / MySQL](#)
- [Example with Ubuntu and Dockerfile](#)
- [Logs in docker - compose](#)
- [docker compose Reference](#)

### 10. Docker - compose (Testprojekte)

- [Testprojekt mit api und mongodb](#)

### 11. Microservices - Daten

- [Überblick shared database / database-per-service](#)
- [Umgang mit Joins bei database-per-service](#)
- [Umgang mit Transaktionen bei database-per-service](#)
- [Event Sourcing](#)

### 12. Microservices (async messaging)

- [Topic/Queue ohne Downtime migrieren](#)
- [Disruptive Änderungen im Schema migrieren](#)

### 13. Microservice - flightapp - concepts

- [Vorgehensweise nach dem SEED-Verfahren](#)
- [Vorgehensweise nach SEED on Detail](#)

#### 14. Microservice - flightapp - reservations

- [Template for microservice with python flask](#)
- [Create microservice - reservations](#)
- [Upload image microservice - reservations](#)
- [Build image reservations with gitlab ci/cd](#)

#### 15. Microservice - flightapp - flights

- [Template for microservice flights with node bootstrap](#)
- [Build flight app](#)
- [Upload image flight app](#)

#### 16. Microservice - flightapp - Deployment Kubernetes

- [Manual deployment](#)
- [gitlab Deployment](#)
- [github Deployment](#)
- [github Deployment-with-secret-not-working](#)

#### 17. Kubernetes - Überblick

- [Warum Kubernetes, was macht Kubernetes](#)
- [Aufbau Allgemein](#)
- [Structure Kubernetes Deep Dive](#)
- [Aufbau mit helm, OpenShift, Rancher\(RKE\), microk8s](#)
- [Welches System ? \(minikube, micro8ks etc.\)](#)

#### 18. Kubernetes - Einsatz

- [Kubernetes Einsatz -> Risiken](#)
- [Kubernetes Datenbanken in Kubernetes oder ausserhalb](#)

#### 19. Kubernetes mit microk8s (Installation und Management)

- [Installation Ubuntu - snap](#)
- [Create a cluster with microk8s](#)
- [Remote-Verbindung zu Kubernetes \(microk8s\) einrichten](#)

#### 20. Kubernetes mit k3s

- [Kubernetes mit k3s](#)

#### 21. Kubernetes - Client Tools und Verbindung einrichten

- [Tools installieren und bash-completion / syntax highlighting](#)
- [Remote-Verbindung zu Kubernetes einrichten](#)
- [Tool zum Konvertion von docker-compose.yaml file manifesten](#)

#### 22. 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](#)
- [Services - Aufbau](#)
- [kubectl/manifest/service](#)
- [DaemonSets \(Devs/Ops\)](#)
- [Hintergrund Ingress](#)
- [Ingress Controller auf Digitalocean \(doks\) mit helm installieren](#)
- [Documentation for default ingress nginx](#)
- [Beispiel Ingress](#)
- [Install Ingress On Digitalocean DOKS](#)
- [Beispiel Ingress mit Hostnamen](#)
- [Achtung: Ingress mit Helm - annotations](#)
- [Permanente Weiterleitung mit Ingress](#)
- [ConfigMap Example](#)
- [Configmap MariaDB - Example](#)
- [Configmap MariaDB my.cnf](#)
- [Secret MariaDB - Example](#)

#### 23. Kubernetes Praxis (Teil 2) - API Objekte

- [Hintergrund Statefulsets](#)
- [Übung Statefulsets](#)

#### 24. Kubernetes Praxis (Teil 3)

- [Using private registry](#)

- 25. Kubernetes Ingress
  - [Ingress Controller on Detail](#)
- 26. ServiceMesh
  - [Why a ServiceMesh ?](#)
  - [How does a ServiceMeshes work? \(example istio\)](#)
  - [istio security features](#)
  - [istio-service mesh - ambient mode](#)
  - [istio-traffic-management](#)
  - [Performance comparison - baseline, sidecar, ambient](#)
- 27. Kubernetes (Debugging)
  - [Netzwerkverbindung zu pod testen](#)
- 28. Kubernetes Netzwerk
  - [DNS - Resolution - Services](#)
- 29. Kubernetes Scaling
  - [Autoscaling Pods/Deployments](#)
- 30. Kubernetes Tipps & Tricks
  - [Oomkiller and maxReadySeconds for safe migration to new pods](#)
  - [Pod-Netzwerk debuggen durch weiteren Pod der daneben liegt kubectl debug](#)
  - [Aus pod mit curl api-server abfragen](#)
- 31. Kubernetes - Monitoring
  - [metrics-server aktivieren \(microk8s und vanilla\)](#)
  - [Prometheus Überblick](#)
  - [Prometheus Kubernetes Stack installieren](#)
  - [Prometheus - Services scrapen die keine Endpunkte für Prometheus haben](#)
- 32. Helm
  - [Helm internals / secret a.s.o](#)
- 33. Kubernetes with ServerLess
  - [Kubernetes with Serverless](#)
- 34. Literatur / Documentation / Information (Microservices)
  - [Sam Newman - Microservices](#)
  - [Sam Newman - Vom Monolithen zu Microservices](#)
  - [Microservices.io Patterns](#)
  - [BFF](#)
  - [Microservices Up and Running](#)
- 35. gitlab ci/cd
  - [Einfaches Beispieldscript](#)

## Backlog

- 1. Praxis Microservices ohne Docker und Kubernetes
  - [Schritt 1: Nodejs aufsetzen](#)
  - [Schritt 2: Codebasis bereitstellen](#)
  - [Schritt 3: Posts - Service testen](#)
- 2. Docker-Installation
  - [Installation Docker unter Ubuntu mit snap](#)
  - [Installation Docker unter SLES 15](#)
- 3. Docker-Grundlagen
  - [Übersicht Architektur](#)
  - [Was ist ein Container ?](#)
  - [Was sind container images](#)
  - [Container vs. Virtuelle Maschine](#)
  - [Was ist ein Dockerfile](#)
- 4. Docker-Befehle
  - [Logs anschauen - docker logs - mit Beispiel nginx](#)
  - [Docker container/image stoppen/löschen](#)
  - [Docker containerliste anzeigen](#)
  - [Docker nicht verwendete Images/Container löschen](#)
  - [Docker container analysieren](#)
  - [Docker container in den Vordergrund bringen - attach](#)
  - [Aufräumen - container und images löschen](#)

- [Nginx mit portfreigabe laufen lassen](#)
- [Docker container/image stoppen/löschen](#)
- [Docker containerliste anzeigen](#)

#### 5. Dockerfile - Examples

- [Ubuntu mit hello world](#)
- [Ubuntu mit ping](#)
- [Nginx mit content aus html-ordner](#)
- [Ubuntu mit hello world](#)
- [Nginx mit content aus html-ordner](#)

#### 6. Docker-Netzwerk

- [Netzwerk](#)

#### 7. Docker-Container Examples

- [2 Container mit Netzwerk anpingen](#)
- [Container mit eigenem privatem Netz erstellen](#)

#### 8. Docker-Daten persistent machen / Shared Volumes

- [Überblick](#)
- [Volumes](#)
- [bind-mounts](#)

#### 9. Docker - Dokumentation

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

#### 10. Docker - Projekt blog

- [posts in blog.dockerisieren](#)

#### 11. Docker Compose (backlog)

- [yaml-format](#)
- [docker-compose und replicas](#)
- [Example with Wordpress / Nginx / Mariadb - wrong](#)

#### 12. Kubernetes Netzwerk

- [Mesh / istio](#)
- [pubsub+ for graph kafka](#)

#### 13. Kubernetes GUI

- [OpenLens](#)

#### 14. Kubernetes - micrsk8s (Installation und Management)

- [Ingress controller in micrsk8s aktivieren](#)

#### 15. Helm (Kubernetes Paketmanager)

- [Helm Grundlagen](#)
- [Helm Warum ?](#)
- [Helm Example](#)

#### 16. Kubernetes - RBAC

- [Nutzer einrichten micrsk8s ab kubernetes 1.25](#)

#### 17. Kubernetes - Netzwerk (CNI's) / Mesh

- [Netzwerk Interna](#)
- [Übersicht Netzwerke](#)
- [Calico - nginx example NetworkPolicy](#)
- [Beispiele Ingress Egress NetworkPolicy](#)
- [Kubernetes Ports/Protokolle](#)
- [IPV4/IPV6 Dualstack](#)

#### 18. kubectl

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

#### 19. Kubernetes - Shared Volumes

- [Shared Volumes with nfs](#)

#### 20. Kubernetes - Wartung / Debugging

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

- [Curl from pod api-server](#)

## 21. Kubernetes - Tipps & Tricks

- [Kubernetes Debuggen ClusterIP/PodIP](#)
- [Debugging,pods](#)
- [Taints und Tolerations](#)

## 22. Kubernetes Advanced

- [Curl api-server kubernetes aus pod heraus](#)

## 23. Kubernetes - Documentation

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

## 24. Kubernetes - Hardening

- [Kubernetes Tipps Hardening](#)
- [Kubernetes Security Admission Controller Example](#)

## 25. Kubernetes Interna / Misc.

- [OCI Container Images Standards](#)
- [Geolocation Kubernetes Cluster](#)

## 26. Documentation

- [Good Doku with Tasks](#)

## 27. Docker-Container Examples

- [2 Container mit Netzwerk anpingen](#)
- [Container mit eigenem privatem Netz erstellen](#)

## 28. Docker-Netzwerk

- [Netzwerk](#)

## 29. Docker Security

- [Scanning docker image with docker scan/snyk](#)

## 30. Docker Compose

- [yaml-format](#)
- [Example with Ubuntu and Dockerfile](#)
- [docker-compose und replicas](#)
- [docker compose Reference](#)

## 31. Docker Swarm

- [Docker Swarm Beispiele](#)

## 32. Docker - Dokumentation

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

## 33. Kubernetes - Überblick

- [Installation - Welche Komponenten from scratch](#)

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

- [kubectl unter windows - Remote-Verbindung zu Kuberenets \(microk8s\) einrichten](#)
- [Arbeiten mit der Registry](#)
- [Installation Kubernetes Dashboard](#)

## 35. Kubernetes - RBAC

- [Nutzer einrichten - kubernetes bis 1.24](#)

## 36. kubectl

- [Tipps&Tricks zu Deployment - Rollout](#)

## 37. Kubernetes - Backups

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

## 38. Kubernetes - Tipps & Tricks

- [Assigning Pods to Nodes](#)

## 39. Kubernetes - Documentation

- [LDAP-Anbindung](#)
- [Helpful to learn - Kubernetes](#)
- [Environment to learn](#)
- [Environment to learn II](#)

- [Youtube Channel](#)

40. Kubernetes -Wann / Wann nicht

- [Kubernetes Wann / Wann nicht](#)

41. Kubernetes - Hardening

- [Kubernetes Tipps Hardening](#)

42. Kubernetes Deployment Scenarios

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

43. Kubernetes Probes (Liveness and Readiness)

- [Übung Liveness-Probe](#)
- [Funktionsweise Readiness-Probe vs. Liveness-Probe](#)

44. Linux und Docker Tipps & Tricks allgemein

- [Auf ubuntu root-benutzer werden](#)
- [IP - Adresse abfragen](#)
- [Hostname setzen](#)
- [Proxy für Docker setzen](#)
- [vim einrückung für yaml-dateien](#)
- [YAML Linter Online](#)
- [Läuft der ssh-server](#)
- [Basis/Parent - Image erstellen](#)
- [Eigenes unsichere Registry-Verwenden, ohne https](#)

45. Linux Tipps & Tricks

- [Grafischen Modus deaktivieren](#)

46. VirtualBox Tipps & Tricks

- [VirtualBox 6.1. - Ubuntu für Kubernetes aufsetzen](#)
- [VirtualBox 6.1. - Shared folder aktivieren](#)

47. CloudInit

- [Kubernetes Client einrichten mit bash](#)

48. Microservices - Messaging

- [EventBus Implementierungen/Überblick](#)

## Grundlagen

### Was sind Microservices ?

1. Microservices oder Microservices-Architekturen sind ein Ansatz zur Anwendungsentwicklung bei dem eine große Anwendung aus modularen Komponenten oder Diensten aufgebaut wird.
2. Jedes Modul unterstützt eine bestimmte Aufgabe oder ein Geschäftsziel und verwendet eine einfache, klar definierte Schnittstelle (Contract / Vertrag), wie zum Beispiel eine Anwendungsprogrammierschnittstelle (API), um mit anderen Diensten zu kommunizieren.

### Grundkonzepte von Microservices

#### Information Hiding

##### Was ?

- Microservices leben das Konzept, so viel wie möglich Informationen in einer Komponenten zu verstecken
- Der Microservice gibt so wenig Informationen wie "nötig" preis.

##### Warum ?

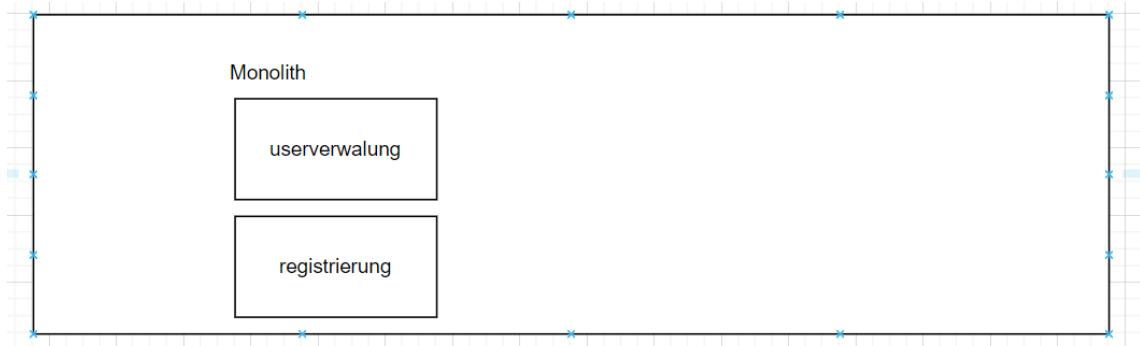
- Dadurch gibt es eine klare Grenze, **was**
  - **einfach zu ändern** ist
  - oder was **komplizierter zu ändern** ist (Änderung des Vertrages)
- Informationen die versteckt ist, können **ohne Absprache** geändert werden.

### Independant Deployability

- Teams unabhängig Änderung in Microservices machen und dieses redeployen und zwar ohne alle anderen
  - zu redeployen
- This is the most important thing and also the (Die wichtigste Sache, der Tipp Nr. 1)

### Monolith vs. Microservices

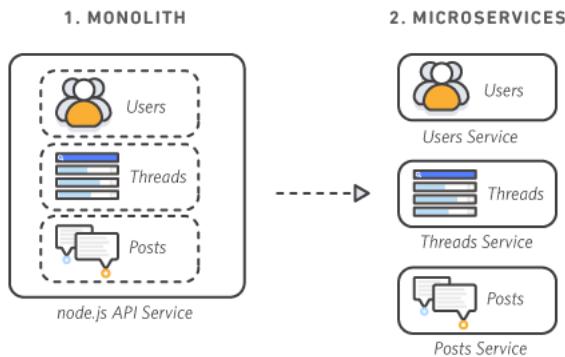
#### Schaubild Monolith mit Nachteilen



#### Nachteile

- Viel Abstimmungbedarf
- Längeres Deployment
- Höhere Gefahr bei Änderungen andere Feature/Logic kaputt zu machen
- Vertical Skalieren geht (mehr CPU, schnellere Platten, mehr Arbeitsspeichern)
- Horizontal Skalieren ist umständlich schwierig und wenn nur alles
- Team gegenseitig ausbremst
- Ich bin an eine bestimmte Sprache gebunden (z.B. JAVA)
- Neue Mitarbeiter gewinnen, ist bei alter Sprache höher

#### 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

### Gut aufgestellt mit Devops

- Weil
  - ansonsten durch alte Strukturen (kein Devops-Team) Geschwindigkeit durch notwendige Klärung, Verantwortlichkeiten verloren geht.

### Nachteile: Microservices

- Höhere Komplexität
- Bei schlechter / nicht automatischer Dokumentation kann man bei einer größeren Anzahl von Microservices den Überblick der Zusammenarbeit verlieren
- Aufwand: Architektur von Monolithisch nach Microservices IST Aufwand !
- Aufwand Wartung und Monitoring (Kubernetes)
- Erhöhte Knowledge bzgl. Debugging.
- Fallback-Aufwand (wenn ein Service nicht funktioniert, muss die Anwendung weiter arbeiten können, ohne das andere Service nicht funktionieren)
- Erhöhte Anforderung an Konzeption (bzgl. Performance und Stabilität)
- Wichtiges Augenmerk (Netzwerk-Performance)

### Nachteile: Microservices in Kubernetes

- andere Anforderungen an Backups und Monitoring

### Praxisbeispiele

#### Auswahl

- Netflix
- Spotify (über 800 microservices, java)
- ebay

#### References

- [https://www.asioso.com/de\\_DE/blog/anwendungen-und-praxisbeispiele-von-microservices-b602](https://www.asioso.com/de_DE/blog/anwendungen-und-praxisbeispiele-von-microservices-b602)

### Was ist devops

I. Ein DevOps-Team besteht aus Entwickler- und IT-Operations-Teams, die während des gesamten Produktlebenszyklus zusammenarbeiten, um die Geschwindigkeit und Qualität des Software-Deployments zu erhöhen.

II. Im Rahmen eines DevOps-Modells sind Entwicklungs- und Operations-Teams nicht mehr voneinander isoliert. Manchmal verschmelzen diese beiden Teams zu einem einzigen Team, in dem die Ingenieure während des gesamten Anwendungslebenszyklus zusammenarbeiten –

### API-Abfrage über REST-API

#### Grundlagen

- synchrone Kommunikation -> bspw. REST-API (zu 95%)

#### Idee dahinter (microservice)

- Nie direkt auf Daten zuzugreifen (immer nur immer API)

#### Damit auch die Möglichkeit haben, Informationen zu verstecken

- Prinzip von Hide Information, nur das wirklich gebraucht wird, kann abgefragt werden und der Rest ist nur im Hintergrund innerhalb des MicroServices zugänglich
- API (klarer Vertrag, wo festgelegt, welche Parameter an die API übergeben werden können/müssen) und welche Information zurückkommt

#### Beispiel

```
## GET - Abfrage
https://api.bitpanda.com/v1/trades
## i.d.R. kriegen wir die Information als json zurück

PUT /shop/products/11 HTTP/1.1
Host: api.predic8.de
Content-Type: application/json

{
  "name": "Red Grapes",
  "price": 1.79,
  "category_url": "/shop/categories/Fruits",
  "vendor_url": "/shop/vendors/501"
}

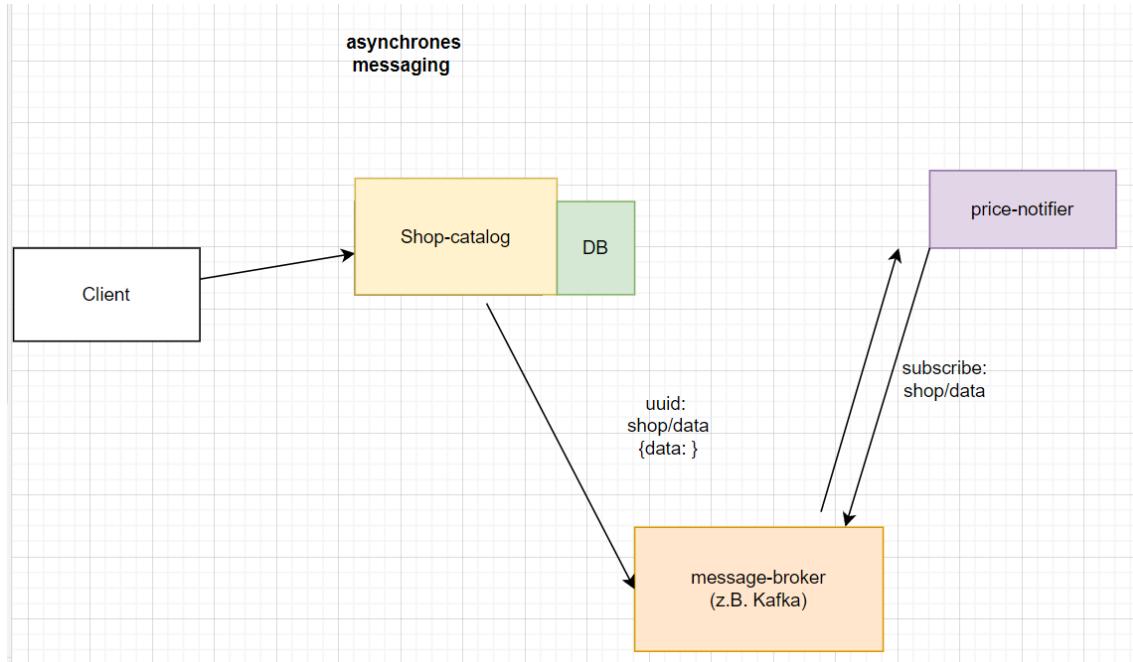
curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
  "name": "Red Grapes",
  "price": 1.79,
  "category_url": "/shop/categories/Fruits",
```

```

    "vendor_url": "/shop/vendors/501"
} 'https://api.predic8.de/shop/products/140'

```

### Asynchrones Messaging



### Microservice and Database

- Der MicroService sollte immer die Hoheit über seine Daten haben (Eine eigene Datenbank), das meint NICHT einen eigenen Datenbankserver
- Und sollte diese verändern (auch Struktur) und nutzen können, ohne sich mit anderen abzusprechen. .

### Grundwissen Microservices (Teil 2)

#### Brainstorming Domäne

#### Prozess aus Domain Driven Design

- Eventstorming
- <https://entwickler.de/ddd/domain-driven-design-in-aktion-mehr-dynamik-mit-event-storming>
- ...

#### Welche Events gibt es ? (in der Vergangenheit)

```

Bewerbungsgespräch geführt
Bewerber akzeptiert
Beispielvertrag erstellt
Beispielvertrag verschickt
Beispielvertrag von zukünftigen Mitarbeiter angenommen

```

#### Wer löst dieses Event aus ?

```

z.B. Button -> Bewerber
Command -> Bewerber auf Button im Webfrontend geklickt

```

### Datenbank - Patterns - Teil 1

#### Pattern Shared Database

- Shared Database: Informations-Hiding ist schwierig
- Achtung: nur in 2 Situationen vernünftig
  - Lesen statischer Referenzdaten (Postleitzahlen, Geschlecht, Bundesländer)
  - Anbieten eines Service, der direkt eine Datenbank als definierten Endpunkt bereitstellt
    - Alternative: Database as-a-Service-Interface Pattern

#### Wie häufig

- Eher selten

#### Referenz

- <https://microservices.io/patterns/data/shared-database.html>

#### Pattern: Database View

- Die Daten werden nicht als Tabelle, sondern als View bereitgestellt
- Datenbank (View) ist dann aber ein öffentlicher Vertrag

#### Wo ?

- Kann man dann machen, wenn man das monolithische Schema nicht auseinander nehmen kann
- Achtung: performance views mysql (nur bei älteren Versionen)
- Wenn der Aufwand für die Aufteilung zu gross ist, kann das der 1. Schritt in die richtige Richtung sein

#### Pattern: Database-as-a-Service Interface

- Manchmal müssen Clients eine Datenbank nur abfragen
- z.B. eine dedizierte Datenbank, als Read-Only-Endpunkt
  - gefüllt wird diese wenn sich Daten in der zugrundeliegenden Datenbank ändern
- Wir sollten die Datenbank, die wird nach draussen anbieten, von der Datenbank
  - getrennt halten, die wir innerhalb unserer Service-Grenzen einsetzen

#### Wie ?

- Umsetzung durch eine Mapping - Engine.

#### Wann ?

- Wenn legacy-client lesenden Zugriff benötigen

#### Reference:

- <https://microservices.io/patterns/data/database-per-service.html>

#### Pattern: Database Wrapping Service

- Eine Datenbank mit einem Service wrappen.
- Damit kann man auch sicherstellen, dass sich die Datenbank nicht verändert.
- Zugriffe müssen jetzt aber geändert werden, von direkt auf die service api

#### Wann ?

- API davor setzen, um Veränderung der Datenbank zu hindern.
- Einschränken, was man machen darf.

#### Pattern: Aggregate Exposing Monolith

- Daten werden über einen Serviceendpunkt vom Monolithen selbst bereitgestellt
  - API oder ein Stream mit Events
- Dadurch wird explizit, welche Informationen der neue Service benötigt

#### Pattern: Change Data Ownership

- Der neue Dienst übernimmt die Ownership für die Daten

#### Pattern Synchronize Data in Application

##### Schritt 1: Daten Bulk - synchronisieren

- z.B. durch Batch-Job
- dann z.B. Change-Data-Capture Prozess

##### Schritt 2: Synchrone Schreiben, aus dem alten Schema lesen

- Erfolgt durch Deployment einer neuen Version der Anwendung

##### Schritt 3: Synchrone Schreiben, aus dem neuen Schema lesen

- Erfolgt wieder durch Deployment einer neuen Version der Anwendung

##### Schritt 4: Alte Schema entfernen

- Altes Schema kann jetzt gefahrlos entfernt werden

#### Pattern: Tracer Write

- Inkrementelle Verschiebung der Source of Truth.
- D.h. nicht komplette Datenbank, sondern einzelne Tabellen

#### Datenbank - Patterns - Teil 2

#### Was sollte ich zuerst aufteilen: Code oder Datenbank, oder gleichzeitig.

- Datenbank nur zuerst, wenn ich Sorge wg. der Performance habe.
- Code in der Regeln zuerst (Machen die meisten Teams so)
  - Vorteil: Ich weiss dann welche Daten der Service braucht
- Gleichzeitig auf keinen Fall

#### Anwendungsfall 1: Zuerst die Daten aufteilen

##### Pattern: Repository per Bounded Context

- Im Code habe ich pro Bounded Context ein Repository - Layer, der auf die Datenbank zugreift

#### Pattern: Database per Bounded Context

- Vorsichtsmaßnahme, falls ich später ein Service draus machen will

#### Anwendungsfall 2: Zuerst den Code aufteilen

#### Pattern: Monolith as Data Access Layer

- [Monolith Data Access Layer - Schritt 1](#)
- [Monolith Data Access Layer - Schritt 2](#)

#### Pattern: Multischema Storage

- Service speichert Teile der Daten selbst
  - und Teile der Daten kommen aus dem Monolithen

#### Pattern: Split Table

- [Split Table - Teil 1](#)
- [Split Table - Teil 2](#)
- [Split Table - Teil 3](#)

#### Grund

Tabellen die über Service - Grenzen hinweg existieren aufteilen

#### Pattern: Move Foreign Key Relationship To Code

- [Schritt 1](#)
- [Schritt 2](#)

#### Anwendungsfall 3: Gemeinsame genutzte statische Daten

#### Pattern: Duplicate Static Reference Data

- Jeder Service hat seine eigenen Ländercodes
- Zwar redundant, aber ändern sich fast nie.
- Und ist es ein Problem, wenn sie sich ändern ?
- Letzte Änderung 2011 (bei Länder-Codes)

#### Pattern: Static Dedicated Reference Data Schema

- Dediziertes Schema für Ländercodes in eigener Datenbank
- Alle Services greifen auf diese Datenbank direkt zu
- Es gibt dort einen Vertrag (Änderungen an der Struktur sind kritisch)

#### Nachteil

- Probleme beim Ändern des Encodings in der Datenbank (Migration von alter auf neue Datenbank)

#### Pattern: Static Reference Data Library

- z.B. Ländercodes wandern von der Datenbank in eine Bibliothek
  - die dann einfach eingebunden wird.

#### Nachteil

- Schwierig, wenn verschiedene Programmier-Sprachen

#### Pattern: Static Reference Data Service

- z.B. dedizierter Service für Ländercodes
- REST-API

#### Aufbau



#### Strategische Patterns

#### 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 Systemn das gleiche ist (z.B. per batch job)

#### Pattern: Decorating Collaborator

- Ansteuerung als nachgelagerten Prozess über einen Proxy

#### Pattern Branch by Abstraction

- Beispiel Notification

#### Schritt 1: Abstraction der zu ersetzennde 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

#### Tests

#### Pyramidenkonzept

- s. Referenz

#### Testkategorien

Klassisch (automatisiertes Testen):

Unit tests  
Integration tests  
End-to-end tests.

Bei microservices kommen noch 2 tests dazu:

Components Tests  
Contract Tests

so dass es dann so aussieht:

End-To-End Tests  
Components Tests  
Integration Tests  
Contract Tests  
Unit Tests

#### Contract

Schnittstelle wird geprüft, ob sie alle Verträge erfüllt

Gibt sie die definierten Antworten mit den definierten Parametern

The contract specifies all the possible inputs and outputs with their data structures and side effects.

The consumer and producer of the service must follow the rules stated in the contract for communication to be possible.

#### Components

Components Test:

A component is a microservice or set of microservices that accomplishes a role within the larger system.

Component testing is a type of acceptance testing in which we examine the component's behavior in isolation by substituting services with simulated resources or mocking.

#### References

- <https://semaphoreci.com/blog/test-microservices>

#### Monolith schneiden microservices

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

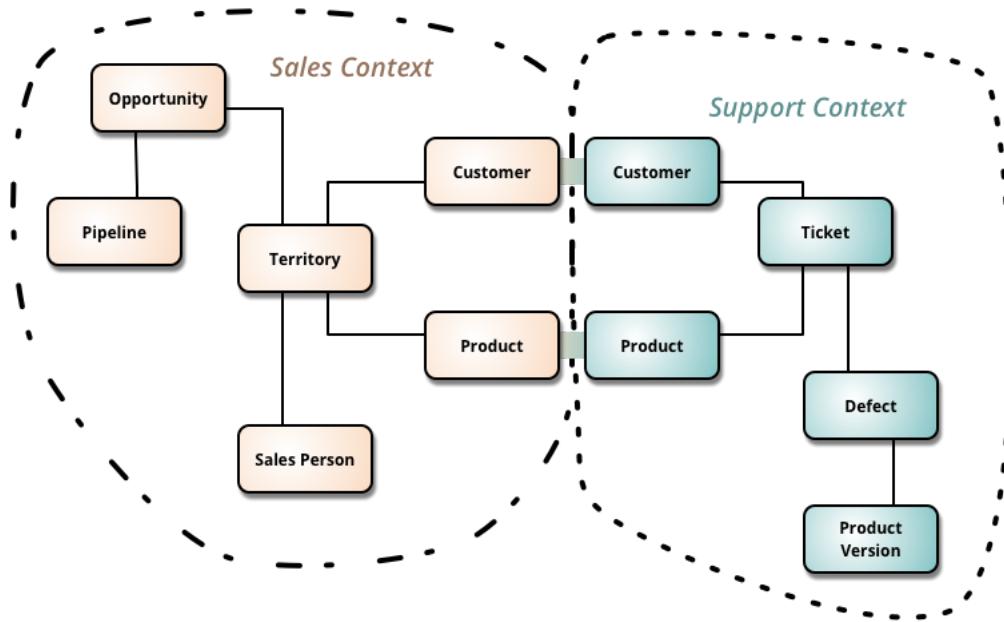
- Code-Größe

- 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 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  
Leih (bounded context 1)  
Suche (bounded context 2)

## EventBus Implementierungen/Überblick

### Fertige Software, die einen Event Bus bereitstellt

- Kafka
- RabbitMQ
- NATS

### Was ist Ihre Aufgabe ?

- Events empfangen
- Events veröffentlichen (publish) für die Zuhörer (listener)

### Wie sehen Events aus ?

- Mit Events meinen wir Informations-Snippets
  - Es ist nicht festgelegt, wie ein Event aussehen soll, es kann
    - Rohe Datenbytes
    - JSON
    - ein String
    - u.a. ... sein (was immer du verwenden willst)

### Was sind Listener ?

- Listener sind Services, die von anderen Events von anderen Services erfahren wollen

## Linux Tipps & Tricks

### In den Root-Benutzer wechseln

```
## kurs>
sudo su -
## password von kurs eingegeben
## wenn wir vorher der benutzer kurs waren
```

## Docker-Grundlagen

### Übersicht Architektur



### Was ist ein Container ?

- vereint in sich Software
- Bibliotheken
- Tools
- Konfigurationsdateien
- keinen eigenen Kernel
- gut zum Ausführen von Anwendungen auf verschiedenen Umgebungen
- Container sind entkoppelt
- Container sind voneinander unabhängig
- Können über wohldefinierte Kommunikationskanäle untereinander Informationen austauschen
- Durch Entkopplung von Containern:
  - o Unverträglichkeiten von Bibliotheken, Tools oder Datenbank können umgangen werden, wenn diese von den Applikationen in unterschiedlichen Versionen benötigt werden.

### Was sind container images

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

### Container vs. Virtuelle Maschine

```
VM's virtualisieren Hardware
Container virtualisieren Betriebssystem
```

### Was ist ein Dockerfile

#### What is it ?

- 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

#### Example

```
## syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

## Docker-Installation

### BEST for Ubuntu : Install Docker from Docker Repo

#### Walkthrough

```
sudo apt-get update
sudo apt-get install -y \
  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
```

### Docker als normaler Benutzer

```

## Wenn dein unprivilegierter Benutzer kurs heisst
sudo su -
usermod -aG docker 11trainingdo
exit

## ich wechsele nochmal in den Benutzer kurs
su - 11trainingdo
## jetzt darf kein Fehler kommen
docker images

```

### docker-compose ?

```

## herausfinden, ob docker compose installieren
docker compose version

```

## Docker-Praxis

### Docker run mit nginx

#### Beispiel (binden an ein terminal), detached

```

docker run -d --name my_nginx nginx:1.23
docker container ls

## falls nicht root
sudo su -
## wo sind die overlays
cd /var/lib/docker
## now find out
ls -la

## in den Container reinwechsel
## interactive
docker exec -it my_nginx bash

## Falls wir Prozesse anschauen wollen mit tool ps
## im container
apt update
apt install -y procps
ps aux | grep nginx
exit

## auf dem Host-System
ps aux | grep nginx

## oder wir führen nur ein Kommando aus
docker exec my_nginx cat /etc/os-release

```

### Die wichtigsten Befehle

```

## docker hub durchsuchen
docker search hello-world

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

```

```
## images die lokal vorhanden sind.
docker images

## container (laufende)
docker container ls
docker ps

## container (vorhanden, aber beendet)
docker container ls -a
docker ps -a

## z.B hilfe für docker run
docker help run

## Informationen zu Docker
## z.B. Was liegt wo ?
docker info
```

### Aufräumen - container und images löschen

#### Alle nicht verwendeten container und images löschen

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

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

## Alle nicht benötigten Daten löschen
docker system prune
```

### Logs des Host-Systems zu den Containern auslesen

```
## e steht für ende // letzte Einträge des Logs
journalctl -eu docker
```

### Logs anschauen - docker logs - mit Beispiel nginx

#### Allgemein

```
## Erstmal nginx starten und container-id wird ausgegeben
docker run -d nginx:1.22.1
a234
docker logs a234 # a234 sind die ersten 4 Ziffern der Container ID
```

#### Laufende Log-Ausgabe

```
docker logs -f a234
## Abbrechen CTRL + c
```

#### Nginx mit portfreigabe laufen lassen

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

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

### Example with Dockerfile

#### Ubuntu mit ping

```
mkdir myubuntu
cd myubuntu/
```

```

nano Dockerfile

FROM ubuntu:22.04
RUN apt-get update; apt-get install -y inetutils-ping
## CMD ["/bin/bash"]

docker build -t fullubuntu:1.0 .
docker images

## 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"]

docker build -t myubuntu:1.0 .
docker images

## -t wird benötigt, damit bash WEITER im Hintergrund im läuft.
## auch mit -d (ohne -t) wird die bash ausgeführt, aber "das Terminal" dann direkt beendet
## -> container läuft dann nicht mehr
docker run -d -t --name container-ubuntu myubuntu:1.0
docker container ls

## docker inspect to find out ip of other container
## 172.17.0.3
docker inspect container-ubuntu | grep -i ipaddress

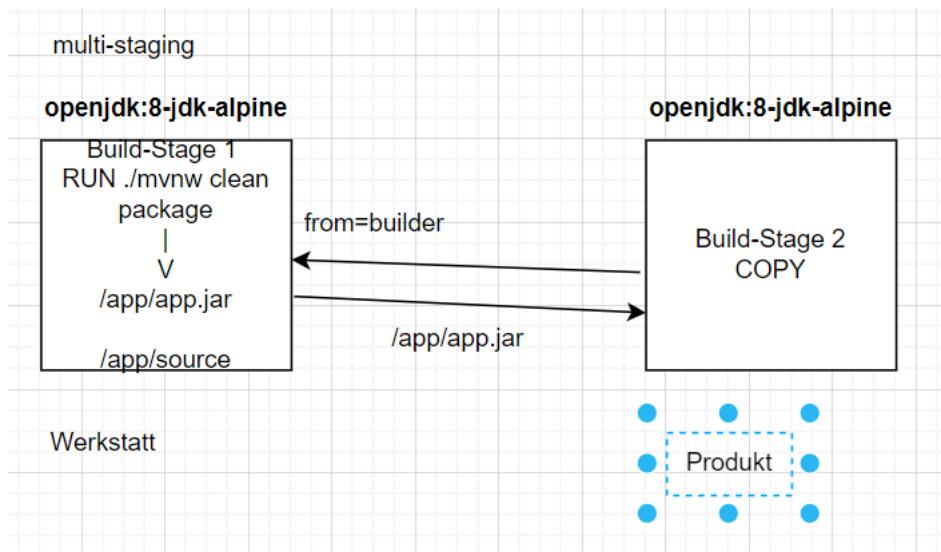
## Zweiten Container starten um 1. anzupingen
docker run -d -t --name container-ubuntu2 myubuntu:1.0

## Ersten Container -> 2. anpingen
docker exec -it container-ubuntu2 bash
## Jeder container hat eine eigene IP
ping 172.17.0.3

```

### Slim multistage-build

#### Overview



#### Step 1:

```

## Clone repo
cd

```

```
git clone https://github.com/jmetzger/multi-stage-example
cd multi-stage-example

## Bauen und vor Target stoppen
docker build . -t multi-stage-example:v1 --target=builder # - Build image using a specific stage

## Bauen
docker build . -t multi-stage-binary:v1
```

### Step 2: Run only binary-version

```
## run
docker run --name app -d -t multi-stage-binary:v1 sh
docker exec -it app sh
```

## Docker Security

### Docker Security

#### Generic

- Kann ich dem Image vertrauen (nur Images verwenden, denen ich vertrauen kann)
  - Im Zweifel eigene Images oder nur images von Docker Official Image / Verified Publisher (Suche auf Docker Hub)
- Container möglichst nicht als Root laufen lassen (bzw. nicht solche Images verwenden)
- Das nur das drinnen ist, was wirklich gebraucht wird (Produktion)
  - Im Idealfall sogar nur das Executable (siehe auch hashicorp/http-echo -> kein sh, keine bash)
- Alle container einer applikation in einem eigenen Netzwerk
- Images to scannen / security scans.

#### Images die nicht als root laufen

- bitnami
- nginx unprivileged

```
## Variante 1:
Erkennbar durch USER - Eintrag in Dockerfile
## oder
docker compose exec database id
docker exec <container> id

## https://hub.docker.com/r/bitnami/mariadb
## https://github.com/bitnami/containers/blob/main/bitnami/mariadb/11.0/debian-11/Dockerfile
USER 1001
```

#### Run container under specific user:

```
## user with id 40000 does not need to exist in container
docker run -it -u 40000 alpine

## user kurs needs to exist in container (/etc/passwd)
docker run -it -u kurs alpine
```

#### Default capabilities

- Set everytime a new container is started as default
- <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json>

#### Run container with less capabilities

```
cd
mkdir captest
cd captest

nano docker-compose.yml

services:
  nginx:
    image: nginx
    cap_drop:
      - CHOWN

docker compose up -d
## start and exits
docker compose ps
##
docker exec -it captest_nginx_1 bash
```

```
##/ touch /tmp/foo; chown 10000 /tmp/foo

## what happened -> wants to do chown, but it is not allowed
docker logs captest_nginx_1
```

```
docker compose down
```

#### Reference:

- [https://cheatsheetseries.owasp.org/cheatsheets/Docker\\_Security\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html)
- <https://www.redhat.com/en/blog/secure-your-containers-one-weird-trick>
- man capabilities

#### Scanning docker image with docker scan/snyx(Deprecated)

**ACHTUNG\_ Deprecated - USE Docker Scout instead (only Docker Desktop ?)**

#### Prerequisites

```
## install docker plugin in some cases
## Ubuntu
apt install docker-scan-plugin
```

```
You need to be logged in on docker hub with docker login
(with your account credentials)
```

#### Example

```
## Snyk (docker scan)
docker help scan
docker scan --json --accept-license dockertrainereu/jm-hello-docker > result.json
```

## Docker Compose

#### Ist docker-compose installiert?

**docker compose direkt als plugin für docker (aktuell die beste Wahl)**

```
## Installiert man docker in der neuesten 20.10.21
## existiert docker als plugin und wird anders aufgerufen
## Je nach distribution als Zusatzplugin von docker
docker compose
```

#### Ist docker-compose installiert (alte Version, nicht in docker integriert)

```
## besser. mehr infos
docker-compose version
docker-compose --version
```

#### Example with Wordpress / MySQL

##### Schritt 1:

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

##### Schritt 2:

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

  wordpress:
    image: wordpress:latest
```

```

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

volumes:
  database_data:
  wordpress_plugins:
  wordpress_themes:
  wordpress_uploads:

```

### Schritt 3:

```

docker compose up -d
## show all containers
docker ps
## show only containers from docker compose project
docker compose ps

```

### Schritt 3.5:

```

docker compose exec -it wordpress bash

apt update
apt install -y iputils-ping
ping -c4 database
exit

```

### Schritt 4: Alles wieder beenden

```
docker compose down
```

### Example with Ubuntu and Dockerfile

#### Schritt 1:

```

cd
mkdir baustest
cd baustest

```

#### Schritt 2:

```

## nano docker-compose.yml
services:
  myubuntu:
    build: ./myubuntu
    restart: always

```

#### Schritt 3:

```

mkdir myubuntu
cd myubuntu

```

```
nano hello.sh
```

```

#!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done

```

```

nano Dockerfile

FROM ubuntu:24.04
RUN apt-get update && apt-get install -y inetutils-ping
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]

```

#### Schritt 4:

```

cd ../
## wichtig, im docker-compose - Ordner seiend
##pwd
##~/bautest
docker compose up -d
## wird image gebaut und container gestartet

## Bei Veränderung vom Dockerfile, muss man den Parameter --build mit angeben
docker compose up -d --build

```

#### Schritt 5: Logs anzeigen

```

docker logs bautest-myubuntu-1
docker compose logs

```

#### Logs in docker - compose

```

##Im Ordner des Projektes
##z.B wordpress-mysql-compose-project
cd ~/wordpress-mysql-compose-project
docker-compose logs
## jetzt werden alle logs aller services angezeigt

```

#### docker compose Reference

- <https://docs.docker.com/compose/compose-file/compose-file-v3/>

### Docker - compose (Testprojekte)

#### Testprojekt mit api und mongodb

##### Was macht es ?

- Das Projekt wird direkt gebaut
- Es startet eine mongodb.
- Daten werden Über api calls geschrieben/gelesen gelöscht

##### Wie verwende ich es ?

#### Schritt 1: Aufsetzen

```

## Auf dem Docker - server direkt klonen und starten
cd
mkdir -p projects
cd projects

git clone https://github.com/jmetzger/multiple-containers-in-docker.md mcid
cd mcid
docker compose up -d

```

#### Schritt 2: [Optional] Datenbankverbindung aufbauen

```

## Z.B. in visual studio code
## Extensions MongoDB installieren

```

#### Schritt 3: Rest API-Calls absetzen

- Diese finden sich bspw. hier: <https://github.com/jmetzger/multiple-containers-in-docker/blob/main/rest.http>

```

## Hierzu kann bspw. der REST-Client in Visual Studio Code verwendet werden

```

### Microservices - Daten

#### Überblick shared database / database-per-service

## Grundlegendes

- Grundlegende Entscheidung, ob
  - Database per Service
  - oder: Shared Database
- Kann auch für einzelne Services unterschiedlich ausfallen

## Database per Service

### Prämissen

- Ein anderer Service kann nur über die API zugreifen.
- Synchronisierung kann auch über andere Weg als synchron erfolgen (z.B. Messaging -> Saga)

### Umsetzung:

- Private-tables-per-service – each service owns a set of tables that must only be accessed by that service
- Schema-per-service – each service has a database schema that's private to that service
- Database-server-per-service – each service has its own database server.

### Vorteile

- Das sichert die große Unabhängigkeit
- D.h. ein unabhängiges Deployment ist problemlos möglich

### Nachteile

- Transaktionen funktionieren auf DB-Ebene nicht mehr.
- JOINS sind schwierig umzusetzen.

### Reference:

- <https://microservices.io/patterns/data/database-per-service.html>

## Shared Database

### Vorteile

- Joins sind einfach möglich
- Transaktionen funktionieren

### Nachteile

- Single Point of Failure (ausser natürlich Cluster)
- Performance Engpässe (kann auch durch gute Optimierung behoben werden)

Development time coupling – a developer working on, for example, the OrderService will need to coordinate schema changes with the developers of other services that access the same tables. This coupling and additional coordination will slow down development.

### Wie ?

- Services teilen sich eine Datenbank

### Ref:

- <https://microservices.io/patterns/data/shared-database.html>

## Umgang mit Joins bei database-per-service

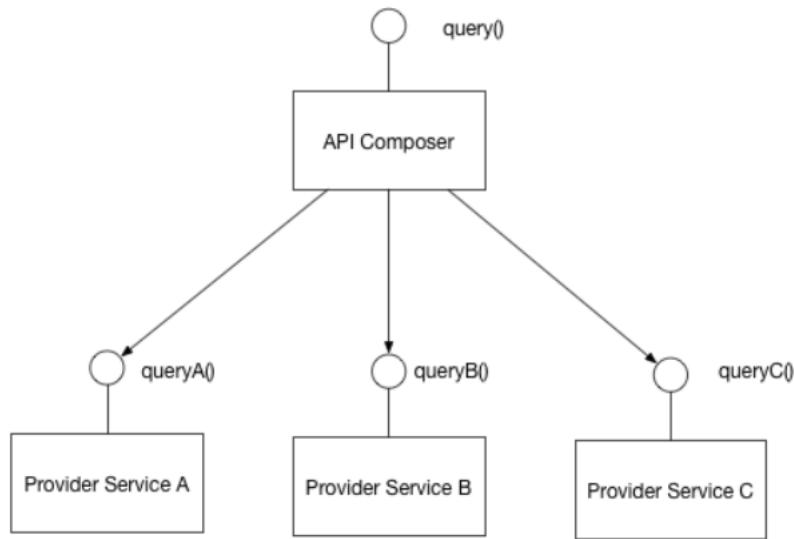
### 1 Pattern: api composition

#### Nachteile:

- Bei grossen Datenmengen, kann das sehr speicher und zeitintensiv sein

#### Schaubild

Implement a query by defining an *API Composer*, which invoking the services that own the data and performs an in-memory join of the results.



#### Generell

- Oftmals wird dafür ein API-Gateway verwendet

#### Ref:

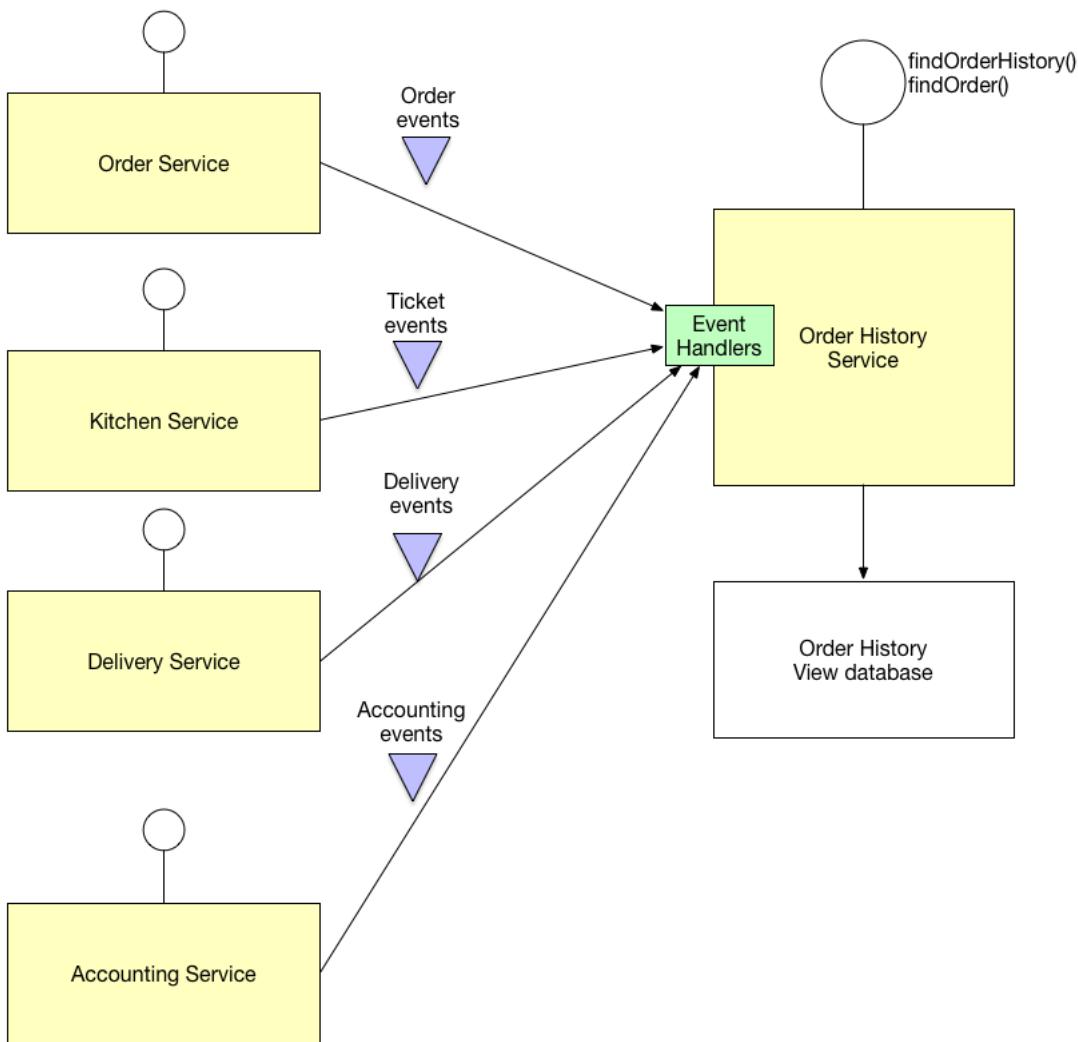
- <https://microservices.io/patterns/data/api-composition.html>

## 2 Pattern: CQRS (Command Query Responsibility Segregation)

#### Wie ?

- Datenbank erzeugen, die nur eine Leseansicht hat.
- Synchronisierung erfolgt über Subsribition zu Domain Events

#### Schaubild



#### Vorteile

- Unterstützt mehrere denormalisierte Views, die performant und skalierbar sind
- Abfragen sind einfache

#### Wann ?

- Notwendig bei Joins, wenn wir das Event Sourcing - Pattern verwenden

#### Nachteile

- Eventuell doppelter Code
- Lag bei den Views / NUR Eventuell Konsistente Views (keine Sicherheit)

#### Ref:

- <https://microservices.io/patterns/data/cqrs.html>

#### Umgang mit Transaktionen bei database-per-service

#### Problem

- When we move to a database per service pattern, we cannot use transactions

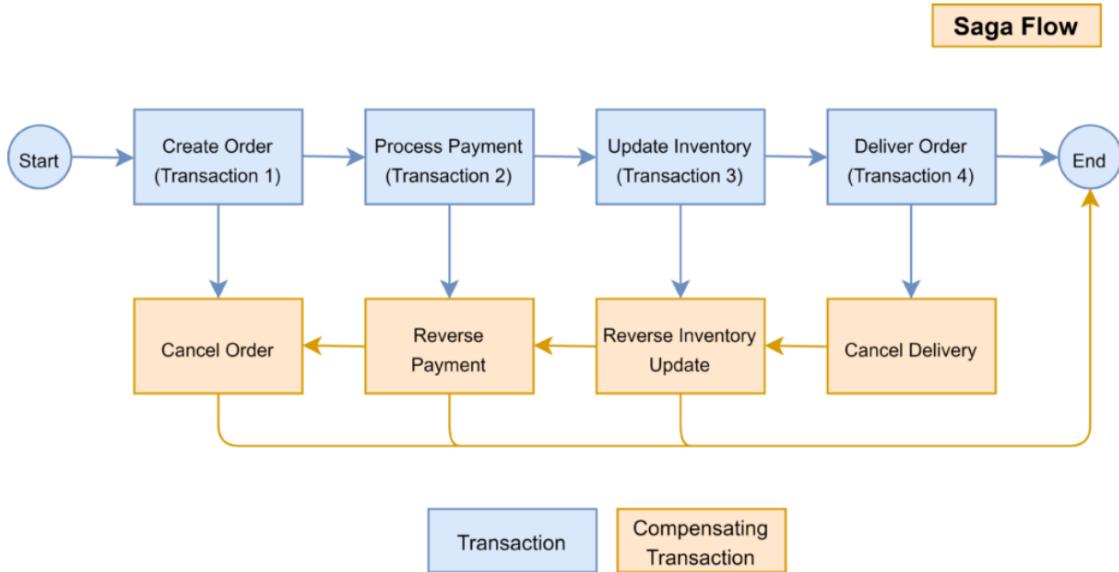
#### Example Problem

- You are using database-per-service-pattern

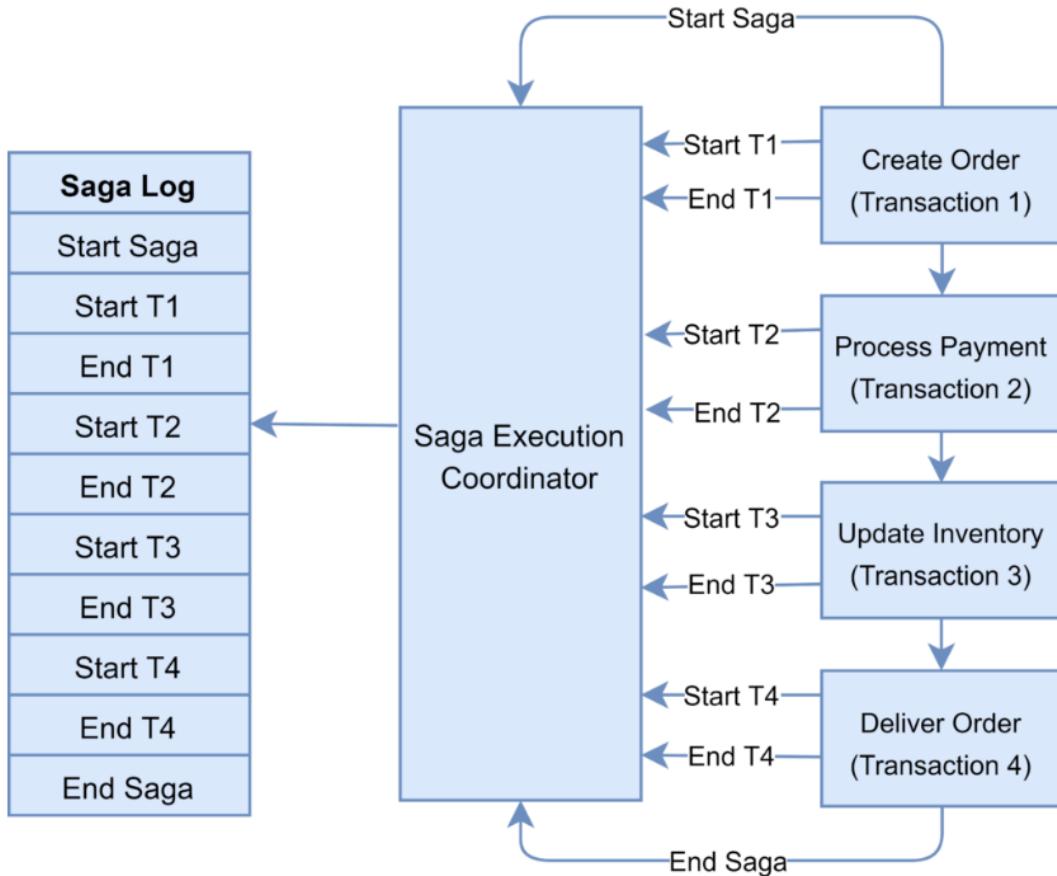
```

x e-commerce store
x customers have a credit limit.
x The application must ensure that a new order will not exceed the customer's credit limit.
x Orders and Customers are in different databases owned by different services
x because of this: application cannot simply use a local ACID transaction.
  
```

Schaubild (Wie funktioniert es ?)



Saga Execution Coordinator (SEC) as central component



- contains a Saga log that captures the sequence of events of a distributed transaction.
- ON FAILURE: the SEC component inspects the Saga log to identify the impacted components and the sequence in which the compensating transactions should run.

- It can then identify transactions successfully rolled back, which ones are pending, and can take appropriate actions

### Implementation as Saga Choreography Pattern

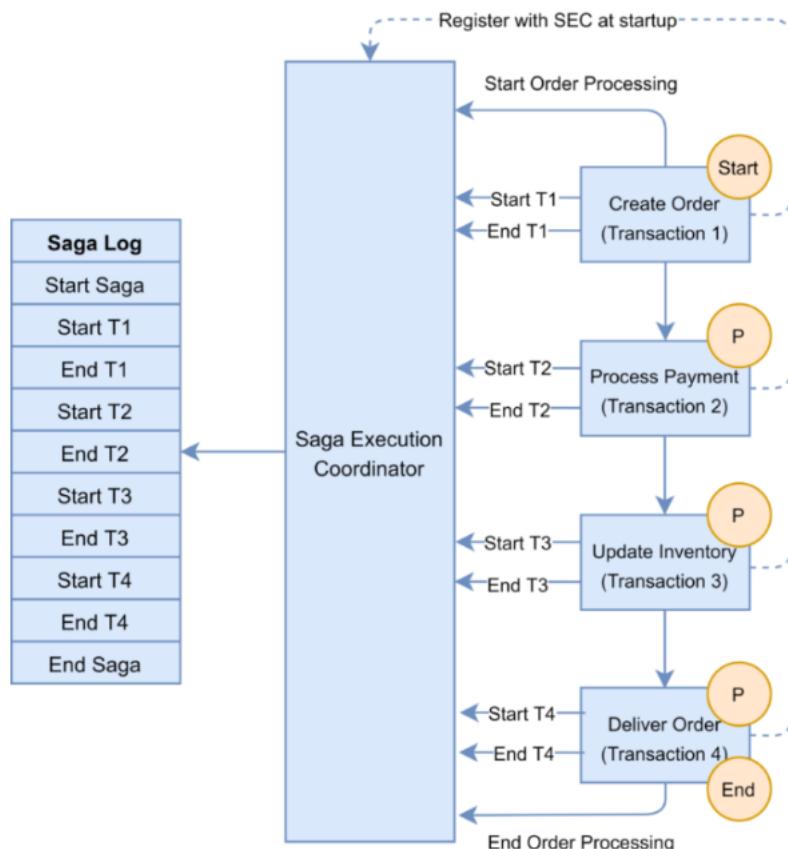
#### When ?

- Greenfield (starting from scratch) microservices development

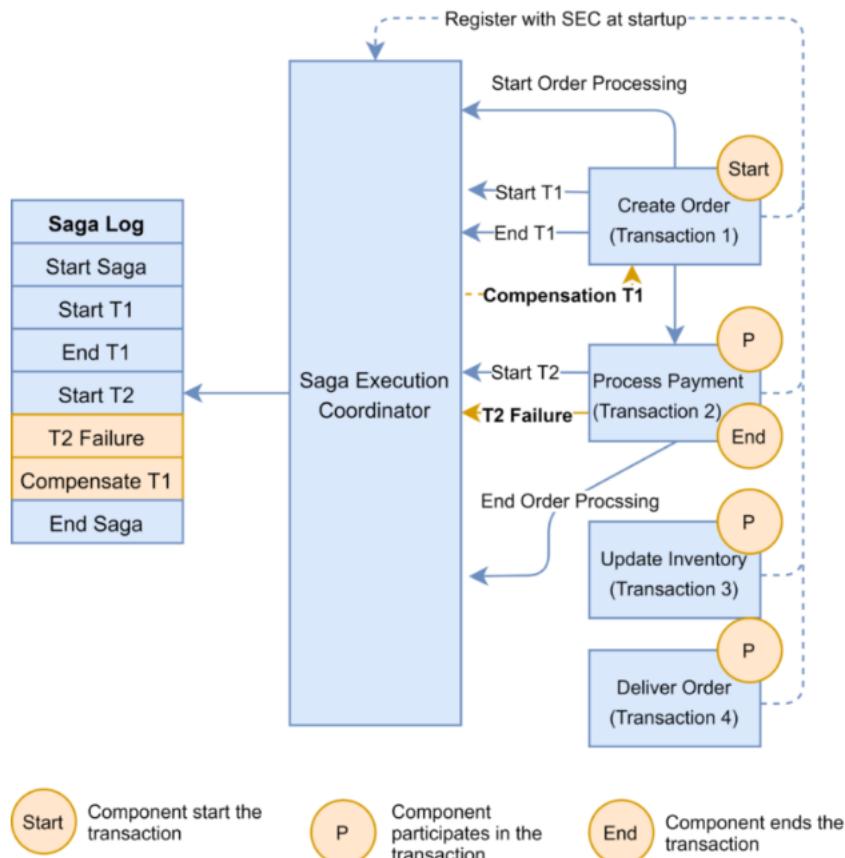
#### How ?

- each microservice that is part of the transaction publishes an event that is processed by the next microservice.

#### Schaubild (Success)



#### Schaubild (Failure)



#### Implementation as Saga Orchestration Pattern

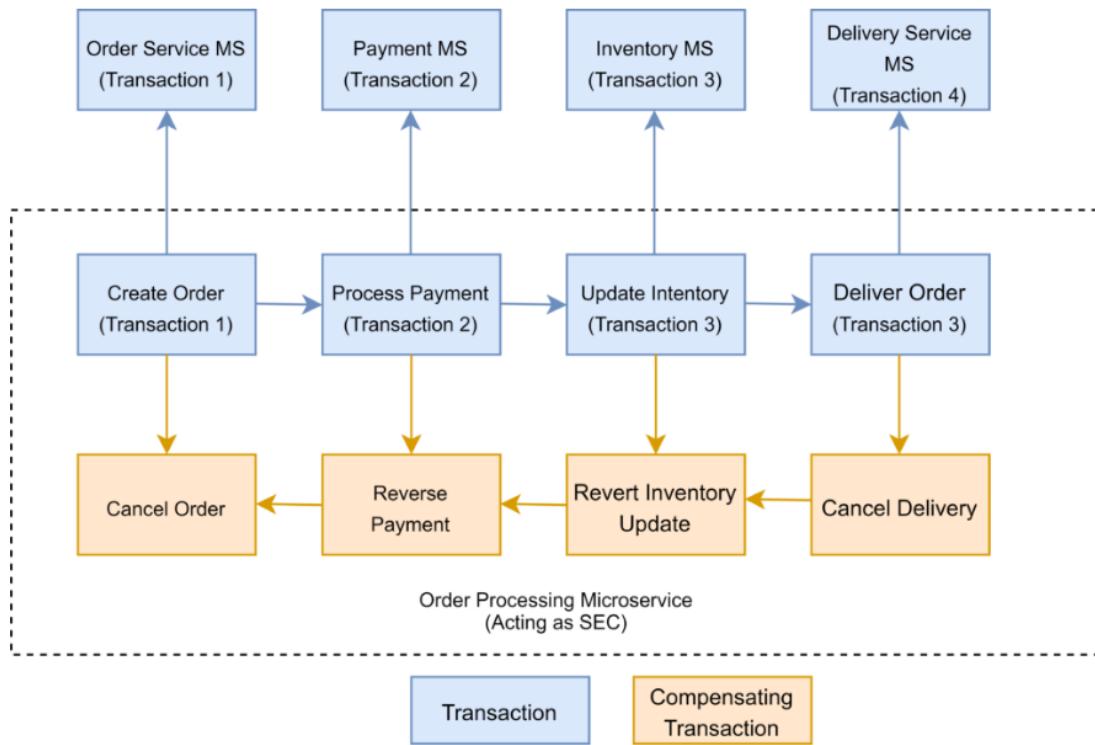
##### When ?

- Brownfield (we already have a set of microservices)

##### How ?

- Orchestrator will be in charge of the whole transactions process

Schaubild



#### Products

- Camunda (framework)
- Apache Camel

#### Reference:

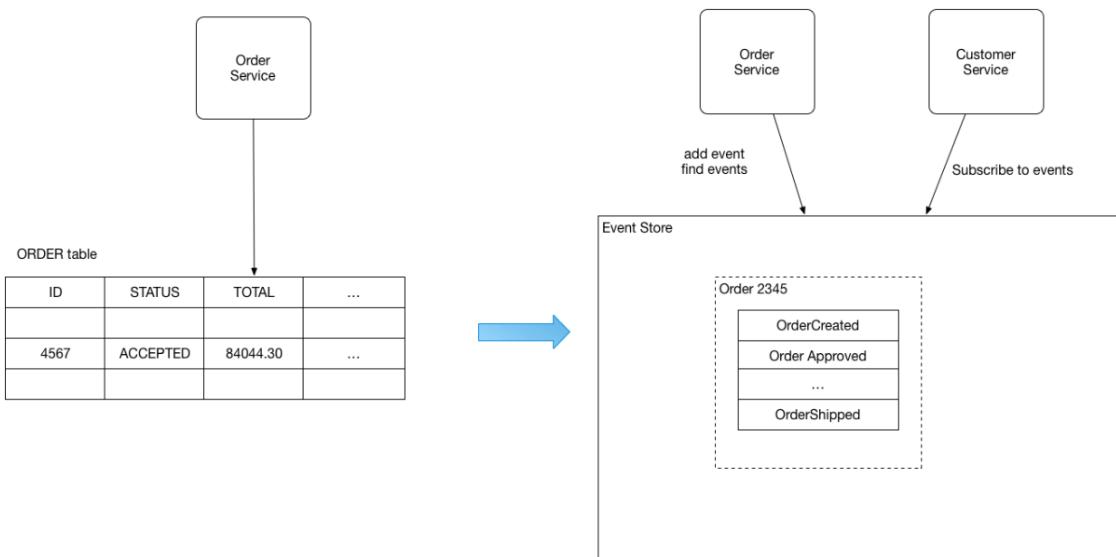
- <https://www.baeldung.com/cs/saga-pattern-microservices#introduction-to-saga>
- <https://microservices.io/patterns/data/saga.html>

#### Event Sourcing

##### Nachteile:

- Um Daten zu bekommen, müssen die verschiedenen Events "geparsed" werden und daraus ein Ergebnis geschrieben werden
- Das ist zeitraubend -> als Ergänzung kann CQRS (Command Query Responsibility Segregation) verwendet werden (Es werden dann finale Daten in ein VIEW basierte Datenbank geschrieben)
  - Hier ist allerdings wieder das Thema der Eventuell konsistenten Daten (Damit müssen die Applikationen umgehen)

#### Schaubild



#### Refs:

- <https://microservices.io/patterns/data/event-sourcing.html>

### Microservices (async messaging)

#### Topic/Queue ohne Downtime migrieren

#### How can you be sure, that consumer goes not get data from the old topic

- Producer should only (!) write to one topic (the newest)

#### How to switch to a new version ? (Part 1: producer)

1. Every topics has a version
2. New Version :: myTopic.v0 -> myTopic.v1
3. Whenever a new version is there, we will write it into topics: \_meta\_version
4. Producer will be consumer for the \_meta\_version and wil get to know about the topic
5. Producer will immediately write to the new topic, BUT ONLY to the new topic

#### How to switch to a new version ? (Part 2: consumer)

1. consumer has to drain the old topic before switching to the new one
2. how to know it is drained ? Ask for "watermark" offset (gives back the last offset)
3. When you have the last message with the watermark offset -> switch to new topic

#### Watermark - Background

1. The offset of the last message in topic, can be retrieved with "watermark"
2. Example Code: [https://github.com/confluentinc/confluent-kafka-python/blob/master/examples/get\\_watermark\\_offsets.py](https://github.com/confluentinc/confluent-kafka-python/blob/master/examples/get_watermark_offsets.py)

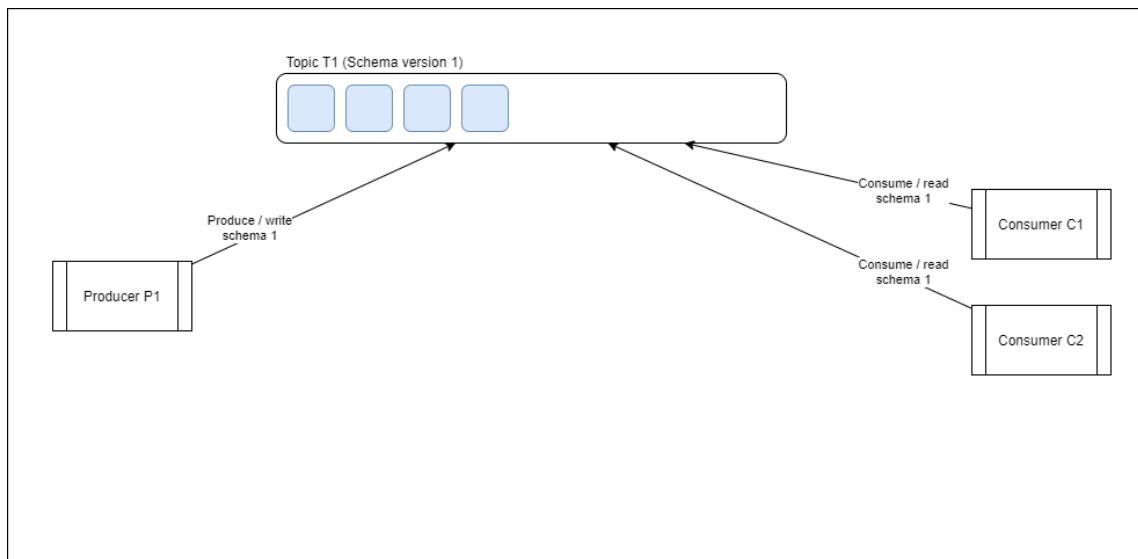
#### Reference:

- <https://gauravssarma1992.medium.com/migrating-kafka-topic-without-downtime-f863819cfb3d>

### Disruptive Änderungen im Schema migrieren

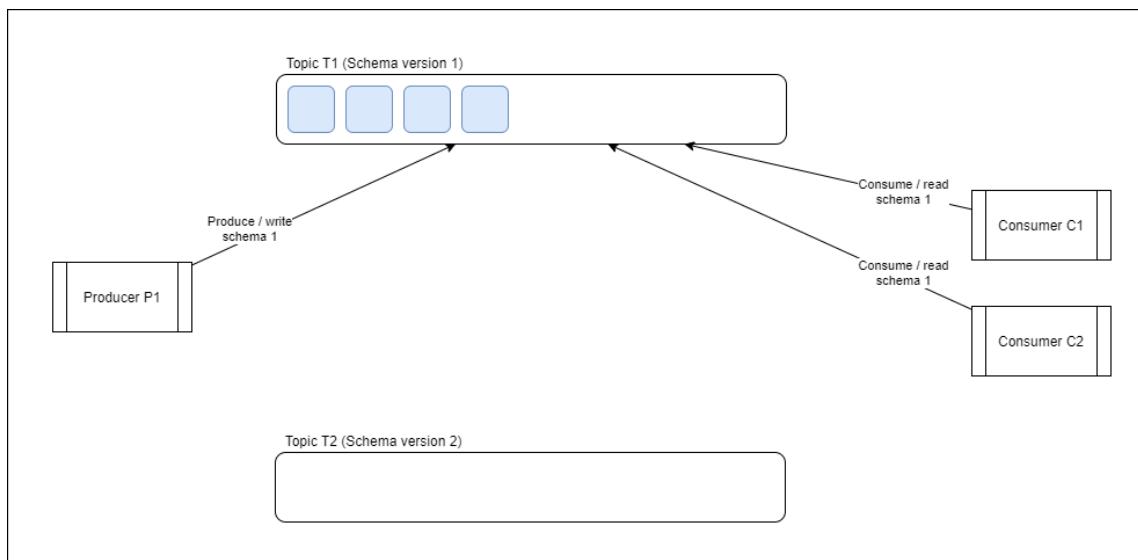
#### Step 1: Status Quo

1. Producer P1 is producing Message to Topic T1
2. Consumer C1 and C2 are consuming this topic



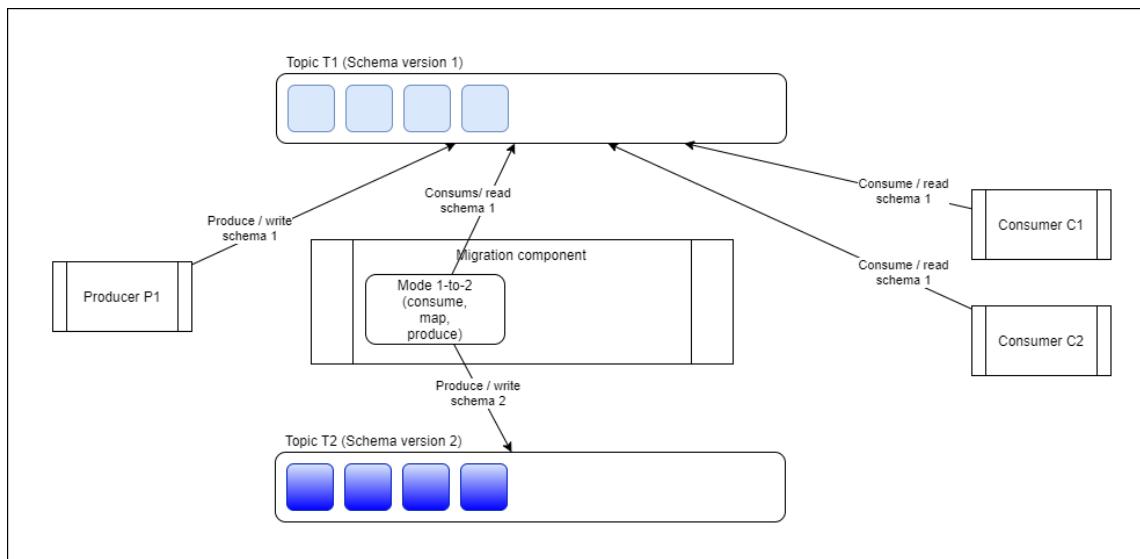
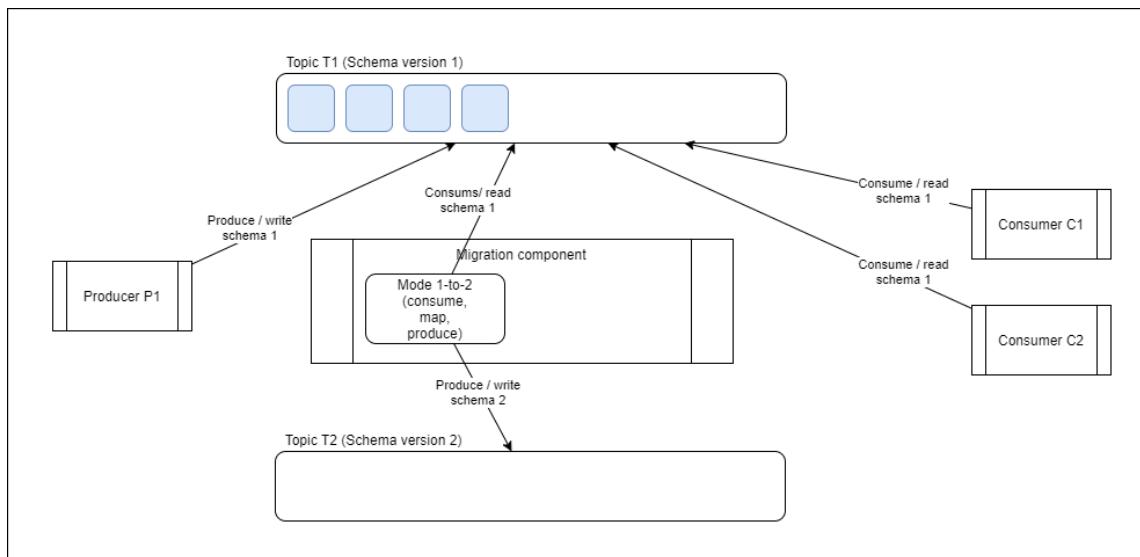
### Step 2: New Topic T2 with breaking Schema S2

1. Topic is currently empty

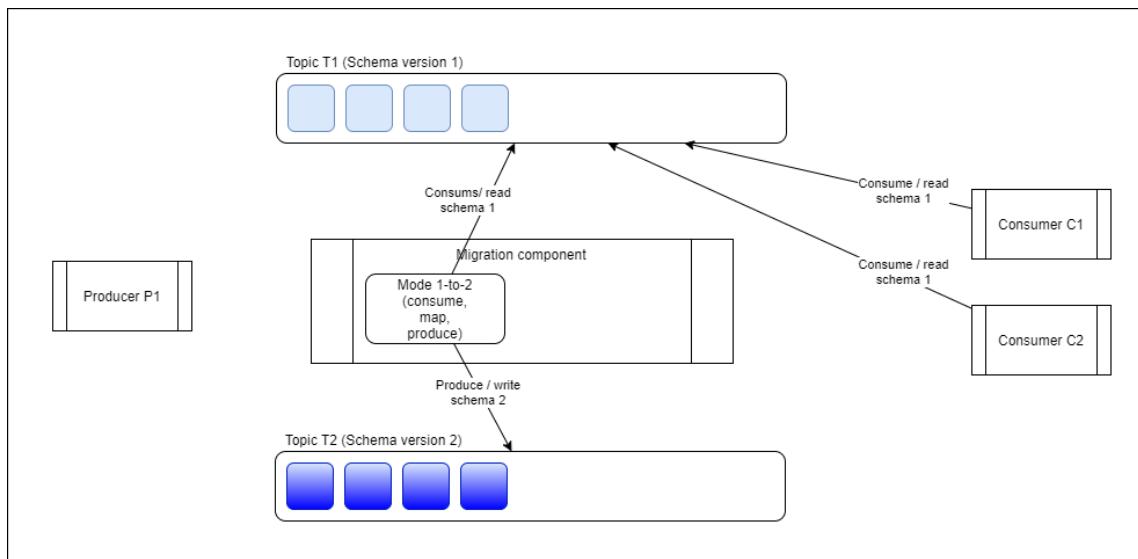


### Step 3: Maintaining history order with migration component (Mode 1 -> 2)

- reproducing all existing records from topic T1 on topic T2
  - by transforming them into messages following schema S2 and producing them on topic T2.
- Same keys are to be used (so it will be in the same partitions)
- reproduced messages are to retain the original timestamp
  - in order to preserve the semantics of the message
- No Downtime: producer P1 can meanwhile still produce new messages that may get consumed by existing consumer
- Every message from P1 will be picked up by the migration component and reproduced in T2

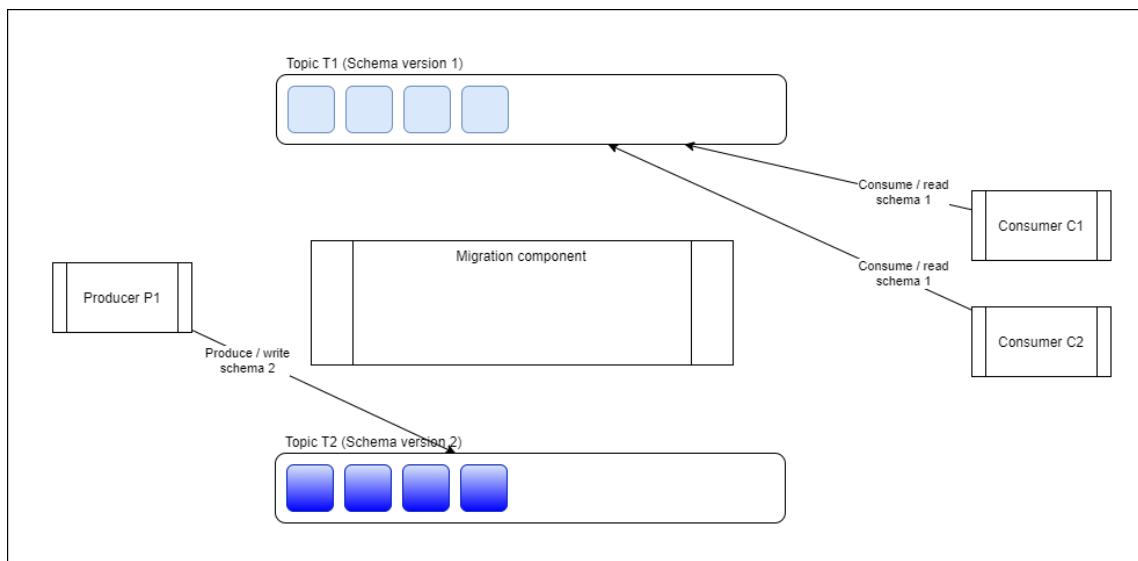


Step 4: New Version P1 (will not send to T1) any more



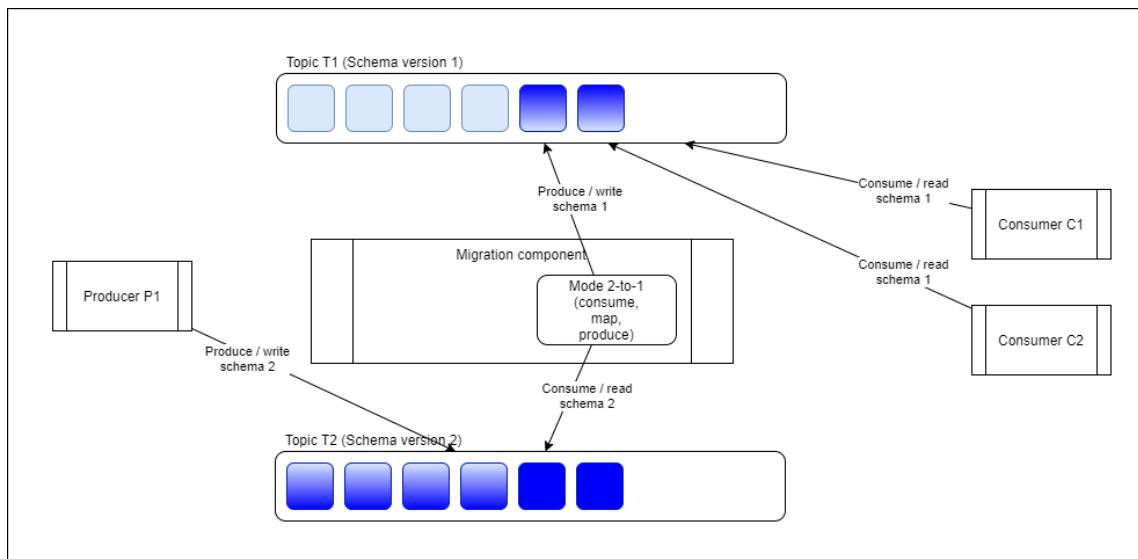
#### Step 5: P1 sends messages to T2 (after migration component has reproduced ...)

1. Migration component has reproduced all messages from T1 -> T2
2. AFTER THAT ! P1 will start to send messages to T2



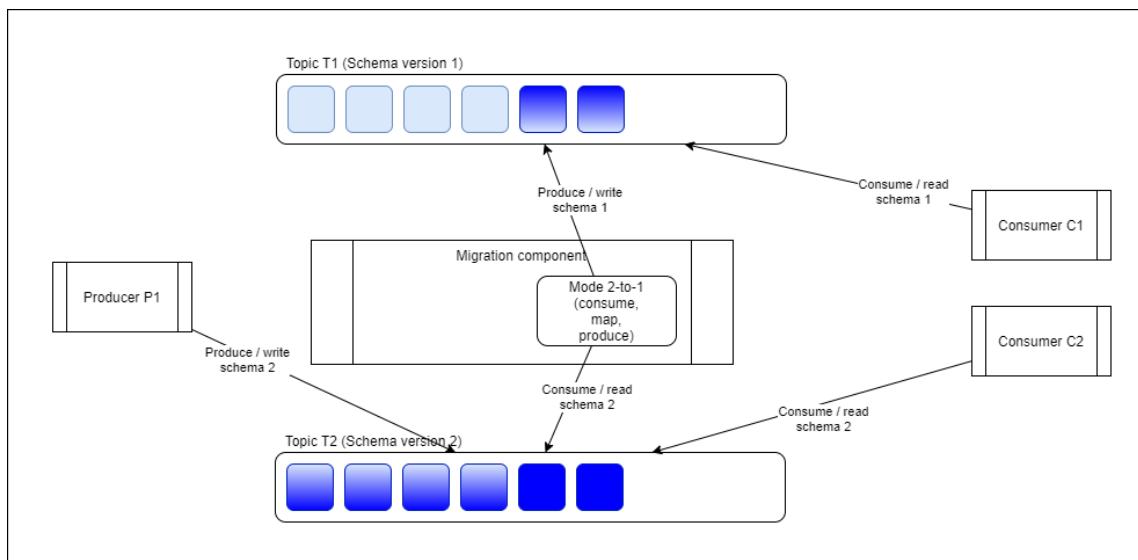
#### Step 6: Switching operation mode of migration component (mode: 2->1)

1. Migration component will from now on consume messages from topic T2 and reproduce them on topic T1.

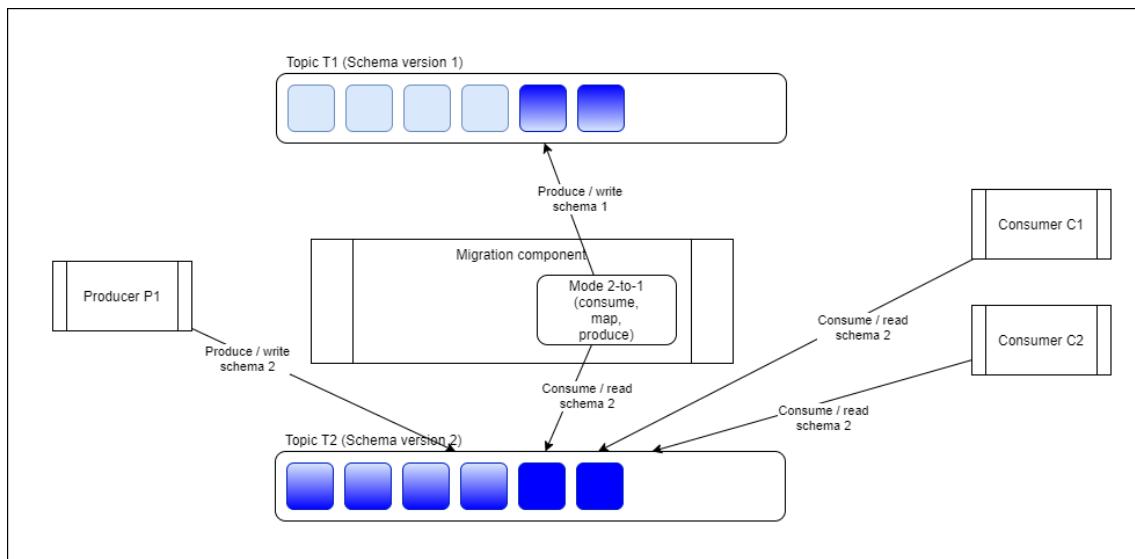


#### Step 7: Ready for consumers

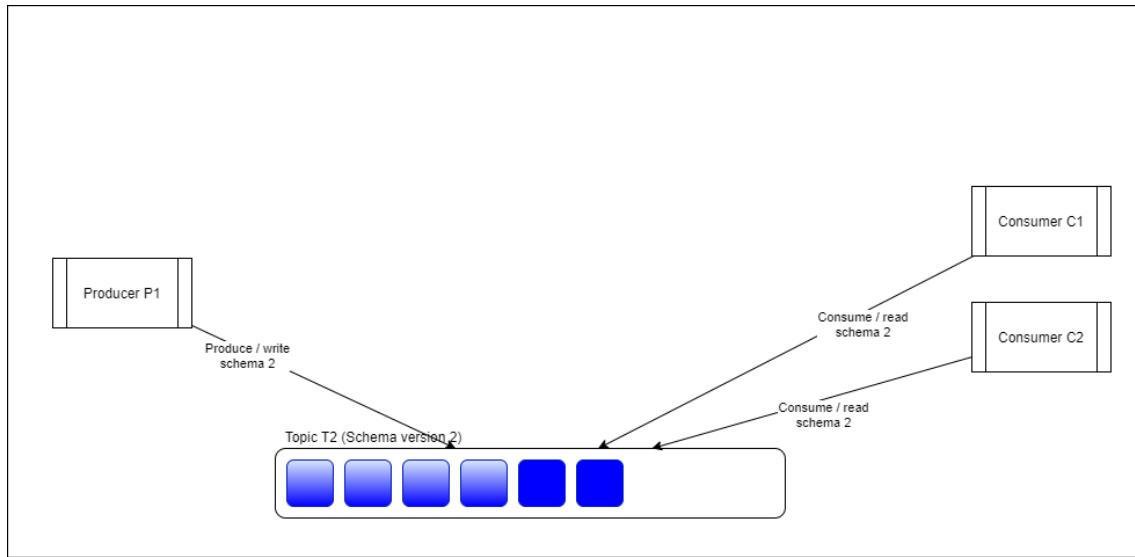
1. It is now possible for the consumers C1 and C2 to switch to T2 at any time



#### Step 8: All consumers have migrated



#### Step 9: Cleanup, when all have migrated



#### Reference

- <https://cymo.eu/blog/a-strategy-for-dealing-with-breaking-schema>

#### Microservice - flightapp - concepts

##### Vorgehensweise nach dem SEED-Verfahren

##### 7 Steps

- 1. Akteure identifizieren
- 2. Jobs identifizieren, die durch Akteure getätigten werden müssen
- 3. Entdecken/Entwickeln von Interaktionsschritte mit Ablaufdiagrammen
- 4. High-Level Aktionen und Abfragen basierend auf den zu tätigen Jobs (2) und den Interaktionsschritten ableiten
- 5. Jede Abfrage und Aktion als Spezifikation beschreiben, mit einem offenen Standard (wie OpenAPI Spezifikation [OAS] oder GraphQL schemata)
- 6. Feedback zur Api erhalten
- 7. Implementieren

#### Reference:

- <https://www.oreilly.com/library/view/microservices-up-and/9781492075448/ch03.html>
- <https://rolfsblog.ch/openapi/>

## Vorgehensweise nach SEED on Detail

### Schritt 1:

Identifizierung der bounded contexts

- o Flights Management
- o Reservations Management

Am Anfang macht es Sinn microservices eher grob/weit (coarse-grained) zu gestalten in Ihrer Funktion.  
(von allgemein zu spezifischer ist nachher einfacher als anders herum)

### Schritt 2: Akteure/Konsumenten mit der SEED - Methode identifizieren

1. Eine Kunden will den Flug buchen
2. Die User-App der Airline (web, mobile, usw.)
3. Die Web API mit der die App interagiert.(dies nennen manche auch "backend for frontends" oder BFF APIs.)
4. The flights management microservice: ms-flights
5. Das Reservierungs Management microservice: ms-reservations

### Schritt 3: Beispiele von JTBD's (Jobs to be Done)

Fiktiv: Gesammelt von Kunden Interviews und Business Analyse Recherche

1. Wenn ein Kunde mit der UI interagiert, muss die app einen Sitzplan anzeigen, welcher die verfügbaren und die belegten Plätze zeigt, so dass der Kunde einen Sitzplatz auswählen kann.
2. Wenn ein Kunde eine Buchung finalisiert, muss die web app einen Sitz reservieren für den Kunden (so kann die App verhindern, dass es im Verlauf Konflikten bei den Sitzten gibt)

### Schritt 4: BFF-Api als Ver-Mittler (JTBD)

(Backend for frontend)

Empfehlung: Eine BFF-Api, die nur eine ganze schlanke Schicht hat ohne business Logik Implementierung.

Sie "orchestriert" nur die microservices

Es gibt also jobs für die die BFF API microservices braucht.

Die folgende Liste an Job, die mehr technischen JTBD's beschreiben die Bedürfnisse zwischen der BFF API und den microservices

1. Wenn die BFF API angefragt wird einen Sitzplan zur Verfügung zu stellen, braucht die API msflights, um einen Plan der Sitze im Flugzeug zu bekommen, so dass die BFF API Verfügbarkeiten abholen kann und das finale Ergebnis erstellen kann.
2. Wenn die BFF API einen Sitzplan rendern soll, braucht die BFF API ms-reservations um eine Liste von bereits reservierten Sitzen zu bekommen, so dass die BFF API diese Daten dem Sitzplatz-Setup hinzufügen kann und ein Sitzplan zurückgibt.
3. Wenn die BFF API gefragt wird, einen Sitzplatz zu reservieren, braucht die BFF API ms-reservations um die Reservierung auszuführen, so dass die API einen Sitzplatz reservieren kann

=====

Achtung:

Wir lassen nicht ms-flights -> ms-reservations aufrufen, um den Sitzplan zusammenzubauen

Stattdessen lassen wir die BFF API diese interaktion tun.

Direkt microservices-to-microservices call sollten vermieden werden

### Schritt 5: Project Flight-Service: UML (Erstellen eines Ablaufdiagramms)

- <https://github.com/jmetzger/training-microservices-docker-kubernetes/blob/main/microservices-flightapp/concept/02-uml.md>

Folgendes kann man hier klar erkennen:

1. flight id muss ich abfragen, weil Kunden oft nicht direkt die Flugnummer eingeben ;o)

```
#####
# Randnotiz
#####
Zwar müssten auch Tasks für die BFF API definiert werden,
lassen wir aber mal jetzt aussen vor.
```

#### Schritt 6: JBTD in actions und queries überführen

```
Wir machen das für ms-flights und ms-reservations
```

##### A. Flights Microservice

###### Get flight details

- o Input flight\_no,  
departure\_local\_date\_time  
(ISO8601 format und in der lokalen zeitzone)
- o Response: A unique flight\_id identifying a  
specific flight on a specific date.  
In der Praxis, wird dieser Endpunkt noch  
mehr relevanten Felder zurückgeben, aber  
diese sind für unseren Context unrelevant  
also überspringen wir diese

###### Get flight seating (the diagram of seats on a flight)

- o Input flight\_id
- o Response: Seat Map Object in JSON Format

##### B. Reservations Microservice

###### Query already reserved seats on a flight

- o Input: flight\_id
- o Response: A list of already-taken seat numbers,  
each seat number in a format like "2A"

###### Reserve a seat on a flight

- o Input: flight\_id, customer\_id, seat\_num
- o Expected outcome: A seat is reserved and unavailable  
to others, or an error fired if the seat was unavailable
- o Response: Success (200 Success) or failure (403 Forbidden)

#### Schritt 7: Ablaufdiagramm erstellen und anzeigen

- Schritte hier: [UML](#)

#### Schritt 8: OpenAPI Spezifikation

- Wir werden das bei der Umsetzung testen.

#### Schritt 9: Implementieren

#### Microservice - flightapp - reservations

##### Template for microservice with python flask

##### Good idea to start microservices with a fixed template within your team

##### Why ?

- Way shorter implementation time for new microservice based on python

##### What do we use it for ?

- We will use it for the reservations (get, put)

##### Refs:

- <https://github.com/inadarei/ms-python-flask-template/tree/master>

#### Create microservice - reservations

##### Part 1: SEAT RESERVATIONS

###### Block 1: The OpenAPI - Definition

###### Getting the OpenAPI - Definition we ;o) created

```
## Get if from:  
https://raw.githubusercontent.com/jmetzger/ms-reservations/master/docs/api.yml
```

###### Use it to render a beautiful out in swagger, and it is also the docs

- <https://editor.swagger.io/>

```

47   content:
48     application/json:
49       schema:
50         type: object
51         properties:
52           flight_id:
53             description: Flight's Unique Identifier.
54             type: string
55             example: "edcc03a4-7f4e-40d1-898d-bf84a266f1b9"
56           customer_id:
57             description: Registered Customer's Unique Identifier
58             type: string
59             example: "2e850e2f-f81d-44fd-bef8-3bb5e90791ff"
60           seat_num:
61             description: seat number
62             type: string
63           example:
64             flight_id: "edcc03a4-7f4e-40d1-898d-bf84a266f1b9"
65             customer_id: "2e850e2f-f81d-44fd-bef8-3bb5e90791ff"
66             seat_num: "8D"
67         responses:
68         '200':
69           description: |
70             Success.
71             content:
72               application/json:
73                 schema:
74                   type: object
75                   properties:
76                     status:
77                       type: string
78                       enum: ["success", "error"]
79             example:

```

## Block 2: Clone ms-reservations and build it.

```

cd
git clone https://github.com/jmetzger/ms-reservations.git msupandrunning
cd msupandrunning
sudo apt install -y make
make

## it then shows the logs
CTRL + C
## app will still be running as it is daemonized (see start in Makefile)

```

## Block 3: Open Client on redis-server to test

Start redis-cli within redis-server

```
make redis
```

These are direct calls to redis through the redis cli

```

echo "in redis client - enter our first seat reservation"

HSETNX flight:40d1-898d-bf84a266f1b9 12B b4cdf96e-a24a-a09a-87fb1c47567c

## this means success -> (integer) 1

HSETNX flight:40d1-898d-bf84a266f1b9 12C e0392920-a24a-b6e3-8b4ebcbe7d5c

```

Now retrieve the occupied seats (in redis cli)

```
HKEYS flight:40d1-898d-bf84a266f1b9
```

This is the output of the reserved seats (keys)

```
1) "12B"
2) "12C"
```

Like so you can also! get the passenger-id of a seat (so including both)

```
HGETALL flight:40d1-898d-bf84a266f1b9
```

That is the output

```
1) "12B"
2) "b4cdf96e-a24a-a09a-87fb1c47567c"
3) "12C"
4) "b4cdf96e-a24a-a09a-87fb1c47567c"
```

Let's try double-booking (of seat)

```
HSETNX flight:40d1-898d-bf84a266f1b9 12B b4cdf96e-a24a-a09a-87fb1c47567c
```

This is how the result looks like

```
## this means success error -> (integer) 0
```

now leave redis-cli again

```
exit
```

#### Block 4: Test microservice with rest-api call

```
## only works when project name is: msupandrunning
make ps

curl --verbose --header "Content-Type: application/json" --request PUT --data '{"seat_num":"12C","flight_id":"werty",
"customer_id": "dfgh"}' http://192.168.56.102:7701/reservations

curl --verbose --header "Content-Type: application/json" --request PUT --data '{"seat_num":"12D","flight_id":"werty",
"customer_id": "dfgh"}' http://192.168.56.102:7701/reservations

## Try once again
## --verbose also shows the headers
curl --verbose --header "Content-Type: application/json" --request PUT --data '{"seat_num":"12D","flight_id":"werty",
"customer_id": "dfgh"}' http://192.168.56.102:7701/reservations
```

#### Reference

- <https://redis.io/docs/latest/commands/hsetnx/>

#### Upload image microservice - reservations

##### Step 1: Upload image to docker hub

```
## eventually
cd msupandrunning
## show all images build through this docker compose
docker compose images

## msupandrunning-ms-reservations      latest
## to upload it to docker hub, we would need to tag it
## one image can have multiple tags
```

##### Das image wird getagged

- Damit klar ist, wo es hingeschickt werden soll und zwar welches Images

```
## Bitte <namenskuerzel> ersetzen, z.B. jm
docker tag msupandrunning-ms-reservations dockertrainereu/reservations-<namenskuerzel>:v14
## now enter dockertrainereu + password-you-will-get-from-your-trainer ;)
docker login
```

```
## push the image to the server
## <namenskuerzel> ersetzen durch z.B. jm
docker push dockertrainereu/reservations-<namenskuerzel>:v14
```

#### Build image reservations with gitlab ci/cd

##### Step 1: Clone Repo from github locally

```
cd
git clone https://github.com/jmetzger/ms-reservations.git
cd ms-reservations
```

##### Step 1.5: Set identity

```
git config --global user.name "Max Mustermann"
git config --global user.email "tn1@t3company.de"
```

##### Step 2: Change origin (target where push data) and push

```
## of your newly created repo on gitlab
git remote -v
git remote set-url origin https://gitlab.com/training.tn1/ms-jochen.git
## find out current branch and use it in next step
## marked with a *
git branch
## enter username + password
git push -u origin master
```

##### Step 3a: build image and push to gitlab registry

```
## modify gitlab-ci.yml with pipeline editor as follows
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:\"$SCI_REGISTRY_USER"
    - echo "pass:\"$SCI_REGISTRY_PASSWORD"
    - echo "registry:\"$SCI_REGISTRY"
    - echo $SCI_REGISTRY_PASSWORD | docker login -u $SCI_REGISTRY_USER $SCI_REGISTRY --password-stdin
    - docker build -t $SCI_REGISTRY_IMAGE .
    - docker images
    - docker push $SCI_REGISTRY_IMAGE
    - echo "BUILD for $SCI_REGISTRY_IMAGE done"

## this will run, when you commit

```

### Step 3b: Build image, when setting a tag and upload to docker hub

#### 3a) Ministep 1 - add variables for docker in SETTINGS -> CI/CD -> Variables

```

## add
DOCKER_USER
DOCKER_PASS
DOCKER_PROJECT # z.B. reservations-jm
in Settings -> CI/CD -> Variables (in your repo)

```

Key	Value	Attributes	Environments	Actions
DOCKER_PASSWORD	*****	Masked	All (default)	<span>edit</span> <span>delete</span>
DOCKER_PROJECT	*****		All (default)	<span>edit</span> <span>delete</span>
DOCKER_USER	*****		All (default)	<span>edit</span> <span>delete</span>

#### 3b) Ministep 2

```

stages:
  - 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:\"$DOCKER_USER"
    - echo "pass:\"$DOCKER_PASSWORD"
    - echo "project:\"$DOCKER_PROJECT"
    - echo $DOCKER_PASSWORD | docker login -u $DOCKER_USER --password-stdin
    - docker build -t $DOCKER_USER/$DOCKER_PROJECT:$SCI_COMMIT_TAG .
    - docker images
    - docker push $DOCKER_USER/$DOCKER_PROJECT:$SCI_COMMIT_TAG
    - echo "BUILD for \"$DOCKER_USER/$DOCKER_PROJECT:$SCI_COMMIT_TAG\" done"

rules:
  # Man muss einen Tag setzen, damit die Pipeline triggered.
  - if: $SCI_COMMIT_TAG

## Jetzt zum Testen (Triggern der Pipeline)
## neuen Tag setzen
CODE -> Tags -> New Tag -> (z.B.) v3
## https://gitlab.com/training.tn1/ms-jochen/-/tags/new

```

## Microservice - flightapp - flights

### Template for microservice flights with node bootstrap

#### Good idea to start microservices with a fixed template within your team

##### Why ?

- Way shorter implementation time for new microservice based on python

##### What do we use it for ?

- Flights

##### Refs (Specifically for microservices)

- <https://nodebootstrap.com>

### Build flight app

#### Step 1: Use bootstrap github template

- <https://github.com/jmetzger/nodebootstrap-microservice>

```
## as unprivileged user / root
cd

## either usetemplate
## but we will just clone it locally
git clone https://github.com/jmetzger/nodebootstrap-microservice ms-flights
```

#### Step 2: Create documentation

```
cd ms-flights/docs

## Replace content in api.yml
## with our own definition from
https://raw.githubusercontent.com/jmetzger/ms-flights/master/docs/api.yml

## now render the docs and open 3939 port with container running
make start
docker container ls

## in browser of rdp
http://192.168.56.102:3939
```

#### Step 3: Cleanup and restructure

##### Step 3.1 Delete and rename unneeded files

```
## we will use the users module for something else
cd
cd ms-flights
mv lib/users lib/flights

## / -> homedoc is no needed, so we delete it
rm -fR lib/homedoc
```

##### Step 3.2 Adjust appConfig.js to change routings

```
nano appConfig.js

## 1. delete this line from appConfig.js
## app.use('/', require('homedoc')); // attach to root route

## 2. change line
## app.use('/users', require('users')); // attach to sub-route
## to ->
## app.use('/flights', require('flights')); // attach to sub-route
```

##### Step 3.3 Edit lib/flights/controllers/mappings.js for input validation and which functions to call

```
cd lib/flights/controllers
rm mappings.js

## the new version will look like this
wget https://raw.githubusercontent.com/jmetzger/ms-flights/master/lib/flights/controllers/mappings.js
```

##### Step 3.4 Edit lib/flights/controllers/actions.js

```

cd
cd ms-flights
mkdir -p lib/flights/controllers/
cd lib/flights/controllers/

## the new version will look like this
wget https://raw.githubusercontent.com/jmetzger/ms-flights/master/lib/flights/controllers/actions.js

```

### Step 3.5 Delete lib/flights/models

```

cd
cd ms-flights
rm -fR lib/flights/models/

```

### Step 3.6 Integrate MySQL - data

#### Step 3.6.1 Delete old stuff in migrations

```
rm -fR migrations/*
```

#### Step 3.6.2 Create files for upgrading/downgrading

```

## create migrations scripts (implemented in bootstrap)
## with correct dates
make migration-create name=seat-maps
make migration-create name=flights
make migration-create name=sample-data

## if you are working as unprivileged user change permissions accordingly
## They are root after this process
cd
cd ms-flights
sudo chown -R 11trainingdo:11trainingdo migrations

```

#### Step 3.6.3 Populate files

```

nano migrations/sqls/[date]-seat-maps-up.sql

## migrations/sqls/[date]-seat-maps-up.sql with data of
https://raw.githubusercontent.com/jmetzger/ms-flights/master/migrations/sqls/20200602055112-seat-maps-up.sql

nano migrations/sql/[date]-flights-up.sql

## migrations/sqls/[date]-flights-up.sql with data of
https://raw.githubusercontent.com/jmetzger/ms-flights/master/migrations/sqls/20200602055121-flights-up.sql

nano migrations/sqls/[date]-sample-data-up.sql

## migrations/sqls/[date]-sample-data-up.sql with data of
https://github.com/jmetzger/ms-flights/blob/master/migrations/sqls/20200602055127-sample-data-up.sql

```

#### Step 3.6.4 Do the migration

```

## Doing make restart instead of make migrate, because new data needs to be in docker container
make restart

```

### Step 3.7 Renaming from ms-nodebootstrap-example to ms-flights

```

cd ms-flights
make stop
## Change all entries that appear here to ms-flights
grep -r ms-nodebootstrap-example .

## when adjusted all entries doublecheck
grep -r ms-nodebootstrap-example
make restart

```

### Step 3.8 Testing

```

## Flight
curl http://192.168.56.102:5501/flights?flight_no=AA34&departure_date_time=2020-05-17T13:20

## Seat map
curl --verbose http://192.168.56.102:5501/flights/AA34/seat_map

```

## Upload image flight app

### Step 1: Upload image to docker hub

```
cd
cd ms-flights

## from the last step 01 Create microservice you should already have an image
docker images | grep flights
## ms-flights-ms-flights           latest
## to upload it to docker hub, we would need to tag it
## one image can have multiple tags
docker tag ms-flights-ms-flights dockertrainereu/flights-jm:v11
docker login
## now enter gittrainereu + password-you-will-get-from-your-trainer ;)
## push the image to the server
docker push dockertrainereu/flights-jm:v11
```

## Microservice - flightapp - Deployment Kubernetes

### Manual deployment

#### Schritt 1: configmap - flights

```
cd
mkdir -p manifests
cd manifests
mkdir flight-app
cd flight-app
mkdir flights
cd flights
nano 01-secret.yml

## you could also create a secret with
## kubectl create secret generic mariadb-secret --from-literal=MARIADB_ROOT_PASSWORD=11abc432 --dry-run=client -o yaml > 01-secrets.yml

### 01-secrets.yml
kind: Secret
apiVersion: v1
metadata:
  name: mariadb-secret
data:
  # als Wertepaare
  MARIADB_ROOT_PASSWORD: MTFhYmM0MzI=

cd
cd manifests/flight-app
## alle Unterverzeichnisse recursiv ausführen
kubectl apply -Rf .
kubectl get secrets
kubectl get secrets mariadb-secret -o yaml
```

#### Schritt 2: PersistentVolumeClaim

```
cd
cd manifests/flight-app
cd flights
nano 02-pvc.yml

## vi 02-pvc.yml
## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-do
spec:
  storageClassName: do-block-storage
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
cd
cd manifests
cd flight-app
kubectl apply -Rf .
```

- Ref: <https://docs.digitalocean.com/products/kubernetes/how-to/add-volumes/>

### Schritt 3: mariadb Deployment

```
cd
cd manifests/flight-app/flights
nano 03-deploy-mariadb.yml
```

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

      volumeMounts:
        - mountPath: "/var/lib/mysql"
          name: do-volume

      envFrom:
        - secretRef:
            name: mariadb-secret

      volumes:
        - name: do-volume
          persistentVolumeClaim:
            claimName: pvc-do
```

```
cd
cd manifests/flight-app/
kubectl apply -Rf .
```

### Schritt 3.1 Service für MariaDB anlegen

```
cd
cd manifests/flight-app/flights
nano 04-service-mariadb.yml
```

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

```
cd
cd manifests/flight-app/
kubectl apply -Rf .
```

### Schritt 3.2: Add flights deployment

```
cd
cd manifests/flight-app/flights
nano 03-deploy-flights.yml
```

```

##deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flights-deployment
spec:
  selector:
    matchLabels:
      app: flights
  replicas: 1
  template:
    metadata:
      labels:
        app: flights
  spec:
    containers:
      - name: app
        image: dockertrainereu/flights-jm:v11
        command: [ "/bin/sh", "-c", "--" ]
        args: [ "while true; do sleep 30; done;" ]
        #volumeMounts:
        #  - mountPath: "/var/lib/mysql"
        #    name: do-volume
    env:
      - name: NODE_ENV
        value: dev
        #  - name: NODE_HOT_RELOAD
        #    value: "1"
        #  - name: NODE_LOGGER_GRANULARLEVELS
        #    value: "1"
      - name: NODE_CONFIG_DISABLE_FILE_WATCH
        value: "Y"
    #volumes:
    #  - name: do-volume
    #persistentVolumeClaim:
    #  claimName: pvc-do

```

```

cd
cd manifests/flight-app/
kubectl apply -Rf .

```

#### Schritt 4: Lokal kompose installieren

- als root

##### [Kompose installieren](#)

```

## alle weiteren Schritte als kurs
su - kurs

```

#### Schritt 5: ms-reservations clonen (zur Hilfe bzgl. der manifests)

```

cd
git clone https://github.com/jmetzger/ms-reservations
cd ms-reservations

## create a dummy folder
mkdir dummy
cp -a docker-compose.yml dummy
cp -a database-dev.env dummy
cd dummy
kompose --file=docker-compose.yml convert

```

#### Schritt 6: config für redis anlegen

```

cd
mkdir -p manifests/flight-app/reservations
cd manifests/flight-app/reservations
nano 01-redis-cm.yml

apiVersion: v1
kind: ConfigMap
metadata:

```

```

name: redis-cm
data:

  REDIS_HOST: "ms-reservations-redis"
  REDIS_PORT: "6379"
  REDIS_DB: "0"
  REDIS_PWD: "4n_ins3cure_P4ss"

  redis-config: |
    appendonly yes

```

```
kubectl apply -f .
```

#### Schritt 7: Redis ausrollen

```

nano 02-redis-deploy.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ms-reservations-redis
spec:
  replicas: 1
  selector:
    matchLabels:
      storage: redis
  strategy:
    type: Recreate

  template:
    metadata:
      labels:
        storage: redis

    spec:
      containers:
        - command:
            - redis-server
            - /usr/local/etc/redis/redis.conf
            - --requirepass
            - 4n_ins3cure_P4ss
        env:
          - name: REDIS_REPLICATION_MODE
            value: master
        image: redis:6-alpine
        name: ms-reservations-redis
        ports:
          - containerPort: 6379
        volumeMounts:
          - mountPath: /data
            name: data
          - mountPath: /usr/local/etc/redis/redis.conf
            name: config
        volumes:
          - name: data
            emptyDir: {}
          - name: config
            configMap:
              name: redis-cm
              items:
                - key: redis-config
                  path: redis.conf

```

```
kubectl apply -f .
```

#### Schritt 8: service für redis

```

nano 03-redis-service.yml

apiVersion: v1
kind: Service
metadata:
  name: ms-reservations-redis
spec:
  ports:
    - name: "redis"

```

```
  port: 6379
  selector:
    storage: redis
```

```
kubectl apply -f .
```

#### Schritt 9: deployment for reservations

```
nano 04-reservations-deploy.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ms-reservations
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reservations
  template:
    metadata:
      labels:
        app: reservations
    spec:
      containers:
        - command: [ "/bin/bash", "-c", "--" ]
          args: [ "while true; do sleep 30; done;" ]
          #       - ./wait-for.sh
          #       - -t
          #       - "60"
          #       - ms-reservations-redis:6379
          #       - --
          #       - gunicorn
          #       - -b
          #       - 0.0.0.0:5000
          #       - --reload
          #       - -w
          #       - "1"
          #       - service:app
      env:
        - name: FLASK_ENV
          value: development
        - name: REDIS_DB
          valueFrom:
            configMapKeyRef:
              key: REDIS_DB
              name: redis-cm
        - name: REDIS_HOST
          valueFrom:
            configMapKeyRef:
              key: REDIS_HOST
              name: redis-cm
        - name: REDIS_PORT
          valueFrom:
            configMapKeyRef:
              key: REDIS_PORT
              name: redis-cm
        - name: REDIS_PWD
          valueFrom:
            configMapKeyRef:
              key: REDIS_PWD
              name: redis-cm
      image: dockertrainereu/reservations-jm:v16
      name: ms-reservations
      ports:
        - containerPort: 5000
      resources: {}
      volumeMounts:
        - mountPath: /app
          name: ms-reservations-claim0
      volumes:
        - name: ms-reservations-claim0
          emptyDir: {}
          #persistentVolumeClaim:
          #  claimName: ms-reservations-claim0
```

```
kubectl apply -f .
```

## gitlab Deployment

### Prerequisites

- 01-xxxx is done by you (manifests created)

### Schritt 1: Neues repo anlegen und manifeste pushen

```
### neues repo in gitlab anlegen
### achtung ohne README -> d.h. leer
z.B.
https://gitlab.com/training.tn1/ms-jochen-k8sdeploy
```

```
cd
cd manifests/
mkdir project-flight-app
mv flight-app project-flight-app
cd project-flight-app
git init
git add .
git status
git config --global user.email "you@email.com"
git config --global user.name "Phantomas"
git commit -am "initial release"
git log
## Wo soll es hingehen, aus Startseite im Repo, wenn keine README gesetzt
git remote add origin https://gitlab.com/training.tn1/ms-jochen-k8sdeploy.git
## In welchem Branch bin ich
git branch
git push -u origin master
```

### Schritt 2: KUBECONFIG\_SECRET einrichten

- in Settings->CI/CD -> Variables -> KUBECONFIG\_SECRET

CI/CD Variables </> 1			Reveal values	Add variable
↑ Key	Value	Attributes	Environments	Actions
KUBECONFIG_SECRET	*****	Expanded	All (default)	 

```
## Inhalt kommt von meinem lokalen System, wo ich auch kubectl verwende
## -> wenn eine Verbindung zum Cluster besteht, ansonsten aus management tool des Clusters , z.B microk8s config
cat .kube/config
```

### Schritt 3: pipeline mit kubectl einrichten

- Ich brauche ein image, das kubectl kann

```
## on gitlab create a new pipeline
## by editing with pipeline editor
```

```
## use the following content
deploy:
  image:
    name: bitnami/kubectl:latest
    entrypoint: ['']
  script:
    - echo "$KUBECONFIG_SECRET" > ~/.kube/config
    - kubectl cluster-info
    # - ls -la
    - cd flight-app
    - kubectl apply -Rf .
```

### Schritt 4: version des images ändern in ...

```
## image-version muss in docker hub vorhanden sein
z.B. v3 -> v4.
```

```
## hier z.B. direkt im repo ändern
flight-app/reservations/04-reservations-deploy.yml
```

#### Ref:

- [https://docs.gitlab.com/ee/user/clusters/agent/cl\\_cd\\_workflow.html#update-your-gitlab-ciyaml-file-to-run-kubectl-commands](https://docs.gitlab.com/ee/user/clusters/agent/cl_cd_workflow.html#update-your-gitlab-ciyaml-file-to-run-kubectl-commands)

#### github Deployment

##### Step 1: Create Repo in github

```
## url
https://github.com/new

## Bitte hier keine Dateien anlegen
## keine README.md
## keine .gitignore
```

##### Step 2: Create personal access token (Optional)

- you can do this here: <https://github.com/settings/tokens/new>

#### New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

What's this token for?

**Expiration \***

30 days The token will expire on Sun, May 26 2024

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input type="checkbox"/> <b>repo:status</b>	Access commit status
<input type="checkbox"/> <b>repo_deployment</b>	Access deployment status
<input type="checkbox"/> <b>public_repo</b>	Access public repositories
<input type="checkbox"/> <b>repo:invite</b>	Access repository invitations
<input type="checkbox"/> <b>security_events</b>	Read and write security events
<input checked="" type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry

##### Step 3: Lokal projekt anlegen (auf dem kubectl - client)

```
cd
mkdir -p github-test
cd github-test

mkdir -p manifests
cd manifests
nano 01-deployment.yaml
```

##### Step 4: Populate project with sample manifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8
  template:
    metadata:
      labels:
```

```

  app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.23
    ports:
    - containerPort: 80

```

#### Step 5: Projekt unter Versionsverwaltung stellen und pushen

```

cd
git init
git config --global user.email test@test.de
git config --global user.name "Jochen from ml"

git remote add origin https://github.com/gittrainereu/microjay2.git

git add .
git commit -am "Initial Release"
## wir werden gefragt nach:
## user-name
## password -> hier bitte den Personal Token verwenden
git push -u origin master

```

#### Step 6: KUBERNETES\_CONFIG als Secret anlegen

```

## kopieren der Ausgabe von server mit kubectl
cat ~/.kube/config

## Enter it here, by adding a new secret: KUBERNETES_CONFIG
## secret für Repository
https://github.com/gittrainereu/<your-repo> /settings/secrets/actions/new

```

The screenshot shows the GitHub repository settings for the 'KUBERNETES\_CONFIG' secret. The left sidebar includes sections for General, Collaborators, Moderation options, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), Security (Code security and analysis, Deploy keys, Secrets and variables - which is selected and expanded to show Actions, Codespaces, and Dependabot), and Actions. The main content area is titled 'Actions secrets and variables'. It has tabs for 'Secrets' (selected) and 'Variables'. The 'Secrets' tab displays the message 'This repository has no environment secrets.' with a 'Manage environment secrets' button. The 'Variables' tab displays the message 'This repository has no secrets.' with a 'New repository secret' button.

#### Step 7: Setup github actions (in web ui of github)

- workflow folder: .github/workflows
- manifests - folder: manifests/

```

## create file .github/workflows/pipeline.yaml
## with content

## adjust
## 1. server-url / use data from last step
## 2. your-name / use your own namespace here
name: CI/CD

```

```

on: push
jobs:
  deploy:
    name: Deploy
    # needs: [ test, build ]
    runs-on: ubuntu-latest
    steps:
      - name: Set the Kubernetes context
        uses: azure/k8s-set-context@v2
        with:
          method: kubeconfig
          kubeconfig: ${{ secrets.KUBERNETES_CONFIG }}

      - name: Checkout source code
        uses: actions/checkout@v3

      - name: Deploy to the Kubernetes cluster
        uses: azure/k8s-deploy@v5
        with:
          namespace: <yournamespace>
          manifests: |
            manifests

```

#### Step 9: watch and enjoy

##### Reference

- <https://github.com/marketplace/actions/deploy-to-kubernetes-cluster>

##### github Deployment-with-secret-not-working

##### What is not working ?

- We get an error in the pipeline, there seems
- to be a misconfiguration about the secret we use

#### Step 1: Create Repo in github

```

## url
https://github.com/new

```

#### Step 2: Create personal access token

- you can do this here: <https://github.com/settings/tokens/new>

#### New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

What's this token for?

**Expiration \***

30 days  The token will expire on Sun, May 26 2024

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> <b>repo:status</b>	Access commit status
<input checked="" type="checkbox"/> <b>repo_deployment</b>	Access deployment status
<input checked="" type="checkbox"/> <b>public_repo</b>	Access public repositories
<input checked="" type="checkbox"/> <b>repo:invite</b>	Access repository invitations
<input checked="" type="checkbox"/> <b>security_events</b>	Read and write security events
<input checked="" type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry

#### Step 3: Clone Repo to local system (machine where we use kubectl )

```
## on local system -> clone to k8s-deploy
cd
mkdir -p github-test
cd github-test
## so we all have the same folder in the training (for our ease)
git clone <your-repo> k8s-deploy
cd k8s-deploy

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

#### Step 4: Populate Repo with sample manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

#### Step 5: Push changes

```
git config --global user.email test@test.de
git config --global user.name "Jochen from m1"

git add -A
git commit -am "Initial Release"
git push -u origin main
```

#### Step 6: Setup authentication in kubernetes (service account) - in kubectl - client

```
## wird in deinem namespace angelegt
## create serviceaccount
kubectl create serviceaccount github-actions-tln<nr>

cd
mkdir -p manifests
cd manifests
mkdir github-account
cd github-account

nano 01-sassecret.yaml

## Secret für service account anlegen / wichtig: muss
## in neueren Versionen von kubernetes gemacht werden
## da secrets nicht mehr automatisch angelegt werden
## beim Erstellen von service account (Stand: 26.04.2024)
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: github-actions-secret
  annotations:
    kubernetes.io/service-account.name: github-actions-tln<nr>

kubectl apply -f .

nano 02-clusterrole.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: continuous-deployment-tln<nr>
rules:
  - apiGroups:
```

```

- ''
- apps
- networking.k8s.io
resources:
- namespaces
- deployments
- replicases
- ingresses
- services
- secrets
verbs:
- create
- delete
- deletecollection
- get
- list
- patch
- update
- watch

```

```
kubectl apply -f .
```

```
kubectl create clusterrolebinding continuous-deployment-tln<nr> \
--clusterrole=continuous-deployment-tln<nr>
--serviceaccount=<dein-namespace>:github-actions-tln<nr>
```

#### Step 7: secrets auslesen und bei github eintragen

```

kubectl get secrets github-actions-secret -o yaml

## Copy the output

## Enter it here, by adding a new secret: KUBERNETES_SECRET
https://github.com/gittrainereu/<your-repo>/settings/secrets/actions/new

## Get the url of your kubernetes cluster
## And Copy it to clipboard
## We will need this for your pipeline
kubectl config view -o 'jsonpath={.clusters[0].cluster.server}'

```

#### Step 8: Setup github actions (in web ui of github)

- workflow folder: github/workflows
- manifests - folder: manifests/

```

## create file .github/workflows/pipeline.yaml
## with content

## adjust
## 1. server-url / use data from last step
## 2. your-name / use your own namespace here
name: CI/CD
on: push
jobs:
  deploy:
    name: Deploy
    # needs: [ test, build ]
    runs-on: ubuntu-latest
    steps:
      - name: Set the Kubernetes context
        uses: azure/k8s-set-context@v2
        with:
          method: service-account
          k8s-url: <server-url>
          k8s-secret: ${{ secrets.KUBERNETES_SECRET }}

      - name: Checkout source code
        uses: actions/checkout@v3

      - name: Deploy to the Kubernetes cluster
        uses: azure/k8s-deploy@v1
        with:
          namespace: <yournamespace>
          manifests: |
            manifests/

```

## Step 9: watch and enjoy

### Reference

- <https://github.com/marketplace/actions/deploy-to-kubernetes-cluster>

## Kubernetes - Überblick

### Warum Kubernetes, was macht Kubernetes

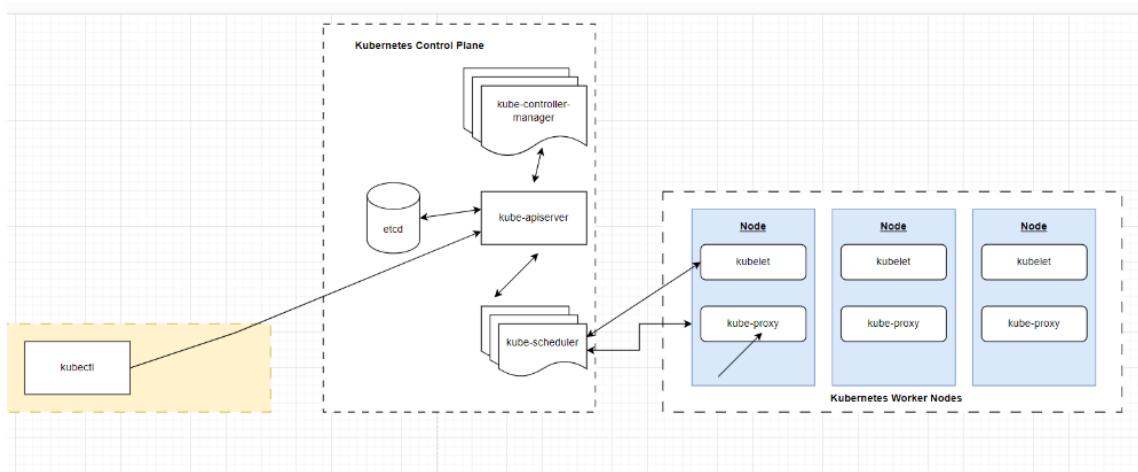
- Virtualisierung von Hardware - einfache bessere Auslastung
- Ist in Google entstanden
- Software 2014 als OpenSource zur Verfügung gestellt (Nachfolger von: Borg)
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbstheilend

### Wozu dient Kubernetes ?

- Orchestrierung von Containern
- am gebräuchlichsten aktuell Docker

### Aufbau 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 Status im Cluster mit Hilfe von endlosen loops.
- Kommuniziert mit dem Cluster über die Kubernetes-API (bereitgestellt vom kube-api-server)

###### KUBE-API-SERVER

- Provides API-frontend for administration (no GUI)
- Expose 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 die Anwendungen ausführen

- Control Plane Node, dort laufen die Tools zur Verwaltung/Beobachtung etc. des Clusters
- Ref: <https://kubernetes.io/de/docs/concepts/architecture/nodes/>

#### Pod/Pods

- Pods sind die kleinsten verwaltbaren 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

#### Control Plane Node (former: master) - components

#### Worker 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.

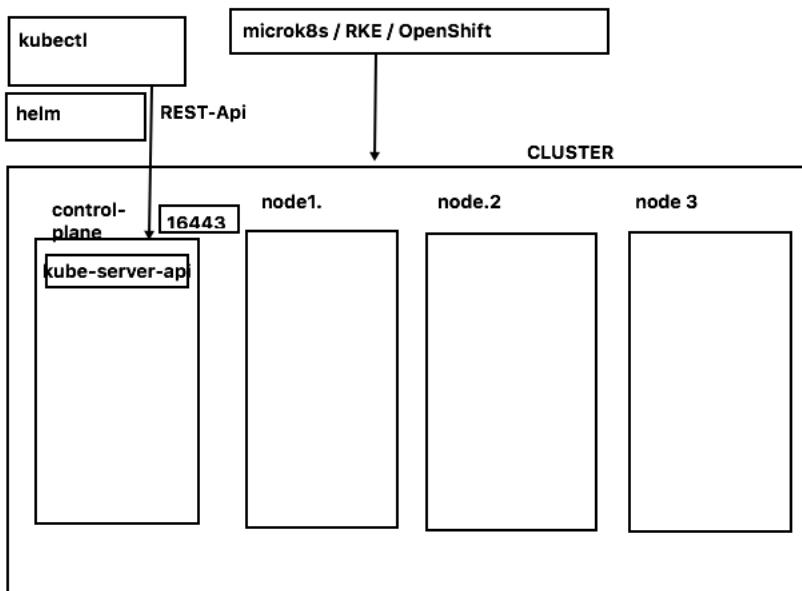
##### Referenzen

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

#### Structure Kubernetes Deep Dive

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

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



#### Welches System ? (minikube, micro8ks etc.)

#### Überblick der Systeme

##### General

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

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

##### Kubeadm

##### General

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

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

### k3s

- Sehr schlanker Footprint
- Sehr einfach und schnell zu installieren (ein binary)
- Auch für die Produktion geeignet.
- Sinnvoll für den Einsatz auf sehr ressourcenarmen Systemen

### 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.

## Kubernetes - Einsatz

### Kubernetes Einsatz - Risiken

#### Kubernetes

#### Leitungsteam

- Risiko: Vertrauen aufbauen.

#### Größter Schmerz

- Know-How Träger gehen weg
  - Lösung: CI/CD Pipeline
- Keine neuen Mitarbeiter zu finden/ nicht genug
  - junge Absolventen wollen hippe Technologie verwenden
  - Lösung: kein Technologie-Login bzgl der Programmierung
- Dauert der Wechsel zu lange ?
  - Lösung: wie gross der Schmerz bzgl. Probleme die auftreten -> Implementierung zu lange (Time to market)
  - Technologie-Schuld: Wo wollen wir in 10 Jahren (und halten wir die Technologie up-to-date)

## Team / Prozesse

- Fehlendes Knowledge im Team für Kubernetes
  - Implikation -> Single Point of Failure
  - Berührungsängste / Nicht-Handlung bzgl. Kubernetes
- Keinen klaren Prozess Updates der Infrastruktur
  - Implikation; Veraltete Versionenstände
  - Implikation: Angriffbarkeit durch Sicherheitslücken
- keinen klaren Prozess auf Updates von Applikation (Prozess Anpassung der Manifeste)
  - Implikation: manifeste funktionieren nicht mehr oder unerwartet auf neueren Version
  - Implikation: bestimmte Feature sind nicht mehr verfügbar (PSP - Pod Security Policies (deprecated), PSA (Pod Security Admission))

## Pods

- Pods (Container) als Root laufen lassen (Angriffsrisiko)
  - Implikation: Wenn jemand den Pod hacked, kann er u.U. Rechte auf der Host-Maschinen bekommen.

## Images / Sicherheit

- Images verwenden alte Versionen von Software und nicht überprüft, gescannt.

## Backup / Monitoring / Observability

- Kubernetes Aware Backups-System (kasten.io) wird nicht verwendet
  - Implikation: Das falsche wird gesichert.
  - Implikation: Hohe Komplexität beim Zurückspielen (falsche Volume wird zurückgesichert, unklar welches)
- Richtige Backupstrategie (manifeste, Grundeinrichtung Server, Konfigurationsdateien (speziell für Cluster-Infrastruktur - besser Infrastructure by Code))

## Performance

- Pods falsch konfiguriert sind.
  - Implikation: die falschen Pods werden verschoben.

## Microservices

### Fehlende Tools zur Analyse

- Entscheidende Tools für Monitoring, Tracing, Observability nicht oder falsch eingerichtet.
- das aller wichtigste Tracing (Jaeger) und pro Anfrage einen durchgängigen Key
- Log Aggregation (ELK / EFK) -> Um aus der Vogelperspektive Problem erfassen und trigger zu setzen
  - Logs vom Server, Logs vom Kubernetes Cluster, Logs von den Pods -> zentral zusammenläuft
- Implikation: Blindflug nach Ausrollen von MicroServices

### Wir machen mal microservices weil es cool ist

- Wichtig: Warum nehme ich microservices
- Warum kein Monolith
- Microservices: Kaufpreis für Effekt
- Implikation: Dat Ding fliegt mir um die Ohren
  - Team: Zu aufwändig.
  - Team: fehlendes Knowledge.
  - Team: Falsche Einschätzung der Performance
  - Team: Grobe Konzeptionsfehler

### Zu schnell zu viel wollen

- Alles von Anfang bis Ende durchkonzipieren
- Zuviele Microservices werden zu schnell online genommen
  - UND: die alte Funktionalität im Monolithen zu schnell abgeschaltet.
- BESSER: Kleine Schritte, Erfahrungen sammeln, MESSEN !

## Kubernetes Datenbanken in Kubernetes oder ausserhalb

### Aspekt: Debugging (Expertise im Team)

- Kann kann ein Killer-Kriterium sein, weil ich jemand brauche,
  - der sowohl die DB beherrscht als auch Kubernetes
- Wenn ich keinen solchen habe, sollte ich es NICHT in Kubernetes betreiben

### Performance - Optimierung / Debugging

- Memory is key (Je mehr Arbeitsspeicher desto besser)
- Funktionieren am besten, wenn alle häufig verwendeten Daten in den Arbeitsspeicher passen

### Nachteil Kubernetes:

- Erhöhte Komplexität (wo muss ich hinlangen)
- Manche Datenbanksystemen kommen nicht gut damit zurecht, wenn pods häufig mit der Datenbank
  - neu erstellt werden

## Wenn Kubernetes:

- Gibt es eine Operator für diesen Datenbank

## Folgende Datenbanken gehen garnicht als Installation innerhalb des Kubernetes Cluster

- Oracle

## Folgende Datenbanken sind nicht so gut geeignet eine Installation innerhalb des Kubernetes Cluster

- mariadb und postgresql

## Referenz:

- <https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider?hl=en>
- <https://operatorhub.io/?keyword=mariadb>

## Kubernetes mit microk8s (Installation und Management)

### Installation Ubuntu - snap

#### Walkthrough

```
sudo snap install microk8s --classic
microk8s status

## Sobald Kubernetes zur Verfügung steht aktivieren wir noch das plugin dns
microk8s enable dns
microk8s status
```

#### Optional

```
## 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>

### 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
##z.B. -> ACHTUNG evtl. IP ändern
microk8s join 10.128.63.86:25000/567a21bdfc9a64738ef4b3286b2b8a69
```

### Auf einem Node addon aktivieren z.B. ingress

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

#### Ref:

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

### 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

```

## Kubernetes mit k3s

### Kubernetes mit k3s

#### Install (Quickstart) - 1 Node Cluster

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

### Flannel

- Kann keine NetworkPolicies
- Angelegte NetzwerkPolicies rennen ins Leere
- Flannel läuft nicht als Pod, sondern ist in der k3s - binary drin

### Wenn Netzwerkolicies dann z.B. calico install

- <https://docs.tigera.io/calico/latest/getting-started/kubernetes/k3s/quickstart>

## Kubernetes - Client Tools und Verbindung einrichten

### Tools installieren und bash-completion / syntax highlighting

#### Tools helm und kubectl

```

snap install kubectl --classic
snap install helm --classic
## Voraussetzung, ist dass das Paket bash-completion installiert
kubectl completion bash > /etc/bash_completion.d/kubectl
helm completion bash > /etc/bash_completion.d/helm

```

### Highlightning und Indenting nano

```

sudo echo "include /usr/share/nano/yaml.nanorc" >> /etc/nanorc
sudo echo "set autoindent" >> /etc/nanorc
sudo echo "set tabs 2" >> /etc/nanorc
sudo echo "set tabstop 8" >> /etc/nanorc

```

### Remote-Verbindung zu Kubernetes einrichten

#### config einrichten

```

## als unprivilegierter Benutzer z.B. kurs
cd
mkdir -p .kube
cd .kube

cp /tmp/config config
ls -la

kubectl cluster-info

```

### Arbeitsbereich konfigurieren

```
kubectl create ns jochen
kubectl get ns
kubectl config set-context --current --namespace jochen
kubectl get pods
```

#### Tool zum Konvertion von docker-compose.yaml file manifesten

```
## als root
sudo su -

curl -L https://github.com/kubernetes/kompose/releases/download/v1.26.0/kompose-linux-amd64 -o kompose
chmod +x kompose
sudo mv ./kompose /usr/local/bin/kompose
```

#### Ref:

- <https://kubernetes.io/docs/tasks/configure-pod-container/translate-compose-kubernetes/>

### Kubernetes Praxis API-Objekte

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

##### Allgemein

```
## Zeige Information über das Cluster
kubectl cluster-info

## Welche api-resources gibt es ?
kubectl api-resources

## Hilfe zu object und eigenschaften bekommen
kubectl explain pod
kubectl explain pod.metadata
kubectl explain pod.metadata.name
```

##### Arbeiten mit manifesten

```
kubectl apply -f nginx-replicaset.yml
## Wie ist aktuell die hinterlegte config im system
kubectl get -o yaml -f nginx-replicaset.yml

## Änderung in nginx-replicaset.yml z.B. replicas: 4
## dry-run - was wird geändert
kubectl diff -f nginx-replicaset.yml

## anwenden
kubectl apply -f nginx-replicaset.yml

## Alle Objekte aus manifest löschen
kubectl delete -f nginx-replicaset.yml
```

##### Ausgabeformate

```
## Ausgabe kann in verschiedenen Formaten erfolgen
kubectl get pods -o wide # weitere informationen
## im json format
kubectl get pods -o json

## gilt natürlich auch für andere commandos
kubectl get deploy -o json
kubectl get deploy -o yaml

## Eigenschaft auslesen
kubectl get pods nginx-deployment-74676ff58f-fxcjv -o jsonpath='{.metadata.ownerReferences[0].name}'
```

##### Zu den Pods

```
## Start einen pod // BESSER: direkt manifest verwenden
## kubectl run podname image=imagename
kubectl run nginx image=nginx

## Pods anzeigen
kubectl get pods
kubectl get pod
```

```
## Format weitere Information
kubectl get pod -o wide
## Zeige labels der Pods
kubectl get pods --show-labels

## Zeige pods mit einem bestimmten label
kubectl get pods -l app=nginx

## Status eines Pods anzeigen
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

### Zu den Pods (Logs)

```
## log eines pods anzeigen
kubectl logs podname

## Logs aller pods im Deployment
## Wichtig Option --prefix
kubectl logs --prefix deploy/web-nginx
```

### Arbeiten mit namespaces

```
## Welche namespaces auf dem System
kubectl get ns
kubectl get namespaces
## Standardmäßig wird immer der default namespace verwendet
## wenn man kommandos aufruft
kubectl get deployments

## Möchte ich z.B. deployment vom kube-system (installation) aufrufen,
## kann ich den namespace angeben
kubectl get deployments --namespace=kube-system
kubectl get deployments -n kube-system

## wir wollen unseren default namespace ändern
kubectl config set-context --current --namespace <dein-namespace>
```

### Arbeiten mit der Config (lokal)

```
## namespace jochen als default setzen
kubectl config set-context --current --namespace jochen
## config anzeigen
kubectl config view
```

### Referenz

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

### kubectl example with run

#### Example (that does work)

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

## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx:1.21

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

#### Example (that does not work)

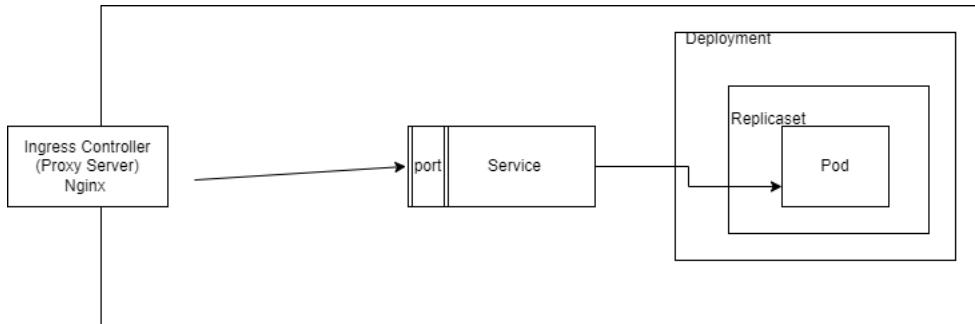
```
kubectl run meinfailscherpod --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
```

```
## Weitere status - info
kubectl describe pods meinfailscherpod
```

#### 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 -p web
cd web
```

```
nano nginx-static.yml
```

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

```
## pod auf Basis von manifest löschen
kubectl delete -f nginx-static.yml
```

#### kubectl/manifest/replicaset

##### Schritt 1: Erstellen

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

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replica-set
spec:
  replicas: 10
  selector:
    matchLabels:
      tier: frontend
```

```

template:
  metadata:
    name: template-nginx-replica-set
    labels:
      tier: frontend
  spec:
    containers:
      - name: nginx
        image: nginx:1.21
        ports:
          - containerPort: 80

```

```
kubectl apply -f rs.yaml
```

## Schritt 2: Erforschen

```

kubectl get all
## Hash entsprechend anpassen
kubectl delete po nginx-replica-set-<hash>
## Dass einer neuer Pod dazugekommen ist (seht ihr an der Zeit)
kubectl get all

```

```

## ändern image in rs.yaml
## vorher
## image: 1.23
## jetzt
image: 1.22

```

```
kubectl apply -f .
```

```

## Gibt es neue Pods ?
kubectl get all
## Welche Image - Version
kubectl describe pods nginx-replica-set-vh6cl

```

```

## FYI
kubectl get rs
kubectl get pods --show-labels

```

## kubectl/manifest/deployments

### Schritt 1: Erstellen

```

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

```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.21
          ports:
            - containerPort: 80

```

```
kubectl apply -f deploy.yaml
```

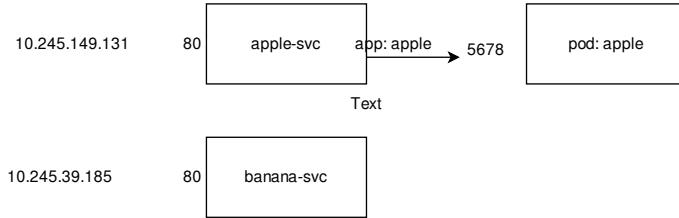
## Schritt 2: Erforschen

```
kubectl get all

## image ändern in deploy.yml
## vorher: image: nginx:1.21
## jetzt
image: nginx:1.23

## Anwenden und wachten
kubectl apply -f . ; kubectl get all; kubectl get pods -w
```

## Services - Aufbau



## kubectl/manifest/service

### Example I : Service with ClusterIP

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

nano deploy.yml

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

```
nano service.yml
```

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

kubectl apply -f .

## find out endpoints, if they are working
kubectl get svc svc-nginx
```

```

kubectl describe svc svc-nginx

## now delete pod and see changes
## -> podip will disappear from service / kubectl describe svc-nginx
kubectl delete po web-nginx-596cd7d5c-2lsr6
kubectl get pods -o wide

kubectl get svc svc-nginx

## New pod (with pod-ip) is detected by service
## and now in the list of the endpoints
kubectl describe svc svc-nginx

```

#### Example II : Short version

```

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

kubectl apply -f .
kubectl get svc
## Über welche externe IP können wir zugreifen ?
kubectl get nodes -o wide
## im Client
curl http://164.92.193.245:30280

```

#### Example II : Service with NodePort (long version)

```

apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: 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 und helm muss eingerichtet sein

#### Walkthrough (Setup Ingress Controller)

```

## Setup repo
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

mkdir -p manifests
cd manifests
mkdir ingress
cd ingress

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-nginx

## Now setup wildcard - domain for training purpose
## inwx.com
*.lab1.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

#### Schritt 1:

```

cd
mkdir -p manifests
cd manifests
mkdir abi
cd abi

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

```

```

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
```

#### Schritt 2:

```

## 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
```

## Install Ingress On DigitalOcean DOKS

### Beispiel Ingress mit Hostnamen

#### Prerequisites

- An IngressController is running in your cluster. Ask the trainer if you are unsure.

#### Aufsetzen mit Ingress-Fehler

##### Step 1: pods and services

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

```
nano apple.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apple-deployment
spec:
  selector:
    matchLabels:
      app: apple
  replicas: 8
  template:
    metadata:
      labels:
        app: apple
    spec:
      containers:
        - name: apple-app
          image: hashicorp/http-echo
          args:
            - "-text=apple-<dein-name>"
```

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

```
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 .
nano banana.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: banana-deployment
spec:
  selector:
    matchLabels:
      app: banana
  replicas: 8
  template:
    metadata:
      labels:
```

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

```
nano banana-service.yml
```

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

```
kubectl apply -f .
```

## Step 2: Ingress

```
nano ingress.yml
```

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
spec:
  ingressClassName: nginx
  rules:
    - host: "<euername>.lab1.t3isp.de"
      http:
        paths:
          - path: /apple
            backend:
              serviceName: apple-service
              servicePort: 80
          - path: /banana
            backend:
              serviceName: banana-service
              servicePort: 80
```

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

## Schritt 3: Wir finden/lösen das Problem

### (Mini-)Schritt 2.1 api-resource und ApiVersion identifizieren

- Den richtige api-resource und die richtige appVersion finden

```
## Bitte nur die ersten 3 Zeilen anzeigen -> head -n 3
kubectl explain ingress | head -n 3
```

```
## Nur Anzeige, nicht eingeben
## GROUP:      networking.k8s.io
## KIND:       Ingress
## VERSION:    v1
```

```
## Wir erkennen das die richtige API-Ressource
## NICHT extensions/v1beta1
## SONDERN networking.k8s.io/v1
```

### (Mini-)Schritt 2.2. api-resource und apiVersion ändern und ausführen

```
## ist und korrigieren das.
nano ingress.yml
```

```
## nur
## apiVersion ändern wie folgt
```

```

apiVersion: networking.k8s.io/v1
## ändern

## Das gesamte File sieht jetzt so aus:
## Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # otherwise it does not know, which controller to use
    # old version... use ingressClassName instead
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "<euename>.lab1.t3isp.de"
    http:
      paths:
        - path: /apple
          backend:
            serviceName: apple-service
            servicePort: 80
        - path: /banana
          backend:
            serviceName: banana-service
            servicePort: 80

```

```
kubectl apply -f .
```

#### (Mini-)Schritt 2.3: Fehler verstehen

```

## Hier der letzte Fehler aus Schritt 2.2.
Error from server (BadRequest): error when creating "ingress.yml": Ingress in version "v1" cannot be handled as a Ingress: strict
decoding error: unknown field "spec.rules[0].http.paths[0].backend.serviceName", unknown field
"spec.rules[0].http.paths[0].backend.servicePort", unknown field "spec.rules[1].http.paths[0].backend.serviceName", unknown field
"spec.rules[1].http.paths[0].backend.servicePort"

```

```

## Auszug
unknown field "spec.rules[0].http.paths[0].backend.serviceName", unknown field "spec.rules[0].http.paths[0].backend.servicePort",
unknown field

```

```

## Bedeutet
## Ich kenne das Feld spec..backend.servicePort nicht.

```

```

## Heisst aber auch:
## Das Feld spec.rules.http.paths.backend .. kenne ich schon

```

```

## Wir forschen
kubectl explain ingress.spec.rules.http.paths.backend

```

```

## Er kennt eine Eigenschaft service, aber eben nich serviceName
FIELD: backend <IngressBackend>
...
service      <IngressServiceBackend>
  service references a service as a backend. This is a mutually exclusive
  setting with "Resource".

```

```

## Was möchte er unter service haben ?
kubectl explain ingress.spec.rules.http.paths.backend.service

```

```

## Bingo: name
GROUP:      networking.k8s.io
KIND:       Ingress
VERSION:    v1

FIELD: service <IngressServiceBackend>
[...]
FIELDS:
  name <string> -required-
    name is the referenced service. The service must exist in the same namespace
    as the Ingress object.

```

```
## Ergebnis:  
## Statt:  
## ingress.spec.rules.http.paths.backend.serviceName  
## Erwartet Kubernetes jetzt:  
## ingress.spec.rules.http.paths.backend.service.name
```

#### (Mini-)Schritt 2.4: Lösung umsetzen

```
nano ingress.yml
```

```
## Das gesamte File sieht jetzt so aus:  
## Ingress  
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  name: example-ingress  
  annotations:  
    ingress.kubernetes.io/rewrite-target: /  
    # with the ingress controller from helm, you need to set an annotation  
    # otherwise it does not know, which controller to use  
    # old version... use ingressClassName instead  
    # kubernetes.io/ingress.class: nginx  
spec:  
  ingressClassName: nginx  
  rules:  
  - host: "<euernname>.lab1.t3isp.de"  
    http:  
      paths:  
      - path: /apple  
        backend:  
          service:  
            name: apple-service  
            servicePort: 80  
      - path: /banana  
        backend:  
          service:  
            name: banana-service  
            servicePort: 80
```

```
kubectl apply -f .  
## --> Fehler
```

#### (Mini-)Schritt 2.5: Nächsten Fehler verstehen und umsetzen (servicePort)

```
## Folgender Fehler nach kubectl apply -f .  
Error from server (BadRequest): error when creating "ingress.yml": Ingress in version "v1" cannot be handled as a Ingress: strict  
decoding error: unknown field "spec.rules[0].http.paths[0].backend.servicePort", unknown field  
"spec.rules[1].http.paths[0].backend.servicePort"  
## <- servicePort kennt er nicht
```

```
## Schrittweise debuggen  
kubectl explain ingress.spec.rules.http.paths.backend  
kubectl explain ingress.spec.rules.http.paths.backend.service  
kubectl explain ingress.spec.rules.http.paths.backend.service.port  
## Und er braucht auch noch Number  
kubectl explain ingress.spec.rules.http.paths.backend.service.port.number  
## so würde die Eigenschaft dann im yml-file aussehen.  
## service:  
##   port:  
##     number: 80
```

```
## Wir setzen das um  
nano service.yml
```

```
## So sieht das korrigierte .yml file aus.  
## Ingress  
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  name: apps-ingress  
  annotations:  
    ingress.kubernetes.io/rewrite-target: /  
spec:  
  ingressClassName: nginx  
  rules:
```

```

- host: app1.dein-training.de
  http:
    paths:
      - path: /apple
        backend:
          service:
            name: apple-service
            port:
              number: 80
      - path: /banana
        backend:
          service:
            name: banana-service
            port:
              number: 80

```

```
kubectl apply -f .
## --> Fehler
```

#### (Mini-)Schritt 2.6: Wir beheben den letzten Fehler

```

## Fehler
The Ingress "apps-ingress" is invalid:
* spec.rules[0].http.paths[0].pathType: Required value: pathType must be specified
* spec.rules[1].http.paths[0].pathType: Required value: pathType must be specified

## Bedeutet
pathType ist jetzt ein Pflichtfeld und wir müssen es ergänzen

```

```

## Welche Werte sind möglich ?
kubectl explain ingress.spec.rules.http.paths.pathType

```

```

[...]
Possible enum values:
- `"Exact"` matches the URL path exactly and with case sensitivity.
- `"ImplementationSpecific"` matching is up to the IngressClass.
[...]
identically to Prefix or Exact path types.
- `"Prefix"` matches based on a URL path prefix split by '/'.

```

```

## Anpassen
nano ingress.yml

```

```

## So sieht das korrigierte .yml file aus.
## Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: apps-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - host: app1.dein-training.de
      http:
        paths:
          - path: /apple
            pathType: Prefix # <- EINGEFUEGT
            backend:
              service:
                name: apple-service
                port:
                  number: 80
          - path: /banana
            pathType: Prefix # <- EINGEFUEGT
            backend:
              service:
                name: banana-service
                port:
                  number: 80

```

```

kubectl apply -f .
kubectl get ingress

```

#### Reference

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

#### Old Version: Find the problem

```
## Hints

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

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

## now we can adjust our config
```

#### Solution

```
## in kubernetes 1.22.2 - ingress.yml needs to be modified like so.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # old version useClassName instead
    # otherwise it does not know, which controller to use
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "app12.lab.t3isp.de"
    http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

#### Achtung: Ingress mit Helm - annotations

#### Permanente Weiterleitung mit Ingress

#### Example

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

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

```

service:
  name: http-svc
  port:
    number: 80
  path: /source
  pathType: ImplementationSpecific

```

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

```

/etc/hosts
45.23.12.12 web.training.local

```

```

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

```

### Umbauen zu google ;o)

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

#### Refs:

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

### ConfigMap Example

#### 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

```

```

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

```

#### Schrit 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 MariaDB - Example

#### Schritt 1: configmap

```
cd
mkdir -p manifests
cd manifests
mkdir cftest
cd cftest
nano 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:latest
          envFrom:
            - configMapRef:
                name: mariadb-configmap

```

```
kubectl apply -f .
```

## Testing

```

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

```

## Important Sidenote

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

## Configmap MariaDB my.cnf

### configmap zu fuß

```

cd
mkdir -p manifests
cd manifests
mkdir mariadb-vol-cm
cd mariadb-vol-cm

```

```
nano 01-mariadb-config2.yaml
```

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: example-configmap
data:
  # als Wertepaare
  database: mongodb
  my.cnf: |
    [mysqld]
    slow_query_log = 1
    innodb_buffer_pool_size = 1G

```

```
kubectl apply -f .
```

```
nano 02-deployment.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:latest
      envFrom:
        - configMapRef:
            name: mariadb-configmap

  volumeMounts:
    - name: example-configmap-volume
      mountPath: /etc/mysql

  volumes:
    - name: example-configmap-volume
      configMap:
        name: example-configmap

kubectl apply -f .
kubectl exec -it deployment/mariadb-deployment -- bash

cd /etc/mysql
ls -la
cat my.cnf

```

### Secret MariaDB - Example

#### Schritt 1: secret

```

cd
mkdir -p manifests
cd manifests
mkdir secrettest
cd secrettest

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

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

```

#### Schritt 2: Deployment

```

nano 02-deploy.yml

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

```

```

kubectl apply -f .

```

#### Testing

```

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

```

### Important Sidenode

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

## Kubernetes Praxis (Teil 2) - API Objekte

### 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.
```

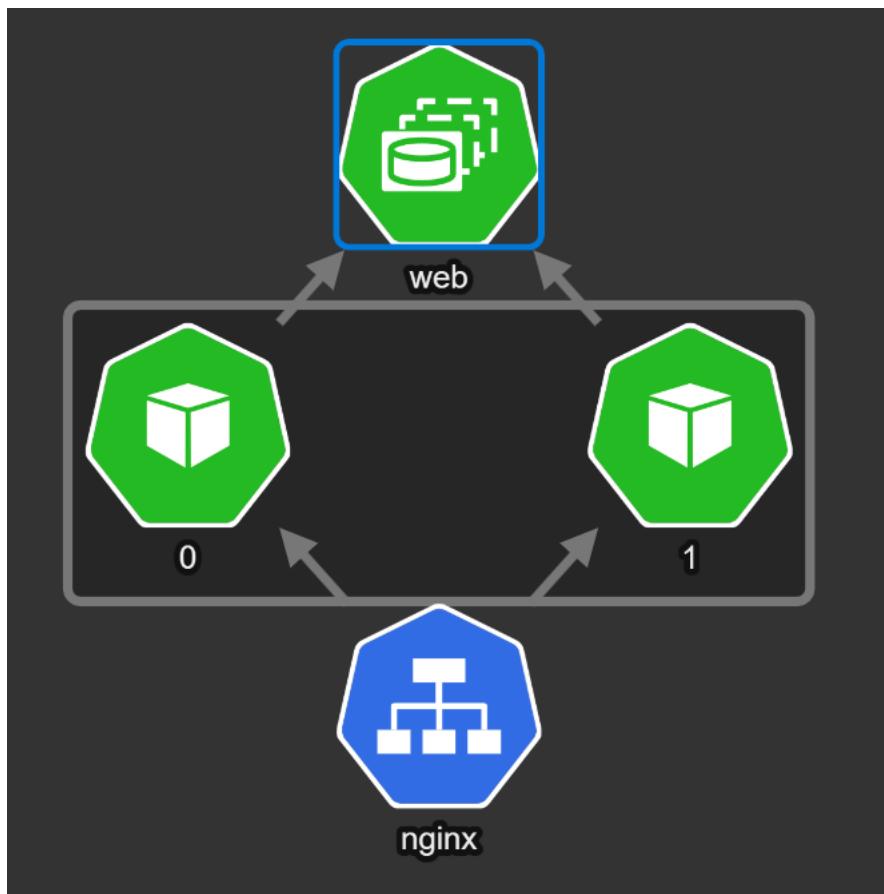
- VolumeClaimTemplate (In addition if the pod is scaled the copies will have their own storage)
  - Plus: When you delete it, it gets recreated and claims the same persistentVolumeClaim

```
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
```

## Übung Statefulsets

### Overview



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

```
nano sts.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  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 .
```

### Auflösung Namen.

```
ping web-0.nginx
ping web-1.nginx
```

### Test der Auflösung

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

```
/ # ping web-0.nginx
/ # ping web-1.nginx
/ # exit
```

### Referenz

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

## Kubernetes Praxis (Teil 3)

### Using private registry

#### Create config.json with login docker

##### Step 1: On docker machine

```
docker login
```

```
cat ~/.docker/config.json
```

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "ZG9....."
    }
}
```

```
copy to kubectl client
```

##### Step 2: on kubectl-client machine

```
cd
mkdir -p manifests
cd manifests
mkdir docker
cd docker
```

```
## Für die Teilnehmer
cp /tmp/config.json .
```

```
kubectl create secret generic docker-credentials \
--from-file=.dockerconfigjson=config.json \
--type=kubernetes.io/dockerconfigjson \
--dry-run=client -o yaml > secret.yaml
```

```
cat secret.yaml
```

```
## umbenennen, damit es nicht von kubectl apply gelesen wird
mv config.json config.json.bkup
```

```
kubectl apply -f .
```

```
nano pod.yaml
```

```

apiVersion: v1
kind: Pod
metadata:
  name: dockertrainereu-pod
spec:
  containers:
    - name: private-container
      image: dockertrainereu/jm-hello-web
  imagePullSecrets:
    - name: docker-credentials

```

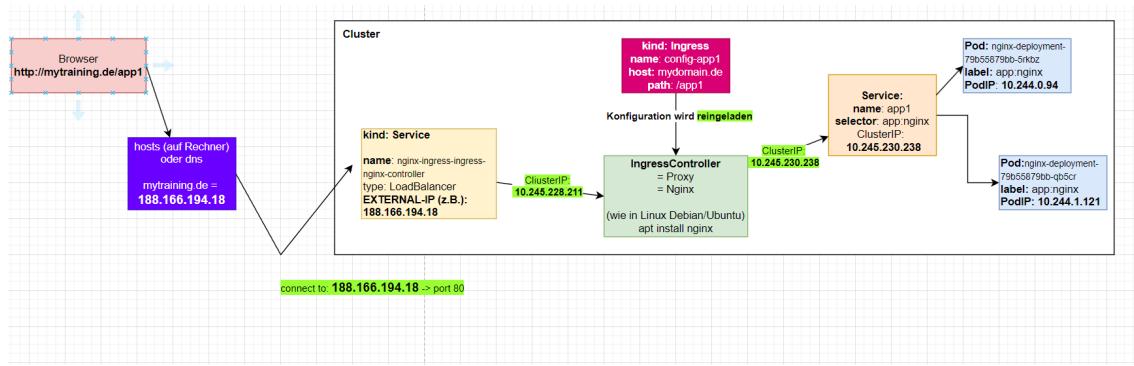
```

kubectl apply -f .
kubectl get pods dockertrainereu-pod
kubectl describe pods dockertrainereu-pod

```

## Kubernetes Ingress

### Ingress Controller on Detail



## ServiceMesh

### Why a ServiceMesh ?

#### What is a service mesh ?

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

#### Advantages / Features

1. Observability & monitoring
2. Traffic management
3. Resilience & Reliability
4. Security
5. Service Discovery

#### Observability & monitoring

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

#### Traffic management

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

#### Resilience & Reliability

- By handling retries, timeouts, and failures,
  - service mesh contributes to the overall stability and resilience of services
  - reducing the impact of potential disruptions.

#### Security

- Service mesh enforces security policies, and handles authentication, authorization, and encryption
  - ensuring secure communication between services and eventually, strengthening the overall security posture of the application.

#### Service Discovery

- With service discovery features, service mesh can simplify the process of locating and routing services dynamically
- adapting to system changes seamlessly. This enables easier management and interaction between services.

#### Overall benefits

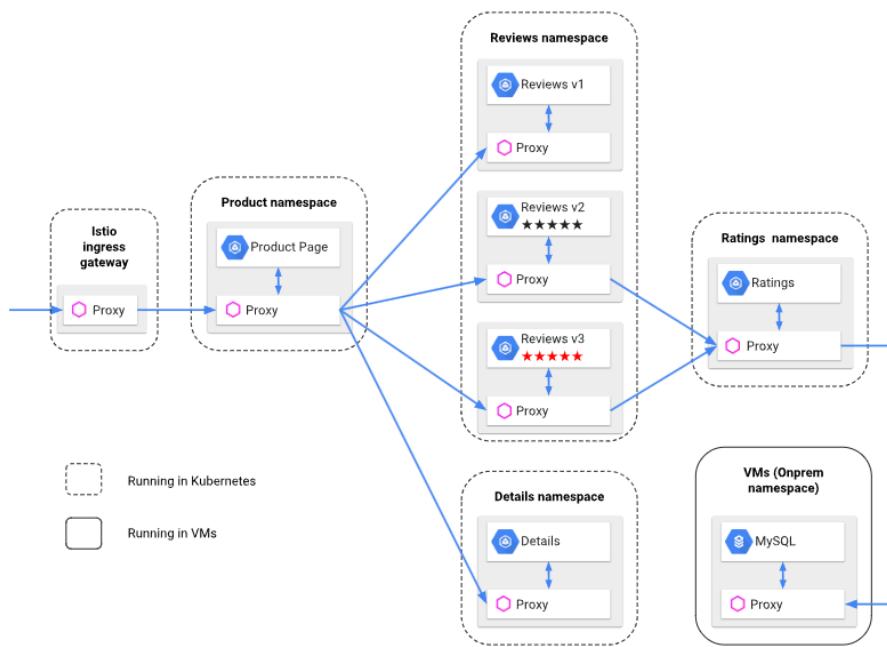
Microservices communication:

Adopting a service mesh can simplify the implementation of a microservices architecture by abstracting away infrastructure complexities.

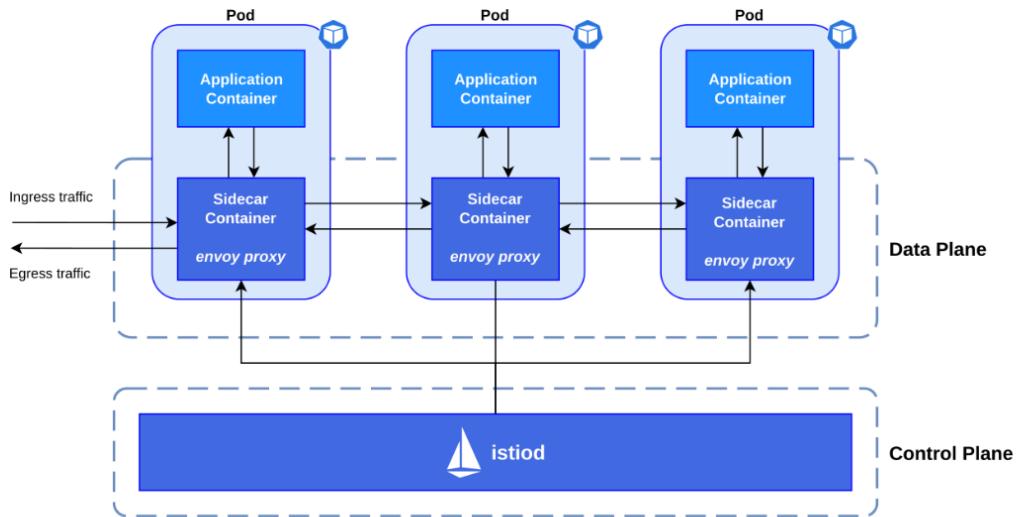
It provides a standardized approach to manage and orchestrate communication within the microservices ecosystem.

How does a ServiceMesh work? (example istio)

## Overview



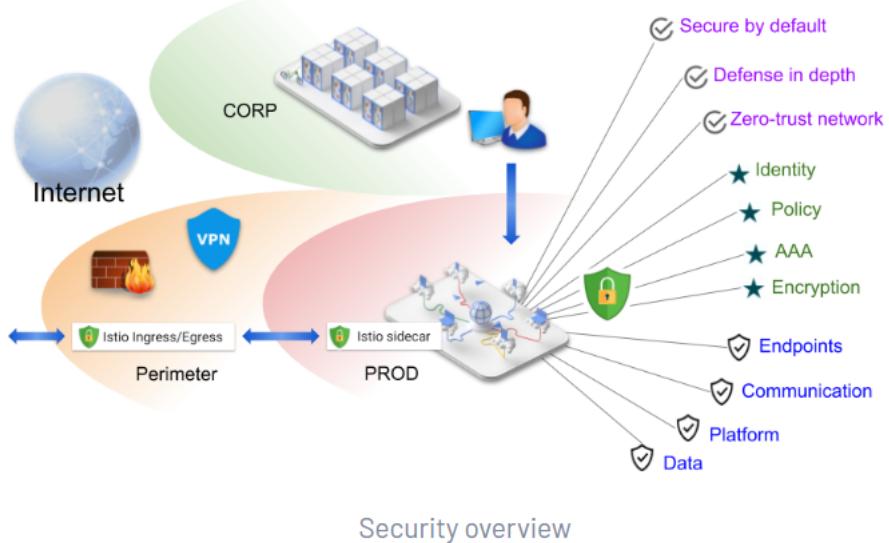
## Istio control plane and data plane



• Source: kubebyexample.com

### istio security features

#### Overview



### Security needs of microservices

- To defend against man-in-the-middle attacks, they need traffic encryption.
- To provide flexible service access control, they need mutual TLS and fine-grained access policies.
- To determine who did what at what time, they need auditing tools.

## Implementation of security

The Istio security features provide strong identity, powerful policy, transparent TLS encryption, and authentication, authorization and audit (AAA) tools to protect your services and data. The goals of Istio security are:

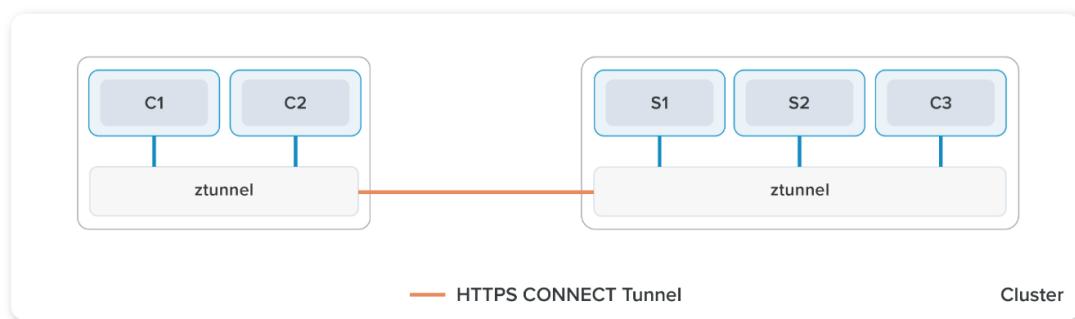
- Security by default: no changes needed to application code and infrastructure
- Defense in depth: integrate with existing security systems to provide multiple layers of defense
- Zero-trust network: build security solutions on distrusted networks

## istio-service mesh - ambient mode

### Light: Only Layer 4 per node (ztunnel)

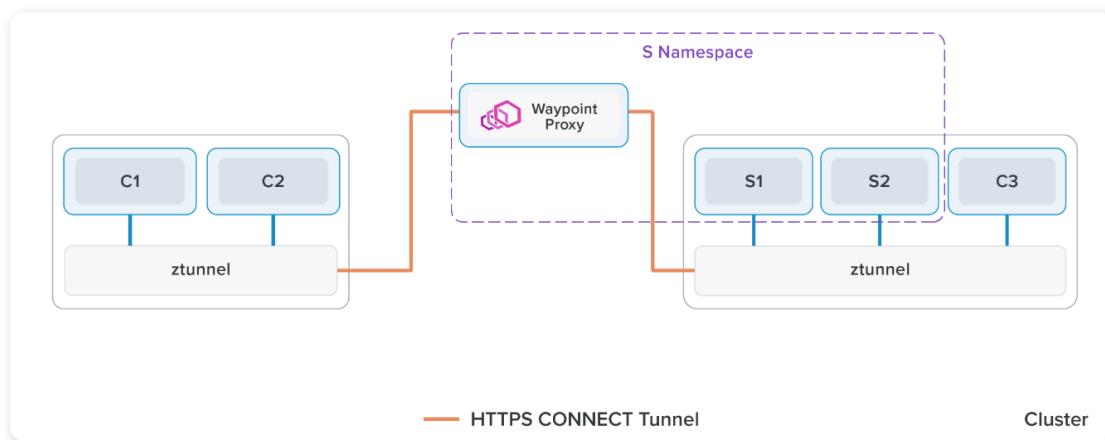
- No sidecar (envoy-proxy) per Pod, but one ztunnel agent per Node (Layer 4)
- Enables security features (mtls, traffic encryption)

Like so:



### Full fledged: Layer 4 (ztunnel) per Node & Layer 7 per Namespace

- One waypoint - proxy is rolled out per Namespace, which connects to the ztunnel agents



When additional features are needed, ambient mesh deploys waypoint proxies, which ztunnels connect through for policy enforcement

### Features in "fully-fledged" - ambient - mode

Application deployment use case	Ambient mode configuration
Zero Trust networking via mutual-TLS, encrypted and tunneled data transport of client application traffic, L4 authorization, L4 telemetry	ztunnel only (default)
As above, plus advanced Istio traffic management features (including L7 authorization, telemetry and VirtualService routing)	ztunnel and waypoint proxies

#### Advantages:

- Less overhead
- One can start step-by-step moving towards a mesh (Layer 4 firstly and if wanted Layer 7 for specific namespaces)
- Old pattern: sidecar and new pattern: ambient can live side by side.

#### istio-traffic-management

- <https://istio.io/latest/docs/concepts/traffic-management/#retries>

#### Performance comparison - baseline,sidecar,ambient

- <https://livewriter.io/blog/2024/06/06/comparison-of-service-meshes-part-two/>
- <https://github.com/livewriter-ops/poc-servicemesh2024/blob/main/docs/test-report-istio.md>

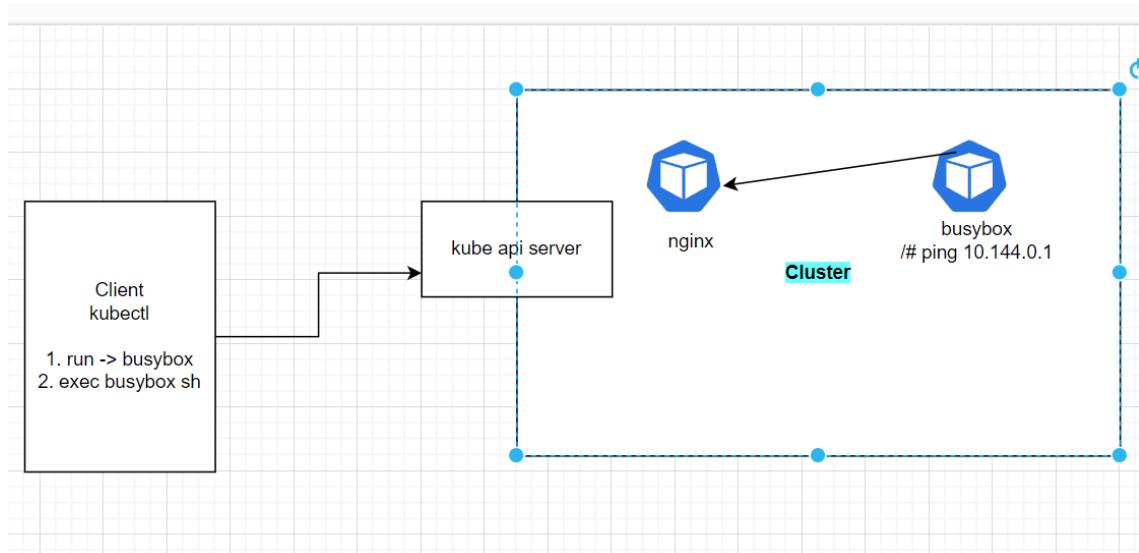
## Kubernetes (Debugging)

### Netzwerkverbindung zu 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

#### Example test connection

##### Schritt 1: Die IP des Pods rausuchen, den ich den testen möchte

kubectl get pods -o wide

##### Schritt 2: Verbindung test

```
## -O -> Output (grosses O (buchstabe))
kubectl run podtest --rm -ti --image busybox
/ # wget -O - http://10.244.0.99
/ # ping -c 4 10.244.0.99
/ # exit
```

## Kubernetes Netzwerk

### DNS - Resolution - Services

#### Generic

- DNS - Names for Services are automatically created from the Service -> Name
  - the namespace the service is in
  - fixed subdomain svc.cluster.local

#### Example:

```
## Service Name: myservice
## Being in Namespace: app
## Results in

myservice
myservice.app
myservice.app.svc.cluster.local
```

### Walkthrough

```
kubectl run podtest --rm -ti --image busybox
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 Scaling

### Autoscaling Pods/Deployments

#### Example: newest version with autoscaling/v2 used to be hpa/v1

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello
          image: k8s.gcr.io/hpa-example
          resources:
            requests:
              cpu: 100m
```

```
---
kind: Service
apiVersion: v1
metadata:
  name: hello
spec:
  selector:
    app: hello
  ports:
    - port: 80
      targetPort: 80
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hello
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hello
  minReplicas: 2
  maxReplicas: 20
  metrics:
    - type: Resource
      resource:
        name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

- <https://docs.digitalocean.com/tutorials/cluster-autoscaling-ca-hpa/>

## Reference

- <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/#autoscaling-on-more-specific-metrics>
- <https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054>
- Alternative: <https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler>

## Kubernetes Tips & Tricks

### Oomkiller and maxReadySeconds for safe migration to new pods

#### What to achieve ?

1. Deploy a working version
2. Deploy a new version that fails with OOM-Killer (but we can be sure pod from old replicaset still works)

#### Step 1: Create deployment that works

```
mkdir -p manifests
cd manifests
mkdir -p stress
cd stress
```

```
kubectl create ns mem-example
nano deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-memtest
  namespace: mem-example
spec:
  ## minReadySeconds: 120
  selector:
    matchLabels:
      app: memtest
  replicas: 3
  template:
    metadata:
      labels:
        app: memtest
    spec:
      containers:
        - name: memory-demo-ctr
          image: polinux/stress
          resources:
            requests:
```

```

    memory: "100Mi"
  limits:
    memory: "200Mi"
  command: ["stress"]
  args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]

kubectl apply -f .
kubectl -n mem-example get all
kubectl get pods

```

### Schritt 2: Now with oom - killer version

- More memory than available
- So new pods fail (normally, old pods would be terminated)
- But: Due to minReadySeconds (each pod must at least 120seconds before state is switched to ready)
  - System will wait / and old pods are still available

```
Change line --args from: --vm-bytes 150M to --vm-bytes 250M
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-memtest
  namespace: mem-example
spec:
  ## minReadySeconds: 120
  selector:
    matchLabels:
      app: memtest
  replicas: 3
  template:
    metadata:
      labels:
        app: memtest
    spec:
      containers:
        - name: memory-demo-ctr
          image: polinux/stress
          resources:
            requests:
              memory: "100Mi"
            limits:
              memory: "200Mi"
          command: ["stress"]
          args: ["--vm", "1", "--vm-bytes", "250M", "--vm-hang", "1"]

```

```

kubectl -n mem-example get all
kubectl apply -f .
kubectl -n mem-example get all
## after a while we will see the new pod being in mode OOMKiller
kubectl -n mem-example get pods -w

```

### Pod-Netzwerk debuggen durch weiteren Pod der daneben liegt kubectl debug

#### Andere Anwendungsfälle

- Tools die nicht auf dem Pod installiert sind, benötigen

#### Walkthrough

```

kubectl run my-nginx --image=nginx
## Daneben einen pod starten, der auf das gleiche Netzwerk zugreift (d.h. die gleiche IP-Adresse hat)
kubectl debug -it my-nginx --image=busybox

## Kann ich rauspingen ?
ping www.google.de

```

#### Reference:

- [https://kubernetes.io/docs/reference/kubectl/generated/kubectl\\_debug/](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_debug/)
- <https://kubernetes.io/docs/tasks/debug/debug-application/debug-running-pod/>

### Aus pod mit curl api-server abfragen

#### Step 1: Prepare Permissions

```
kubectl create ns app
```

```

cd
mkdir -p manifests
cd manifests
mkdir curltest
cd curltest

nano 01-clusterrole.yml

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: service-reader
rules:
- apiGroups: [""]
  resources: ["services", "endpoints"]
  verbs: ["get", "list"]

kubectl -n app apply -f .

## Einfacher hack, wir verwenden den default-service - account
kubectl -n app create rolebinding api-service-explorer:default --clusterrole service-reader --serviceaccount app:default

```

### Schritt 2: curlimage/curl starten

```
kubectl run -it --rm curltest --image=curlimages/curl -- sh
```

### Schritt 3: in curl - shell

```

cd /var/run/secrets/kubernetes.io/serviceaccount
TOKEN=$(cat token)
env | grep KUBERNETES_SERVICE
curl https://$KUBERNETES_SERVICE_HOST/openapi/v2 --header "Authorization: Bearer $TOKEN" --cacert ca.crt
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/ --header "Authorization: Bearer $TOKEN" --cacert ca.crt

## Now look into one of the services
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt

## We will get the pod ip's from the endpoints
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/endpoints/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt

```

### Reference

- <https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c>

## Kubernetes - Monitoring

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

#### Warum ? Was macht er ?

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

```

kubectl top pods
kubectl top nodes

```

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

### Walktrough

```

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

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

```

### Kubernetes

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

## Prometheus Überblick

### What does it do ?

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

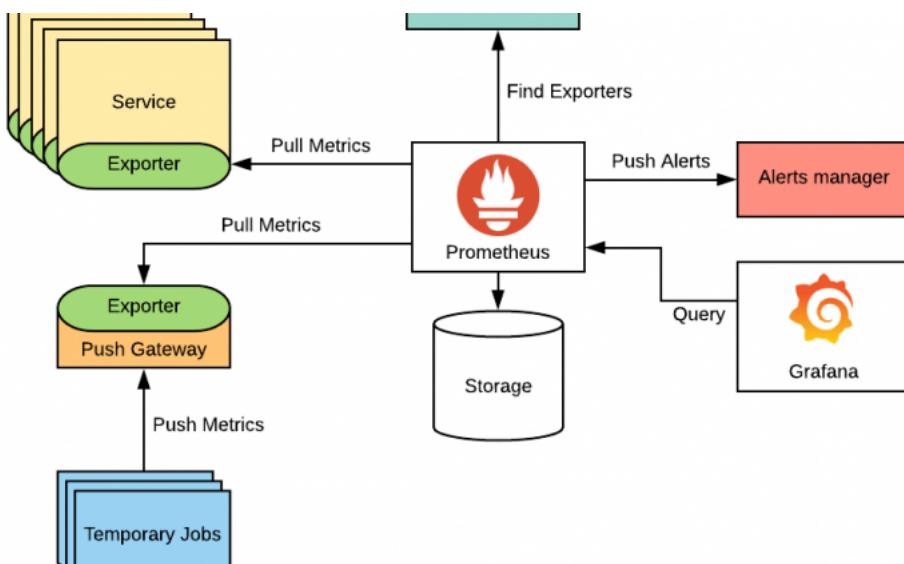
### Technical

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

### What are time series ?

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

### Komponenten von Prometheus



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

### Prometheus Server

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

### Grafana ?

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

### Prometheus Kubernetes Stack installieren

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

### Step 1: Prepare values-file

```

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

vi values.yml

fullnameOverride: prometheus

alertmanager:
  fullnameOverride: alertmanager

grafana:
  fullnameOverride: grafana

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

prometheus-node-exporter:
  fullnameOverride: node-exporter

```

## Step 2: Install with helm

```

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

```

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

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

```

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

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

```

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

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

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

```

## in browser
http://localhost:9090

```

## Step 4: Connect to the grafana from the outside world

### Step 4.1: Start proxy to connect

```

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

```

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

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

## References:

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

## Prometheus - Services scrapen die keine Endpunkte für Prometheus haben

### Prerequisites

- prometheus setup with helm

### Step 1: Setup

```

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install my-prometheus-blackbox-exporter prometheus-community/prometheus-blackbox-exporter --version 8.17.0 --namespace
monitoring --create-namespace

```

## Step 2: Find SVC

```
kubectl -n monitoring get svc | grep blackbox

my-prometheus-blackbox-exporter  ClusterIP  10.245.183.66  <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

```
## From within curlimages/curl pod
curl http://my-prometheus-blackbox-exporter.monitoring:9115/probe?target=apple-service.app&module=http_2xx
```

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

```
prometheus:
  prometheusSpec:
    additionalScrapeConfigs:
      - job_name: "blackbox-microservices"
        metrics_path: /probe
        params:
          module: [http_2xx]
        # Autodiscovery through kube-api-server
        # https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config
        kubernetes_sd_configs:
          - role: service
        relabel_configs:
          # Example relabel to probe only some services that have "example.io/should_be_probed = true" annotation
          - source_labels: [__meta_kubernetes_service_annotation_example_io_should_be_probed]
            action: keep
            regex: true
          - source_labels: [__address__]
            target_label: __param_target
            target_label: __address__
            replacement: my-prometheus-blackbox-exporter:9115
          - source_labels: [__param_target]
            target_label: instance
            action: labelmap
            regex: __meta_kubernetes_service_label_(.+)
          - source_labels: [__meta_kubernetes_namespace]
            target_label: app
          - source_labels: [__meta_kubernetes_service_name]
            target_label: kubernetes_service_name
```

## Step 6: Test with relabeler

- <https://relabeler.promlabs.com>

## Step 7: Scrapeconfig einbauen

```
## von kube-prometheus-grafana in values und upgraden
helm upgrade prometheus prometheus-community/kube-prometheus-stack -f values.yml --namespace monitoring --create-namespace --
version 61.3.1
```

## Step 8: annotation in service einfügen

```
kind: Service
apiVersion: v1
metadata:
  name: apple-service
  annotations:
    example.io/should_be_probed: "true"
spec:
```

```
selector:
  app: apple
ports:
  - protocol: TCP
    port: 80
    targetPort: 5678 # Default port for image
```

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

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

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

#### Step 10: Unter <http://64.227.125.201:30090/targets?search=> gucken

- .. ob das funktioniert

#### Step 11: Hauptseite (status code 200)

- Metrik angekommen `?
- [http://64.227.125.201:30090/graph?g0.expr=probe\\_http\\_status\\_code&g0.tab=1&g0.display\\_mode=lines&g0.show\\_exemplars=0&g0.range\\_input=1h](http://64.227.125.201:30090/graph?g0.expr=probe_http_status_code&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h)

#### Step 12: pod vom service stoppen

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apple-deployment
spec:
  selector:
    matchLabels:
      app: apple
  replicas: 8
  template:
    metadata:
      labels:
        app: apple
    spec:
      containers:
        - name: apple-app
          image: hashicorp/http-echo
          args:
            - "-text=apple-<dein-name>"
```

```
kubectl apply -f apple.yml # (deployment)
```

#### Step 13: status\_code 0

- Metrik angekommen `?
- [http://64.227.125.201:30090/graph?g0.expr=probe\\_http\\_status\\_code&g0.tab=1&g0.display\\_mode=lines&g0.show\\_exemplars=0&g0.range\\_input=1h](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)

## Helm

### Helm internals / secret a.s.o

### Kubernetes with ServerLess

#### Kubernetes with Serverless

##### What is serverless ?

- Serverless-Computing ist eine Möglichkeit, Code auszuführen, ohne sich um Server zu kümmern.
- Mit am beliebtesten ist: FaaS (Function as a service)
- Serverless ist eine Kategorie von Cloud-Computing, die eine Plattform für die Entwicklung und Bereitstellung von Anwendungen bereitstellt, ohne sich um die zugrunde liegende Infrastruktur zu sorgen.

##### Products to install on Kubernetes to use the Serverless concept.

###### OpenFaas

###### Example

###### OpenWhisk

###### Example

- <https://github.com/appvia/serverless-kube/tree/master/openwhisk>

###### Kubeless

- Described as the most Kubernetes-native. Easy installation and simple architecture using Custom Resource Definitions. No scale-to-zero functionality at the time of writing.

###### Fission

###### Example

- <https://github.com/fission/fission#getting-started>

#### Knative

##### Example

- <https://github.com/appvia/serverless-kube/tree/master/knative#template-for-autoscaling-flask-app-using-knative>

#### fn

##### Example

- <https://github.com/appvia/serverless-kube/tree/master/fn#example-app-for-deployment-to-fn>

#### Reference

- <https://www.appvia.io/blog/serverless-on-kubernetes>

## Literatur / Documentation / Information (Microservices)

### Sam Newman - Microservices

- <https://www.amazon.de/Building-Microservices-English-Sam-Newman-ebook/dp/B09B5L4NVT/>

### Sam Newman - Vom Monolithen zu Microservices

- <https://www.amazon.de/Vom-Monolithen-Microservices-bestehende-umzugsstellen/dp/3960091400/>

### Microservices.io Patterns

- <https://microservices.io>

#### BFF

- <https://blog.bitsrc.io/bff-pattern-backend-for-frontend-an-introduction-e4fa965128bf>

### Microservices Up and Running

- [https://www.amazon.de/Kubernetes-Running-Dive-Future-Infrastructure/dp/109811020X/ref=sr\\_1\\_1](https://www.amazon.de/Kubernetes-Running-Dive-Future-Infrastructure/dp/109811020X/ref=sr_1_1)

## gitlab ci/cd

### Einfaches Beispielscript

#### Edit .gitlab-ci.yml with pipeline editor

```
## with this content
stages:           # List of stages for jobs, and their order of execution
- build
- test
- deploy

build-job:        # This job runs in the build stage, which runs first.
stage: build
script:
- ls -la
- pwd
```

## Praxis Microservices ohne Docker und Kubernetes

### Schritt 1: Nodejs aufsetzen

#### Prerequisites

```
## We have set an ubuntu 22.04 LTS server with a user 11trainingdo
## on digitalocean
## When you set up a server you can use a script under advanced options

## or: we have another server at hand
```

```
## just in case curl is not installed
apt install -y curl
cd
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
source ~/.bashrc
## Install latest stable version
nvm install lts/hydrogen

## test if it is installed
nvm list
```

```
## test node
node
>.exit
```

## Schritt 2: Codebasis bereitstellen

### Cloning

```
cd
## Wird in den Ordner blog geklont
git clone https://github.com/jmetzger/training-microservices-docker-kubernetes-uebungen blog
```

## Schritt 3: Posts - Service testen

### Start service posts

```
cd
cd posts
npm start
## output will listening to :4000
```

### Find external ip

```
## in most cases look for eth0 or enp0s3 or enp0s8
## Example: 134.122.93.133
ip -br a
```

### Open postmon web interface or download it.

```
## get it from here or use web there
https://www.postman.com/
```

```
## Step 1: Send a post

- Url: POST http://<ip-of-server>/posts
- Set: Header -> Content-Type -> to application/json # this is need otherwise json is not detected from server
- Body: set a title
- Body: set dropdown (at the outer right to -> JSON)
{
  "title": "my title is great"
}

Send ;o)
```

You should get a response with an id and a title

```
## Step 2: Get all posts
- Url: GET http://<ip-of-server>/posts
- Change http-request to send
- Set: Header -> Content-Type -> to application/json # this is need otherwise json is not detected from server
```

## Docker-Installation

### Installation Docker unter Ubuntu mit snap

```
sudo su -
snap install docker

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

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

systemctl stop snap.docker.dockerd.service
systemctl status snap.docker.dockerd.service
systemctl start snap.docker.dockerd.service

## wird der docker-dienst beim nächsten reboot oder starten des Server gestartet ?
systemctl is-enabled snap.docker.dockerd.service
```

### Installation Docker unter SLES 15

### Walkthrough

```
sudo zypper search -v docker*
sudo zypper install docker

## Dem Nutzer /z.B. Nutzer kurs die Gruppe docker hinzufügen
## damit auch dieser den Docker-daemon verwenden darf
sudo groupadd docker
sudo usermod -aG docker $USER

### Unter SLES werden Dienste nicht automatisch aktiviert und gestartet !!!
## Service für start nach Boot aktivieren
newgrp docker
sudo systemctl enable docker.service
## Docker dienst starten
sudo systemctl start docker.service
```

## Ausführlich mit Ausgaben

```
.....  
[fertig]  
  
Überprüfung auf Dateikonflikte läuft:  
.....  
[fertig]  
(1/7) Installieren: libshadetectcoll1-1.0.3-2.18.x86_64  
.....  
[fertig]  
(2/7) Installieren: catatonit-0.1.5-3.3.2.x86_64  
.....  
[fertig]  
(3/7) Installieren: runc-1.1.4-150000.33.4.x86_64  
.....  
[fertig]  
(4/7) Installieren: containerd-1.6.6-150000.73.2.x86_64  
.....  
[fertig]  
(5/7) Installieren: git-core-2.35.3-150300.10.15.1.x86_64  
.....  
[fertig]  
Updating /etc/sysconfig/docker ...  
(6/7) Installieren: docker-20.10.17_ce-150000.166.1.x86_64  
.....  
[fertig]  
(7/7) Installieren: docker-bash-completion-20.10.17_ce-150000.166.1.noarch  
.....  
[fertig]  
  
sudo groupadd docker  
sudo usermod -aG docker $USER  
// logout  
  
newgrp docker  
sudo systemctl enable docker.service  
sudo systemctl start docker.service
```

## Docker-Grundlagen

### Übersicht Architektur



### Was ist ein Container ?

- vereint in sich Software
- Bibliotheken
- Tools
- Konfigurationsdateien
- keinen eigenen Kernel
- gut zum Ausführen von Anwendungen auf verschiedenen Umgebungen
  
- Container sind entkoppelt
- Container sind voneinander unabhängig
- Können über wohldefinierte Kommunikationskanäle untereinander Informationen austauschen
  
- Durch Entkopplung von Containern:
  - o Unverträglichkeiten von Bibliotheken, Tools oder Datenbank können umgangen werden, wenn diese von den Applikationen in unterschiedlichen Versionen benötigt werden.

### Was sind container images

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

### Container vs. Virtuelle Maschine

```
VM's virtualisieren Hardware  
Container virtualisieren Betriebssystem
```

### Was ist ein Dockerfile

### What is it ?

- 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

## Example

```
## syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

## Docker-Befehle

### Logs anschauen - docker logs - mit Beispiel nginx

#### Allgemein

```
## Erstmal nginx starten und container-id wird ausgegeben
docker run -d nginx:1.22.1
a234
docker logs a234 # a234 sind die ersten 4 Ziffern der Container ID
```

#### Laufende Log-Ausgabe

```
docker logs -f a234
## Abbrechen CTRL + c
```

### Docker container/image stoppen/löschen

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

## Get nur, wenn der Container nicht mehr läuft
docker rm ubuntu-container

## oder alternative
docker rm -f ubuntu-container

## image löschen
docker rmi ubuntu:xenial

## falls Container noch vorhanden aber nicht laufend
docker rmi -f ubuntu:xenial
```

### Docker containerliste anzeigen

```
## besser
docker container ls
## Alle Container, auch die, die beendet worden sind
docker container ls -a

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

### Docker nicht verwendete Images/Container löschen

```
docker system prune
## Löscht möglicherweise nicht alles

## d.h. danach nochmal prüfen ob noch images da sind
docker images
## und händisch löschen
docker rmi <image-name>
```

### Docker container analysieren

```
docker run -t -d --name mein_container ubuntu:latest
docker inspect mein_container # mein_container = container name
```

#### Docker container in den Vordergrund bringen - attach

##### docker attach - walkthrough

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

docker attach 1a4d

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

#### interaktiven Prozess nicht beenden (statt exit)

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

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

#### Reference:

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

#### Aufräumen - container und images löschen

##### Alle nicht verwendeten container und images löschen

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

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

## Alle nicht benötigten Daten löschen
docker system prune
```

#### Nginx mit portfreigabe laufen lassen

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

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

#### Docker container/image stoppen/löschen

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

## Get nur, wenn der Container nicht mehr läuft
docker rm ubuntu-container

## oder alternative
docker rm -f ubuntu-container

## image löschen
docker rmi ubuntu:xenial

## falls Container noch vorhanden aber nicht laufend
docker rmi -f ubuntu:xenial
```

#### Docker containerliste anzeigen

```
## besser
docker container ls
## Alle Container, auch die, die beendet worden sind
docker container ls -a

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

## Dockerfile - Examples

### Ubuntu mit hello world

#### Simple Version

##### Schritt 1:

```
cd
mkdir hello-world
cd hello-world
```

##### Schritt 2:

```
## nano Dockerfile
FROM ubuntu:22.04

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

##### Schritt 3:

```
nano hello.sh

#!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done
```

##### Schritt 4:

```
## dockertrainereu/<dein-name>-hello-docker .
## Beispiel
##
docker build -t dockertrainereu/<dein-name>-hello-docker .

docker images
docker run dockertrainereu/<dein-name>-hello-docker
```

##### Schritt 5:

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

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

## und wir schauen online, ob wir das dort finden
```

### Ubuntu mit ping

```
mkdir myubuntu
cd myubuntu/
```

```

nano Dockerfile

FROM ubuntu:22.04
RUN apt-get update; apt-get install -y inetutils-ping
## CMD ["/bin/bash"]

docker build -t fullubuntu:1.0 .
docker images

## 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"]

docker build -t myubuntu:1.0 .
docker images

## -t wird benötigt, damit bash WEITER im Hintergrund im läuft.
## auch mit -d (ohne -t) wird die bash ausgeführt, aber "das Terminal" dann direkt beendet
## -> container läuft dann nicht mehr
docker run -d -t --name container-ubuntu myubuntu:1.0
docker container ls

## docker inspect to find out ip of other container
## 172.17.0.3
docker inspect container-ubuntu | grep -i ipaddress

## Zweiten Container starten um 1. anzupingen
docker run -d -t --name container-ubuntu2 myubuntu:1.0

## Ersten Container -> 2. anpingen
docker exec -it container-ubuntu2 bash
## Jeder container hat eine eigene IP
ping 172.17.0.3

```

## Nginx mit content aus html-ordner

### Schritt 1: Simple Example

```

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

Text, den du rein haben möchtest

cd ..
vi Dockerfile

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

## nameskürzel z.B. jml
docker build -t nginx-test .
docker images

```

### Schritt 2: docker laufen lassen

```

## und direkt aus der Registry wieder runterladen
docker run --name hello-web -p 8080:80 nginx-test

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

```

```
curl http://localhost:8080
```

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

### Ubuntu mit hello world

#### Simple Version

##### Schritt 1:

```
cd  
mkdir hello-world  
cd hello-world
```

##### Schritt 2:

```
## nano Dockerfile  
FROM ubuntu:22.04  
  
COPY hello.sh .  
RUN chmod u+x hello.sh  
CMD ["/hello.sh"]
```

##### Schritt 3:

```
nano hello.sh
```

```
#!/bin/bash  
let i=0  
  
while true  
do  
    let i=i+1  
    echo $i:hello-docker  
    sleep 5  
done
```

##### Schritt 4:

```
## dockertrainereu/<dein-name>-hello-docker .  
## Beispiel  
##  
docker build -t dockertrainereu/<dein-name>-hello-docker .  
  
docker images  
docker run dockertrainereu/<dein-name>-hello-docker
```

##### Schritt 5:

```
docker login  
user: dockertrainereu  
pass: --bekommt ihr vom trainer--  
  
## docker push dockertrainereu/<dein-name>-hello-docker  
## z.B.  
docker push dockertrainereu/jm-hello-docker  
  
## und wir schauen online, ob wir das dort finden
```

### Nginx mit content aus html-ordner

#### Schritt 1: Simple Example

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

```
Text, den du rein haben möchtest
```

```
cd ..  
vi Dockerfile  
  
FROM nginx:latest  
COPY html /usr/share/nginx/html  
  
## nameskürzel z.B. jm1  
docker build -t nginx-test .  
docker images
```

### Schritt 2: docker laufen lassen

```
## und direkt aus der Registry wieder runterladen  
docker run --name hello-web -p 8080:80 -d nginx-test  
  
## laufenden Container anzeigen lassen  
docker container ls  
## oder alt: deprecated  
docker ps  
  
curl http://localhost:8080  
  
##  
docker rm -f hello-web
```

## Docker-Netzwerk

### Netzwerk

#### Übersicht

```
3 Typen  
o none  
o bridge (Standard-Netzwerk)  
o host  
  
### Additionally possible to install  
o overlay (needed for multi-node)
```

#### Kommandos

```
## Netzwerk anzeigen  
docker network ls  
  
## bridge netzwerk anschauen  
## Zeigt auch ip der docker container an  
docker inspect bridge  
  
## im container sehen wir es auch  
docker inspect ubuntu-container
```

#### Eigenes Netz erstellen

```
docker network create -d bridge test_net  
docker network ls  
  
docker container run -d --name nginx --network test_net nginx  
docker container run -d --name nginx_no_net --network none nginx  
  
docker network inspect none  
docker network inspect test_net  
  
docker inspect nginx  
docker inspect nginx_no_net
```

#### Netzwerk rausnehmen / hinzufügen

```
docker network disconnect none nginx_no_net  
docker network connect test_net nginx_no_net  
  
### Das Löschen von Netzwerken ist erst möglich, wenn es keine Endpoints
```

```
### d.h. container die das Netzwerk verwenden
docker network rm test_net
```

## Docker-Container Examples

### 2 Container mit Netzwerk anpingen

```
clear
docker run --name dockerserver1 -dit ubuntu
docker run --name dockerserver2 -dit ubuntu
docker network ls
docker network inspect bridge
## dockerserver1 - 172.17.0.2
## dockerserver2 - 172.17.0.3
docker container ls
docker exec -it dockerserver1 bash
## in container
apt update; apt install -y iputils-ping
ping 172.17.0.3
```

### Container mit eigenem privatem Netz erstellen

```
clear
## use bridge as type
## docker network create -d bridge test_net
## by bridge is default
docker network create test_net
docker network ls
docker network inspect test_net

## Container mit netzwerk starten
docker container run -d --name nginx1 --network test_net nginx
docker network inspect test_net

## Weiteres Netzwerk (bridged) erstellen
docker network create demo_net
docker network connect demo_net nginx1

## Analyse
docker network inspect demo_net
docker inspect nginx1

## Verbindung lösen
docker network disconnect demo_net nginx1

## Schauen, wir das Netz jetzt aussieht
docker network inspect demo_net
```

## Docker-Daten persistent machen / Shared Volumes

### Überblick

#### Overview

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

#### Disadvantages

```
stored only on one node
Does not work well in cluster
```

#### Alternative for cluster

```
glusterfs
cephfs
nfs

## Stichwort
ReadWriteMany
```

## Volumes

### Storage volumes verwalten

```
docker volume ls
docker volume create test-vol
docker volume ls
docker volume inspect test-vol
```

### Storage volumes in container einhängen

```
## Schritt 1
docker run -it --name container-test-vol --mount target=/test_data,source=test-vol ubuntu bash
1234ad# touch /test_data/README
exit
## stops container
docker container ls -a

## Schritt 2:
## create new container and check for /test_data/README
docker run -it --name=container-test-vol2 --mount target=/test_data,source=test-vol ubuntu bash
ab45# ls -la /test_data/README
```

### Storage volume löschen

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

## bind-mounts

```
## andere Verzeichnis als das Heimatverzeichnis von root funktionieren aktuell nicht mit
## snap install docker
## wg. des Confinements
docker run -d -it --name devtest --mount type=bind,source=/root,target=/app nginx:latest
docker exec -it devtest bash
/# cd /app
```

## Docker - Dokumentation

### Vulnerability Scanner with docker

- <https://docs.docker.com/engine/scan/#prerequisites>

### Vulnerability Scanner mit snyk

- <https://snyk.io/plans/>

### Parent/Base - Image bauen für Docker

- <https://docs.docker.com/develop/develop-images/baseimages/>

## Docker - Projekt blog

### posts in blog dockerisieren

#### Walkthrough

```
sudo -i
cd
git clone https://github.com/jmetzger/training-microservices-docker-kubernetes-uebungen blog

## Create Dockerfile in posts
cd posts
nano Dockerfile

FROM node:16-alpine

WORKDIR /app
## COPY package.json ./
COPY ./ ./ 
RUN npm install

CMD ["npm", "start"]

nano .dockerignore
```

```
package-lock.json

docker build -t dockertrainereu/<namenskuerzel>-posts:0.0.1 .
docker run -d -p 4000:4000 --name posts dockertrainereu/<namenkuerzel>-posts:0.0.1
docker logs posts
```

## Docker Compose (backlog)

### yaml-format

```
## Kommentare

## Listen
- rot
- gruen
- blau

## Mappings
Version: 3.7

## Mappings können auch Listen enthalten
expose:
- "3000"
- "8000"

## Verschachtelte Mappings
build:
  context: .
  labels:
    label1: "bunt"
    label2: "hell"
```

### docker-compose und replicas

#### Beispiel

```
version: "3.9"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
      configs:
        - my_config
        - my_other_config
  configs:
    my_config:
      file: ./my_config.txt
    my_other_config:
      external: true
```

#### Ref:

- <https://docs.docker.com/compose/compose-file/compose-file-v3/>

### Example with Wordpress / Nginx / Mariadb - wrong

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

version: "3.7"

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

```

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

```

```

## now start the system
docker compose up -d
## we can do some test if db is reachable
docker exec -it wp-wordpress-1 bash
## within shell do
apt update
apt-get install -y telnet
## this should work
telnet database 3306

## and we even have logs
docker compose logs

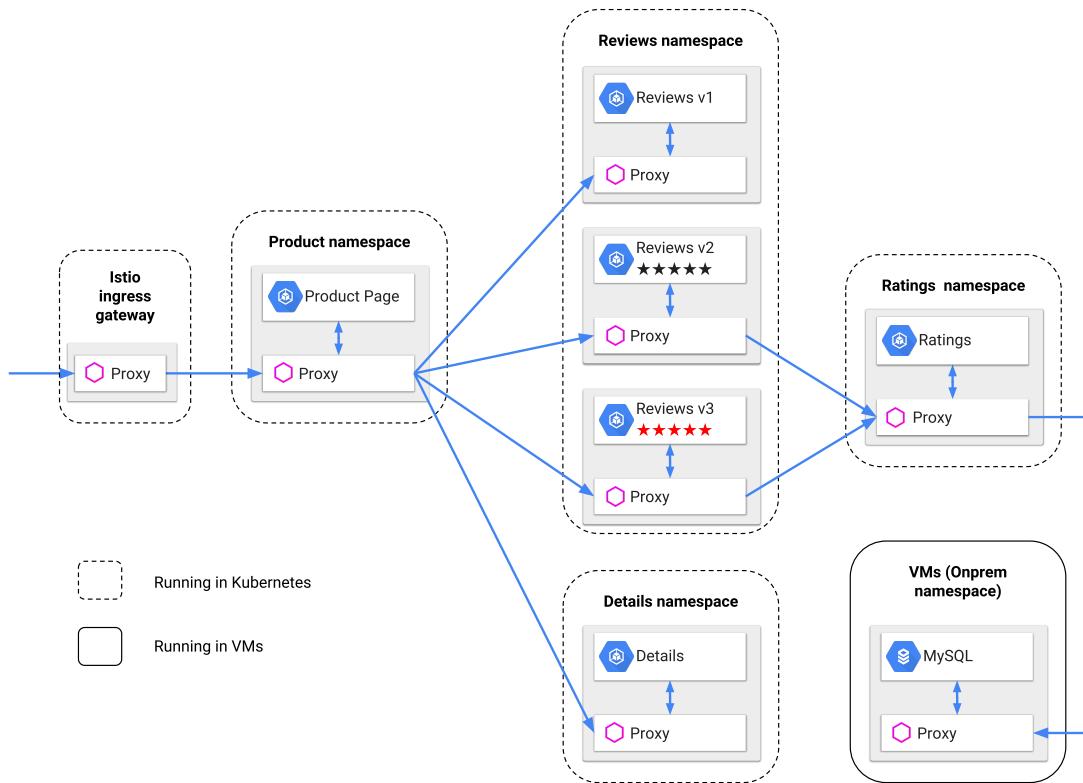
##
docker compose down

```

## Kubernetes Netzwerk

**Mesh / istio**

**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/>

## pubsub+ for graph kafka

- <https://solace.com/blog/how-a-financial-services-giant-cleaned-up-their-kafka-with-pubsub-event-portal/>

## Kubernetes GUI

### OpenLens

#### Why not use lens ?

Interestingly the source of lens is opensource, but the binary is freeware.  
As of 2023 there is a license key needed

But OpenLens - client (binary) is opensource, so we use that one

#### Install openlens (Windows)

```
## The easiest way to do so on windows is:
```

```
winget install openlens
```

```
After that it will be available from Windows (not cmd.exe)
```

## References

- <https://github.com/MuhammedKalkan/OpenLens>

## Kubernetes - microk8s (Installation und Management)

### Ingress controller in microk8s aktivieren

#### Aktivieren

```
microk8s enable ingress
```

#### Referenz

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

## Helm (Kubernetes Paketmanager)

### Helm Grundlagen

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

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

## Test
kubectl cluster-info
```

### Helm Warum ?

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

### Helm Example

#### 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)

```
## 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
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
```

### 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

```
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 4: Install Wordpress with values and auth

```
## mkdir helm-mysql
## cd helm-mysql
## vi values.yml
```

```

persistence:
  enabled: false

wordpressUsername: admin
wordpressPassword: password
mariadb:
  primary:
    persistence:
      enabled: false
  auth:
    rootPassword: secretpassword

helm uninstall my-wordpress
helm install my-wordpress bitnami/wordpress -f values

```

#### Referenced

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

### Kubernetes - RBAC

#### Nutzer einrichten microk8s ab kubernetes 1.25

##### Enable RBAC in microk8s

```

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

```

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

```

cd
mkdir -p manifests/rbac
cd manifests/rbac

```

##### Mini-Schritt 1: Definition für Nutzer

```

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

```

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

##### Mini-Schritt 1.5: Secret erstellen

- From Kubernetes 1.25 tokens are not created automatically when creating a service account (sa)
- You have to create them manually with annotation attached
- <https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token>

```

## vi secret.yml
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: trainingtoken
  annotations:
    kubernetes.io/service-account.name: training

```

```
kubectl apply -f .
```

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

```

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

## vi pods-clusterrole.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-clusterrole
rules:

```

```
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

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

#### Mini-Schritt 3: Die 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
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pods-clusterrole
subjects:
- kind: ServiceAccount
  name: training
  namespace: default
```

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

#### Mini-Schritt 4: Testen (klappt der Zugang)

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

### Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

#### Mini-Schritt 1: kubeconfig setzen

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

## extract name of the token from here

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

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

#### Mini-Schritt 2:

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

#### Mini-Schritt 3: Zurück zum alten Default-Context

```
kubectl config get-contexts

  CURRENT  NAME          CLUSTER        AUTHINFO      NAMESPACE
*        microk8s        microk8s-cluster  admin2
*        training-ctx    microk8s-cluster  training2

kubectl config use-context microk8s
```

#### Refs:

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

#### Ref: Create Service Account Token

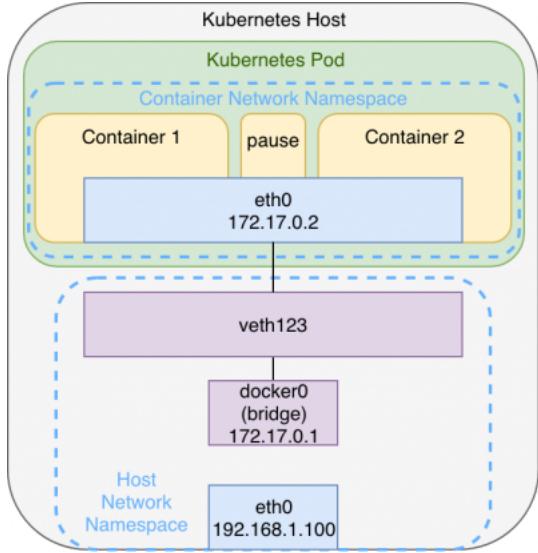
- <https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token>

## Kubernetes - Netzwerk (CNI's) / Mesh

### Netzwerk Interna

#### Network Namespace for each pod

## Overview



## General

- Each pod will have its own network namespace
  - with routing, network devices
- Connection to default namespace to host is done through veth - Link to bridge on host network
  - similar like on docker to docker0

Each container is connected to the bridge via a veth-pair. This interface pair functions like a virtual point-to-point ethernet connection and connects the network namespaces of the containers with the network namespace of the host

- Every container is in the same Network Namespace, so they can communicate through localhost
  - Example with hashicorp/http-echo container 1 and busybox container 2 ?

## Pod-To-Pod Communication (across nodes)

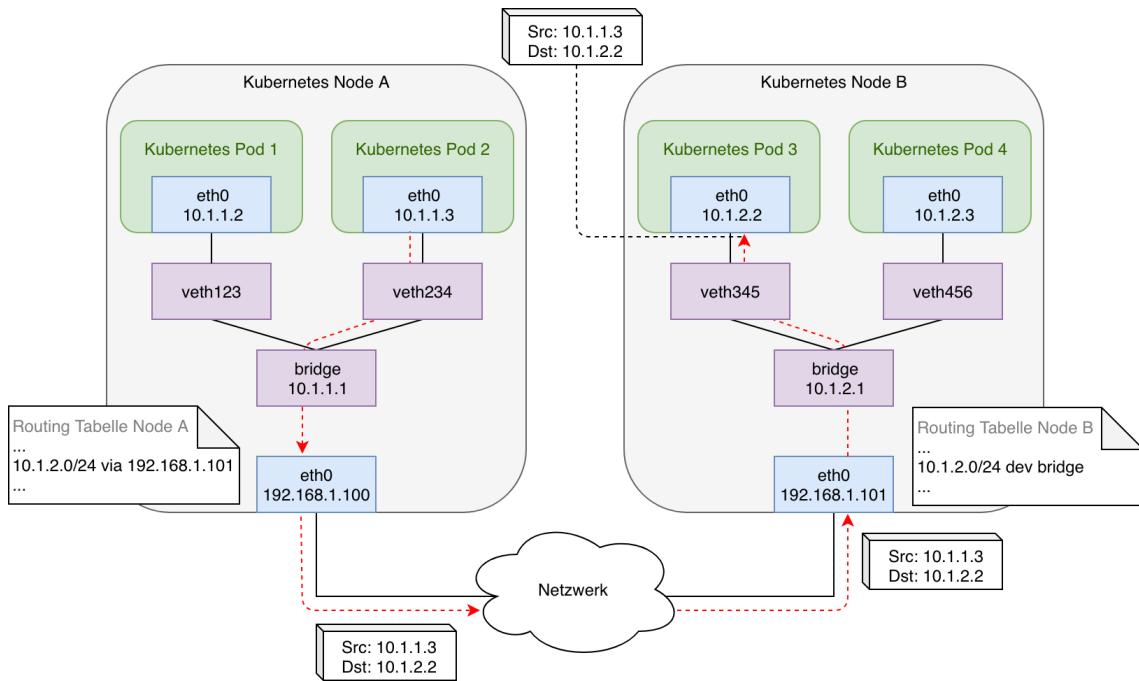
### Prerequisites

- pods on a single node as well as pods on a topological remote can establish communication at all times
- Each pod receives a unique IP address, valid anywhere in the cluster. Kubernetes requires this address to not be subject to network address translation (NAT)
- Pods on the same node through virtual bridge (see image above)

### General (what needs to be done) - and could be done manually

- local bridge networks of all nodes need to be connected
- there needs to be an IPAM (IP-Address Management) so addresses are only used once
- The need to be routes so, that each bridge can communicate with the bridge on the other network
- Plus: There needs to be a rule for incoming network
- Also: A tunnel needs to be set up to the outside world.

### General - Pod-to-Pod Communication (across nodes) - what would need to be done



#### General - Pod-to-Pod Communication (side-note)

- This could of cause be done manually, but it is too complex
- So Kubernetes has created an Interface, which is well defined
  - The interface is called CNI (common network interface)
  - Functionally is achieved through Network Plugin (which use this interface)
    - e.g. calico / cilium / weave net / flannel

#### CNI

- CNI only handles network connectivity of container and the cleanup of allocated resources (i.e. IP addresses) after containers have been deleted (garbage collection) and therefore is lightweight and quite easy to implement.
- There are some basic libraries within CNI which do some basic stuff.

#### Hidden Pause Container

##### What is for ?

- Holds the network - namespace for the pod
- Gets started first and falls asleep later
- Will still be there, when the other containers die

```
cd
mkdir -p manifests
cd manifests
mkdir pausetest
cd pausetest
nano 01-nginx.yml
```

```
## vi nginx-static.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pausetest
  labels:
    webserver: nginx:1.21
spec:
  containers:
  - name: web
    image: nginx
```

```
kubectl apply -f .
```

```
ctr -n k8s.io c list | grep pause
```

#### References

- [https://www.inovex.de/de/blog/kubernetes-networking.part-1-en/](https://www.inovex.de/de/blog/kubernetes-networking-part-1-en/)
- <https://www.inovex.de/de/blog/kubernetes-networking-2-calico-cilium-weavenet/>

## Übersicht Netzwerke

### CNI

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

### Docker - Container oder andere

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

### Welche gibt es ?

- Flannel
- Canal
- Calico
- Cilium
- Weave Net

### Flannel

#### Overlay - Netzwerk

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

#### Vorteile

- Guter einfacher Einstieg
- redziert 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>

### Weave Net

- Ähnlich calico
- Verwendet overlay netzwerk
- Sehr stabil bzgl IPV4/IPV6 (Dual Stack)
- Sehr grosses Feature-Set
- mit das älteste Plugin

### microk8s Vergleich

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

```
snap.microk8s.daemon-flanneld
Flanneld is a CNI which gives a subnet to each host for use with container runtimes.

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

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

### Calico - nginx example NetworkPolicy

```
## Schritt 1:
kubectl create ns policy-demo
kubectl create deployment --namespace=policy-demo nginx --image=nginx
```

```
kubectl expose --namespace=policy-demo deployment nginx:1.21 --port=80
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

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

## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: policy-demo
spec:
  podSelector:
    matchLabels: {}
EOF
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

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

## Schritt 3: Zugriff erlauben von pods mit dem Label run=access
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
  namespace: policy-demo
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
    - from:
        - podSelector:
            matchLabels:
              run: access
EOF
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
## pod hat durch run -> access automatisch das label run:access zugewiesen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

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

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

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

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

#### Ref:

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

#### Beispiele Ingress NetworkPolicy

#### Links

- <https://github.com/ahmetb/kubernetes-network-policy-recipes>
- <https://k8s-examples.container-solutions.com/examples/NetworkPolicy/NetworkPolicy.html>

#### Example with http (Cilium !!)

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L7 policy to restrict access to specific HTTP call"
metadata:
  name: "rule1"
spec:
```

```

endpointSelector:
  matchLabels:
    type: l7-test
ingress:
- fromEndpoints:
  - matchLabels:
    org: client-pod
  toPorts:
- ports:
  - port: "8080"
    protocol: TCP
  rules:
    http:
      - method: "GET"
        path: "/discount"

```

### Downside egress

- No valid api for anything other than IP's and/or Ports
- If you want more, you have to use CNI-Plugin specific, e.g.

### Example egress with ip's

```

## Allow traffic of all pods having the label role:app
## egress only to a specific ip and port
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
      cidr: 10.10.0.0/16
    ports:
    - protocol: TCP
      port: 5432

```

### Example Advanced Egress (cni-plugin specific)

#### Cilium

```

apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: "fqdn-pprof"
  namespace: msp
spec:
  endpointSelector:
    matchLabels:
      app: pprof
  egress:
  - toFQDNs:
    - matchPattern: '*.baidu.com'
  - toPorts:
    - ports:
      - port: "53"
        protocol: ANY
    rules:
      dns:
        - matchPattern: '*'

```

#### Calico

- Only Calico enterprise
  - Calico Enterprise extends Calico's policy model so that domain names (FQDN / DNS) can be used to allow access from a pod or set of pods (via label selector) to external resources outside of your cluster.
  - <https://projectcalico.docs.tigera.io/security/calico-enterprise/egress-access-controls>

### Using istio as mesh (e.g. with cilium/calico )

#### Installation of sidecar in calico

- <https://projectcalico.docs.tigera.io/getting-started/kubernetes/hardway/istio-integration>

## Example

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  policyTypes:
    - Egress
  egress:
    - to:
        - ipBlock:
            cidr: 10.10.0.0/16
      ports:
        - protocol: TCP
          port: 5432
```

## Kubernetes Ports/Protokolle

- <https://kubernetes.io/docs/reference/networking/ports-and-protocols/>

## IPV4/IPV6 Dualstack

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

## kubectl

### Start pod (container with run && examples)

#### Example (that does work)

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

## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx:1.21

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

#### Example (that does not work)

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

#### Ref:

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

## Bash completion for kubectl

### Walkthrough

```
## Eventuell, wenn bash-completion nicht installiert ist.
apt install bash-completion
source /usr/share/bash-completion/bash_completion
## is it installed properly
type _init_completion

## als root
## 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
```

```
## alternativ z.B. als Benutzer kurs
## activate for all users
sudo -i
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
exit

## verifizieren - neue login shell
su - kurs

## 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 Spickzettel

##### Allgemein

```
## Zeige Information über das Cluster
kubectl cluster-info

## Welche api-resources gibt es ?
kubectl api-resources

## Hilfe zu object und eigenschaften bekommen
kubectl explain pod
kubectl explain pod.metadata
kubectl explain pod.metadata.name
```

##### Arbeiten mit manifesten

```
kubectl apply -f nginx-replicaset.yml
## Wie ist aktuell die hinterlegte config im system
kubectl get -o yaml -f nginx-replicaset.yml

## Änderung in nginx-replicaset.yml z.B. replicas: 4
## dry-run - was wird geändert
kubectl diff -f nginx-replicaset.yml

## anwenden
kubectl apply -f nginx-replicaset.yml

## Alle Objekte aus manifest löschen
kubectl delete -f nginx-replicaset.yml
```

##### Ausgabeformate

```
## Ausgabe kann in verschiedenen Formaten erfolgen
kubectl get pods -o wide # weitere informationen
## im json format
kubectl get pods -o json

## gilt natürlich auch für andere kommandos
kubectl get deploy -o json
kubectl get deploy -o yaml

## Eigenschaft auslesen
kubectl get pods nginx-deployment-74676ff58f-fxcjv -o jsonpath='{.metadata.ownerReferences[0].name}'
```

##### Zu den Pods

```
## Start einen pod // BESSER: direkt manifest verwenden
## kubectl run podname image=imagename
kubectl run nginx image=nginx

## Pods anzeigen
kubectl get pods
```

```

kubectl get pod
## Format weitere Information
kubectl get pod -o wide
## Zeige labels der Pods
kubectl get pods --show-labels

## Zeige pods mit einem bestimmten label
kubectl get pods -l app=nginx

## Status eines Pods anzeigen
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

```

### Zu den Pods (Logs)

```

## log eines pods anzeigen
kubectl logs podname

## Logs aller pods im Deployment
## Wichtig Option --prefix
kubectl logs --prefix deploy/web-nginx

```

### Arbeiten mit namespaces

```

## Welche namespaces auf dem System
kubectl get ns
kubectl get namespaces
## Standardmäßig wird immer der default namespace verwendet
## wenn man kommandos aufruft
kubectl get deployments

## Möchte ich z.B. deployment vom kube-system (installation) aufrufen,
## kann ich den namespace angeben
kubectl get deployments --namespace=kube-system
kubectl get deployments -n kube-system

## wir wollen unseren default namespace ändern
kubectl config set-context --current --namespace <dein-namespace>

```

### Arbeiten mit der Config (lokal)

```

## namespace jochen als default setzen
kubectl config set-context --current --namespace jochen
## config anzeigen
kubectl config view

```

### Referenz

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

### Tipps&Tricks zu Deployment - Rollout

#### Warum

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

#### Beispiele

```

## Deployment nochmal durchführen
## z.B. nach kubectl uncordon n12.training.local
kubectl rollout restart deploy nginx-deployment

## Rollout rückgängig machen
kubectl rollout undo deploy nginx-deployment

```

### Kubernetes - Shared Volumes

#### Shared Volumes with nfs

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

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

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

exportfs -av
```

#### On all nodes (needed for production)

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

#### On all nodes (only for testing)

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

## for testing
mkdir /mnt/nfs
## 10.135.0.18 is our nfs-server
mount -t nfs 10.135.0.18:/var/nfs /mnt/nfs
ls -la /mnt/nfs
umount /mnt/nfs
```

#### Persistent Storage-Step 1: Setup PersistentVolume in cluster

```
cd
cd manifests
mkdir -p nfs
cd nfs
nano 01-pv.yml

apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln<nr>
  labels:
    volume: nfs-data-volume-tln<nr>
spec:
  capacity:
    # storage size
    storage: 1Gi
  accessModes:
    # ReadWriteMany (RW from multi nodes), ReadWriteOnce (RW from a node), ReadOnlyMany (R from multi nodes)
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    # retain even if pods terminate
    Retain
  nfs:
    # NFS server's definition
    path: /var/nfs/tln<nr>/nginx
    server: 10.135.0.18
    readOnly: false
    storageClassName: ""

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

#### Persistent Storage-Step 2: Create Persistent Volume Claim

```
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
kubectl get pvc

```

### Persistent Storage-Step 3: Deployment

```

## deployment including mount
## vi 03-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # tells deployment to run 4 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
      volumeMounts:
      - name: nfsvol
        mountPath: "/usr/share/nginx/html"
    volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: pv-nfs-claim-tln<tln>

```

```

kubectl apply -f 03-deploy.yml

```

### Persistent Storage Step 4: service

```

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

```

```

kubectl apply -f 04-service.yml

```

### Persistent Storage Step 5: write data and test

```

## connect to the container and add index.html - data
kubectl exec -it deploy/nginx-deployment -- bash

```

```

## in container
echo "hello dear friend" > /usr/share/nginx/html/index.html
exit

## now try to connect
kubectl get svc

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

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

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

```

#### Persistent Storage Step 6: retest after redeployment

```

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

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

```

### Kubernetes - Wartung / Debugging

#### kubectl drain/uncordon

```

## Achtung, bitte keine pods verwenden, dies können "ge"-drained (ausgetrocknet) werden
kubectl drain <node-name>
z.B.

## Daemonsets ignorieren, da diese nicht gelöscht werden
kubectl drain n17 --ignore-daemonsets

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

## Wenn fertig bin:
kubectl uncordon n17

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

```

#### Alte manifeste konvertieren mit convert plugin

##### What is about?

- Plugins needs to be installed 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>

##### Curl from pod api-server

## Step 1: Prepare Permissions

```
kubectl create ns app

cd
mkdir -p manifests
cd manifests
mkdir curltest
cd curltest

nano 01-clusterrole.yml

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: service-reader
rules:
- apiGroups: [""]
  resources: ["services", "endpoints"]
  verbs: ["get", "list"]

kubectl -n app apply -f .

## Einfacher hack, wir verwenden den default-service - account
kubectl -n app create rolebinding api-service-explorer:default --clusterrole service-reader --serviceaccount app:default
```

## Schritt 2: curlimage/curl starten

```
kubectl run -it --rm curltest --image=curlimages/curl -- sh
```

## Schritt 3: in curl - shell

```
cd /var/run/secrets/kubernetes.io/serviceaccount
TOKEN=$(cat token)
env | grep KUBERNETES_SERVICE
curl https://$KUBERNETES_SERVICE_HOST/openapi/v2 --header "Authorization: Bearer $TOKEN" --cacert ca.crt
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/ --header "Authorization: Bearer $TOKEN" --cacert ca.crt

## Now look into one of the services
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt

## We will get the pod ip's from the endpoints
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/endpoints/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt
```

## Reference

- <https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c>

## Kubernetes - Tips & Tricks

### Kubernetes Debuggen ClusterIP/PodIP

#### Situation

- Kein Zugriff auf die Nodes, zum Testen von Verbindungen zu Pods und Services über die PodIP/ClusterIP

#### Lösung

```
## Wir starten eine Busybox und fragen per wget und port ab
## busytester ist der name
## long version
kubectl run -it --rm --image=busybox busytester
## wget <pod-ip-des-zieles>
## exit

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

## Debugging pods

### How ?

- Which pod is in charge

2. Problems when starting: kubectl describe po mypod
3. Problems while running: kubectl logs mypod

## Taints und Tolerations

### Taints

```
Taints schliessen auf einer Node alle Pods aus, die nicht bestimmte taints 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 Advanced

### Curl api-server kubernetes aus pod heraus

#### Step 1: Prepare Permissions

```

kubectl create ns app

cd
mkdir -p manifests
cd manifests
mkdir curltest
cd curltest

nano 01-clusterrole.yml

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: service-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["services", "endpoints"]
  verbs: ["get", "list"]

kubectl -n app apply -f .

```

```
## Einfacher hack, wir verwenden den default-service - account
kubectl -n app create rolebinding api-service-explorer:default --clusterrole service-reader --serviceaccount app:default
```

### Schritt 2: curlimage/curl starten

```
kubectl run -it --rm curltest --image=curlimages/curl -- sh
```

### Schritt 3: in curl - shell

```
cd /var/run/secrets/kubernetes.io/serviceaccount
TOKEN=$(cat token)
env | grep KUBERNETES_SERVICE
curl https://$KUBERNETES_SERVICE_HOST/openapi/v2 --header "Authorization: Bearer $TOKEN" --cacert ca.crt
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/ --header "Authorization: Bearer $TOKEN" --cacert ca.crt

## Now look into one of the services
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/services/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt

## We will get the pod ip's from the endpoints
curl https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/app/endpoints/apple-service/ --header "Authorization: Bearer $TOKEN" --
cacert ca.crt
```

### Reference

- <https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cf398c>

## Kubernetes - Documentation

### Documentation zu microk8s plugins/addons

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

### Shared Volumes - Welche gibt es ?

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

## Kubernetes - Hardening

### Kubernetes Tipps Hardening

### PSA (Pod Security Admission)

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

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

### Möglichkeiten in Pods und Containern

```
## für die Pods
kubectl explain pod.spec.securityContext
kubectl explain pod.spec.containers.securityContext
```

### Example (seccomp / security context)

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

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

  containers:
    - name: test-container
      image: hashicorp/http-echo:0.2.3
      args:
```

```
- "-text=just made some syscalls!"  
securityContext:  
  allowPrivilegeEscalation: false
```

### SecurityContext (auf Pod Ebene)

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

### NetworkPolicy

```
## Firewall Kubernetes
```

### Kubernetes Security Admission Controller Example

#### 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

```
mkdir -p manifests  
cd manifests  
mkdir psa  
cd psa  
nano 01-ns.yml
```

```
## Schritt 1: Namespace anlegen  
## vi 01-ns.yml  
  
apiVersion: v1  
kind: Namespace  
metadata:  
  name: test-ns<tln>  
labels:  
  pod-security.kubernetes.io/enforce: baseline  
  pod-security.kubernetes.io/audit: restricted  
  pod-security.kubernetes.io/warn: restricted
```

```
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-nginx.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-ns<tln>
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-nginx.yml
kubectl apply -f 02-nginx.yml
kubectl -n test-ns<tln> get pods
kubectl -n test-ns<tln> describe pods nginx

```

#### Praktisches Beispiel für Version ab 1.2.23 -Lösung - Container als NICHT-Root laufen lassen

- Wir müssen ein image, dass auch als NICHT-Root laufen 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-ns<tln>
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<tln> get pods

```

## Kubernetes Interna / Misc.

### OCI,Container,Images Standards

#### Schritt 1:

```

cd
mkdir bautest
cd bautest

```

#### Schritt 2:

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

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

#### Schritt 3:

```
mkdir myubuntu
cd myubuntu

nano hello.sh

#!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done

## nano Dockerfile
FROM ubuntu:latest
RUN apt-get update; apt-get install -y inetutils-ping
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]
```

#### Schritt 4:

```
cd ..
## wichtig, im docker-compose - Ordner seiend
##pwd
##~/bautest
docker-compose up -d
## wird image gebaut und container gestartet

## Bei Veränderung vom Dockerfile, muss man den Parameter --build mit angeben
docker-compose up -d --build
```

#### Geolocation Kubernetes Cluster

- <https://learnk8s.io/bite-sized/connecting-multiple-kubernetes-clusters>

#### Documentation

##### Good Doku with Tasks

- <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

#### Docker-Container Examples

##### 2 Container mit Netzwerk anpingen

```
clear
docker run --name dockerserver1 -dit ubuntu
docker run --name dockerserver2 -dit ubuntu
docker network ls
docker network inspect bridge
## dockerserver1 - 172.17.0.2
## dockerserver2 - 172.17.0.3
docker container ls
docker exec -it dockerserver1 bash
## im container
apt update; apt install -y iputils-ping
ping 172.17.0.3
```

##### Container mit eigenem privatem Netz erstellen

```
clear
## use bridge as type
## docker network create -d bridge test_net
```

```
## by bridge is default
docker network create test_net
docker network ls
docker network inspect test_net

## Container mit netzwerk starten
docker container run -d --name nginx1 --network test_net nginx
docker network inspect test_net

## Weiteres Netzwerk (bridged) erstellen
docker network create demo_net
docker network connect demo_net nginx1

## Analyse
docker network inspect demo_net
docker inspect nginx1

## Verbindung lösen
docker network disconnect demo_net nginx1

## Schauen, wir das Netz jetzt aussieht
docker network inspect demo_net
```

## Docker-Netzwerk

### Netzwerk

#### Übersicht

```
3 Typen
o none
o bridge (Standard-Netzwerk)
o host

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

#### Kommandos

```
## Netzwerk anzeigen
docker network ls

## bridge netzwerk anschauen
## Zeigt auch ip der docker container an
docker inspect bridge

## im container sehen wir es auch
docker inspect ubuntu-container
```

#### Eigenes Netz erstellen

```
docker network create -d bridge test_net
docker network ls

docker container run -d --name nginx --network test_net nginx
docker container run -d --name nginx_no_net --network none nginx

docker network inspect none
docker network inspect test_net

docker inspect nginx
docker inspect nginx_no_net
```

#### Netzwerk rausnehmen / hinzufügen

```
docker network disconnect none nginx_no_net
docker network connect test_net nginx_no_net

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

## Docker Security

Scanning docker image with docker scan/snyx

**ACHTUNG\_ Deprecated - USE Docker Scout instead (only Docker Desktop ?)**

#### Prerequisites

```
## install docker plugin in some cases
## Ubuntu
apt install docker-scan-plugin
```

You need to be logged in on docker hub with docker login  
(with your account credentials)

#### Example

```
## Snyk (docker scan)
docker help scan
docker scan --json --accept-license dockertrainereu/jm-hello-docker > result.json
```

## Docker Compose

#### yaml-format

```
## Kommentare

## Listen
- rot
- gruen
- blau

## Mappings
Version: 3.7

## Mappings können auch Listen enthalten
expose:
- "3000"
- "8000"

## Verschachtelte Mappings
build:
  context: .
  labels:
    label1: "bunt"
    label2: "hell"
```

#### Example with Ubuntu and Dockerfile

##### Schritt 1:

```
cd
mkdir bautest
cd bautest
```

##### Schritt 2:

```
## nano docker-compose.yml
services:
  myubuntu:
    build: ./myubuntu
    restart: always
```

##### Schritt 3:

```
mkdir myubuntu
cd myubuntu
```

```
nano hello.sh
```

```
#!/bin/bash
let i=0

while true
do
  let i=i+1
```

```

echo $i:hello-docker
sleep 5
done

nano Dockerfile

FROM ubuntu:24.04
RUN apt-get update && apt-get install -y inetutils-ping
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]

```

#### Schritt 4:

```

cd ../
## wichtig, im docker-compose - Ordner seind
##pwd
##~/bautest
docker compose up -d
## wird image gebaut und container gestartet

## Bei Veränderung vom Dockerfile, muss man den Parameter --build mitangeben
docker compose up -d --build

```

#### Schritt 5: Logs anzeigen

```

docker logs bautest-myubuntu-1
docker compose logs

```

#### docker-compose und replicas

##### Beispiel

```

version: "3.9"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
      configs:
        - my_config
        - my_other_config
    configs:
      my_config:
        file: ./my_config.txt
      my_other_config:
        external: true

```

##### Ref:

- <https://docs.docker.com/compose/compose-file/compose-file-v3/>

#### docker compose Reference

- <https://docs.docker.com/compose/compose-file/compose-file-v3/>

## Docker Swarm

#### Docker Swarm Beispiele

##### Generic examples

```

## should be at least version 1.24
docker info

## only for one network interface
docker swarm init

## in our case, we need to decide what interface
docker swarm init --advertise-addr 192.168.56.101

## is swarm active
docker info | grep -i swarm
## When it is -> node command works
docker node ls
## is the current node the manager
docker info | grep -i "is manager"

```

```
## docker create additional overlay network
docker network ls

## what about my own node -> self
docker node inspect self
docker node inspect --pretty self
docker node inspect --pretty self | less

## Create our first service
docker service create redis
docker images
docker service ls
## if service-id start with j
docker service inspect j
docker service ps j
docker service rm j
docker service ls

## Start with multiple replicas and name
docker service create --name my_redis --replicas 4 redis
docker service ls
## Welche tasks
docker service ps my_redis
docker container ls
docker service inspect my_redis

## delete service
docker service rm
```

#### Add additional node

```
## on first node, get join token
docker swarm join-token manager

## on second node execute join command
docker swarm join --token SWMTKN-1-07jy3ym29au7u3isf1hfhd7wpfggc1nia2kwtqfnfc8hxfcw-2kuhwlnr9i0nkje81z437d2d5
192.168.56.101:2377

## check with node command
docker node ls

## Make node a simple worker
## Does not make, because no highavailable after crush node 1
## Take at LEAST 3 NODES
docker node demote <node-name>
```

#### expose port

```
docker service create --name my_web \
    --replicas 3 \
    --publish published=8080,target=80 \
    nginx
```

#### Ref

- <https://docs.docker.com/engine/swarm/services/>

#### Docker - Dokumentation

##### Vulnerability Scanner with docker

- <https://docs.docker.com/engine/scan/#prerequisites>

##### Vulnerability Scanner mit snyk

- <https://snyk.io/plans/>

##### Parent/Base - Image bauen für Docker

- <https://docs.docker.com/develop/develop-images/baseimages/>

#### Kubernetes - Überblick

##### Installation - Welche Komponenten from scratch

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

```

## Installation Ubuntu - Server

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

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

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

## Standard vo Installation microk8s
lo      UNKNOWN      127.0.0.1/8 ::1/128
## public ip / interne
eth0    UP          164.92.255.234/20 10.19.0.6/16 fe80::c:66ff:fea4:cbce/64
## private ip
eth1    UP          10.135.0.3/16 fe80::8081:aaff:fea4:780/64

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

## Funktioniert microk8s
microk8s status

```

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

```

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

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

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
  - echo " " >> /etc/ssh/sshd_config
  - echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
  - echo "AllowUsers root" >> /etc/ssh/sshd_config
  - systemctl reload sshd
  - sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWFaGkZF3LFAXM4hg13d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo

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

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

```

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

```

Weiteren Server hochgezogen.
Vanilla + BASIS

```

```

## Installation Ubuntu - Server

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

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

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

## Standard vo Installation mikrok8s
lo      UNKNOWN      127.0.0.1/8 ::1/128
## public ip / interne
eth0     UP          164.92.255.232/20 10.19.0.6/16 fe80::c:66ff:fed4:cbce/64
## private ip
eth1     UP          10.135.0.5/16 fe80::8081:aaff:feaa:780/64

##### Installation von kubectl aus dem snap
## NICHT .. keine mikrok8s - keine control-plane / worker-node
## NUR Client zum Arbeiten
snap install kubectl --classic

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

## Mini-Schritt 1:
## Auf dem Server 1: kubeconfig ausspielen
mikrok8s config > /root/kube-config
## auf das Zielsystem gebracht (client 1)
scp /root/kubeconfig 11trainingdo@10.135.0.5:/home/11trainingdo

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

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

```

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

```

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

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

/etc/hosts
10.135.0.3 n1.training.local n1
10.135.0.4 n2.training.local n2
10.135.0.5 n3.training.local n3

```

#### Schritt 5: Cluster aufbauen

```

## Mini-Schritt 1:
## Server 1: connection - string (token)
mikrok8s add-node
## Zeigt Liste und wir nehmen den Eintrag mit der lokalen / öffentlichen ip
## Dieser Token kann nur 1x verwendet werden und wir auf dem ANDEREN node ausgeführt
## mikrok8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 2:
## Dauert eine Weile, bis das durch ist.
## Server 2: Den Node hinzufügen durch den JOIN - Befehl

```

```

microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

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

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

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

```

#### Ergänzend nicht notwendige Scripte

```

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

## Digitalocean - unter user_data reingepastet beim Einrichten

##cloud-config
users:
- name: 11trainingdo
  shell: /bin/bash

runcmd:
- sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
- echo " " >> /etc/ssh/sshd_config
- echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
- echo "AllowUsers root" >> /etc/ssh/sshd_config
- systemctl reload sshd
- sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFaxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBK1.SYbhS52u70:17476:0:99999:7:::'
/etc/shadow
- echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
- chmod 0440 /etc/sudoers.d/11trainingdo

```

### Kubernetes - microk8s (Installation und Management)

#### kubectl unter windows - Remote-Verbindung zu Kuberenets (microk8s) einrichten

##### Walkthrough (Installation)

```

## Step 1
chocolatry installiert.
(powershell als Administrator ausführen)
## https://docs.chocolatey.org/en-us/choco/setup
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

## Step 2
choco install kubernetes-cli

## Step 3
testen:
kubectl version --client

## Step 4:
## powershell als normaler benutzer öffnen

```

##### Walkthrough (autocomplete)

```

in powershell (normaler Benutzer)
kubectl completion powershell | Out-String | Invoke-Expression

```

##### kubectl - config - Struktur vorbereiten

```

## in powershell im heimatordner des Benutzers .kube - ordnern anlegen
## C:\Users\<dein-name>\
mkdir .kube
cd .kube

```

## IP von Cluster-Node bekommen

```
## auf virtualbox - maschine per ssh einloggen
## öffentliche ip herausfinden - z.B. enp0s8 bei HostOnly - Adapter
ip -br a
```

## config für kubectl aus Cluster-Node auslesen (microk8s)

```
## auf virtualbox - maschine per ssh einloggen / zum root wechseln
## abfragen
microk8s config

## Alle Zeilen ins clipboard kopieren
## und mit notepad++ in die Datei \Users\<dein-name>\.kube\config
## schreiben

## Wichtig: Zeile cluster -> clusters / server
## Hier ip von letztem Schritt eintragen:
## z.B.
Server: https://192.168.56.106/.....
```

## Testen

```
## in powershell
## kann ich eine Verbindung zum Cluster aufbauen ?
kubectl cluster-info
```

- <https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>

## Arbeiten mit der Registry

### Installation Kubernetes Dashboard

#### Reference:

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

## Kubernetes - RBAC

### Nutzer einrichten - kubernetes bis 1.24

#### Enable RBAC in microk8s

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

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

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

#### Mini-Schritt 1: Definition für Nutzer

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

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

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

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

## vi pods-clusterrole.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-clusterrole
rules:
- apiGroups: [""]
  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
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pods-clusterrole
subjects:
- kind: ServiceAccount
  name: training
  namespace: default
```

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

#### Mini-Schritt 4: Testen (klappt der Zugang)

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

### Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

#### Mini-Schritt 1: kubeconfig setzen

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

## extract name of the token from here

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

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

#### Mini-Schritt 2:

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

#### Refs:

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

#### Ref: Create Service Account Token

- <https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token>

## kubectl

### Tipps&Tricks zu Deployment - Rollout

#### Warum

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

#### Beispiele

```
## Deployment nochmal durchführen
## z.B. nach kubectl uncordon n12.training.local
kubectl rollout restart deploy nginx-deployment

## Rollout rückgängig machen
kubectl rollout undo deploy nginx-deployment
```

## Kubernetes - Backups

## Kubernetes - Tipps & Tricks

### Assigning Pods to Nodes

#### Walkthrough

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

kubectl get nodes --show-labels

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

## Let's rewrite that to deployment
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
    - name: nginx
      image: nginx
      imagePullPolicy: IfNotPresent
  nodeSelector:
    rechenzentrum=rz2
```

#### Ref:

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

## Kubernetes - Documentation

### LDAP-Anbindung

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

### Helpful to learn - Kubernetes

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

### Environment to learn

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

### Environment to learn II

- <https://killercoda.com/>

### Youtube Channel

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

## Kubernetes -Wann / Wann nicht

### Kubernetes Wann / Wann nicht

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

#### Rechtliche Regulatorien

#### Nationale Grenzen

#### Cloud oder onPrem (private Cloud)

#### Gegenfragen:

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

Kubernetes : weniger sinnvoll <-> sehr sinnvoll.

#### Kosten:

- o Konzeption / Planung
- o Cluster / Manpower (Cluster-Kompetenz)
- o Neue Backup-Strategie / Software
- o Monitoring (ELK / EFK - Stack (Elasticsearch / Logstash-Fluent))

#### Anforderungen an Last

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

#### Nutzt mir Skalierung und kann ich skalieren

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

#### Kubernetes -> Kategorien. Warum ?

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

## Kubernetes - Hardening

### Kubernetes Tipps Hardening

#### PSA (Pod Security Admission)

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

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

#### Möglichkeiten in Pods und Containern

```
## für die Pods  
kubectl explain pod.spec.securityContext  
kubectl explain pod.spec.containers.securityContext
```

#### Example (seccomp / security context)

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

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

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

#### SecurityContext (auf Pod Ebene)

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

#### NetworkPolicy

```
## Firewall Kubernetes
```

### Kubernetes Deployment Scenarios

#### Deployment green/blue,canary,rolling update

##### Canary Deployment

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

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

##### Blue / Green Deployment

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

New Version (GREEN) will be tested and if it works  
the traffic will be switch completley 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 Probes (Liveness and Readiness)

### Übung Liveness-Probe

#### Übung 1: Liveness (command)

What does it do ?

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

```

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

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

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

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

```

#### Übung 2: Liveness Probe (HTTP)

```

## Step 0: Understanding Prerequisite:
This is how this image works:
## after 10 seconds it returns code 500

```

```

http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([]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/>

#### Funktionsweise Readiness-Probe vs. Liveness-Probe

##### Why / Howto /

- Readiness checks, if container is ready and if it's not READY
  - SENDS NO TRAFFIC to the container

##### Difference to LiveNess

- They are configured exactly the same, but use another keyword
  - readinessProbe instead of livenessProbe

#### Example

```

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

```

#### Reference

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

## Linux und Docker Tipps & Tricks allgemein

### Auf ubuntu root-benutzer werden

```

## kurs>
sudo su -

```

```
## password von kurs eingegeben
## wenn wir vorher der benutzer kurs waren
```

### IP - Adresse abfragen

```
## IP-Adresse abfragen
ip a
```

### Hostname setzen

```
## als root
hostnamectl set-hostname server.training.local
## damit ist auch sichtbar im prompt
su -
```

### Proxy für Docker setzen

#### Walktrough

```
## as root
systemctl list-units -t service | grep docker
systemctl cat snap.docker.dockerd.service
systemctl edit snap.docker.dockerd.service
## in edit folgendes reinschreiben
[Service]
Environment="HTTP_PROXY=http://user01:password@10.10.10.10:8080/"
Environment="HTTPS_PROXY=https://user01:password@10.10.10.10:8080/"
Environment="NO_PROXY= hostname.example.com,172.10.10.10"

systemctl show snap.docker.dockerd.service --property Environment
systemctl restart snap.docker.dockerd.service
systemctl cat snap.docker.dockerd.service
cd /etc/systemd/system/snap.docker.dockerd.service.d/
ls -la
cat override.conf
```

#### Ref

- <https://www.thegeekdiary.com/how-to-configure-docker-to-use-proxy/>

### vim einrückung für yaml-dateien

#### Ubuntu (im Unterverzeichnis /etc/vim/vimrc.local - systemweit)

```
hi CursorColumn cterm=None ctermfg=lightred ctermbg=white
autocmd FileType y?ml setlocal ts=2 sts=2 sw=2 ai number expandtab cursorline cursorcolumn
```

#### Testen

```
vim test.yml
Eigenschaft: <return> # springt eingerückt in die nächste Zeile um 2 spaces eingerückt

## evtl funktioniert vi test.yml auf manchen Systemen nicht, weil kein vim (vi improved)
```

### YAML Linter Online

- <http://www.yamllint.com/>

### Läuft der ssh-server

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

### Basis/Parent - Image erstellen

#### Auf Basis von debootstrap

```
## Auf einem Debian oder Ubuntu - System
## folgende Schritte ausführen
## z.B. virtualbox -> Ubuntu 20.04.

## alle mit root durchführen
apt install debootstrap
cd
debootstrap focal focal > /dev/null
tar -C focal -c . | docker import - focal
```

```
## er gibt eine checksumme des images
## so kann ich das sehen
## müsste focal:latest heissen
docker images

## teilchen starten
docker run --name my_focal2 -dit focal:latest bash

## Dann kann ich danach reinwechseln
docker exec -it my_focal2 bash
```

#### Virtuelle Maschine Windows/OSX mit Vagrant erstellen

```
## Installieren.
https://vagrantup.com
## ins terminal
cd
cd Documents
mkdir ubuntu_20_04_test
cd ubuntu_20_04_test
vagrant init ubuntu/focal64
vagrant up
## Wenn die Maschine oben ist, kann direkt reinwechseln
vagrant ssh
## in der Maschine kein pass notwendig zum Wechseln
sudo su -
## wenn ich raus will
exit
exit

## Danach kann ich die maschine wieder zerstören
vagrant destroy -f
```

#### Ref:

- <https://docs.docker.com/develop/develop-images/baseimages/>

#### Eigenes unsichere Registry-Verwenden. ohne https

##### Setup insecure registry (snap)

```
systemctl restart
```

#### Spiegel - Server (mirror -> registry-mirror)

```
https://docs.docker.com/registry/recipes/mirror/
```

#### Ref:

- <https://docs.docker.com/registry/insecure/>

## Linux Tipps & Tricks

### Grafischen Modus deaktivieren

- Besser: komplett deinstallieren

### Das geht immer

```
## also root
sudo su -
## target ohne Grafik
systemctl isolate multi-user
## Beim Start auch diese Target setzen
systemctl set-default multi-user
```

## VirtualBox Tipps & Tricks

### VirtualBox 6.1. - Ubuntu für Kubernetes aufsetzen

#### Vorbereitung

- Ubuntu Server 22.04 LTS - ISO herunterladen

#### Schritt 1: Virtuelle Maschine erstellen

In VirtualBox Manager -> Menu -> Maschine -> Neu (Oder Neu icon)

Seite 1:  
Bei Name Ubuntu Server eingeben (dadurch wird gleich das richtige ausgewählt, bei den Selects)  
Alles andere so lassen.  
Weiter

Seite 2:  
Hauptspeicher mindest 4 GB , d.h. 4096 auswählen (für Kubernetes / microk8s)  
Weiter

Seite 3:  
Festplatte erzeugen ausgewählt lassen  
Weiter

Seite 4:  
Dateityp der Festplatte: VDI ausgewählt lassen  
Weiter

Seite 5:  
Art der Speicherung -> dynamisch alloziert ausgewählt lassen  
Weiter

Seite 6:  
Dateiname und Größe -> bei Größe mindestens 30 GB einstellen (bei Bedarf größer)  
-> Erzeugen

## Schritt 2: ISO einhängen / Netzwerk und starten / installieren

Neuen Server anklicken und ändern klicken:

1.  
Massenspeicher -> Controller IDE -> CD (Leer) klicken  
CD - Symbol rechts neben -> Optisches Laufwerk (sekundärer Master) -> klicken -> Abbild auswählen  
Downgeloadetes ISO Ubuntu 22.04 auswählen -> Öffnen klicken

2.  
Netzwerk -> Adapter 2 (Reiter) anklicken -> Netzwerkadapter aktivieren  
Angeschlossen an -> Host-only - Adapter

3.  
unten rechts -> ok klicken

## Schritt 3: Starten klicken und damit Installationsprozess beginnen

Try or install Ubuntu Server -> ausgewählt lassen

Seite 1:  
Use up -.... Select your language  
-> English lassen  
Enter eingeben

Seite 2: Keyboard Configuration  
Layout auswählen (durch Navigieren mit Tab-Taste) -> Return  
German auswählen (Pfeiltaste nach unten bis German, dann return)  
Identify Keyboard -> Return  
Keyboard Detection starting -> Ok  
Jetzt die gewünschten tasten drücken und Fragen beantworten  
Layout - Variante bestätigen mit OK

-> Done

Seite 3: Choose type of install  
Ubuntu - Server ausgewählt lassen

-> Done

Seite 4: Erkennung der Netzwerkkarten  
(192.168.56.1x) sollte auftauchen

-> Done

Seite 5: Proxy

leer lassen

```

-> Done

Seite 6: Mirror Address

kann so bleiben

-> Done

Seite 7:

Guided Storage konfiguration
Entire Disk

-> Done

Seite 8: File System Summary

-> Done

Seite 9: Popup: Confirm destructive action
Bestätigen, dass gesamte Festplatte überschrieben wird
(kein Problem, da Festplatte ohnehin leer und virtuell)

-> Continue

Seite 10: Profile Setup

User eingeben / einrichten
Servernamen einrichten

-> Done

Seite 11: SSH Setup

Haken bei: Install OpenSSH Server
setzen

-> Done

Seite 12: Featured Server Snaps

Hier brauchen wir nichts auswählen, alles kann später installiert werden

-> Done

Seite 13: Installation

Warten bis Installation Complete und dies auch unten angezeigt wird (Reboot Now):
(es dauert hier etwas bis alle Updates (unattended-upgrades) im Hintergrund durchgeführt worden sind)

-> Reboot Now

Wenn "Failed unmounting /cdrom" kommt
dann einfach Server stoppen
-> Virtual Box Manager -> Virtuelle Maschine auswählen -> Rechte Maustaste -> Schliessen -> Ausschalten

```

#### Schritt 4: Starten des Gast-Systems in virtualbox

- \* Im VirtualBox Manager auf virtuelle Maschine klicken
- \* Neben dem Start - Pfeil -> Dreieck anklicken und Ohne Gui starten wählen
- \* System startet dann im Hintergrund (kein 2. Fenster)

#### Erklärung

- Console wird nicht benötigt, da wir mit putty (ssh) arbeiten zum Administrieren des Clusters
- Putty-Verbindung muss nur auf sein, wenn wir administrieren
- Verwendung des Clusters (nutzer/Entwickler) erfolgt ausschliesslich über kubectl in powershell !!

#### VirtualBox 6.1. - Shared folder aktivieren

##### Prepare

```

Walkthrough

## At the top menu of the virtual machine
## Menu -> Geräte -> Gasterweiterung einlegen

## In the console do a

```

```

mount /dev/cdrom /mnt
cd /mnt
sudo apt-get install -y build-essential linux-headers-`uname -r`
sudo ./VBoxLinuxAdditions.run

sudo reboot

```

## Configure

```

Geräte -> Gemeinsame Ordner
Hinzufügen (blaues Ordnersymbol mit +) ->
Ordner-Pfad: C:\Linux (Ordner muss auf Windows angelegt sein)
Ordner-Name: linux
checkbox nicht ausgewählt bei: automatisch einbinden, nur lesbar
checkbox ausgewählt bei: Permanent erzeugen

Dann rebooten

In der virtuellen Maschine:
sudo su -
mkdir /linux
## linux ist der vergebene Ordnername
mount -t vboxsf linux /linux

## Optional, falls du nicht zugreifen kannst:
sudo usermod -aG vboxsf root
sudo usermod -aG vboxsf <your-user>

```

## persistent setzen (beim booten mounten)

```

echo "linux      /linux  vboxsf  defaults      0      0" >> /etc/fstab
reboot

```

## Reference:

- <https://gist.github.com/estorgo/1d679f962e8209f8a9232f7593683265>

## CloudInit

### Kubernetes Client einrichten mit bash

```

#!/bin/bash

groupadd sshadmin
USERS="11trainingdo $(echo tln{1..20})"
echo $USERS
for USER in $USERS
do
  echo "Adding user $USER"
  useradd -s /bin/bash --create-home $USER
  usermod -aG sshadmin $USER
  echo "$USER:deinpassword" | chpasswd
done

## We can sudo with 11trainingdo
usermod -aG sudo 11trainingdo

## Now let us do some generic setup
echo "Installing kubectl"
snap install --classic kubectl

echo "Installing helm"
snap install --classic helm

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

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

```

```

usermod -aG sshadmin root

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

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

##### BASH Completion #####
## update repo
apt-get update
apt-get install -y bash-completion
source /usr/share/bash-completion/bash_completion
## is it installed properly
type _init_completion

## 1. kubectl completion -> activate for all users
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

## 2. helm completion -> activate for all users
helm completion bash | sudo tee /etc/bash_completion.d/helm > /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/autoindenting for nano
## v1 - old version / remove if new version works
##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

#####
## v2 - new version / more simplistic
#####
echo "include /usr/share/nano/yaml.nanorc" >> /etc/nanorc
echo "set autoindent" >> /etc/nanorc
echo "set tabsize 2" >> /etc/nanorc
echo "set tabstop=8" >> /etc/nanorc

## Install nfs-common for mounting, just in case we need it for persistant storage exercise
apt-get install -y nfs-common

```

## Microservices - Messaging

### EventBus Implementierungen/Überblick

#### Fertige Software, die einen Event Bus bereitstellt

- Kafka
- RabbitMQ
- NATS

#### Was ist Ihre Aufgabe ?

- Events empfangen
- Events veröffentlichen (publish) für die Zuhörer (listener)

#### Wie sehen Events aus ?

- Mit Events meinen wir Informations-Snippets
  - Es ist nicht festgelegt, wie eine Event aussehen soll, es kann
    - Rohe Datenbytes
    - JSON
    - ein String
    - u.a. ... sein (was immer du verwenden willst)

#### Was sind Listener ?

- Listener sind Services, die von anderen Events von anderen Services erfahren wollen