

KUBERNETES WORKSHOP

VORTRAGENDE

- Sandro Koll

- Pascal Rimann

TEILNEHMER

- Kurze Vorstellung
- Erfahrungen?
 - Docker
 - Kubernetes
- Erwartungen?

TAG 1

AGENDA

1. Setup
2. Container
3. Monolithen vs. Microservices
4. Container-Orchestrierung
5. Prinzipien hinter Kubernetes

SETUP

```
sudo usermod -aG docker ${USER}
git clone https://github.com/x-cellent/k8s-workshop.git
cd k8s-workshop

make
mkdir -p ~/bin
mv bin/w6p ~/bin/
echo "export PATH=$PATH:~/bin" >> ~/.bashrc
source ~/.bashrc
w6p
```

OUTPUT

Usage:

```
w6p [flags]  
w6p [command]
```

Available Commands:

cluster	Runs the workshop cluster or exercises
exercise	Runs the given exercise
help	Help about any command
install	Installs tools on local machine
slides	Shows or exports workshop slides

Flags:

```
-h, --help    help for w6p
```

WAS IST W6P?

Go CLI executable ausschließlich für diesen Workshop

- w6p install TOOL
 - lokale Installation von gebräuchlichen k8s Tools
- w6p exercise CONTEXT -n NUMBER
 - Startet Aufgaben aus dem jeweiligen Kontext
(docker oder k8s)
- w6p cluster
 - Startet/stoppt Single Node Kubernetes Cluster in

- w6p slides
 - Startet Webserver Container, der die Workshop Slides hosted
 - erreichbar unter localhost:8080

CONTAINER

Software wird schon seit Jahrzehnten in Archive oder Single-Binaries verpackt

- Einfache Auslieferung
- Einfache Verteilung

ABER

- Installation notwendig
- Dependency Hell
- No cross platform functionality

LÖSUNG

- Verpacken der Software mitsamt aller Dependencies (Image)
 - Nichts darüber hinaus (Betriebssystem notwendig?)
- Container-Runtime für alle Plattformen

UMSETZUNG

- Linux
- Idee: Container teilen sich Kernel
- LXC: basierend auf Kernel-Funktionalitäten
 - namespaces
 - cgroups
- Docker erweitert LXC um
 - CLI zum Starten und Verwalten von Containern

VORTEILE CONTAINER

1. Geringere Größe
2. Erhöhte Sicherheit
3. Funktional auf allen Systemen
4. Immutable
 - Damit Baukastenprinzip möglich (DRY)

VORTEILE GEGENÜBER VMS

1. Geringere Größe
2. Geringerer Ressourcenverbrauch
3. Viel schnellere Startup-Zeiten
4. Auch geeignet für Entwicklung und Test

NACHTEILE GEGENÜBER VMS

1. Geringere Sicherheit
2. Keine echte Trennung
 - z.B. kein Block-Storage möglich

Container und VMs schließen sich aber nicht gegenseitig aus

DOCKER KOMPONENTEN

1. Image

- Layer
- Dockerfile

2. Container

3. Image Registry

DOCKERFILE

- Referenz
- Image-Rezept mit u.a. folgenden Zutaten:
 - FROM
 - COPY/ADD
 - RUN
 - USER
 - WORKDIR

BEISPIEL

```
# Basis-Image
FROM alpine:3.15

# Installiert busybox-extras ins Basis-Image
RUN apk add --no-cache busybox-extras
# ...und committed den FS-Diff als neuen Layer

# Installiert auf dem obigen Layer mysql-client
RUN apk add --no-cache mysql-client
# ...und committed das FS-Diff als neuen Layer

# Default command
ENTRYPOINT [ "mysql" ]

# Default arg(s)
CMD [ "--help" ]
```

EINIGE DOCKER COMMANDS

- docker build
 - Baut ein Image von Dockerfile
- docker images / docker image ls
 - Listet alle (lokalen) Images
- docker tag
 - Erstellt Image "Kopie" unter anderem Namen
- docker rmi / docker image rm

- docker run
 - Startet ein Image -> Container
- docker ps [-q]
 - Listet alle (laufenden) Container
- docker rm
 - Löscht einen Container
- docker logs

- docker [image] inspect
 - Zeigt Metadaten von Container/Images
- docker cp
 - Kopiert eine Datei aus Container ins Host-FS und umgekehrt
- docker save/load
 - Erzeugt Tarball aus Image und umgekehrt

IMAGE BAUEN

```
docker build -t [REPOSITORY_HOST/] IMAGENAME:IMAGETAG \
[-f path/to/Dockerfile] path/to/context-dir
```

- Kontext-Verzeichnis wird zum Docker Daemon hochgeladen
 - lokal oder remote (via DOCKER_HOST)
 - Nur darin enthaltene Dateien können im Dockerfile verwendet werden (COPY/ADD)
 - Nach Möglichkeit keine ungenutzten Dateien hochladen

AUFGABE 1

```
w6p exercise docker -n 1
```

Lösung nach 5 min

CONTAINER STARTEN

```
docker run [--name NAME] [-i] [-t] [-d|--rm] [--net host|NETWO  
[-p HOST_PORT:CONTAINER_PORT] [-u UID:GID] IMAGE [arg(s)]
```

- Mehr Optionen möglich
- Referenz

EXECUTE CMD IN CONTAINER

```
docker exec [-i] [-t] CONTAINER COMMAND
```

Via Bash in den Container "springen":

```
docker exec -it CONTAINER /bin/bash
```

DATEIEN KOPIEREN

...vom Host in den Container:

```
docker cp HOST_FILE CONTAINER_NAME:CONTAINER_FILE
```

z.B.:

```
docker cp ~/local-index.html my-server:/static/index.html
```

...vom Container in das Host-FS:

```
docker cp CONTAINER_NAME:CONTAINER_FILE HOST_FILE
```

z.B.:

```
docker cp my-server:/static/index.html ~/local-index.html
```

AUFGABE 2

```
w6p exercise docker -n2
```

Lösung nach 20 min

METADATEN

```
docker inspect IMAGE|CONTAINER
```

- Image Metadaten
 - ID
 - Architecture
 - Layers
 - Env
 - ...

- Container Metadaten
 - ID
 - Image ID
 - NetworkSettings
 - Mounts
 - State

AUFGABE 3

```
w6p exercise docker -n3
```

Lösung nach 15 min

LINTING

- [hadolint](#)
- Erhältlich als Docker Image:

```
docker run ... hadolint/hadolint hadolint path/to/Dockerfile
```

AUFGABE 4

```
w6p exercise docker -n4
```

Lösung nach 5 min

MULTI-STAGE DOCKERFILE

```
# Build as usual in the first stage
FROM golang:1.17 AS builder # named stage

WORKDIR /work
# Copy source code into image
COPY app.go .

# Compile source(s)
RUN go build -o bin/my-app

# Further builder images possible, e.g.
# FROM nginx AS webserver
# ...

# Final stage: all prior images will be discarded after build
FROM scratch

# ...but here we can copy files from builder image(s)
COPY --from=builder /work/bin/my-app /

ENTRYPOINT [ "/my-app" ]
```

VORTEILE

1. Kompakte Imagegröße
2. Erhöhte Sicherheit

AUFGABE 5

```
w6p exercise docker -n5
```

Lösung nach 15 min

DOCKER REGISTRY

- Docker-Hub
 - öffentlich
- private Registries möglich
 - Image registry
 - absicherbar
 - praktisch in jeder Firma eingesetzt

AUFGABE 6

```
w6p exercise docker -n6
```

Lösung nach 15 min

WAS DOCKER NICHT BIETET:

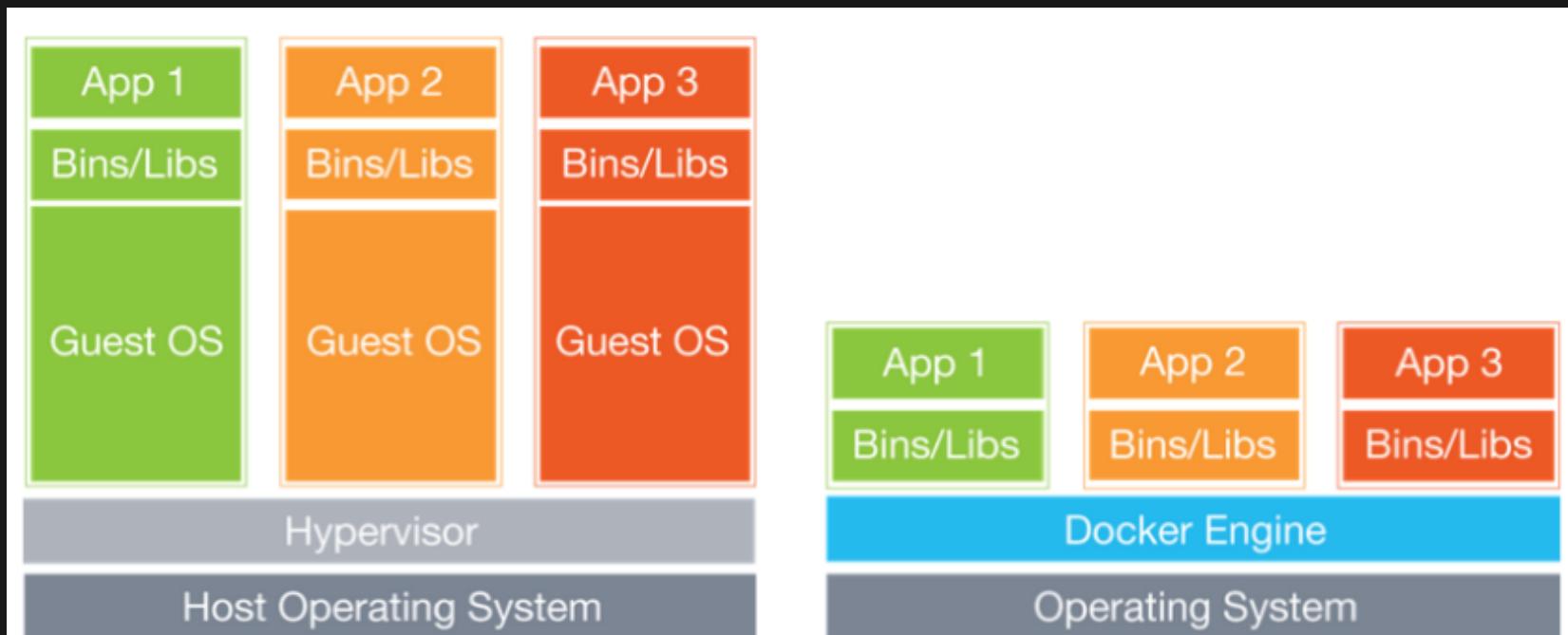
1. Orchestrierung
2. Ausfallsicherheit

=> Kubernetes - bietet beides - ...und noch viel mehr

DOCKER COMPOSE

- Für Multi-Container Docker Anwendungen
- docker-compose.yaml
 - Definition der Container
- docker-compose up/down
 - Start/Stop aller Anwendungen in einem Rutsch
- Rudimentäre Funktionalitäten
- Geeignet für sehr kleine (Dev / Test) Umgebungen

MONOLITH VS MICROSERVICES



CONTAINER ORCHESTRIERUNG

WIESO?

- Orchestrierung von Containern

WARUM KUBERNETES?

- Warum nicht Docker Swarm?
- Mehr Flexibilität
- Eingebautes Monitoring und Logging
- Bereitstellung von Storage
- Größere User-Base

PRINZIPIEN HINTER KUBERNETES

DER POD

- kein Container
- beinhaltet mindestens einen Container
- kann init container beinhalten
- kann sidecar container beinhalten
- kleinste Einheit in Kubernetes

WO LAUFEN PODS?

- Auf (Worker-)Nodes

ORDNUNGSELEMENTE

- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job
- CronJob

FRAGEN

- Habt ihr noch Fragen an uns?

AUSBLICK TAG 2

- Architektur von Kubernetes
- Basis Objekte von Kubernetes

TAG 2

AGENDA

1. Rewind
2. Architektur von Kubernetes
3. Einrichtung eurer Umgebung
4. kubectl
5. k9s
6. Basisobjekte Kubernetes

REWIND

DOCKER

- Container Runtime Engine
- docker CLi
 - Bau von Images
 - Start von Images (Container)
 - Image Registry (Verteilung von Images, vgl. AppStore)

IMAGE BAUEN

- Schreibe ein Dockerfile (Rezept)
 - Instruktionen (Zutaten)
 - FROM, COPY, RUN, ENTRYPOINT, ...
 - jede Instruktion = neuer Layer (vgl. mit Binärdatei)
- docker build [-t TAG] CONTEXT
 - Image = N übereinander gelegte PQLayer (overlay)

IMAGE STARTEN

- docker run [OPTS] IMAGE [COMMAND] [ARGS]
 - Container = N Image RO Layers plus ein leerer RW Layer on top
 - RW Layer kann zur Laufzeit modifiziert werden
 - Delete File/Dir
 - Löscht, wenn im RW Layer vorhanden
 - Versteckt, wenn in einem darunter liegenden

KUBERNETES

- Container Orchestrierungstool
 - Verwaltet Pods
 - besteht aus 1 bis N Containern
 - eigene IP, eigenes Netzwerk
 - Startet, stoppt und überwacht Pods
 - Verteilung auf Worker-Nodes
 - Garantiert Pods Ressourcen (CPU/Memory)

KUBERNETES OBJEKTE

- Pod
 - kleinste deploybare Einheit
 - beinhaltet 1 bis N Container
 - eigener Netzbereich und IP
- ReplicaSet
 - Stellt sicher, dass zu jeder Zeit genau N Pods laufen

- Deployment
 - Managed ein ReplicaSet
 - Bietet Versionierung und zero downtime Rollouts
- DeamonSet
 - Spec wie Deployment nur ohne Replica Count
 - Managed damit kein ReplicaSet
 - Stattdessen je ein Replica pro Node

- Service
 - Loadbalancer für Pods
 - Auch hier Matching via Labels
 - Typen
 - None (headless)
 - ClusterIP (Default)
 - NodePort

- StatefulSet
 - Spec ähnlich zu Deployment
 - Geordnetes Starten (einer nach dem anderen)
 - Geordnetes Stoppen in umgekehrter Reihenfolge
- ConfigMap
 - Plain-Text Key-Value Store
 - Kann in Pods, Deployments, STSs und DSSs

KUBERNETES TOOLS

- kubectl
 - CLI zur Interaktion mit k8s Clustern
- krew
 - kubectl Plugin Manager
- k9s
 - Terminal UI zur Interaktion mit k8s Clustern
- kind (*Kubernetes in Docker*)

- helm
 - Paket-Manager für Kubernetes
 - vgl. mit apt für Ubuntu oder apk für Alpine
- Lens
 - Graphical UI zur Interaktion mit k8s Clustern
 - Nicht Teil des Workshops

KUBECTL PLUGINS

- [Liste](#) von verfügbaren Plugins
- Installation
 - `kubectl krew install NAME [NAME...]`

WAS IST W6P?

Go CLI executable ausschließlich für diesen Workshop

- w6p install TOOL
 - lokale Installation von gebräuchlichen k8s Tools
- w6p exercise CONTEXT -n NUMBER
 - Startet Aufgaben aus dem jeweiligen Kontext
(docker oder k8s)
- w6p cluster
 - Startet/stoppt Single Node Kubernetes Cluster in

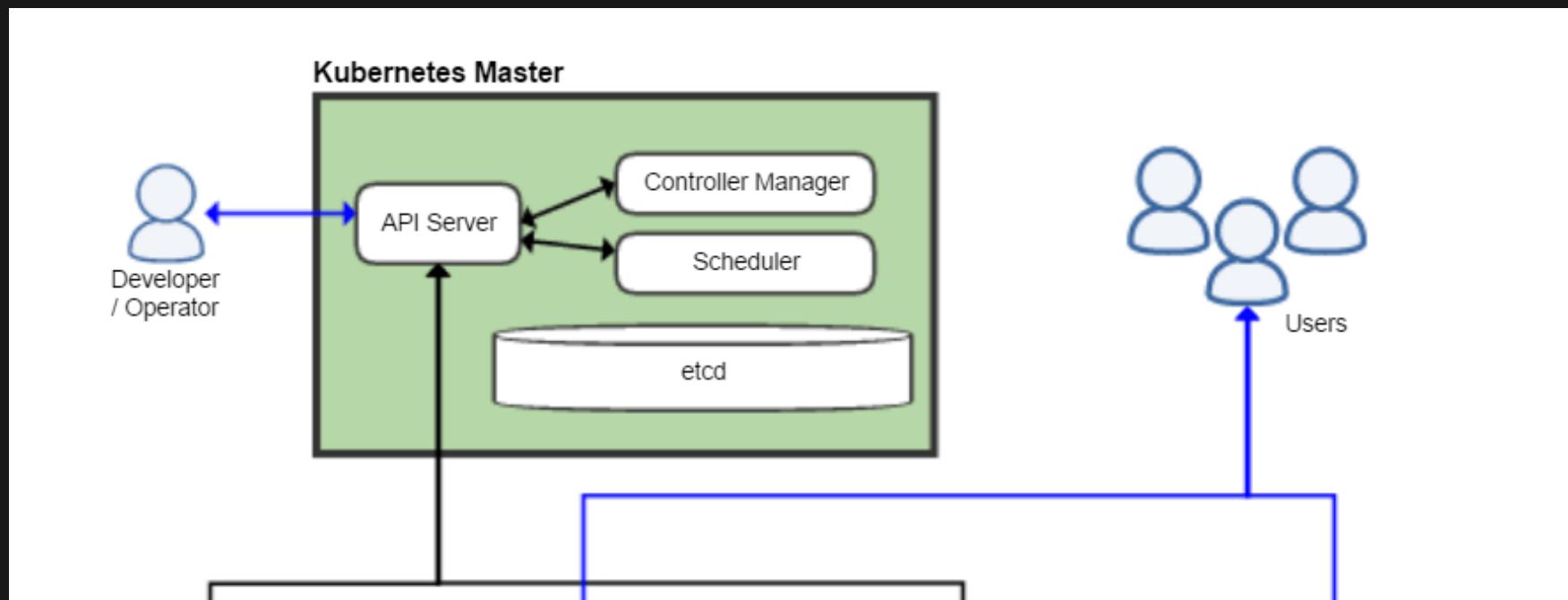
KUBERNETES

- Ursprünglich 2014 entwickelt von Google
- Abgegeben 2015 an die Cloud Native Compute Fondation (CNCF)

CNCF

- Cloud Native Computing Foundation
- 2015 Gegründet
- über 500 Hersteller und Betreiber

ARCHITEKTUR VON KUBERNETES



ARCHITEKTUR DES CLUSTERS

1. Modular und austauschbar

1. Control-Plane

- etcd
- API-Server
- Scheduler
- Kube-Controller-Manager

2. Nodes

CONTROL-PLANE

ETCD

- entwickelt von CoreOS
- key-value Database
- kann nicht getauscht werden
- speichert stand von cluster
- Consistency notwendig

API-SERVER

- Ansprechpunkt des Users
- Validation der Daten
- bekanntester ist kube-apiserver
- horizontale Skalierbarkeit

SCHEDULER

- Verteilt workload
- verantwortlich für pods ohne node

KUBE-CONTROLLER-MANAGER

- bringt cluster von "ist" -> "soll"
- Managed Nodes
- mitteilung an scheduler wenn node down

NODES

KUBELET

- verwaltet pods
- auf jeden node installiert
- verantwortlich für status

KUBE PROXY

- verwaltet Netzwerkanfragen
- routet traffic zu gewünschten pod
- loadbalancer

WEITERE KOMPONENTEN

- CNI
- Container-Runtime

NAMESPACES

- separierungseinheit in Kubernetes
- Objekte können welche in anderem Namespace nicht sehen
- 4 standart Namespaces
 - default
 - kube-node-lease
 - kube-public

DNS

- CoreDNS und KubeDNS
- FQDN in cluster
 - POD.NAMESPACE.pod.cluster.local
 - SERVICE.NAMESPACE.svc.cluster.local

OPENSOURCE

AUSFALLSICHERHEIT

- Container Health Check
 - readiness
 - liveness
- Hostsystemausfall
- Update

YAML

- Alle k8s Objekte werden in **YAML** definiert
 - Wird Manifest genannt
 - Jedes Objekt hat eigene YAML Spec
- API Server versteht diese Specs
 - `kubectl apply -f spec.yaml`
 - Schickt den Inhalt von `spec.yaml` an k8s API Server

MANIFEST: BEISPIELE

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
```

```
metadata:  
  labels:  
    app: nginx # has to match .spec.selector.matchLabels  
spec:  
  terminationGracePeriodSeconds: 10  
  containers:  
    - name: nginx  
      image: nginx-slim:0.8  
      ports:  
        - containerPort: 80  
          name: web  
      volumeMounts:  
        - name: www  
          mountPath: /usr/share/nginx/html
```

```
apiVersion: apps/v1
kind: StatefulSet

metadata:
  name: web
  namespace: testing
```

```
spec:  
  selector:  
  
    matchLabels:  
      app: nginx # has to match .spec.template.metadata.labels  
  replicas: 3 # by default is 1
```

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

EINRICHTUNG EURER

UMGEBUNG

w6p exercise k8s

KUBECTL

CLI Tool für das Management von k8s Clustern

Man kann kubectl über folgende Wege mitteilen, mit welchem Cluster es sich verbinden soll:

- via Command-Line Argument '--kubeconfig path/to/kubeconfig'
- via Umgebungsvariable KUBECONFIG
- via Default Kubeconfig-Datei ~/.kube/config

WEITERE PARAMETER

- Welcher Namespace?
 - Alle: '--all-namespaces' oder '-A'
 - X: '--namespace X' oder '-n X'
- Was will ich tun?
 - Ressource anschauen
 - 'get pod', 'get deploy', 'get secret'
 - Ressource erstellen

KONKRETE BEISPIELE

- Anlegen einer ConfigMap im NS 'webapp':

```
kubectl create -n webapp cm db-config \
--from-literal=db-user=dba \
--from-literal=db-password=OH-NO
```

- Erstelle Pod Manifest:

```
export do='--dry-run=client -o yaml'  
kubectl run my-pod --image ubuntu:20.04 $do > pod.yaml
```

Dieses Manifest kann jetzt bequem angepasst/vervollständigt werden

- Erstelle Deployment Manifest:

```
export do='--dry-run=client -o yaml'  
kubectl create deploy --image alpine:3.15 my-deploy $do > dp.y
```

AUFGABE 1

w6p exercise k8s -n1

Lösung nach 20m

K9S

Terminal UI zum Management eines k8s Clusters

- Start

```
k9s [--kubeconfig path/to/kubeconfig] [-n NAMESPACE]
```

- Stop: Ctrl^c
- Hilfe: '?'
- Navigation mit Pfeiltasten
- Tiefer reinspringen mit ENTER

- Ressourcen-Navigation
 - `:`
 - Gefolgt von der gewünschten Ressource
 - Context: 'context' oder 'ctx'
 - Namespace: 'namespace' oder 'ns'
 - Pod: 'pod' oder 'po'
 - Deployment: 'deployment' 'deploy' oder 'dp'

- Context-Switch
 - ':ctx' ENTER Navigiere-zu-Context ENTER
 - Verbindet sich mit dem ausgewählten Cluster
 - In der Folge werden nur noch Objekte dieses Clusters angezeigt
- Namespace-Switch
 - ':ns' ENTER Navigiere zu Namespace ENTER

- Wenn eine Ressource X selektiert ist
 - Describe: 'd' (k describe X)
 - Zeige YAML: 'y' (k get X -o yaml)
 - Delete: 'Ctrl^d' (k delete X)
 - Edit: 'e' (k edit X)
 - vi oder vim

- Je nachdem welche Resource ausgewählt ist, weitere Optionen möglich
 - Für Pods z.B:
 - Logs: 'l' (k logs X)
 - Shell: 's' (k exec -ti X sh)
 - Port Forward: 'Shift^f' (k port-forward X 80)
 - Für Deployments z.B:

AUFGABE 2

w6p exercise k8s -n2

Lösung nach 20m

TAG 3

AGENDA

- Recap
 - Docker (high-level)
 - Kubernetes und API Objekte

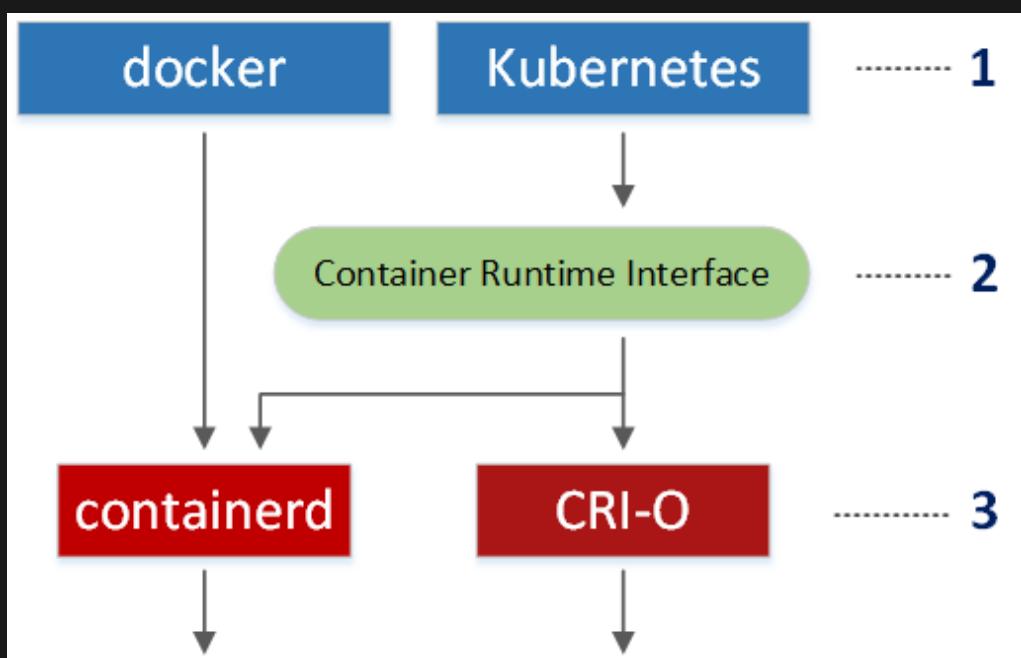
RECAP

DOCKER

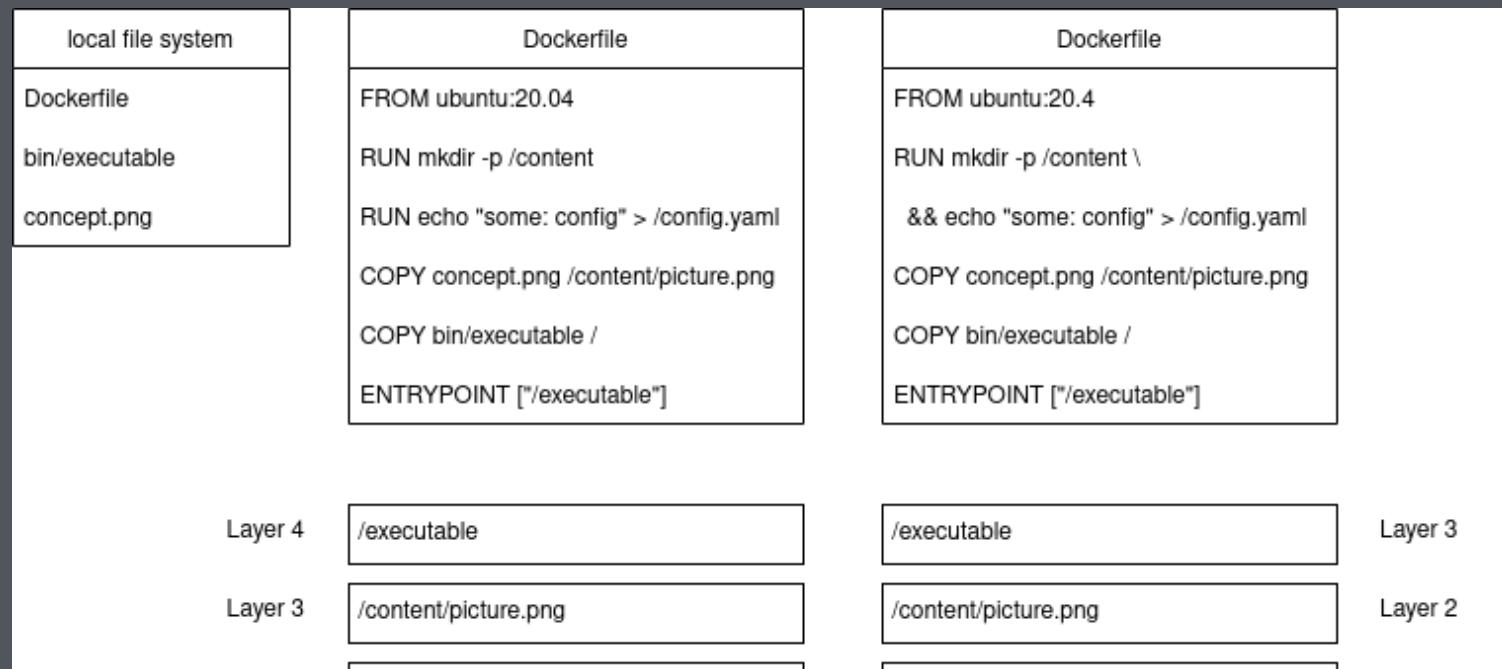
- Bündelt Software mitsamt aller Abhängigkeiten
 - evtl. auch inklusive OS
 - damit über Rechnergrenzen hinweg funktional
- Bau und Starten von Images
 - Container Runtime Engine
- Image-Sharing (Docker Registry)
- Client-Server Architektur

CONTAINER ÖKOSYSTEM

- Container Runtime Interface (CRI)
 - Wird von Kubernetes unterstützt
 - Konkrete Implementierung damit austauschbar
- Container Runtimes
 - containerd (von Docker entwickelt), CRI-O, ...
- Open Container Initiative (OCI) spec
 - enthält runtime spec (runc) und image spec



BEISPIEL DOCKER-IMAGE



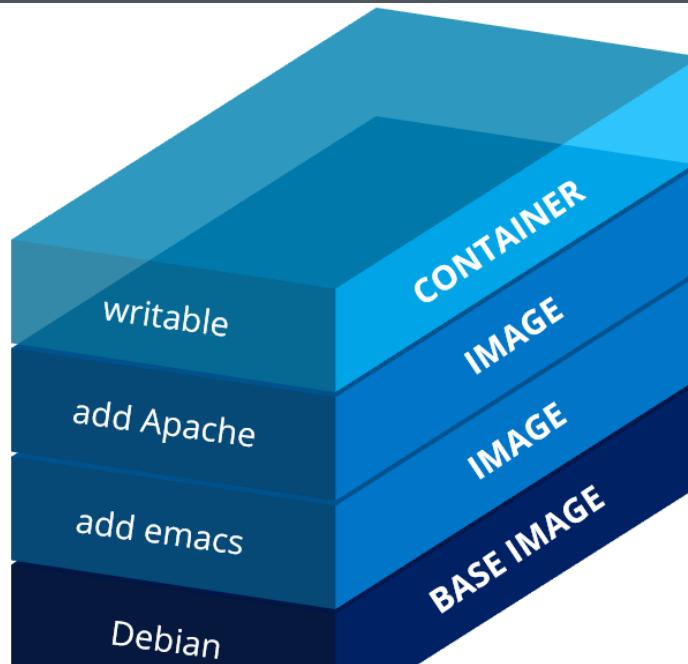
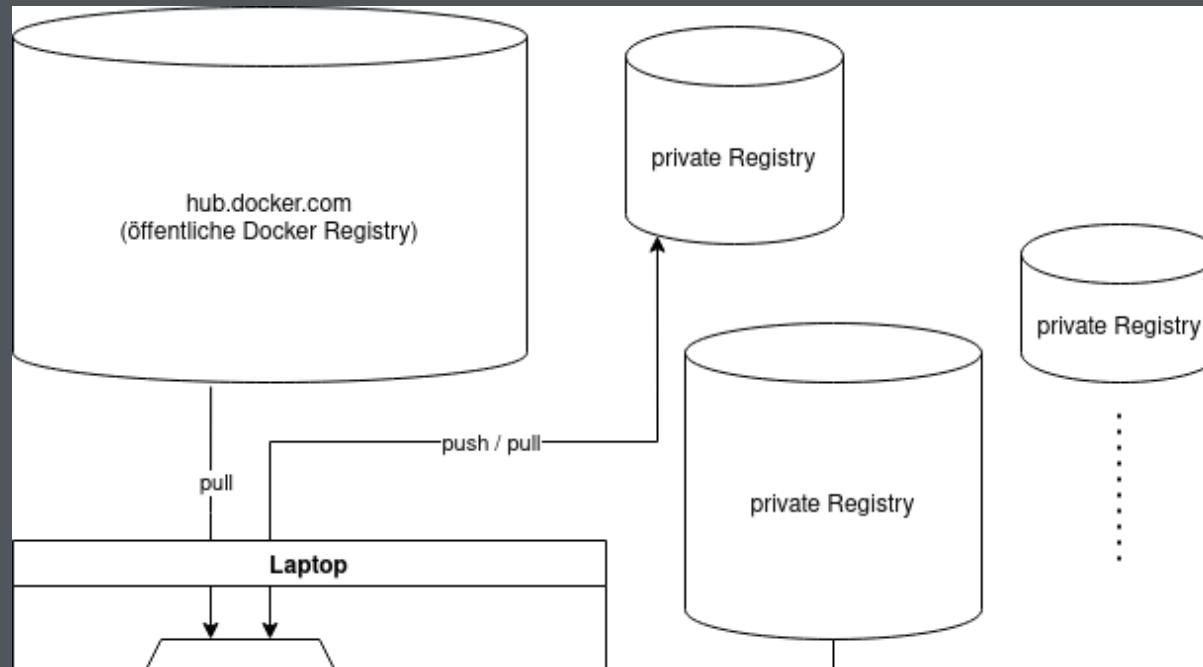


IMAGE-SHARING

```
docker pull nginx
```

```
docker tag my-image my-registry-host/my-image:v1
docker login my-registry-host
docker push|pull my-registry-host/my-image:v1
docker logout my-registry-host
```

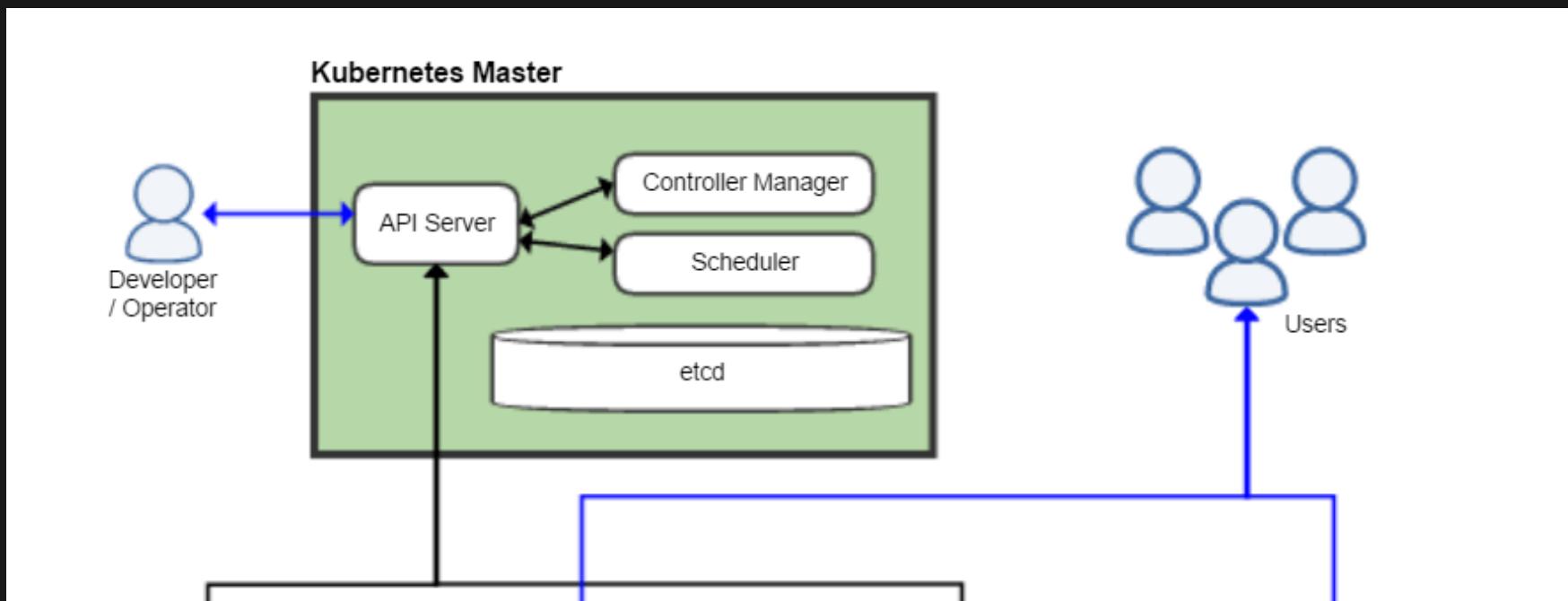




kubernetes

- Container Orchestrierungstool
 - Verwaltung von Pods
 - Reconciliation-Loop / Control-Loop
 - Starten, stoppen und Überwachen
 - Self-Healing
 - Dynamische Skalierung

ARCHITEKTUR VON KUBERNETES



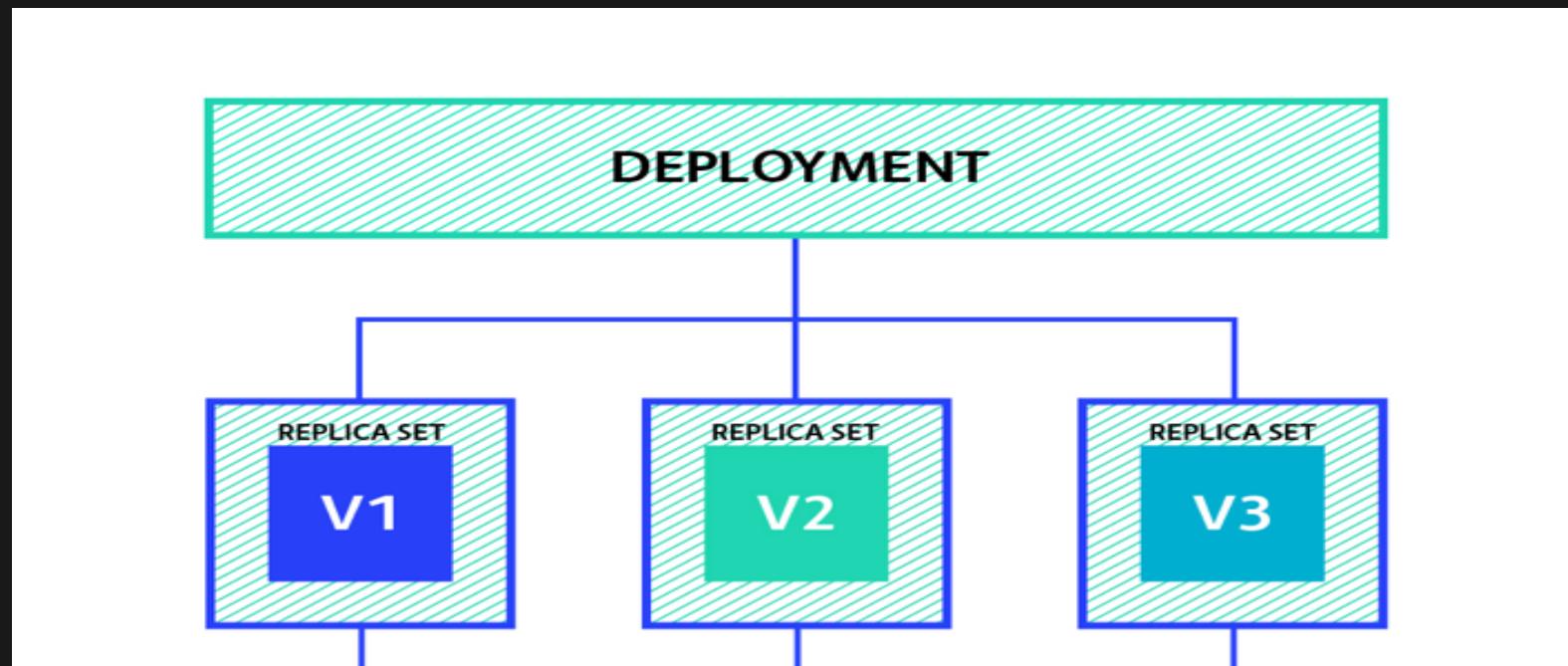
KUBECONFIG

- Text-Datei
- Beinhaltet URLs und Credentials zu k8s Clustern
- "Login zum Cluster"
 - `~/.kube/config` (default)
 - via KUBECONFIG Umgebungsvariable
 - via `--kubeconfig` flag

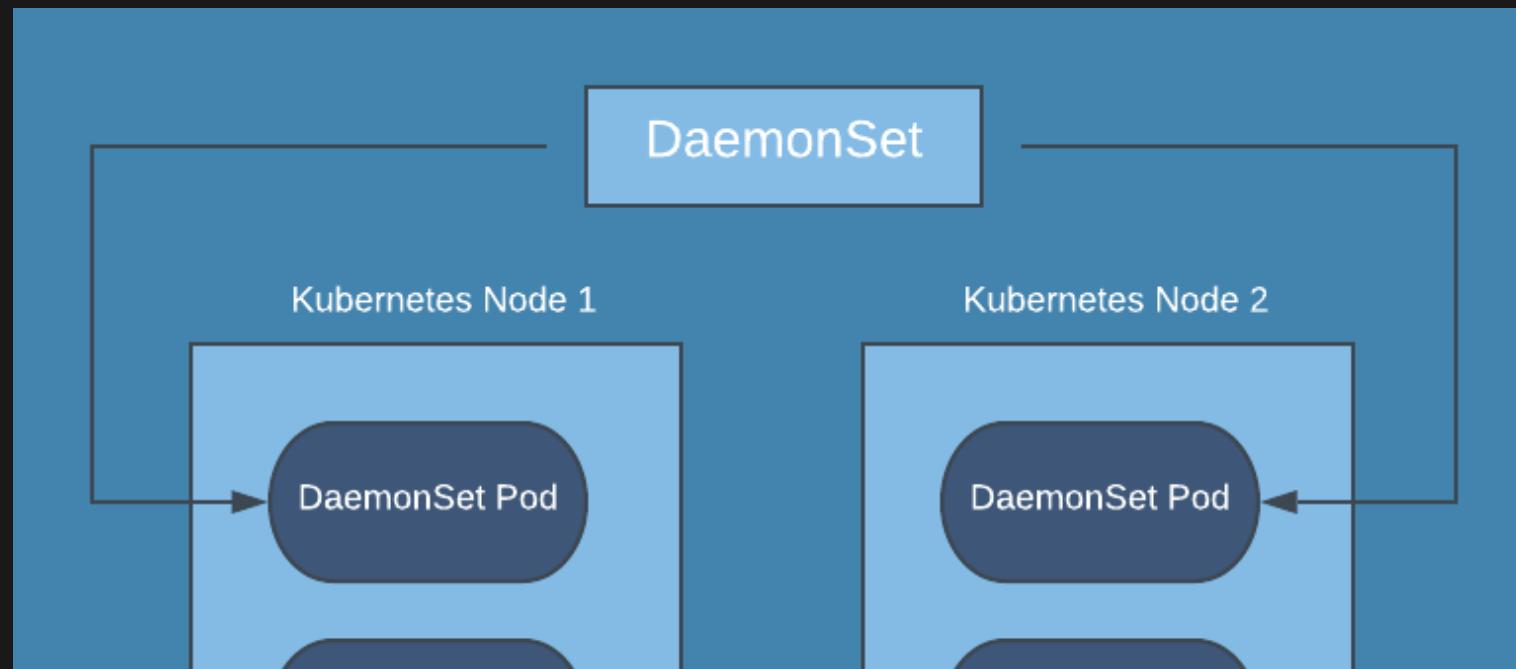
KUBERNETES API OBJEKTE

- Pod
 - kleinste deploybare Einheit
 - beinhaltet 1 bis N Container
 - eigener Netzbereich und IP

- ReplicaSet
 - Stellt sicher, dass zu jeder Zeit genau N Pods laufen
 - Matching über Labels
- Deployment
 - Managed stateless Pods
 - Managed ein ReplicaSet



- DaemonSet
 - Spec fast analog zu Deployment, u.a. ohne Replicas
 - Managed damit kein ReplicaSet
 - Stattdessen je ein Replica pro Node

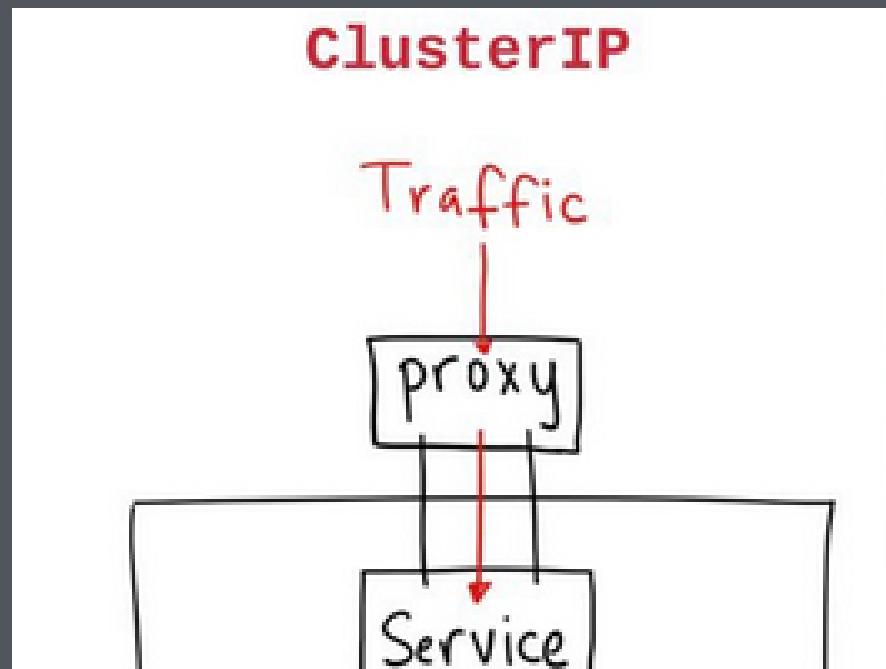


- Service
 - Loadbalancer für Pods (Layer 3/4)
 - Matching via Labels
 - Typen
 - None (headless)
 - ClusterIP (Default)
 - NodePort

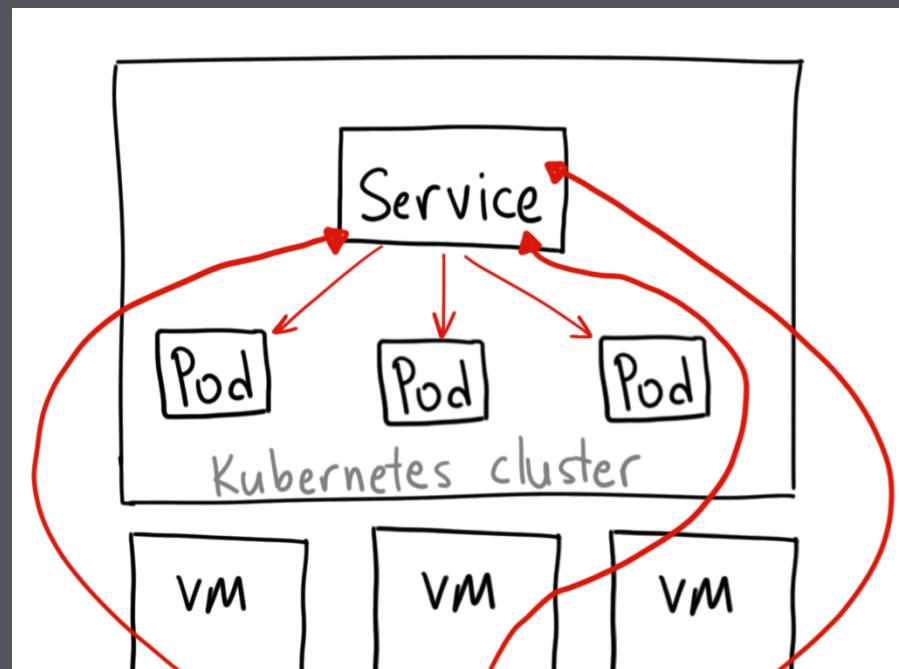
SERVICE TYPEN

- None (headless)
 - erstellt für jeden Pod einen DNS Eintrag innerhalb des Clusters (coredns)
 - kein externer Zugriff möglich

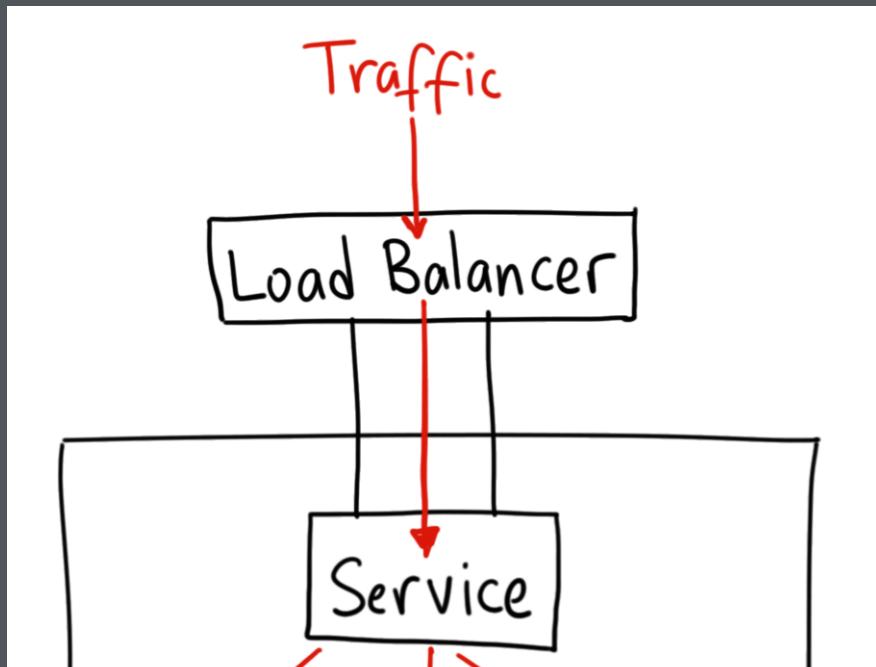
- ClusterIP
 - routet über die clusterinternen Pod IPs
 - kein externer Zugriff möglich



- NodePort
 - öffnet auf jedem Node einen zufälligen Port zwischen 30000 und 32767
 - ist auf allen Nodes immer derselbe
 - Traffic wird von dort an den Service weitergeleitet

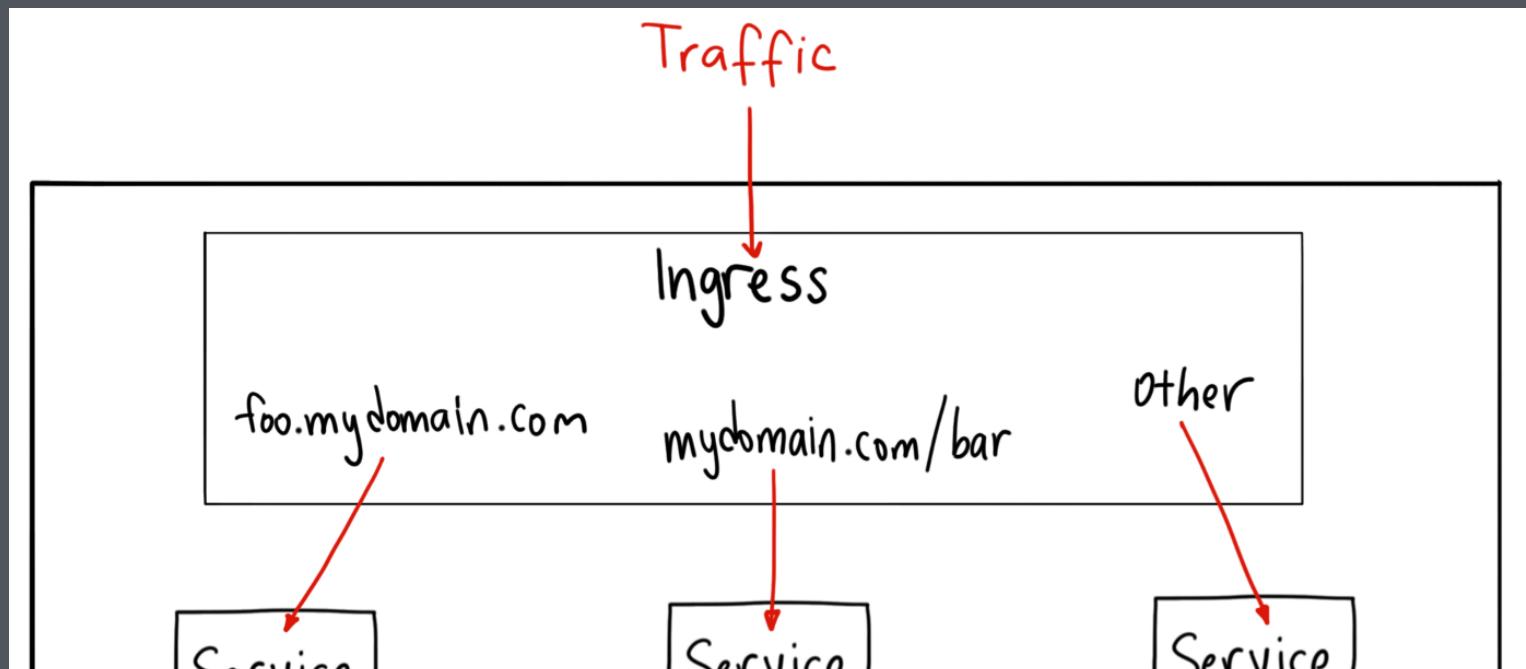


- Loadbalancer
 - macht den Service von außen zugänglich
 - bedarf eines Loadbalancers, der den Traffic zum Service routet
 - k8s bietet keine Standardimplementierung (Cloud Provider)
 - Kosten

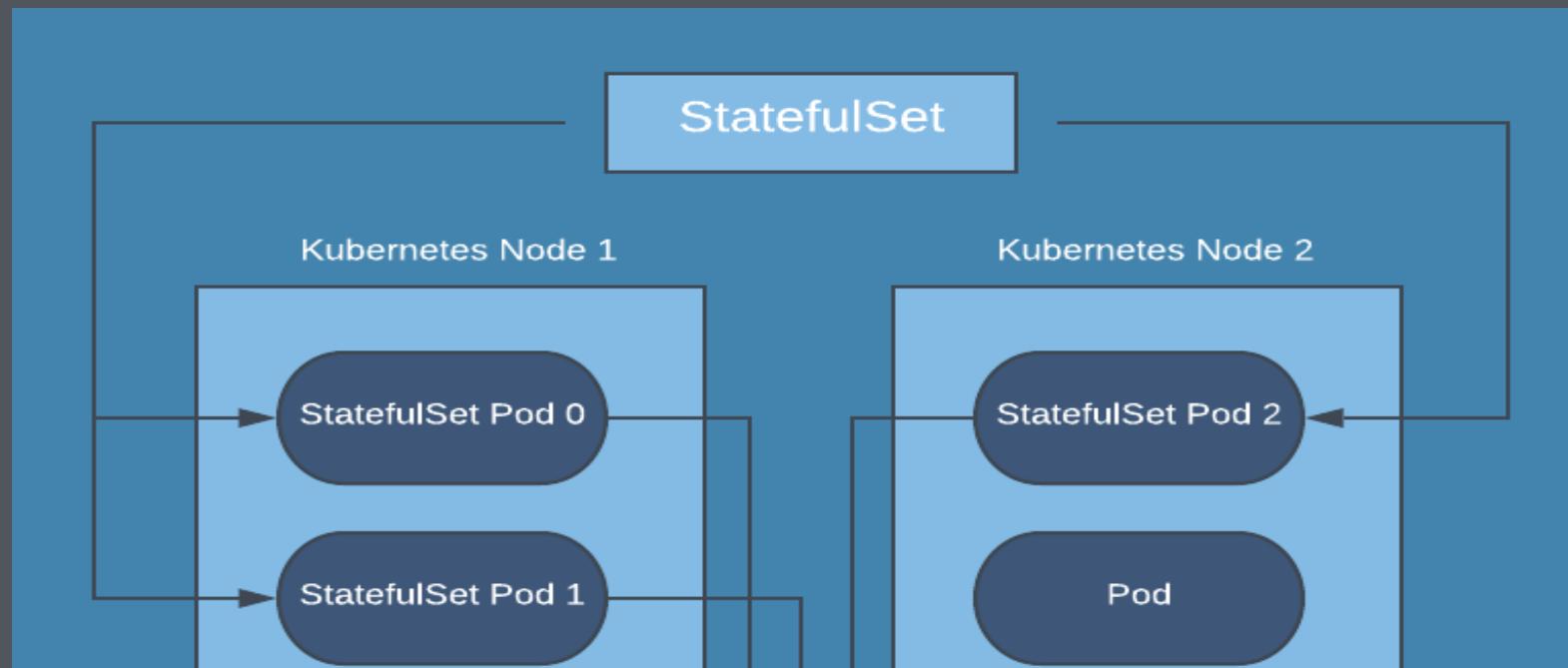


AUSBLICK: INGRESS

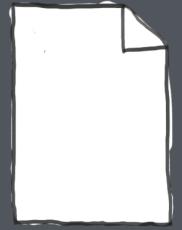
- HTTP level router (Level 7)
- Host und Path basiertes Routing
- Auch für verschiedene Hosts (SNI)
- TLS Terminierung möglich



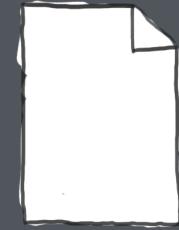
- StatefulSet
 - Managed stateful Pods
 - Spec ähnlich zu Deployment
 - Geordnetes Starten
 - Geordnetes Stoppen in umgekehrter Reihenfolge
 - Je Pod ein DNS Eintrag der Bauart POD-



- ConfigMap
 - Plain-Text Key-Value Store
 - Kann für Pods, Deployments, STSs und DSs definiert werden
- Secret
 - base64 encoded Data Store
 - Kann für Pods, Deployments, STSs und DSs



ConfigMap Definition



Pod Definition



Secret Definition

KUBERNETES TOOLS

- kubectl
 - CLI zur Interaktion mit k8s Clustern
- krew
 - kubectl Plugin Manager
- k9s
 - Terminal UI zur Interaktion mit k8s Clustern
- kind (*Kubernetes in Docker*)

- helm
 - Paket-Manager für Kubernetes
 - vgl. mit apt für Ubuntu oder apk für Alpine
- Lens
 - Graphical UI zur Interaktion mit k8s Clustern
 - Nicht Teil des Workshops

TAG 4

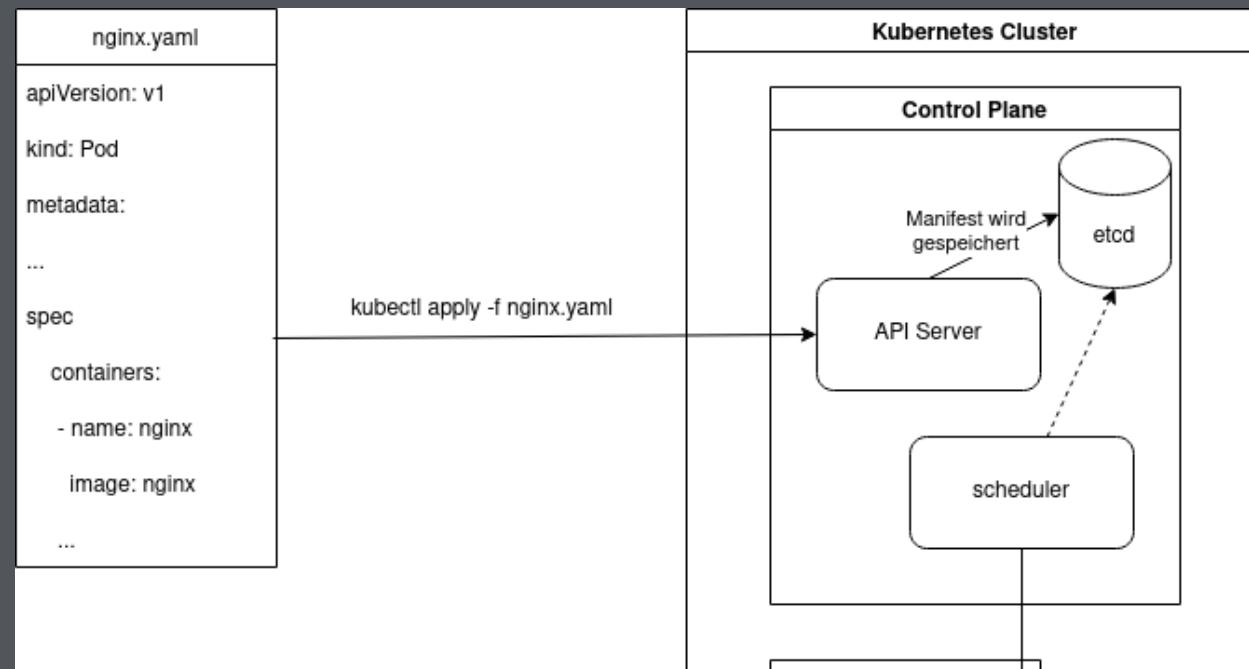
AGENDA

- Weitere API Objekttypen
- Admission Controller
- Sicherheit (RBAC)
- Helm
- Cert-Manager + Ingress
- MetalStack / Gardener / FCN
- Logging/Monitoring

WEITERE
OBJEKTYPEN IN
K8S

ERINNERUNG POD

- Kleinste deploybare Einheit
- Kann per Manifest werden



AUFGABE 1

- Fehlerhaftes Manifest `pod.yaml`
- Reparieren und in Namespace der Wahl deployen

LÖSUNG 1

```
apiVersion: v1 #Typo in apiVersion
kind: Pod #Typo
metadata:
  labels:
    app: frontend
  name: web
  namespace: ex1 #Optional
spec:
  containers: #Einrückung beachten
  - name: web #Listenelemente werden mit ` - ` eingeleitet
    image: nginx:latest #Image und Tag mit ` : ` getrennt
  ports:
  - containerPort: 80 #hier auch falsch eingerückt
  resources:
```

- Erstelle Pod Manifest:

```
export do='--dry-run=client -o yaml'  
kubectl run my-pod --image ubuntu:20.04 $do > pod.yaml
```

Dieses Manifest kann jetzt bequem angepasst/vervollständigt werden

- Erstelle Deployment Manifest:

```
export do='--dry-run=client -o yaml'  
kubectl create deploy --image alpine:3.15 my-deploy $do > dp.y
```

AUFGABE 2

- Erstelle ein Deployment (Objekt) des Pods aus vorangegangener Aufgabe
 - **Pod Manifest**
- Erstelle anschließend ein Service (Objekt) um die Pods zu loadbalancen
 - **Deployment Manifest**

LÖSUNG 2

- Erstelle **deployment.yaml**
 - `kubectl apply -f deployment.yaml -n web`
- Erstelle **service.yaml**
 - `kubectl apply -f service.yaml -n web`

PORT-FORWARDING

- Zugriff auf Container (Tunnel)
- Debugging
- via kubectl und k9s möglich

AUFGABE 3

- Deployment und Service aus letzter Aufgabe muss im selben Namespace deployed sein
 - `kubectl create ns web`
 - `kubectl apply -f deployment.yaml -n web`
 - `kubectl apply -f service.yaml -n web`
- Anschließend bitte ein Port-Forwarding zu den Pods des Deployments einrichten

LÖSUNG 3

- `kubectl -n web port-forward pod/web-6779b45f74-bvc7p :80`
 - lokaler, zufällig bestimmter Port wird an Pod Port 80 gebunden
- `kubectl -n web port-forward svc/web 8081`
 - lokaler Port 8081 -> von Service bestimmter Service Port 8081

DAEMONSET

- Jeder Node bekommt ein Replica
 - Log-Shipper
 - Monitoring Agent

AUFGABE 4

- Deploye ein DaemonSet mit einem nginx Pod in einen Namespace deiner Wahl
- Scale das DaemonSet auf 3 Pods
 - ist dies Möglich?
 - Warum? Warum nicht?

LÖSUNG 4

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: web-ds
  labels:
    app: web-ds
spec:
  selector:
    matchLabels:
      name: web-ds
  template:
    metadata:
      labels:
        name: web-ds
```

- Scaling ist nicht möglich, da DaemonSets mit den Nodes scalen

STATEFULSET

- Persistente Pods
- Geordnetes Update/Shutdown

JOB

- Einmalige Ausführung eines Commands in einem Pod
 - Datenbank Backup

AUFGABE 5

- Erstelle einen Job, welcher einmalig die Zahl Pi auf 5000 Stellen genau berechnet.
- gebe Pi aus

LÖSUNG 5

- Von Kubernetes Doku das Manifest übernehmen und anpassen

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      labels:
        job: pi
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi("
```

- kubectl get logs -n ex7 pi-

CRONJOBS

- Mischung aus klassischen CronJobs und Jobs
- Regelmäßige Ausführung eines Jobs
 - Datenbank Backups

AUFGABE 6

- erstelle einen CronJob welcher minütlich das Datum und deinen Namen ausgibt
- dieser Cronjob soll 5 erfolgreiche und 8 fehlgeschlagene Versuche behalten
- teste diesen CronJob ohne eine Minute zu warten

LÖSUNG 6

- aus kubernetes Doku Manifest kopieren und anpassen

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  successfulJobsHistoryLimit: 5
  failedJobsHistoryLimit: 8
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            cronjob: hello
```

- Erstellen eines einmaligen runs

```
kubectl create job -n ex8 --from=cronjob/hello hello
```

- Output wieder sichtbar mit

```
kubectl logs -n ex8 hello-
```

CONFIGMAPS

- Speicherung von nicht vertraulichen Daten
- Einbindung in Pods als
 - Umgebungsvariable
 - command-line argument
 - Datei (Volume)
- Kein Reload der Pods bei Änderung

AUFGABE 7

- dieses Deployment möchte eine ConfigMap einbinden
 - [Deployment.yaml](#)
- diese ConfigMap
 - [configmap.yaml](#)
- ändere die WorkerConnection und deploye die beiden Ressourcen

LÖSUNG 7

- WorkerConnection in Zeile 13 Updaten,
anschließend zuerst die Configmap deployen:

```
kubectl apply -f configmap.yaml -n ex9
```

- anschließend das Deployment Objekt deployen

```
kubectl apply -f deployment.yaml -n ex9
```

- Wichtig! Beides in den gleichen Namespace

SECRET

- Speicherung vertraulicher Daten
- Unverschlüsselt in etcd DB
- Bessere Separierung mittels Rollen
 - User darf Configmaps sehen aber keine Secrets

PERSISTENTVOLUME (PV)

- kapselt Storage in einer API
- sehr viele Volume Typen
 - NFS share, iSCSI, Host-Path, emptyDir, ...
 - Lightbits, S3 und lokal bei FI-TS
- Vorab oder dynamisch (StorageClass) erstellt
- Kann von Pods via PersistentVolumeClaims (PVC) angefordert werden

ZUGRIFFSTYPEN

- ReadWriteOnce
 - Once, nur ein Node darf auf das Volume schreiben
- ReadWriteMany
 - Many, mehrere dürfen
- ReadOnlyMany
 - mehrere Nodes können das Volume ReadOnly mounten

AUFGABE 8

- Erstelle ein local PV mit 10 GB Capacity
- Erstelle das Verzeichnis auf der Node
- Dieses soll ReadWriteOnce sein
- Dieses muss einen eindeutigen Namen haben

LÖSUNG 8

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
  labels:
    storage: local

spec:
  storageClassName: standard
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
```

PERSISTENTVOLUMECLAIM (PVC)

- Reserviert Ressourcen eines PV`s
- wird anschließend ins Deployment eingebaut
- Verknüpfung PV und PVC mit Selector labels oder direkt mit Namen
 - bei local kein dynamisches (selector) mapping möglich
- Verknüpfung ist eine 1 zu 1 Verknüpfung

AUFGABE 10

- Erstelle ein PVC
- Erstelle ein postgresql statefulset
 - Tipp: Configmap und Secret müssen auch erstellt sein um env Variablen in den Container zu übergeben
- Welches das PVC einbindet
- Lasse die Daten welche in der DB sind anzeigen

LÖSUNG 10

- Der PV der letzten Aufgabe muss erstellt sein

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pv-claim
  labels:
    app: postgres
spec:
  storageClassName: standard
  capacity:
  accessModes:
```

- Configmap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-configuration
  labels:
    app: postgres
data:
  POSTGRES_DB: topdb
  POSTGRES_USER: user23
```

- StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-statefulset
  labels:
    app: postgres
spec:
  serviceName: "postgres"
  replicas: 1
  selector:
```

- Daten anzeigen lassen

```
kubectl exec -n postgresql -it postges-statefulset-0 -- /bin/psql -U user23 topdb
```

RESOURCE QUOTAS

- Limitiert die Ressourcen (CPU/Memory/Anzahl Pods/...) für Pods auf NS Ebene
- LimitRange einsetzen für Min/Max und default Werte
- Vorsicht bei zu knapper Bemessung

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-rq
  namespace: my-namespace

spec:
  hard:
    requests.cpu: "1"
    requests.memory: "1Gi"
    limits.cpu: "2"
    limits.memory: "2Gi"
```

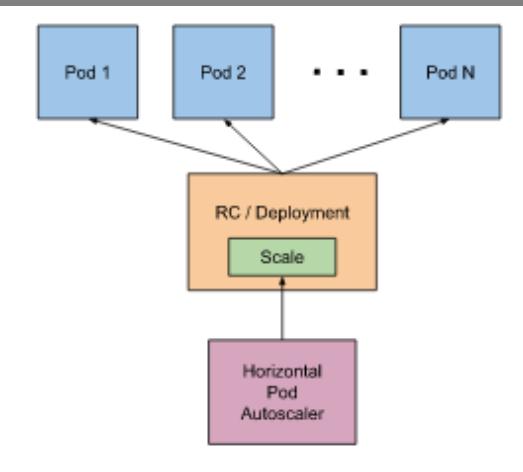
```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-cpu-lr
  namespace: my-namespace
spec:
  limits:
  - default:
      memory: "512Mi"
    defaultRequest:
      memory: "256Mi"
  type: Container
```

VERTICAL POD AUTOSCALER

- Passt Ressourcen-Nutzung (CPU/Memory) der Pods automatisch an
- Effiziente Nutzung der Nodes
- Löst das Problem mit den ResourceQuotas/LimitRange
- Noch nicht für Produktion empfohlen
 - wegen möglicher eingeschränkter Sichtbarkeit

HORIZONTAL POD AUTOSCALER

- Automatisches Rescaling von Deployments
 - nach CPU Auslastung
 - nach Custom Metriken (v2)



HPA ERSTELLEN

```
kubectl autoscale deployment nginx --cpu-percent=50 --min=1 --  
  
apiVersion: autoscaling/v1  
kind: HorizontalPodAutoscaler  
metadata:  
  
  name: nginx  
spec:  
  scaleTargetRef:  
    apiVersion: apps/v1  
    kind: Deployment  
    name: nginx  
  minReplicas: 1  
  maxReplicas: 10  
  targetCPUUtilizationPercentage: 50
```

PROBES

- Überwachung und Einschätzung der Pods
- LivenessProbe
 - Wann ist ein Pod gestartet und gesund?
- ReadinessProbe
 - Wann kann ein Pod Traffic entgegennehmen?
 - z.B. wenn Pod von anderen Pods noch abhängt
- StartupProbe

BEISPIEL

```
startupProbe:  
  httpGet:  
    path: /health  
    port: 80  
  failureThreshold: 30  
  periodSeconds: 10
```

BEISPIEL

```
livenessProbe:  
  exec:  
    command:  
    - cat  
      - /tmp/health  
  initialDelaySeconds: 5  
  periodSeconds: 5  
  
readinessProbe:  
  httpGet:  
    path: /  
    port: 80
```

SICHERHEIT

Role Based Access Control (RBAC)

- Authentifizierung (Wer bin ich?)
 - Analogie Ausreise: Perso
- Autorisierung (Was darf ich?)
 - Analogie Ausreise: Visa
- Admission Control
 - Stellt Authentifizierung und Autorisierung sicher
 - Analogie Ausreise:

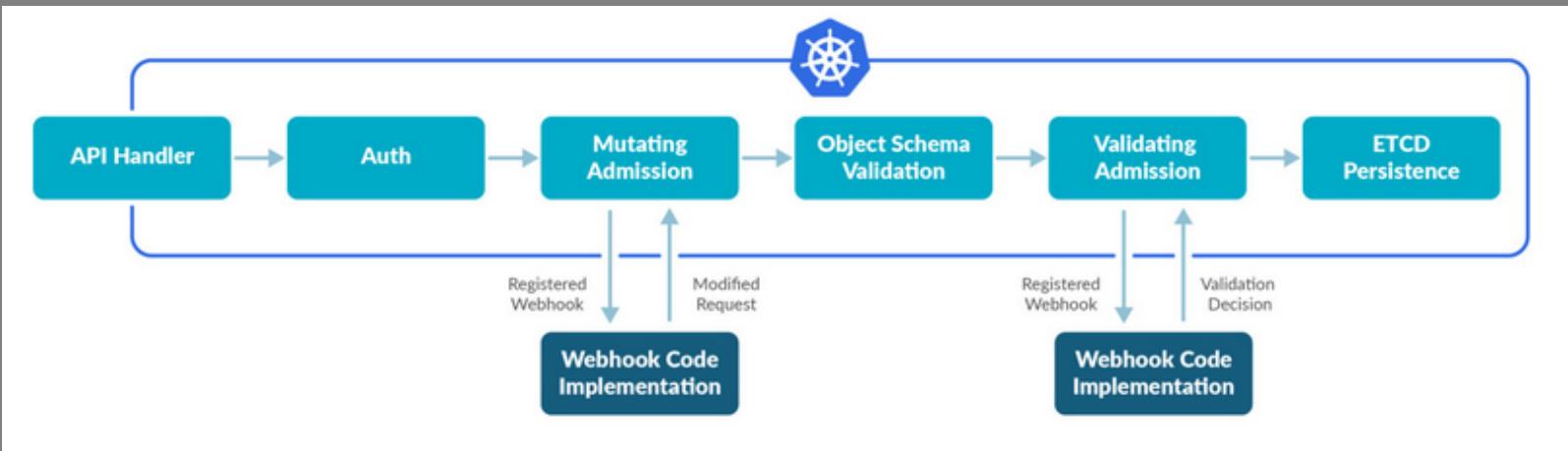
MODELL

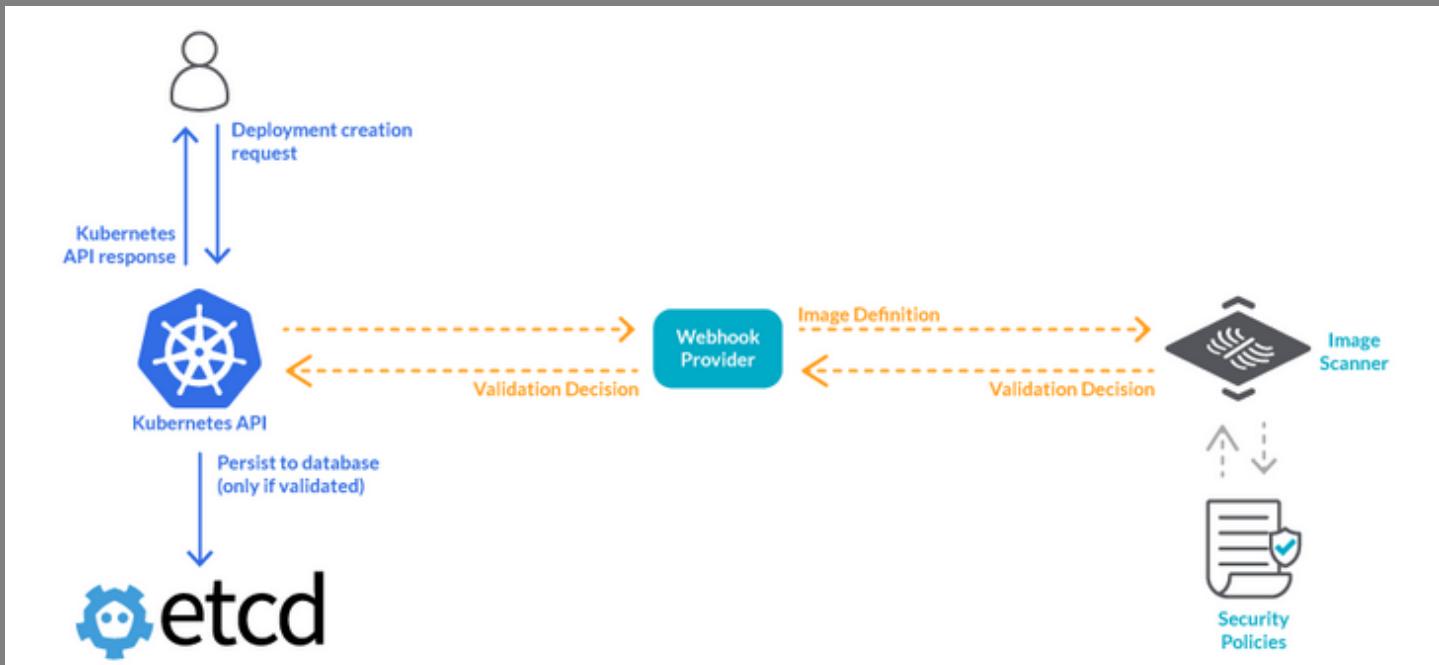
- ServiceAccount (SA) / User
 - technischer / User Account im Cluster
- (Cluster)Role
 - Feingranulare Berechtigungen auf Ressourcen
 - get, create, read, watch, ...
 - pod, secret, configmap, deployment, ...
- (Cluster)RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader

  namespace: my-ns
rules:
- apiGroups: [ "" ]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

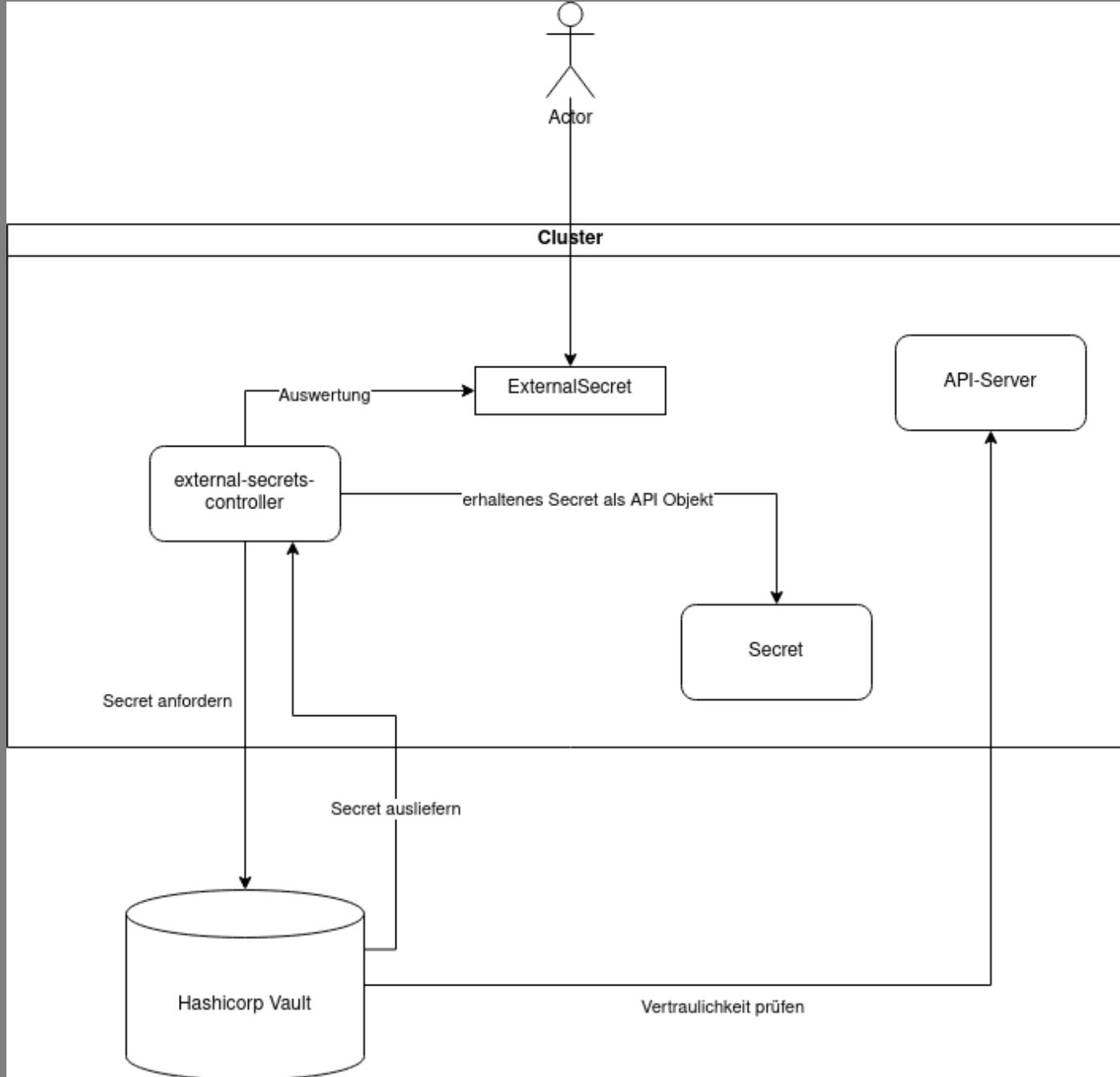
```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: my-ns
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io` ```
---
```





CUSTOMRESOURCEDEFINITION

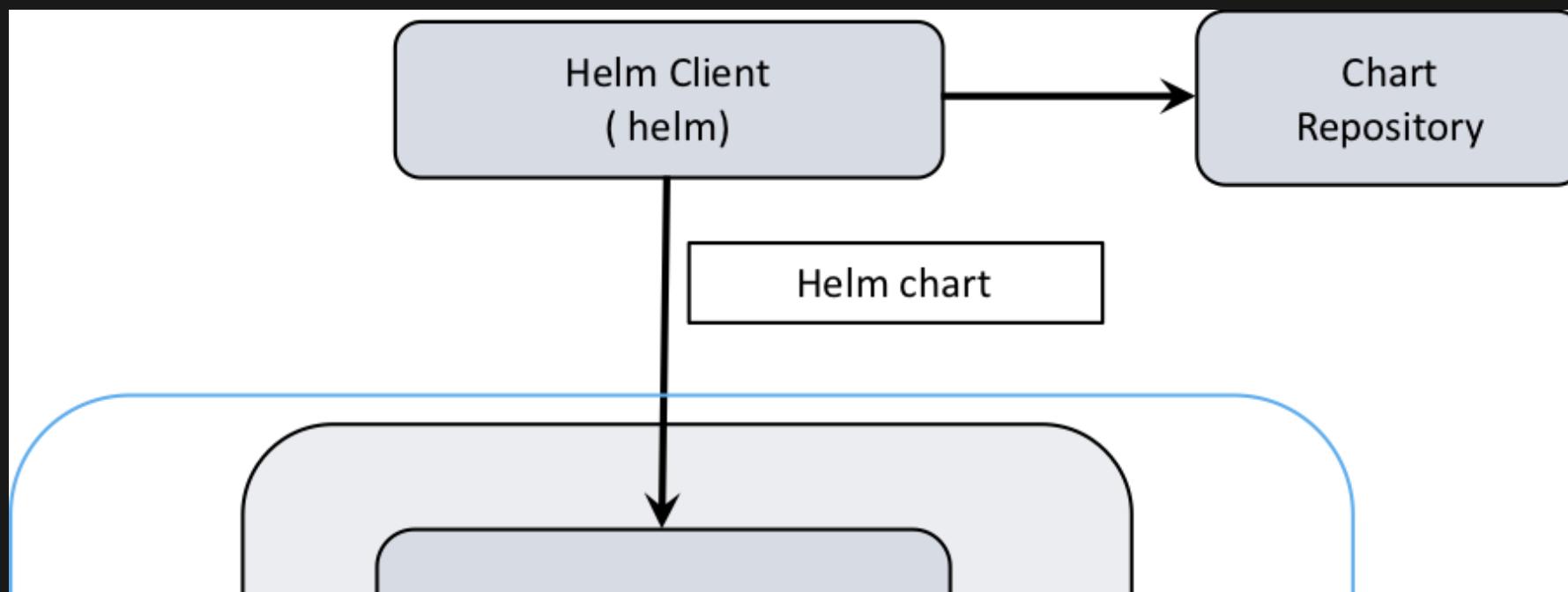
- Erfordert eigenen Admission Controller
- Damit selbst definierbare Manifste möglich
- Beispiele
 - external-secrets-controller
 - cert-manager
 - anwendungsspezifische Funktionalitäten



HELM

- Package Manager für Kubernetes
- Gegliedert in sogenannten Charts
- Große Softwarehersteller schreiben eigene Helm Charts
 - z.B. Gitlab

- Praktisch, um eine Anwendung mit wenigen Änderungen in verschiedenen Umgebungen zu deployen
 - test/staging/production
- Helm Charts sind in sogenannten Repos gespeichert
 - Chart Ersteller haben meistens eigene Repos
 - Nutzung ähnlich wie bei apt in ubuntu



AUFBAU EINES HELM CHARTS

```
name
├── charts
├── Chart.yaml
└── templates
    ├── deployment.yaml
    |
    ├── _helpers.tpl
    ├── hpa.yaml
    ├── ingress.yaml
    ├── NOTES.txt
    ├── serviceaccount.yaml
    ├── service.yaml
    └── tests
        └── test-connection.yaml
└── values.yaml
```

AUFBAU EINES HELM CHARTS

- Das Meiste spielt sich im templates-Ordner ab

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "schulung.fullname" . }}
  labels:
    {{- include "schulung.labels" . | nindent 4 }}
spec:
{{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
{{- end }}
```

AUFBAU EINES HELM CHARTS

- wie so oft im yaml-Format
- das Meiste bis alles templates
- Anpassungen in der values.yaml

```
# Default values for schulung.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
    repository: nginx
    pullPolicy: IfNotPresent
    # Overrides the image tag whose default is the chart appVersion.
    tag: ""

imagePullSecrets: []
nameOverride: ""
```

- Beispiel an Container part des Deployment template

```
containers:
- name: {{ .Chart.Name }}
  securityContext:
    {{- toYaml .Values.securityContext | indent 12 }}
  image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
  imagePullPolicy: {{ .Values.image.pullPolicy }}
  ports:
    - name: http
      containerPort: 80
      protocol: TCP
  livenessProbe:
    httpGet:
      path: /
      port: http
```

```
image:
  repository: nginx
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion
  tag: ""

resources: {}

  # We usually recommend not to specify default resources and
  # choice for the user. This also increases chances charts run
  # resources, such as Minikube. If you do want to specify res
  # lines, adjust them as necessary, and remove the curly brac
  # limits:
  #   cpu: 100m
  #   memory: 128Mi
```

HELM COMMANDS

- helm install
 - Installiert ein Helm Chart
 - mit "-n namespace" angebar
 - mit "--dry-run --debug" kann man überprüfen ob das deployment klappen sollte
 - mit "--version" Versionspinning
 - Syntax `helm install -n NAMESPACE`

- helm upgrade
 - Upgraden eines Helm Charts auf neue Revision
 - "--install" wichtiges Flag. Macht, dass Chart installiert wird wenns nicht da ist
 - mit "--version" Versionspinning
- helm create
 - Erstellen eines Helm Charts

- helm uninstall
 - Deinstalliert ein Chart und löscht alle Ressourcen
- helm rollback
 - Zurückspielen der alten Version des Helm Charts
- helm list
 - Zeigt installierte Helm Charts
 - entweder mit " -A" für alle Namespaces oder " -n"

- helm repo
 - add
 - Hinzufügen eines Repos
 - z.B. "helm repo add bitnami"
 - update
 - Herunterladen, welche Charts im Repo sind
 - z.B. in bitnami gibt es ein postgresql Chart

AUFGABE 11

1. erstelle ein Helm Chart für ein nginx deployment mit Service
2. deploye dies in einen Namespace deiner Wahl

LÖSUNG 11

1. Erst das Chart erstellen

```
helm create NAME  
helm create nginx-deployment
```

2. dann installieren

```
helm install -n NAMESPACE RELEASE_NAME PFAD_ZUM_HELM_CHART  
helm install -n helm-namespace nginx-deployment ./nginx-depl
```

AUFGABE 12

1. Passe die Replicas mit helm an
2. Verifiziere, dass mehr Pods laufen

LÖSUNG 12

1. Dann die values.yaml anpassen und upgrade

```
helm upgrade -n NAMESPACE RELEASE_NAME PFAD_ZUM_HELM_CHART  
helm upgrade -n helm-namespace nginx-deployment ./nginx-depl
```

2. Mit "kubectl" oder "k9s" anzeigen, dass die angegebenen Pods vorhanden sind

```
kubectl get pods -n NAMESPACE  
kubectl get pods -n helm-namespace
```

AUFGABE 13

1. Mache ein Rollback auf eine alte Helm-Version

LÖSUNG 13

1. Mit "helm rollback" auf alte Revision gehen

```
helm rollback -n NAMESPACE RELEASE_NAME REVISION  
helm rollback -n helm-namespace nginx-deployment 1
```

ZUSAMMENFASSUNG HELM

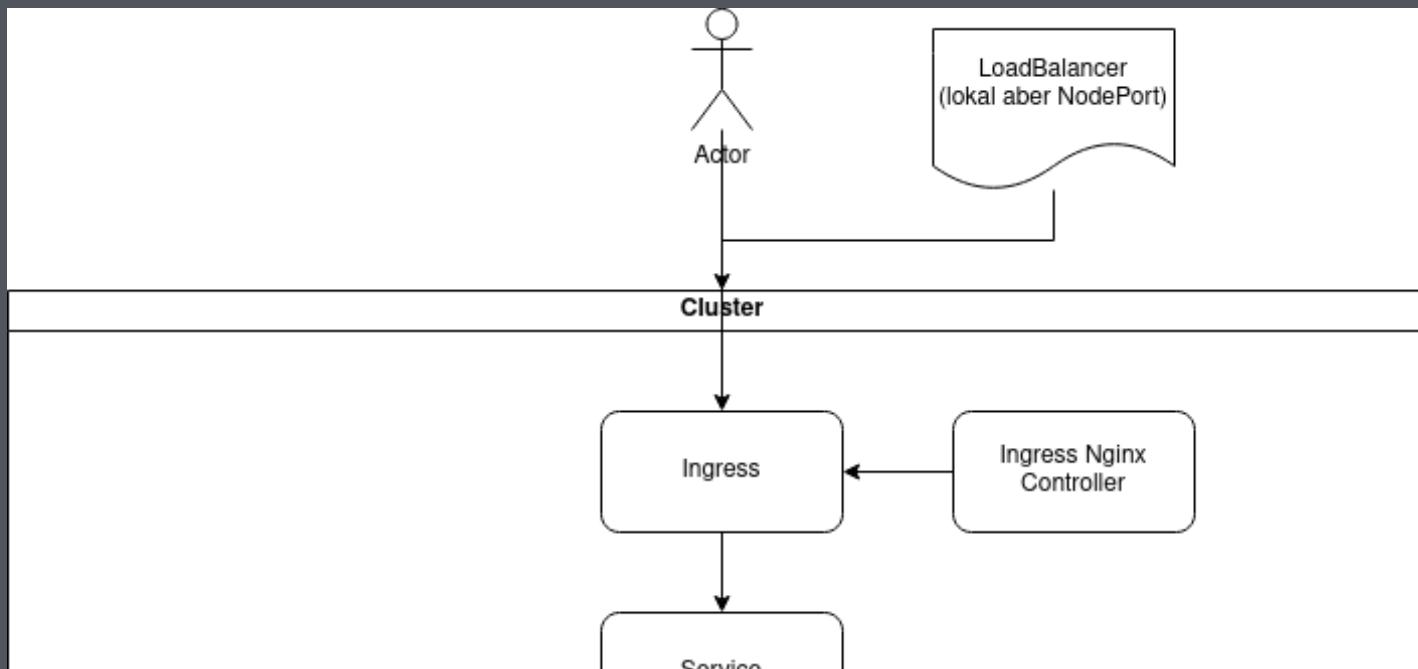
- Ist ein Packetmanager
- arbeitet mit Templates
- eine zentrale Datei (values.yaml) um komplexe Anwendungen zu deployen
- wird in Repos verwaltet

INGRESS

- HTTP layer 7 router
- Host und Path basiertes Routing (HTTP/HTTPS)
- TCP Routing von Ports != 80/443 über
- Erfordert Installation Ingress Controller
 - traefik
 - ingress-nginx

INGRESS ÜBUNG

- Installiere letsencrypt cert manager via Helm
- Installiere ingress-nginx-controller
- Mache den nginx Server nach außen hin via HTTPS mit Zertifikat zugreifbar via Ingress
 - Weil kein LoadBalancer zur Verfügung steht, ändere den ingress-nginx Service in den Typ NodePort



METALSTACK

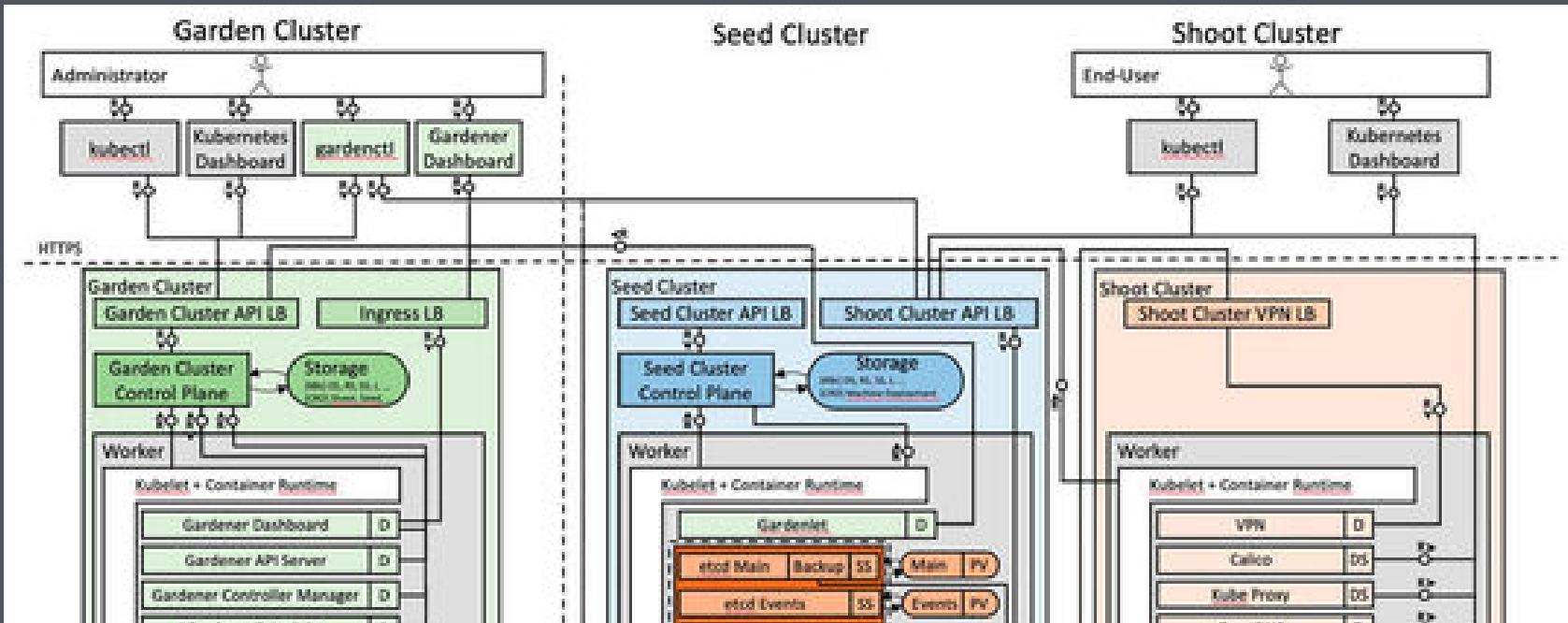
- von der x-cellent für die FI-TS entwickelte bare-metal cloud
- OpenSource
- Server-Racks in drei Rechenzentren (Nürnberg, Stuttgart, Fellbach)
- ersetzt Nimbus-Cloud
- schon zwei Jahre produktiv im Einsatz

GARDENER

- Kubernetes-as-a-Service (KaaS) von SAP (Open Source)
- Eine Abstraktionsebene höher
 - Shoot-Cluster ~ Pod
 - beinhaltet die tenant Cluster
 - Seed-Cluster ~ Node
 - beinhaltet control planes der tenant Cluster

- Garden Cluster
 - Verwaltet die Seed-Cluster
 - Seed-Cluster isoliert lauffähig
 - Umgekehrung der Kommunikation
- Unterstützt alle großen CloudProvider, und...

SAP Gardener beinhaltet ein vollständig validiertes Erweiterungs-[Framework](#), welches sich an jeden beliebigen programmatischen Cloud- oder Infrastrukturdiensst anpassen lässt. Gardener kann derzeit den gesamten Lebenszyklus konformer Kubernetes-Cluster auf AliCloud, AWS, Azure, GCP, [OpenStack](#), Packet, MetalStack und vSphere in einer as-a-Service-Bereitstellung mit minimalen Gesamtbetriebskosten verwalten.



FINANCE CLOUD NATIVE (FCN)

- Bietet cloudctl als zentrales Verwaltungstools von Tenant-Clustern
- Bereitstellung (via Gardener -> MetalStack)
- Netzwerkeinrichtung
- Update
 - Kubernetes Version der einzelnen Nodes (auto), Network

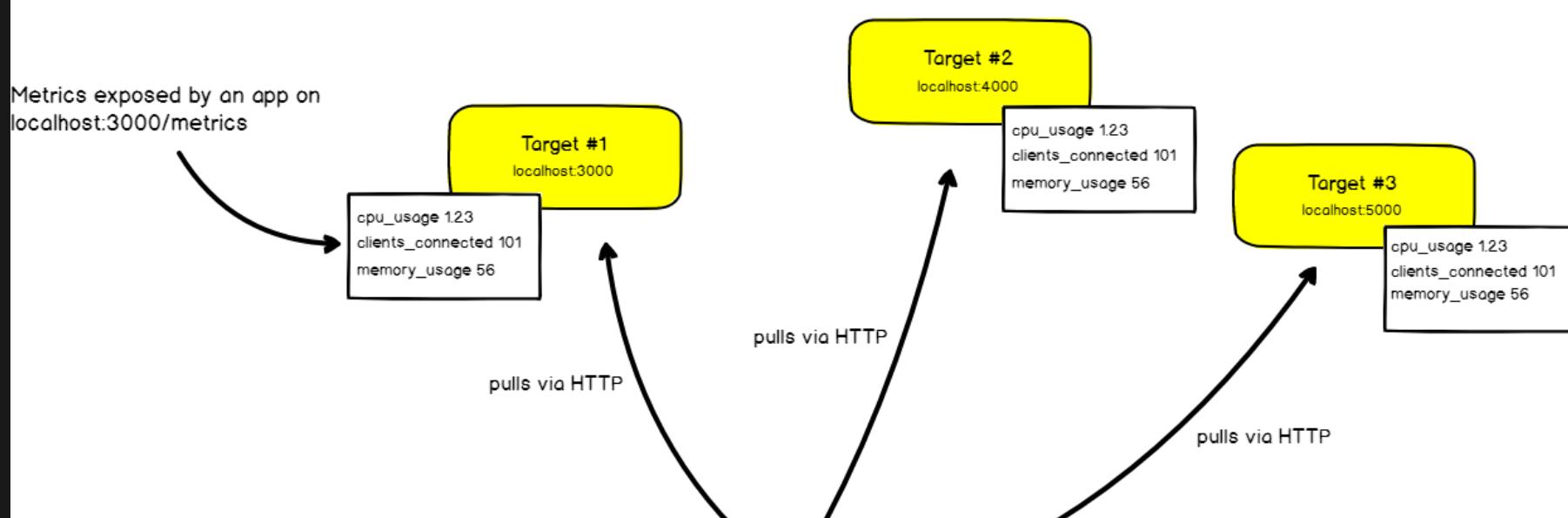
MONITORING UND LOGGING

PROMETHEUS

- time series Database
- speichert metric daten einzelner services
- Abrufen der daten mittels PromQL

```
sum(rate(container_cpu_usage_seconds_total{container!=""} [5m])) by (name)
```

What does Prometheus do?

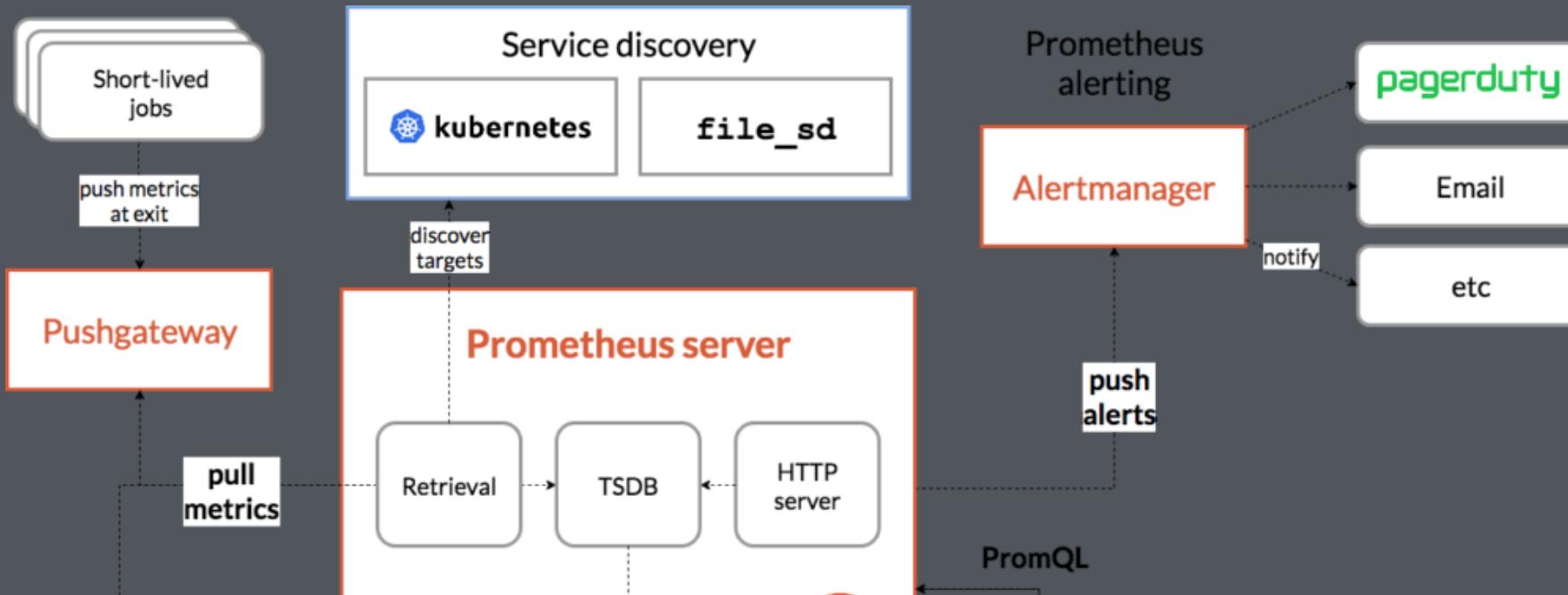


ALERTMANAGER

- Alamiert nach erstellten Vorgaben
- Diverse endpoints lassen sich hinzufügen
 - Email
 - div. Instant messenger
 - slack
 - MS teams
 - Telegram

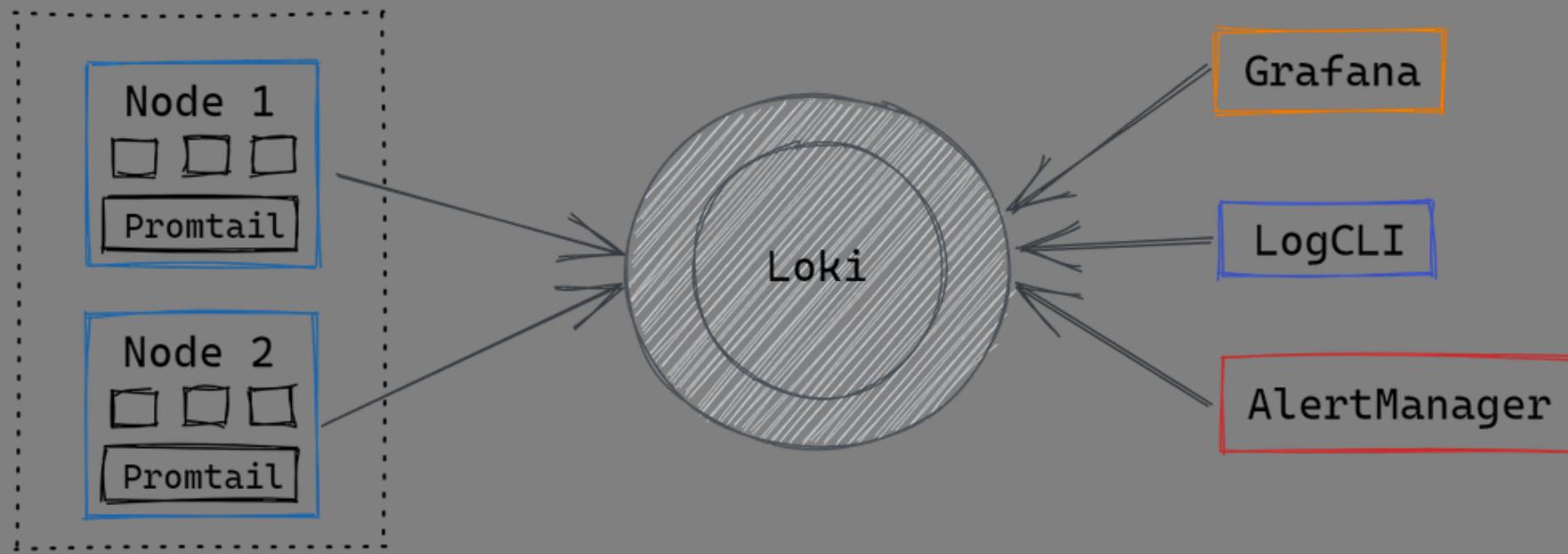
GRAFANA

- liest daten von Datensammelstationen
 - Prometheus
 - Loki
 - MSSQL
 - InfluxDB
 - viele weitere [DataSources](#)
- erstellt Grafiken



LOGGING MIT LOKI UND PROMTAIL

- auch loki-stack genannt
- log aggregation system
- indexiert nicht den Inhalt der Logs
- indexiert Metadaten der Logs
- Integriert sich mit Prometheus und Grafana
 - ununterbrochener Wechsel zwischen Logs und Metrics möglich



AUFGABE

- installiert mit Helm Prometheus, Grafana, loki-stack(lokistack und promtail)
- in grafana Datasources Prometheus und Loki hinzufügen
- erstellt ein Dashboard welches den Ressourcenverbrauch anzeigt
- erstellt ein Log Dashboard

LÖSUNG

- für Prometheus:
 - helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>
 - helm repo update
 - helm install -n NAMESPACE prometheus-community prometheus-community/prometheus

- für Loki-stack (installiert loki und promtail):
 - ist auch in grafana helm repo
 - helm install -n NAMESPACE loki grafana/loki-stack

- um das grafana Dashboard erreichbar zu machen braucht man forward
 - `export POD_NAME=$(kubectl get pods --namespace NAMESPACE --selector "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=grafana" -o jsonpath=".items[0].metadata.name")`
 - `kubectl --namespace NAMESPACE port-forward $POD_NAME 3000:3000`
 - anschließend grafana auf localhost:3000 erreichbar

- Datasources auf der linken Seite bei Configuration
 - prometheus braucht in diesem Fall nur die URL:
dieses Schema: <http://PROMETHEUS-SERVER-SERVICE:PORT>
 - loki braucht ebenfalls nur URL: diese Schema:
<http://LOKI-SERVICE:PORT>

- Auslastungsdashboard erstellen:
 - auf grafischer Oberfläche entweder ein Dashboard mit PromQL erstellen
 - oder auf grafischer Oberfläche ein Dashboard importieren
 - [Dashboard](#)
- Log Dashboard erstellen

LITERATUR

- Kubernetes Up & Running
- Kubernetes Best Practices
- Online

FRAGEN

- Habt ihr noch Fragen an uns?