

Kubernetes Basis-Schulung

Dozenten

Dominik Bittl

- DevOps Consultant
- Hobbies: Mein Hund, 3D-Druck



Sören Jentzsch

- IT Freelancer
- Kubernetes / Cloud Native Engineer
- Hobbies: Laufen, Zocken (AoE2, SC2) :-)



Vorstellungsrunde :-)

Vorstellungsrunde

Erfahrungen

Z.B. was hast du mit Kubernetes schon gemacht?

Aufgaben

Was sind deine Aufgaben im Job?

Erwartungen

Mit was oder wie moechtest du den Kurs verlassen?

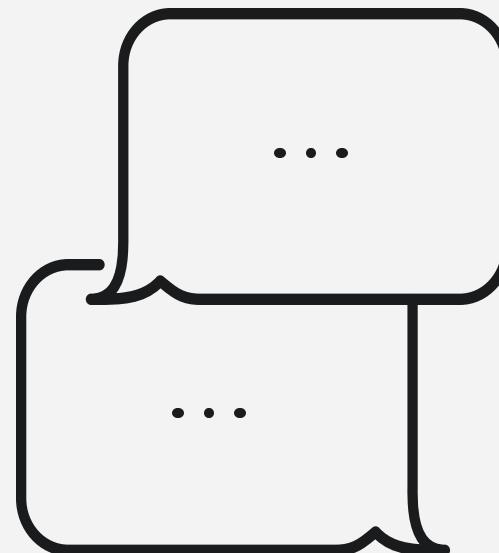
Hobbys

Was machst du in deiner Freizeit gerne?

Organisatorisches

Fragen?

Bitte jederzeit stellen!
Im Teams die Hand heben.



Organisatorisches

Zeiteinteilung

- 09:00 bis 17:00 Uhr
- Es wird genuegend Pausen geben!
- Mittag: 12:00 bis 13:00 Uhr?!



Agenda

1. Grundlagen
2. K8s Architektur
3. K8s Objekte
4. Paketierung und Auslieferung (z.B. Helm)
5. Tools
6. Ausblick

Grundlagen



Agenda

1. Warum Container
2. Container Grundlagen
3. Was ist Kubernetes?
4. Warum Kubernetes?
5. CNCF Landscape

Warum Container?



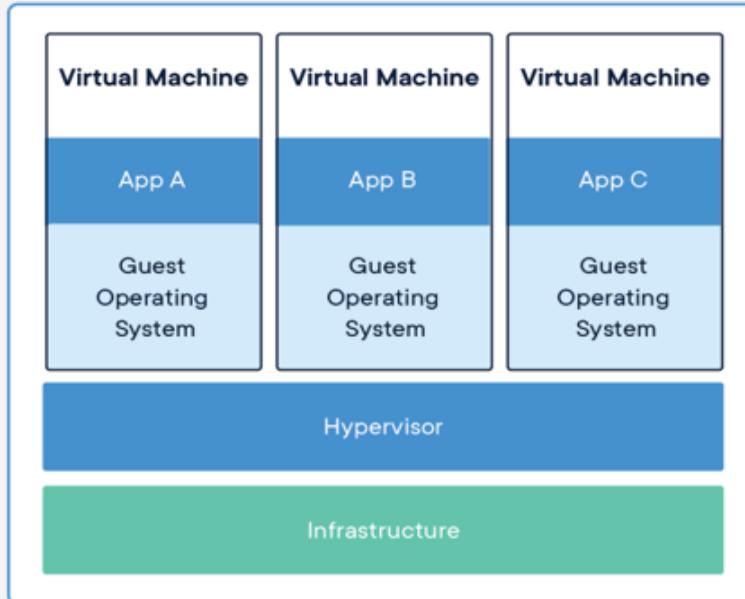
Warum Containerisierung?

Vorteile Containerisierung:

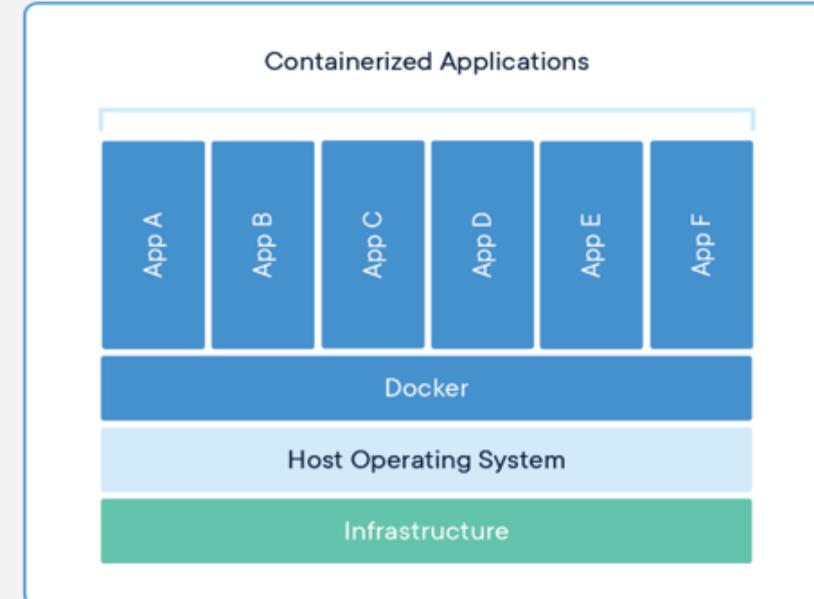
- Geringerer Ressourcenverbrauch (als VM oder klassischer Server)
- Skalierbar und wiederverwendbar
- Ideal für Microservices
- Einfach auf- und abbaubar
- Immer gleiche Umgebung für die Applikation
- Alle Abhängigkeiten sind mit dabei
- Einfache Auslieferbarkeit
- Untereinander und auf dem Host isoliert und bieten dadurch schon eine grundlegende Sicherheit

Unterschied VM vs. Container

VM



Container



Container Grundlagen

Dockerfile

Hier wird beschrieben, was alles in ein Image rein soll

Image

das erfolgreich gebaute Dockerfile als Gesamtbild übereinander gestapelter Layer

Layer

Jede Instruktion im Dockerfile resultiert in einem Layer

Container-Registry

Applikation, in der Images hochgeladen und heruntergeladen werden können (Speicher für Images)

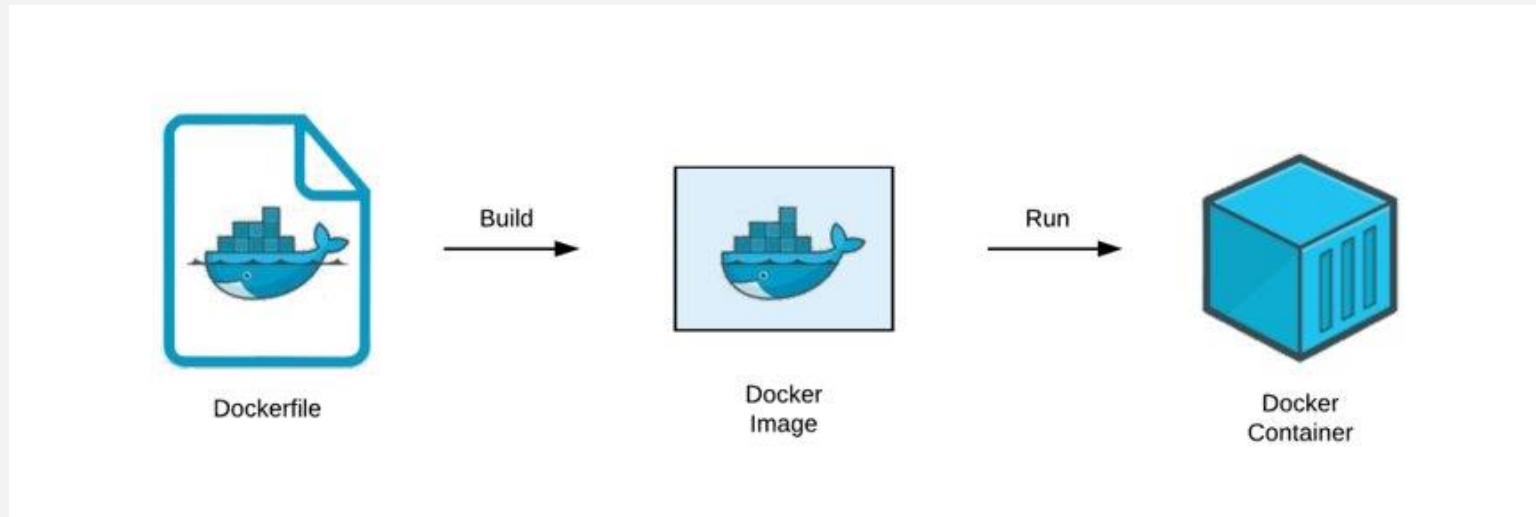
Container

ein Image, das gestartet wurde (Aus einem Image können beliebig viele Container gestartet werden)

Volumes

ein Dateisystem, das in einen Container gemountet werden kann

Dockerfile -> Image -> Container



Aufgabe Container (Optional)

```
# Dateien
```

```
https://github.com/x-cellent/Kubernetes-Basis-Schulung/tree/main/Docker
```

```
# Build Image
```

```
docker build -t nginx-test .
```

```
# Run Container
```

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

```
# Delete Container
```

```
docker rm -f nginx-test-container
```

```
# Run Container + Volume
```

```
docker run -d -p 80:80 --name nginx-test-container -v $PWD/nginx-volume:/usr/share/nginx/html nginx-test
```

```
# Access Container
```

```
docker exec -it nginx-test-container /bin/bash
```

Monolithische Architektur vs. Microservices

Früher:

- Monolithische Software-Architekturen
- Ein Programm bzw. Codebasis ist für alles zuständig

Heute:

- Trend geht dahin, Microservices zu verwenden
- Programm wird in verschiedene Teilmodule aufgeteilt, die getrennt voneinander laufen
- Sie kommunizieren über bestimmte Schnittstellen untereinander

Microservices Vor- und Nachteile



Vorteile

- Microservice kann sich auf seine Aufgabe konzentrieren
- Detail-Änderungen nur an einem Microservice, nicht an ganzer Applikation
- Die Komponenten können je nach Frequentierung unterschiedlich skaliert werden
- Einzelne Microservices können unabhängig voneinander deployt werden

Nachteile

- Ggf. komplexer (Auslieferung, Kommunikationsbeziehungen)
- Ist in manchen Fällen zu viel Overhead (z.B. für kleine Projekte)

Was ist Kubernetes?

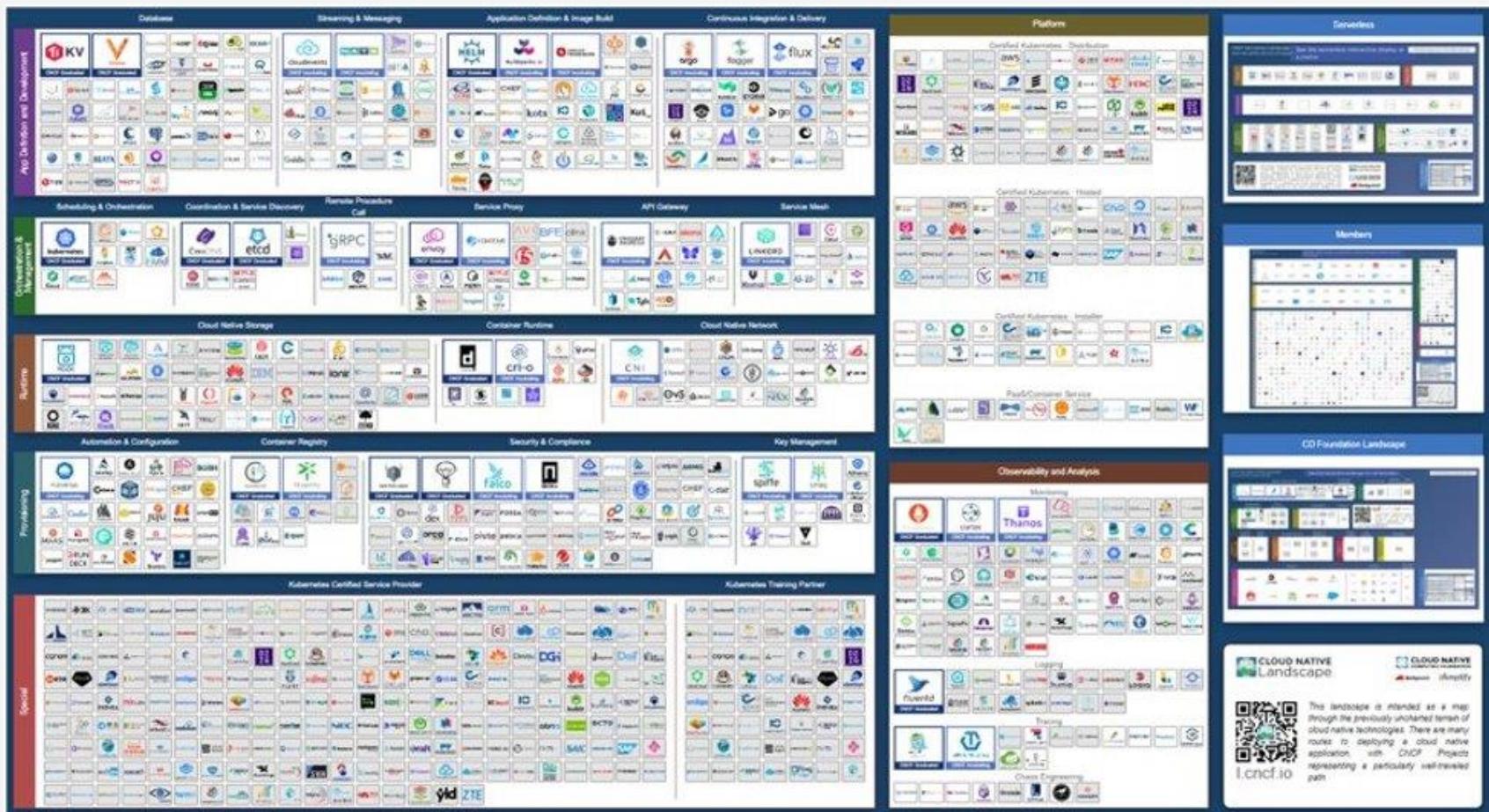


- Kubernetes wird auch mit K8s abgekürzt
- K8s ist ein Open-Source System zum Management von containerisierten Applikationen über mehrere Hosts hinweg
- Kubernetes basiert auf einem Google System (Borg) und wird zusammen mit der Community weiterentwickelt
- Inzwischen gehört das Projekt der CNCF
- Es übernimmt Basisaufgaben wie:
 - Deployment
 - Maintenance
 - Scaling von Applikationen

Warum Kubernetes?

- Vorteile Kubernetes:
 - Container über mehrere Hosts hinweg orchestrieren (⇒ Ausfallsicherheit, Autoscaling, Erweiterbarkeit)
 - Hardware-Ressourcen effizienter nutzen
 - Die Bereitstellung und Aktualisierung von Applikationen steuern und automatisieren (möglichst 0% Downtime)
 - Storage mounten und Speicherkapazitäten hinzufügen, um zustandsbehaftete Applikationen auszuführen (man muss sich um Storage keine/wenig Gedanken machen => Standardisiert, wird bereitgestellt)
 - Applikations-Container und deren Ressourcen skalieren (automatisch, je nach Ressourcenverbrauch oder manuell)

CNCF (CLOUD NATIVE COMPUTING FOUNDATION)



<https://landscape.cncf.io/>

Architektur

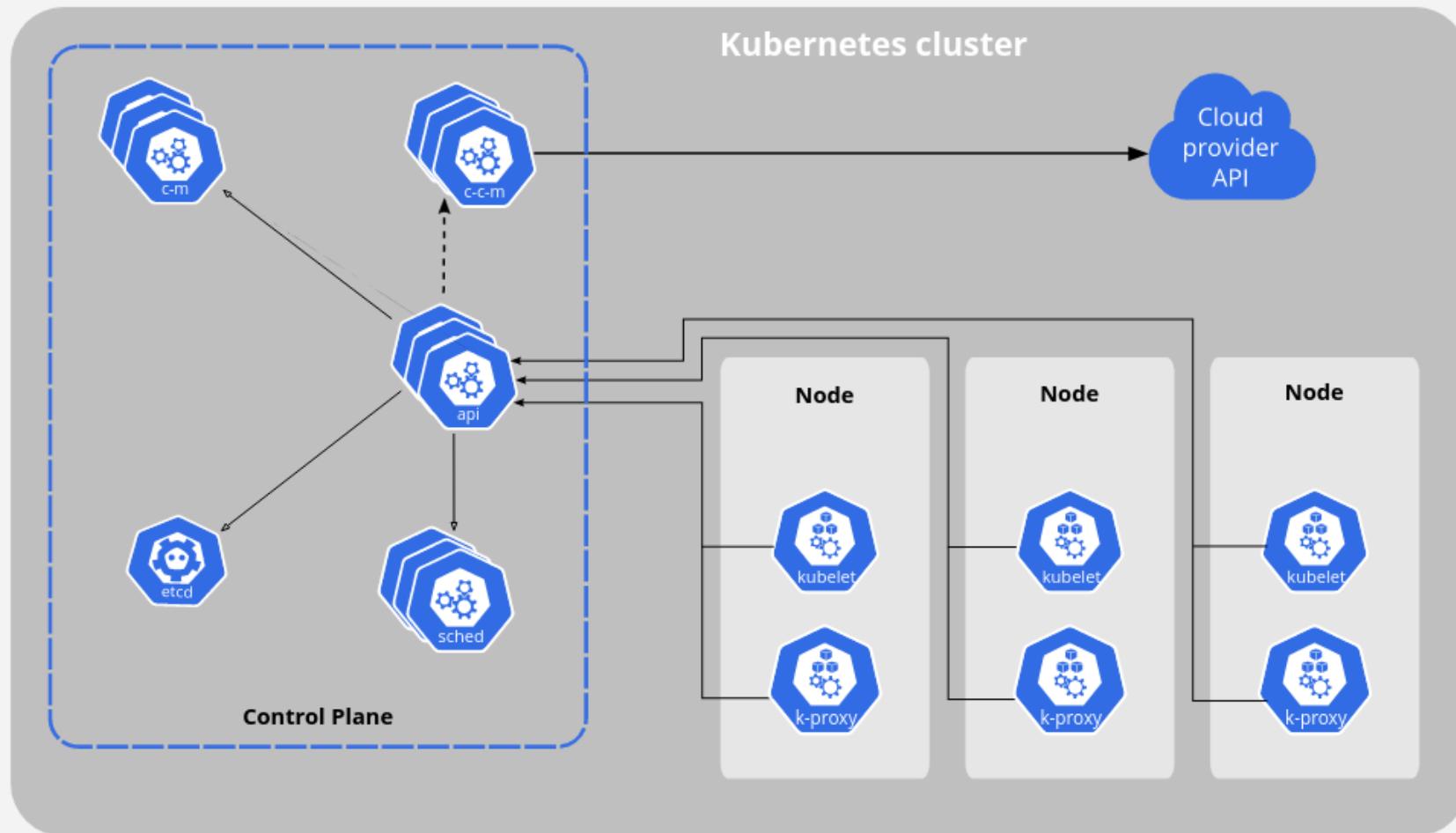


Kubernetes Cluster-Komponenten

Gesamtarchitektur

- Kubelet
- Kube-proxy
- etcd
- DNS
- Network CNI
- Ausfallsicherheit

Architektur



Quelle:<https://kubernetes.io/docs/concepts/architecture/cloud-controller/>

API-Server

- Authentifizierung
- stellt die Kubernetes API bereit
- verarbeitet REST-Anfragen, validiert diese, und updates die dazugehörigen Objekte in der Kubernetes Datenbank (etcd)
- z.B. [kube-apiserver](#)
- ist skalierbar

Scheduler

- Wenn ein neues Kubernetes Objekt "Pod" erstellt wird, hat dies noch keinen Node (=Host) zugeteilt, auf dem dieser starten soll
- Der Scheduler sucht ständig nach solchen Pods. Wenn er einen solchen gefunden hat, versucht er dafür einen passende Node zu finden. Dabei gibt es einige Kriterien bei der Auswahl:
 - Es muss genug CPU / RAM auf dem Node verfügbar sein
 - Er versucht "gleichmäßig" zu verteilen / gleichmäßige Node-Lastverteilung
 - Er achtet auf Informationen im Pod: man kann in dem Pod-Manifest beschreiben, auf welche Nodes er starten darf und auf welchen nicht (taints), z.B. nicht oder ausschließlich auf Master-Nodes, oder man könnte einen dedizierten Node bereitstellen, der nur Pods einer bestimmten Applikation zulässt)
- Diese Zuweisung schreibt er dann ins Pod-Manifest (etcd).
Von dort bekommt das kubelet auf dem jeweiligen Node die Information, dass er nun diesen Pod starten soll

Kubelet

"The kubelet manages pods and their containers, their images, their volumes, etc."
- kubernetes.io

Ist der "Kubernetes Agent" auf den Hostsystemen
=> Ohne ihn ist der Node nicht funktionstüchtig

Meldet die laufenden Container und den Node Status an die controlplane (api, etcd)

Installation: z.B. Systemd service

Schaut auf den "manifest-path" und startet statische Pods von dort (z.B. api server, cni, ...)

Kube-Controller-Manager

- Versucht stetig den Status des Kubernetes Clusters vom "Ist" auf den "Soll" Stand zu bringen
- Discovery and management of nodes
- Verwaltung von endpoints
- Verwaltung von replicas
 - Sind immer so viele Pods da, wie sein sollen?
- Verwaltung von Serviceaccounts

etcd – Die Kubernetes Datenbank

- Konsistente und hochverfügbare Key-Value-Datenbank
- Der Clusterstatus und alle Kubernetes Objekte sind hier persistiert
- Wichtig: Backup bedenken! Sonst kann man den Clusterstatus nicht mehr wiederherstellen
- Sollte schnellen Speicher haben, da Kubernetes anfällig für hohe Latenzen beim schreiben/lesen ist

DNS

- DNS für Kubernetes Services
- Alternativ könnte man auch via Environment Variablen diese erreichbar machen
==> Somit theoretisch optionales Add-on, aber defacto gehört dies auch in jedes Cluster
- Standardmäßig wird dieser DNS von Pods im Cluster genutzt

Network CNI



CNI

- Das Container Network Interface (CNI) ist ein Open-Source Projekt der CNCF
- Das CNI ist eine Sammlung von Spezifikationen und Bibliotheken, die das Schreiben von Plugins für Container Networking regelt
- Zusätzlich gibt es schon unterstützte Plugins (z.B. für Kubernetes)
- Befasst sich nur mit der Netzwerk Konnektivität der Container sowie der Verwaltung der zugehörigen Ressourcen
- Beispiele:
Calico, Cilium, Flannel

Ausfallsicherheit

Container Health Check (readyness, liveness)

- K8s bemerkt, wenn ein Container nicht mehr "healthy" ist und startet diesen neu.

Hostsystemausfall

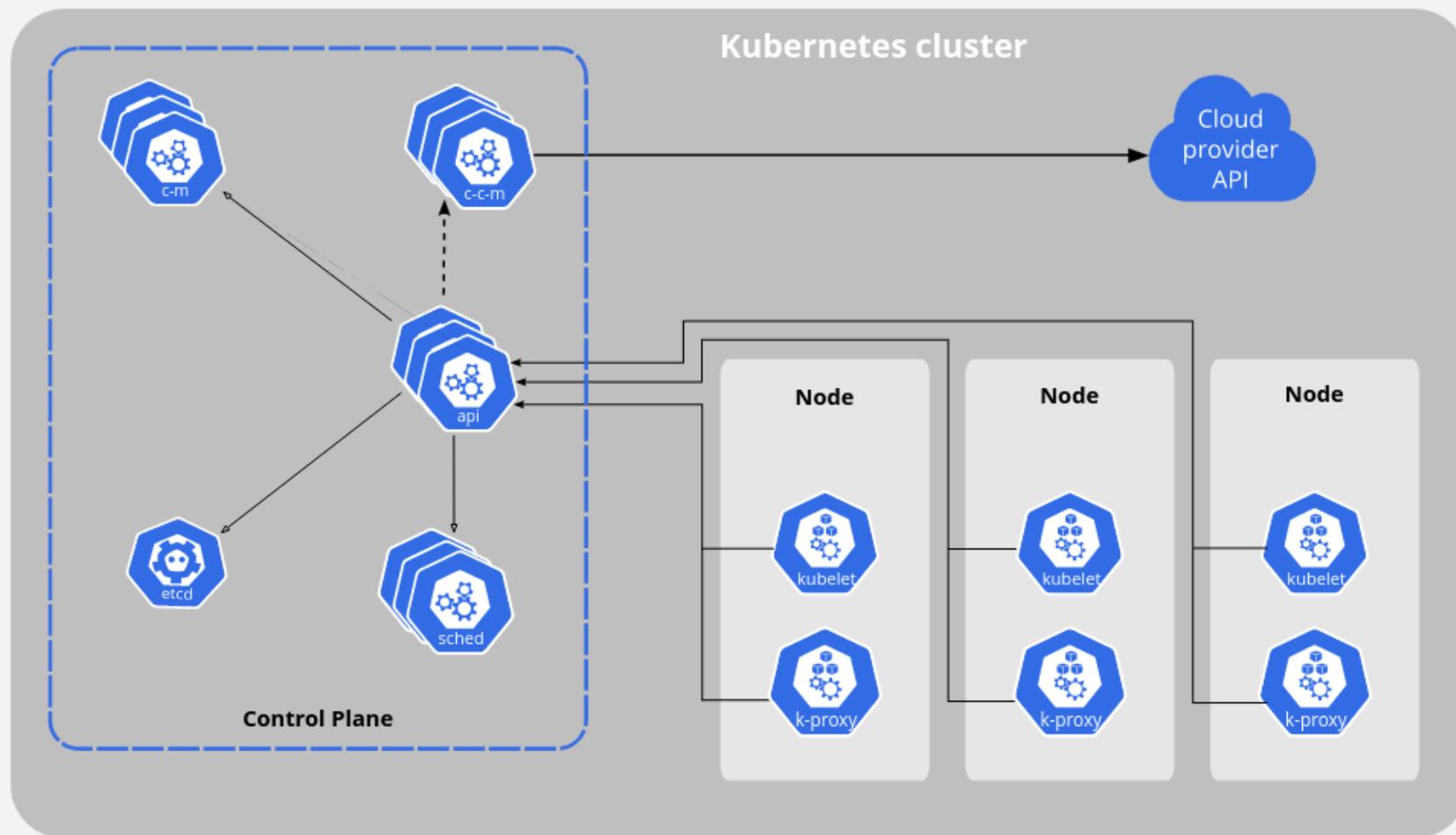
- K8s merkt, dass ein Host nicht mehr erreichbar ist. Somit sind auch alle Pods dort weg. K8s weiß welche Pods dort liefen und weist diese automatisch anderen Nodes zu, auf denen sie dann neu gestartet werden. Im besten Fall waren die Pods vorher schon auf mehreren Nodes vorhanden => kein Ausfall

Update

- Bei einem Update von Deployments werden erst neue Pods gestartet und dann alte gelöscht (jedoch sind insgesamt immer X Pods verfügbar, wobei X konfigurierbar ist. Default: X=75%) Dies erfolgt so lange, bis alle ausgetauscht wurden.
Damit wird sichergestellt, wenn das neue Deployment defekt ist und nicht hochkommt, dass die alten Pods weiterlaufen.
Der Service merkt anhand des Status des Pods, ob dieser erreichbar ist oder nicht => Nimmt diesen somit aus dem Loadbalancing

<https://kubernetes.io/de/docs/tutorials/kubernetes-basics/update/update-intro/>

Architektur (Wiederholung)



Quelle:<https://kubernetes.io/docs/concepts/architecture/cloud-controller/>

Kubernetes Objekte



Kubernetes Objekte

Vorbereitung

- Cheatsheet, kubectl Dokumentation, Kubernetes Zugriff, GIT

Basis Objekte

- Pod, Konfiguration (Secret, ConfigMap, Environment), Service (Label & Selektoren / Zugriff), persistente Datenhaltung (Volume / PersistentVolume / PersistentVolumeClaim), Namespace

Workload Resources

- ReplicaSet, Deployment, DaemonSet, StatefulSet, Job

Weitere...

- Ingress, RBAC, (ClusterWide)NetworkPolicy, PodSecurityPolicy (PSP)

Kubernetes Cheat-Sheet

- https://github.com/lathspell/kubernetes_test/blob/master/Kubernetes-Cheat-Sheet_07182019.pdf
- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Kubernetes Dokumentation

Kubectl reference:

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

Kubernetes Api reference:

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/>

Kubernetes Dokumentation:

<https://kubernetes.io/docs/concepts/overview/>

kubectl Setup

```
source <(kubectl completion bash) # setup autocomplete
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently

alias k=kubectl # set short alias for kubectl
complete -o default -F __start_kubectl k # set completion for 'k'

export do="--dry-run=client -o yaml" # k create deploy nginx --image=nginx $do
export now="--force --grace-period 0" # k delete pod x $now
```

vim Setup

```
> cat ~/.vimrc

syntax on

autocmd FileType yaml setlocal ts=2 sts=2 sw=2 expandtab

set foldlevelstart=20

let g:ale_echo_msg_format='[%linter%] %s [%severity%]'
let g:ale_sign_error='✗'
let g:ale_sign_warning='⚠'
let g:ale_lint_on_text_changed='never'
```

Kubernetes Ausprobieren

Online

- z.b. Killercoda: <https://killercoda.com/>

Minikube:

- Virtualbox: https://www.virtualbox.org/wiki/Linux_Downloads
- Minikube: <https://minikube.sigs.k8s.io/docs/start/>

Kind

- <https://kind.sigs.k8s.io/>

kind

```
~ > git clone https://github.com/x-cellent/Kubernetes-Basis-Schulung
~ > cd Kubernetes-Basis-Schulung
~/Kubernetes-Basis-Schulung > cat kind/my-cluster.yaml
```

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
name: my-cluster
nodes:
- role: control-plane
- role: worker
- role: worker
- role: worker
```

```
~/Kubernetes-Basis-Schulung > kind create cluster --config ./kind/my-cluster.yaml
```

Git

Clonen oder öffnen (wer es noch nicht hat):

<https://github.com/x-cellent/Kubernetes-Basis-Schulung>

Namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: TestApp
```

- Namespaces stellen einen abgegrenzten Raum für Namen dar
- Innerhalb eines Namespace müssen die Namen von Objekten einzigartig sein, aber dürfen außerhalb eines Namensbereich identisch sein
- Namespaces werden verwendet, um Cluster Ressourcen unter verschiedenen Usern aufzuteilen
- Die meisten Ressourcen sind Namespace-gebunden (außer der Namespace selbst und manche "low-level" Objekte wie Nodes), aber es gibt auch Clusterweite Ressourcen (CWNP, ClusterRoles, ...)
- Per Default sind Objekte im "default" Namespace
- Namespaces können nicht geschachtelt werden
- Besonderer Namespace "kube-system"

Namespace: Aufgabe

- Benötigte Befehle: create, get, apply, delete
1. Erstellen Sie einen Namespace mit dem Namen "training" per command line
 2. Exportieren Sie diesen Namespace in die Datei namespace.yaml
 3. Ändern Sie darin den namen zu "schulung"
 4. Wenden Sie diese Datei an und löschen Sie anschließend den alten Namespace
 5. Schauen Sie sich nun den folgenden Namespace an und reparieren Sie diesen: "tofix-namespace.yaml" (git)
- Hinweis: namespace.yaml anschauen und vergleichen (oder Dokumentation anschauen)

Namespace: Lösung

```
kubectl create ns training
```

```
kubectl get -o yaml ns training > namespace.yaml
```

```
vi namespace.yaml
```

```
kubectl apply -f namespace.yaml
```

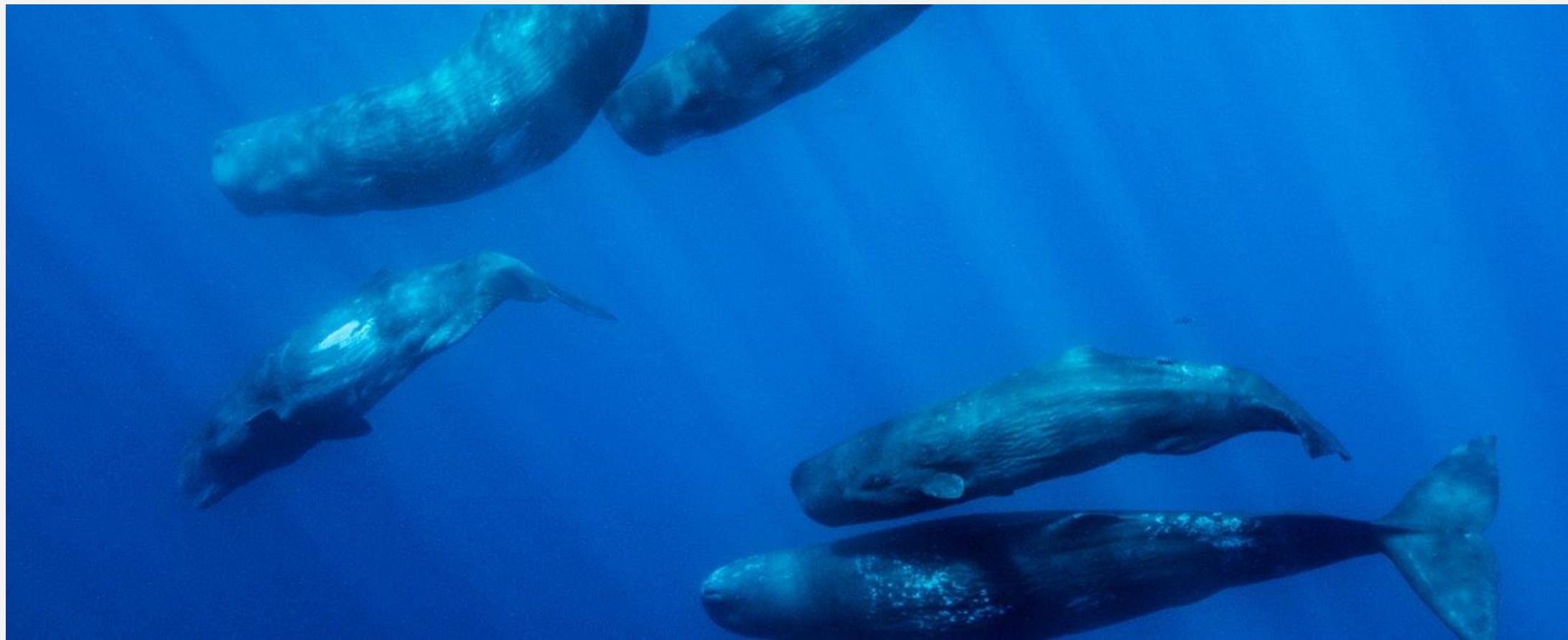
```
kubectl get ns
```

```
kubectl delete ns training
```

#namespace.yaml Fix:

#Diverse Syntaxfehler können mit "kubectl apply" oder durch yaml linter genauer analysiert werden

Pod of whales



Pod

- Pods bestehen aus einem oder mehreren Containern und Volumes
- Diese Container stellen eine Applikation dar (z. B. Frontend, Backend)
- Jeder Pod hat seine eigene IP-Adresse
- Kleinster Bestandteil im Scheduling von Kubernetes
- Wird man selten selber managen, da dies automatisch von den Controllern übernommen wird (Deployment, ReplicaSet, DaemonSet, StatefulSet, Job)

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - image: example-image
      name: example-name
    - image: myimage
      name: myname
```

Pod: Aufgabe

Benötigte Befehle: run, get, apply, delete

1. Erstellen Sie einen Pod mit dem Namen "nginx", der ein nginx Image startet und auf port 80 lauscht
2. Exportieren Sie diesen Pod in die Datei pod.yaml
3. Ändern Sie darin den Namen zu "webserver"
4. Wenden Sie diese Datei an und löschen Sie, wenn nötig, anschließend den alten Pod "nginx"
5. Schauen Sie sich nun den Pod tofix-pod.yaml an. Machen Sie diesen lauffähig.

Pod: Lösung

```
kubectl run -n schulung nginx --image=nginx --port=80 --dry-run -o yaml > pod.yaml
```

```
kubectl run -n schulung nginx --image=nginx --port=80
```

```
kubectl get -n schulung -o yaml pod nginx > pod.yaml
```

```
vi pod.yaml
```

```
kubectl apply -f pod.yaml -n schulung
```

```
kubectl get pods -n schulung
```

```
kubectl delete pod -n schulung nginx
```

pod.yaml fix:

Diverse Syntaxfehler können mit "kubectl apply" oder durch yaml Linter genauer analysiert werden

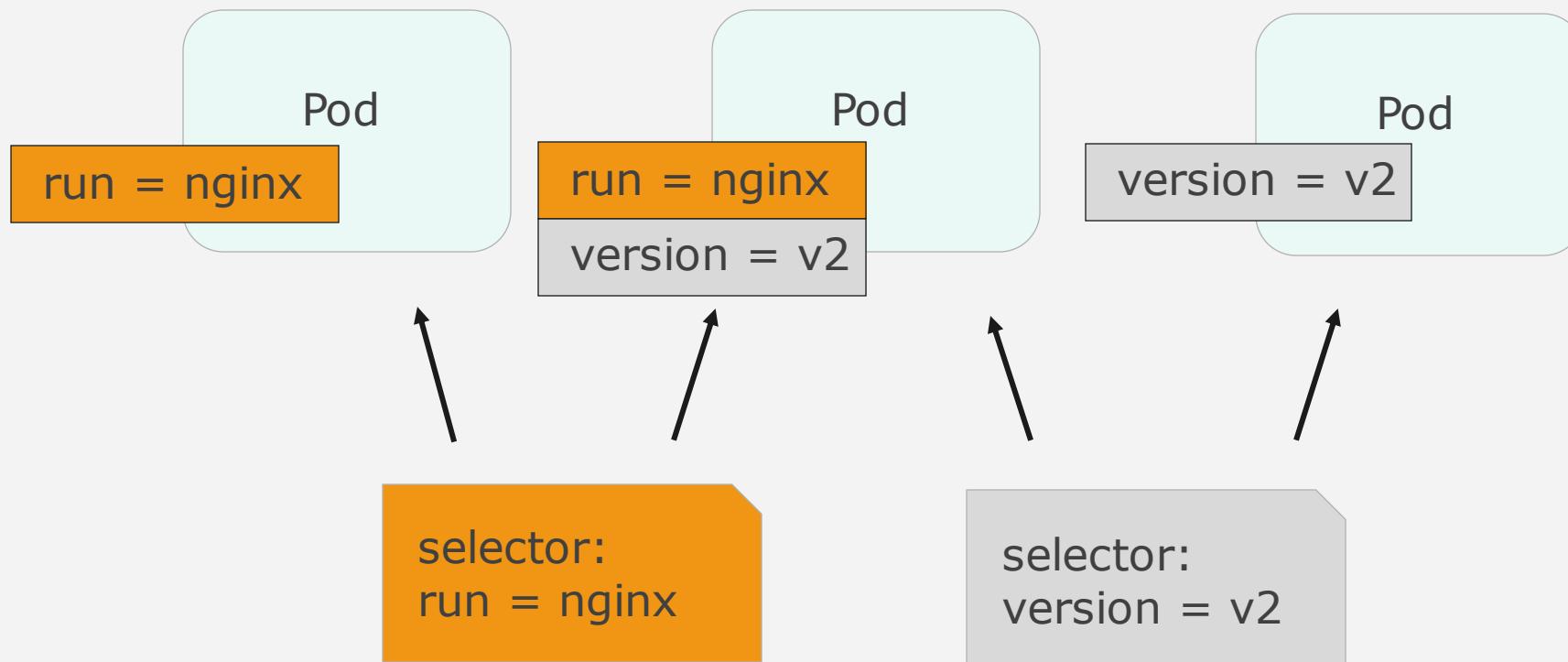
Selektoren

- Pods werden üblicherweise via Services angesprochen, die gleichzeitig als Loadbalancer fungieren
- Es gibt keine harte Abhängigkeit zwischen Pods und Services
- Die Zuordnung geschieht anhand von Pod-Labels und Service-Selektoren:

Wenn alle Labels in der "selector" Sektion des Service auch in der "labels" Sektion eines Pod enthalten sind, wird dieser Pod automatisch Teil des Service-Loadbalancings

```
apiVersion: v1
kind: Service
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 80
    selector:
      run: nginx
    type: ClusterIP
  ---
  kind: Pod
  metadata:
    labels:
      run: nginx
  spec:
    containers:
      - image: nginx
        name: nginx
      ports:
        - containerPort: 80
      resources: {}
    dnsPolicy: ClusterFirst
    restartPolicy: Always
```

Beispiel Selektoren



Service

- "An abstract way to expose an application running on a set of Pods as a network service."
- = Loadbalancer auf mehrere Pods
- Immer auf Services gehen, nie direkt auf Pods!
- Vgl. Loadbalancer (apache, nginx) (wobei weniger Funktionsumfang)
- Sucht anhand von Labels die dazugehörigen Pods (=selector)
- Alternative: Services ohne Selektoren:
Manuelle Endpunkte erstellen und mappen (Sonderfall, wird selten benötigt!) z.B.
externe Datenbanken/Services

Warum ein Service?

Pods sind dynamisch => IPs, Namen und Status ändert sich, dynamisches Hinzufügen und Löschen => benötigt Loadbalancer, der erkennt, wann Pods erreichbar/funktionsfähig sind und wann nicht, und dynamisch erkennt, wie viele und vor allem welche derzeit vorhanden sind / hinzugefügt/ gelöscht wurden

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Service: Aufgabe

Benötigte Befehle: expose, get

1. Erstellen Sie einen Service mit dem Namen "nginx", der auf port 8080 lauscht und auf den zuvor erstellten pod loadbalanced
2. Hinweis: ggf. muss der Pod um Labels erweitert werden!
3. Ändern Sie darin den Namen zu "webserver"

Service: Lösung

```
kubectl expose pod webserver -n schulung --port=8080 --target-port=80 --type=ClusterIP --dry-run -o yaml  
kubectl expose pod webserver -n schulung --port=8080 --target-port=80 --type=ClusterIP  
kubectl get service -n schulung  
kubectl get pod -n schulung -o wide
```

Port-Forward

- Beispiele:
 - Pod: `kubectl port-forward mongo-75f59d57f4-4nd6q 28015:27017`
 - Service: `kubectl port-forward service/mongo 28015:27017`
- Man kann unter anderem folgende Objekte forwarden: pod, replicaset, deployment, service
- Tipp: Durch verschiedene Stellen, an denen man den port-forward macht, kann man leicht debuggen, ob der Service kaputt ist, die App, oder etwas außerhalb des Clusters!
- Vorsicht: Wird im Normalfall nur zum Debuggen bzw. Austesten genutzt. Sollte sonst mit Ingress-Objekten gemacht werden.

<https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/>

Port-Forward: Aufgabe

1. "forwarden" Sie den Service nach localhost und greifen Sie darauf zu (z.B. mit curl)

Port-Forward: Lösung

```
kubectl port-forward -n schulung service/webserver -n schulung 8080
```

```
curl localhost:8080
```

Persistenz

Wie kann man nun Pods konfigurieren, oder Daten persistieren?

- Persistenz:
 - PersistentVolumes / PersistentVolumeClaim
 - StorageClass
- Hostsystempfade
- Konfiguration:
 - ConfigMaps
 - Secrets
 - Environment Variablen

Persistent volumes (PV)

Persistent Volume

- vgl: docker volumes
- ist praktisch ein konkreter Storage Platz
- z.B. ein NFS Pfad/Ordner
- Manuell erstellbar (z.B. NFS Pfad und IP angeben)
- Oder automatisiert nach Bedarf erstellbar (storageclass)
- Unterstützt verschiedene Storage-Anbindungen: NFS, iSCSI, cloud provider, ...
- Attribute:
 - Size
 - Mode: RWO – ReadWriteOnce, ROX – ReadOnlyMany, RWX – ReadWriteMany, RWOP – ReadWriteOncePod
 - ReclaimPolicy: Was soll nach Löschen des PVCs erfolgen?

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes>

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 1.2.3.4
    path: "/"
```

Persistent Volume Claim (PVC)

PersistentVolumeClaim:

- Ist praktisch eine Anfrage, um Storage zu bekommen
- PVCs konsumieren PVs (claimen/binden diese)
- Pod referenziert ein PVC, das PVC bindet ein PV
- Binding: Es wird ständig geschaut, ob es neue PVCs gibt, die ein PV benötigen. Dann wird versucht diesen PVCs ein passendes PV hinzuzufügen. Das nennt man dann "Bound"
- Der Pod kommt i.d.R. erst hoch, wenn sein PVC "Bound" ist, also er für seinen Request auch ein konkretes PV erhalten hat

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
resources:
  requests:
    storage: 1Mi
```

Storage Class

Können von einem PVC genutzt werden, um dynamisch PVs zu generieren, die dann geclaimed werden können

Es gibt schon diverse vorgefertigte Typen:

AWSElasticBlockStore, AzureFile, AzureDisk, CephFS, Cinder, FC, FlexVolume, Flocker, GCEPersistentDisk, Glusterfs, iSCSI, Quobyte, NFS, RBD, VsphereVolume, PortworxVolume, ScaleIO, StorageOS, Local

<https://kubernetes.io/docs/concepts/storage/storage-classes/>

Bei manchen Typen wird ein Provisioner Pod benötigt, z.B. für NFS:

<https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner>

Parameter:

- Reclaim Policy: Was soll mit den PVs geschehen, nachdem das PVC gelöscht wurde?

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

Storage Class

Beispiel (PVC)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
  matchExpressions:
  - {key: environment, operator: In, values: [dev]}
```

HostPath

1) Ist praktisch ein PV

PV-Typ: Local / HostPath

Bitte NICHT nutzen, macht meistens keinen Sinn

2) Kann man aber auch direkt im Pod angeben

Es wird kein extra PV/PVC gebraucht

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - image: nginx:0.1
      name: nginx
      volumeMounts:
        - mountPath: /data
          name: volume
  volumes:
    - name: volume
      hostPath:
        path: /data
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - example-node
```

Environment Variablen

- Cloud native Anwendungen sollten über env varibalen konfigurierbar sein
- Kann direkt in der Container-Spec gesetzt werden
- Man kann Environment Variablen auch mit Werten aus ConfigMaps und Secrets befüllen
- Metadaten können z.B. auch genutzt werden

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envvars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from
the                                     environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
    - name: MY_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
```

ConfigMaps (cm)

- Beinhaltet daten (key:value)
- Value kann auch der Inhalt einer Datei sein
- Die Daten können z.B. in Pods verwendet werden
- Einsatzzwecke:
 - Environment Variablen setzen
 - Configurationsfiles mounten

Wann sollte ich ConfigMaps nutzen?

- Konfigurationen für eine Applikation speichern
- Sollte keine Geheimnisse enthalten (Passwörter, private keys, ...)
=> secrets

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key
  # maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-
  interface.properties"
  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

ConfigMaps (cm)

Beispiel:

Pod mit Env aus einer ConfigMap und einer Datei, die aus einer ConfigMap in das Filesystem gemountet wird

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: alpine
      env:
        - name: PLAYER_INITIAL_LIVES
          valueFrom:
            configMapKeyRef:
              name: game-demo # The ConfigMap
              key: player_initial_lives # The key to fetch
      volumeMounts:
        - name: config
          mountPath: "/config"
          readOnly: true
      volumes:
        - name: config
          configMap:
            name: game-demo
            # An array of keys from the ConfigMap to create as files
            items:
              - key: "game.properties"
                path: "game.properties"
```

Secrets

- Sehr ähnlich zu ConfigMaps
- Werden in der Rechteverwaltung (RBAC) von k8s gesondert betrachtet, z.B. darf die clusterrole view zwar nahezu alles sehen, aber keine Secrets
- Wird base64 encoded => deswegen ist oft "echo ... |base64 -d" nötig um den Klartext zu sehen
- Es gibt verschiedene Typen von Secrets:
Opaque, service-account-token, dockercfg, basic-auth, tls, token, ...

Wofür sollte man dies nutzen?

- Sensitive Daten wie Passwörter, Token, und private keys
- Solche Daten sollten NICHT in die Pod spec! (und auch nicht in einer cm)
- Kann als Datei in einen Pod gemountet werden, oder als environment variable, oder z.B. ein docker pull secret um Images aus einer privaten, abgesicherten Registry pullen zu dürfen

Übung

Commands: create, get, apply

Ressources: ConfigMap, Pod

1. Erstellen Sie eine ConfigMap mit einer index.html Datei und nutzen Sie diese im Deployment mit volumeMount und volume
2. Hinweis: Editieren Sie hierfür nach der Erstellung der ConfigMap eine pod.yaml, und erweitern Sie diese um "volumeMounts" und "volumes"
3. Optional: Ggf. mit volumes / hostpaths / secrets / env-variablen üben

Lösung

```
kubectl create cm -n schulung indexhtml --from-file index.html  
kubectl get -o yaml pod -n schulung webserver >pod.yaml  
vi pod.yaml  
kubectl delete -f pod.yaml  
kubectl apply -f pod.yaml
```

```
...  
spec:  
  containers:  
  ...  
  volumeMounts:  
  - name: nginx-index-file  
    mountPath:/usr/share/nginx/html/  
  ...  
  volumes:  
  - name: nginx-index-file  
    configMap:  
      name: indexhtml  
  ...
```

Workload Resources

Diese werden genutzt, um auf einer höheren Ebene mit mehr Funktionalität Pods zu verwalten
z.B. Ausfallsicherheit, Skalierbarkeit, Update-Prozess, ...

Beispiele:

- Deployment
- ReplicaSet
- StatefulSet
- Jobs / Cronjobs

ReplicaSet

Besteht aus...

- Selector (welche Pods verwalte ich?)
- Replicas (Wieviele dieser Pods sollen da sein?)
- Pod template (Wie sollen die Pods aussehen?)

Aufgabe:

- Erstellt/löscht Pods, um auf die gewünschte Anzahl zu kommen
- Nutzt das Pod template, um neue Pods zu erstellen
- Setzt "metadata.ownerReferences" in der Pod-Spec, die er verwaltet
- Sucht anhand vom Selector (labels) nach Pods, die er verwalten soll (und nicht schon von anderen verwaltet werden)

Wird normalerweise von einem Deployment erstellt / verwaltet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: nginx:latest
```

Deployment

"A Deployment provides declarative updates for [Pods](#) and [ReplicaSets](#)."

"You describe a *desired state* in a Deployment, and the Deployment [Controller](#) changes the actual state to the desired state at a controlled rate."

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

- Deployment => ReplicaSet => Pods
- Die Zuordnung erfolgt anhand von Labeln
- Bei Änderungen im Deployment wird das ReplicaSet angepasst, was wiederum die Pods anpasst
- Ist der "Standard"-Weg wie man Pods im Cluster betreibt (Ausnahme: Status gebundene Pods wie Datenbanken)
- "per Hand" sollte man nicht an Pods arbeiten, immer am Deployment, DaemonSet oder StatefulSet!

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Deployment: Aufgabe

Commands: create, get, apply

Ressources: deployment

- Erstellen Sie ein Deployment des nginx Pods mit 3 replicas
- Hinweis: plain deployment yaml file erstellen und mit der pod.yaml anpassen

Lösung

```
kubectl create deployment webserver --image=nginx \
--dry-run=client -o yaml > deployment.yaml

vi deployment.yaml

# auf replicas, label, ns und volume mounts achten

kubectl delete -f deployment.yaml

kubectl apply -f deployment.yaml

kubectl apply -f deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: nginx
    name: webserver
    namespace: schulung
spec:
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: nginx
    spec:
      containers:
        - image: nginx
          name: nginx
          resources: {}
        volumeMounts:
          - name: nginx-index-file
            mountPath: /usr/share/nginx/html/
      volumes:
        - name: nginx-index-file
          configMap:
            name: indexhtml
  status: {}
```

StatefulSet

"StatefulSet is the workload API object used to manage stateful applications."

Aufgabe:

- "deployment and scaling of a set of Pods"
- "provides guarantees about the ordering and uniqueness of these Pods"
- maintains a sticky identity for each of their Pods.

=> Pods bleiben bestehen, werden nicht gelöscht, nur verändert und neu gestartet.

Dadurch ist das Mapping von Volumes zu Pods auch einfacher. Man weiß genau welches Volume zu welchem Pod gehören muss

Wann brauche ich StatefulSets statt Deployments?

- Stable, unique network identifiers (bleibt nach Neustart erhalten)
- Stable, persistent storage (ein Pod soll immer das gleiche Volume erhalten! Auch nach Neustarts.)
- Ordered, graceful deployment and scaling (starte ein Pod nach dem anderen in einer vorgegebenen Reihenfolge)
- Ordered, automated rolling updates (aktualisiere ein Pod nach dem anderen in einer vorgegebenen Reihenfolge)

StatefulSet

```
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
  spec:
    terminationGracePeriodSeconds: 10
    containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
          - containerPort: 80
            name: web
        volumeMounts:
          - name: www
            mountPath: /usr/share/nginx/html
    volumeClaimTemplates:
      - metadata:
          name: www
        spec:
          accessModes: [ "ReadWriteOnce" ]
          storageClassName: "my-storage-class"
        resources:
          requests:
            storage: 1Gi
```

Jobs

- Werden i.d.R. nur einmal ausgeführt => Kein dauerhafter Container
- Z.B. backups, migration, Scans, audits, installer, ...
- Deleting a Job will clean up the Pods it created
- A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate

Wann brauche ich Jobs?

Wenn

ich sicher gehen will, dass ein bestimmter Task ausgeführt wird (restartet so lange, bis es klappt)

Spezialfall: Cronjobs

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: Never
          backoffLimit: 4
---
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              imagePullPolicy: IfNotPresent
              command:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

SECURITY

Serviceaccounts

Rolebindings

Pod security policy

Privileged?

Capabilities?

Run as non root?

Was darf ich mounten? (hostsystempfade)

Network policies

ServiceAccounts (SA)

- = Technischer user in K8s
- Jeder Namespace hat einen default SA
- Man kann selber SAs anlegen (und diese auch berechtigen)
- Die Rechteverwaltung in k8s bezieht sich immer auf einen SA oder einen user
- Jedem Pod kann man einen SA mitgeben => der Pod erhält dann die Rechte von diesem SA
- Eine Applikation für k8s liefert oft auch einen/mehrere SAs mit, die nur Minimalrechte für diese Applikation besitzen
- Jeder SA besitzt normalerweise einen Token als secret. Mit diesem kann er sich beim API server authentifizieren
- Neben den Rechten für Pods könnten SAs auch für andere Sachen genutzt werden, wie z.B. für den Zugriff von einer CI/CD Pipeline auf ein Cluster zum Deployen/Ausrollen der Applikation

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
secrets:
- name: build-robot-token-bvbk5
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: build-robot
...
```

ServiceAccount: Aufgabe

- Erstellen Sie einen SA und eine Applikation, die diesen SA nutzt

ServiceAccount: Lösung

```
kubectl create sa -n schulung webserver --dry-run=client -o yaml > sa.yaml  
kubectl create sa -n schulung webserver  
vi deployment.yaml  
kubectl delete -f deployment.yaml  
kubectl apply -f deployment.yaml
```

Roles & RoleBindings | RBAC

Was ist eine Rolle? (role)

- Eine Rolle ist eine Sammlung von Rechten in Kubernetes
- Ein Recht besteht aus einem/mehreren Verben (verbs) und Ressourcen
- z.B. get (verb) Pods (resource)
- Eine Rolle ist namespace-bezogen
- Das clusterweite Pendant dazu nennt man ClusterRole
- Eine ClusterRole kann man auch für einen Namespace nutzen, indem man im RoleBinding diese referenziert und einen "namespace" angibt

Was ist ein RoleBinding?

- Dies ist die Verknüpfung von einer Rolle (role) zu einem/mehreren Accounts (user/ServiceAccount)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

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

Roles & RoleBindings | RBAC

There are 4 different RBAC combinations and 3 valid ones:

1. **Role + RoleBinding** (available in single Namespace, applied in single Namespace)
2. **ClusterRole + ClusterRoleBinding** (available cluster-wide, applied cluster-wide)
3. **ClusterRole + RoleBinding** (available cluster-wide, applied in single Namespace)
4. **Role + ClusterRoleBinding** (**NOT POSSIBLE**: available in single Namespace, applied cluster-wide)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

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

Roles & RoleBindings | RBAC

```
# commands
# audit
kubectl auth can-i -n schulung 'create' 'pod' --as=system:serviceaccount:schulung:webserver
kubectl auth can-i -n schulung 'get' '/healthz' --as=system:serviceaccount:schulung:webserver
kubectl auth can-i 'create' 'pod' -n schulung

# Configs auslesen
k get clusterrolebindings [-o yaml] [name]
k get rolebindings -n default [-o yaml] [name]
```

Roles & Rolebindings: Aufgabe

- Welche Rechte hat der für eine App erstellte ServiceAccount?
Wie können Sie sich das anzeigen lassen? Darf er Pods erstellen (create)?
- Erstellen Sie einen neuen SA "cirobot"
- Erstellen Sie eine ClusterRole, die die gleichen Rechte wie die ClusterRole "admin" hat
- Erstellen Sie ein dazugehöriges RoleBinding von dieser Rolle zu dem SA
- Ist das so in der Praxis ein gutes Vorgehen? Warum (nicht)?
- Wie kann man überprüfen, ob der SA nun z.B. Pods erstellen darf?

Roles & Rolebindings: Lösung

```
kubectl auth can-i -n schulung 'create' 'pod' --as=system:serviceaccount:schulung:webserver  
kubectl auth can-i -n schulung 'get' '/healthz' --as=system:serviceaccount:schulung:webserver  
kubectl auth can-i 'create' 'pod' -n schulung  
kubectl get clusterrolebindings [-o yaml] [name]  
kubectl get rolebindings -n default [-o yaml] [name]
```

```
# alle Ausgaben fuer webserver  
  
kubectl get  
rolebinding,clusterrolebinding --all-  
namespaces -o jsonpath='{range  
.items[?(@.subjects[0].name=="webserver"  
)]}{.roleRef.kind},{.roleRef.name}}{end  
}'  
  
# alle für serviceaccounts  
  
kubectl get clusterrolebindings -o json  
| jq -r '  
.items[] |  
select(  
.subjects // [] | .[] |  
[.kind,.name] ==  
["Group","system:serviceaccounts"]  
) |  
.metadata.name'  
  

```

Roles & Rolebindings: Lösung

```
kubectl create sa -n schulung cirobot
kubectl get clusterrole admin -o yaml > cr.yaml
vi cr.yaml
# n amen ändern, nicht benötigtes löschen
kubectl apply -f cr.yaml
kubectl create rolebinding -n schulung myadmin --clusterrole=myadmin --serviceaccount=schulung:cirobot
# nein und ja. Grundsätzlich ist es gut, aber der SA hat zu viele Rechte. Sollte nur die Rechte haben, die man für das Deployment braucht und nur in diesem NS
kubectl auth can-i -n schulung 'create' 'pod' --as=system:serviceaccount:schulung:cirobot
```

Pod Security Policy (PSP) (deprecated v1.21)

- Eine pod security policy beschreibt, welche Rechte ein Container haben darf.
- Vgl. Docker: privileged containers, capabilities

Wie wird eine PSP zugewiesen?

1. PSP erstellen
2. (optional) ServiceAccount für den Pod erstellen und diesen im Pod eintragen
3. RoleBinding / ClusterRoleBinding erstellen, um den SA zu berechtigen (verb=use, ressource=psp, und am besten den Namen angeben)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

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

Pod Security Admission (stable v1.25)

- Bewertet Pod-Spezifikationen anhand eines vordefinierten Satzes von Pod-Sicherheitsstandards
- Wird bei der Erstellung von Pods auf Namespace-Ebene angewendet
- Einfach und leicht zu bedienen
- Unterstützt und fördert bewährte Kubernetes-Praktiken
- Pod-Sicherheitsstandards Stufen:
 - privilegiert - offen und uneingeschränkt
 - baseline - deckt bekannte Privilegien-Eskalationen ab und minimiert die Einschränkungen
 - restricted - stark eingeschränkt, Härtung gegen bekannte und unbekannte Privilegieneskalationen (kann zu Kompatibilitätsproblemen führen)
- Modi:
 - enforce - alle Pods, die gegen die Richtlinie verstößen, werden zurückgewiesen
 - audit - Verstöße werden als Anmerkung in den Audit-Protokollen vermerkt, haben aber keinen Einfluss darauf, ob der Pod zugelassen wird.
 - warn - Verstöße werden mit einer Warnmeldung an den Benutzer zurückgemeldet, haben aber keinen Einfluss darauf, ob der Pod zugelassen wird.

```
# The per-mode level label indicates which policy level
# to apply for the mode.
#
# MODE must be one of `enforce`, `audit`, or `warn`.
# LEVEL must be one of `privileged`, `baseline`, or
# `restricted`.
pod-security.kubernetes.io/<MODE>: <LEVEL>
```

```
# Optional: per-mode version label that can be used to
pin the policy to the
# version that shipped with a given Kubernetes minor
version (for example v1.25).
#
# MODE must be one of `enforce`, `audit`, or `warn`.
# VERSION must be a valid Kubernetes minor version, or
`latest`.
pod-security.kubernetes.io/<MODE>-version: <VERSION>
```

```
~ > k label ns very-secure pod-security.kubernetes.io/warn=restricted
~ > k run -n very-secure --image busybox --privileged busybox
```

Warning: would violate PodSecurity "restricted:latest": privileged (container "busybox" must not set securityContext.privileged=true), allowPrivilegeEscalation!=false (container "busybox" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "busybox" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "busybox" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container "busybox" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost") pod/busybox created

Pod: init container

- Werden vor dem eigentlichen Anwendungscontainer gestartet
- Wird zum "initialisieren", oder z.B. vorbereiten genutzt
- Mehrere möglich (sequenziell)
- Init-Container müssen erfolgreich durchlaufen, bevor der eigentliche Container starten kann
=> Kann somit auch zu Ausfällen führen, wenn im Init-Container Fehler sind!
- z.B. erwartet die Applikation, dass bestimmte Filesystem permissions gesetzt sind (die man nicht schon im Image / durch die Spec setzen konnte). Dann könnte der Init-Container dies vor dem eigentlichen Start tun
- z.B. ein DB init script ausführen, bevor die DB startet
- Sollte klein/schnell sein (um Restartzeiten gering zu halten)

Pod: init container : Aufgabe (optional)

- Erstellen Sie einen init container, der das gleiche Volume wie die eigentliche App mountet, und dort eine Datei hinterlegt mit z.B. dem Inhalt "init war hier".

Multi-Container Pod (sidecar)

- = mehrere Container in einem Pod
- Mehrere Container in einem Pod werden eigentlich nur dann genutzt, wenn die Container sehr eng zusammenarbeiten und sich Ressourcen teilen müssen. Z.B. gibt ein Container via Webserver eine index.html aus, die im Filesystem liegt. Ein zweiter Container aktualisiert diese index.html. Dies kann er nur, wenn er auf dasselbe Filesystem Zugriff hat
- Wird eher selten benötigt, für viele UseCases sind zwei separate Pods eher an der Microservice Architektur angelehnt
- Weitere Pods sollten die Hauptaufgabe nur unterstützen bzw. erweitern/verbessern!
=> Also um nicht funktionale Anforderungen erweitern, z.B. Logging hinzufügen, Aktualisierungsmechanismen, Statusausgaben, Monitoring Endpunkte, ...
=> ein Pod sollte trotzdem noch nur eine Hauptaufgabe haben!
- Wie sieht das aus?
In der Pod Spezifikation (spec) werden Container in einer Liste angegeben. Somit kann man einfach ein zweites Listenelement der Spec hinzufügen

Multi-Container Pod: Aufgabe (optional)

Erstellen Sie einen zweiten Container für den nginx webserver.

Dieser soll in die Webseite des nginx alle 5 Sekunden eine Zeile schreiben.

Der command dafür kann z.B. folgendermaßen aussehen:

command: ["/bin/sh"]

```
args: ["-c", "while true; do echo $(date -u) 'Hi I am from Sidecar container' >> /var/log/index.html; sleep 5; done"]
```

Multi-Container Pod: Lösung

- Quelle:

<https://kubernetes.io/docs/concepts/workloads/pods/>

```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-container-demo
spec:
  containers:
    - image: busybox
      command: ["/bin/sh"]
      args: ["-c", "while true; do echo $(date -u) 'Hi from Sidecar container' >> /var/log/index.html; sleep 5; done"]
      name: sidecar-container
      resources: {}
      volumeMounts:
        - name: var-logs
          mountPath: /var/log
    - image: nginx
      name: main-container
      resources: {}
      ports:
        - containerPort: 80
      volumeMounts:
        - name: var-logs
          mountPath: /usr/share/nginx/html
      dnsPolicy: Default
      volumes:
        - name: var-logs
          emptyDir: {}
```

Paketierung und Auslieferung



Helm



Helm

Was ist Helm?

- Ein Paketierungswerkzeug für Kubernetes
- Kann man sich vorstellen wie das, was ein Container für eine Applikation ist:
Ein Container ist eine Sammlung aller relevanten Abhängigkeiten an Software und Betriebssystemkomponenten für eine Applikation.
Ein Helm Chart ist eine Sammlung aller relevanten Abhängigkeiten an Kubernetes Objekten und Containern, damit die Applikation lauffähig ist.

Vorteile:

- Versionierung
- Rollbacks
- Einfachere Installation
- Alle Ressourcen zusammen an einem Ort
- Mehrfachaushandlungen möglich => Nur die Konfiguration muss geändert werden (values.yaml), das Chart bleibt gleich
- Einfaches Verteilen & Ausliefern (Helm Chart Repository = ähnlich wie die Docker Container Registry)

Alternativen zur Installation:

- Operators, Kustomize, Acorn

Helm: Chart Aufbau

```
somename/  
Chart.yaml      # A YAML file containing information about the chart  
LICENSE         # OPTIONAL: A plain text file containing the license for the chart  
README.md       # OPTIONAL: A human-readable README file  
values.yaml     # The default configuration values for this chart  
values.schema.json # OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file  
charts/          # A directory containing any charts upon which this chart depends.  
crds/            # Custom Resource Definitions  
templates/        # A directory of templates that, when combined with values,  
                  # will generate valid Kubernetes manifest files.  
templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

Helm: Chart.yaml

```
apiVersion: The chart API version (required)
name: The name of the chart (required)
version: A SemVer 2 version (required)
kubeVersion: A SemVer range of compatible Kubernetes versions (optional)
description: A single-sentence description of this project (optional)
type: The type of the chart (optional)
keywords:
  - A list of keywords about this project (optional)
home: The URL of this projects home page (optional)
sources:
  - A list of URLs to source code for this project (optional)
dependencies: # A list of the chart requirements (optional)
  - name: The name of the chart (nginx)
    version: The version of the chart ("1.2.3")
    repository: (optional) The repository URL ("https://example.com/charts") or alias ("@repo-name")
    condition: (optional) A yaml path that resolves to a boolean, used for enabling/disabling charts
(e.g. subchart1.enabled)
tags: # (optional)
  - Tags can be used to group charts for enabling/disabling together
import-values: # (optional)
  - ImportValues holds the mapping of source values to parent key to be imported. Each item can be a string or pair of child/parent sublist items.
alias: (optional) Alias to be used for the chart. Useful when you have to add the same chart multiple times
maintainers: # (optional)
  - name: The maintainers name (required for each maintainer)
    email: The maintainers email (optional for each maintainer)
    url: A URL for the maintainer (optional for each maintainer)
icon: A URL to an SVG or PNG image to be used as an icon (optional).
appVersion: The version of the app that this contains (optional). Needn't be SemVer. Quotes recommended.
deprecated: Whether this chart is deprecated (optional, boolean)
annotations:
  example: A list of annotations keyed by name (optional).
```

Helm: Commands

helm install # installiert ein Chart

helm upgrade --install # upgraded ein bereits installiertes Chart, oder installiert dieses, wenn noch nicht vorhanden

helm list # zeigt die installierten Helm Charts im Cluster

helm lint # Linter für Helm Charts

helm pull # download a Chart

helm seach repo # sucht nach Charts/Versionen in allen hinzugefügten Repos

helm repo add|list|remove # fügt Helm Chart Repository hinzu, z.B. bitnami, private (harbor),...

helm rollback# geht zu der vorherigen deployten Version zurück

helm dependency build|list|update # baut, zeigt, updated die Dependencies (z.B. andere Sub Charts)

helm get values # downloaded die values Datei eines Releases (Deployments)

helm show values # zeigt die default values Datei eines Helm Charts (nicht deployed)

helm template# zeigt die .yaml Dateien an, die durch "helm install" applied werden würden

helm create# erstellt lokal ein default Helm Chart

Helm: Beispiel Apps

Helm Charts anschauen => <https://artifacthub.io/>

Helm: Aufgabe 1

Helm Chart erstellen

- Install helm binary
- Helm Chart aus den derzeitigen Objekten erstellen und deployen
- Kopieren Sie die gesamte, derzeitige Applikation in den templates Ordner und löschen Sie die anderen, darin enthaltenen Dateien
 - webserver deployment + (sidecar) + (init)
 - Service
 - ServiceAccount (SA)
 - ConfigMap (cm)
 - RoleBinding (optional)
- Installieren Sie nun dieses Helm Chart in das Cluster

Helm: Aufgabe 2

- Möglichst alles konfigurierbar machen
- Hinweis: Ggf. nochmal "helm create" und die meisten Sachen so lassen und nur anpassen

Helm: Aufgabe 3 (optional)

- Helm Chart deployen (und konfigurieren!)
 - z.B. Prometheus
 - <https://artifacthub.io/packages/helm/prometheus-community/prometheus>
 - z.B. Wordpress
 - <https://artifacthub.io/packages/helm/bitnami/wordpress>

Netzwerk (Ingress)

- Ist dafür da, eine Applikation nach außen aufrufbar zu machen
- Wird von einem "ingress controller" Pod verwaltet
- Ist sozusagen ein vhost für einen Webserver (apache, nginx)
- Kann dynamisch zur Laufzeit angepasst werden
- Wird normalerweise nur für http/https traffic genutzt
- Man kann nahezu alles im Ingress konfigurieren, was man auch in einem Webserver konfigurieren könnte z.B. basic-auth, ip whitelisting, ssl-Terminierung (Pod vs im Ingress controller), bodysizes, ...
- Außerdem kommt bei einigen auch eine einfache WAF mit, die man leicht aktivieren und mitnutzen kann, z.B. modsecurity bei nginx

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
    - host: foo.bar.com
      http:
        paths:
          - path: /foo
            pathType: Prefix
            backend:
              service:
                name: service1
                port:
                  number: 4200
          - path: /bar
            pathType: Prefix
            backend:
              service:
                name: service2
                port:
                  number: 8080
```

Ingress: Aufgabe (optional)

- Deployen Sie ein "nginx Ingress Controller" Helm Chart
- Erstellen Sie ein Ingress-Objekt
- Schauen Sie nun via port-forward auf den Ingress Controller, ob die Verbindung richtig erfolgt

Ingress: Lösung

- Wichtig: Es gibt verschiedene Ingress Controller und verschiedene Arten, diese zu installieren (helm chart, minikube addon, yaml datei, ...). Hier ein Beispiel mit minikube:
- `kubectl apply -f ./Kubernetes-Basis-Schulung/Kubernetes/10_ingress/ingress.yaml`
- `minikube addons enable ingress`
- `kubectl port-forward -n ingress-nginx service/ingress-nginx-controller 8080:80`
- `curl localhost:8080`

Network Policies

- Bestimmt, wer mit wem kommunizieren darf. Eingehender und ausgehender Traffic auf IP:Port Ebene zwischen Pods und externen Systemen
- Das verwendete Netzwerk CNI muss Network Policies supporten, sonst hat diese Ressource keine Wirkung

Auf welchen Ebenen kann man Regeln erstellen?

- Namespace (selector!)
- Pods (selector!)
- IP-Block

Per default darf jeder Pod mit jedem überall sprechen (allow all)!

Dies kann man jedoch einschränken, dass erstmal nichts erlaubt ist und man explizit erlauben muss, wer mit wem reden darf (=whitelisting). Dies ist ein Grundprinzip der Netzwerksicherheit und sollte für eigene Applikationen stats angewandt werden.

NPs ergänzen sich: Ein Pod, für den zwei NPs gelten, darf beides machen. (1. NP erlaubt Kommunikation mit a; 2. NP erlaubt Kommunikation mit b => Pod darf mit a und b reden)

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: ksm-can-be-accessed-by-my-app
  namespace: kube-system
spec:
  podSelector:
    matchLabels:
      app: kube-state-metrics
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: my-app-that-needs-
access-to-ksm
  ports:
    - protocol: TCP
      port: 10301
```

Network Policies

- Default deny all ingress traffic
- Default deny all ingress & egress traffic

```
---  
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: default-deny-ingress  
spec:  
  podSelector: {}  
  policyTypes:  
  - Ingress
```

```
---  
apiVersion:  
networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: default-deny  
spec:  
  podSelector: {}  
  policyTypes:  
  - Ingress  
  - Egress
```

Network Policies: Aufgabe

- Erstellen Sie eine sehr restriktive NetworkPolicy für diesen Namespace (deny all)
- Können die Pods noch miteinander reden?
- Warum ja/nein?
- Was wäre der Soll Zustand?
- Erstellen Sie nun eine Regel, die folgende Verbindung erlaubt: nginx ingress => webserver pod

Network Policies: Lösung

- kubectl apply -n schulung2 -f ./Kubernetes-Basis-Schulung/Kubernetes/11_networkpolicy/np-deny-all-ingress.yaml
- kubectl apply -n schulung2 -f ./Kubernetes-Basis-Schulung/Kubernetes/11_networkpolicy/allow-web-ingress.yaml
- kubectl port-forward -n ingress-nginx service/ingress-nginx-controller 8080:80
- curl localhost:8080
- Wichtig: das verwendete Network CNI muss die NP unterstützen

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allowwebserver
  namespace: schulung2
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/name: webserver
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          app.kubernetes.io/name: ingress-nginx
  ports:
  - protocol: TCP
    port: 80
```

Custom Resource Definitions (CRDs)

- Erweiterung der K8s-API um eigene Objekte
- Scope: cluster oder namespace
- <https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/>

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: 5
```

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.com
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer
            # either Namespaced or Cluster
            scope: Namespaced
            names:
              # plural name to be used in the URL: /apis/<group>/<version>/<plural>
              plural: crontabs
              # singular name to be used as an alias on the CLI and for display
              singular: crontab
              # Kind is normally the CamelCased singular type. Your resource manifests use this.
              kind: CronTab
              # shortNames allow shorter string to match your resource on the CLI
              shortNames:
                - ct
```

Operators

"Operators are software extensions to Kubernetes that make use of [custom resources](#) to manage applications and their components. Operators follow Kubernetes principles, notably the [control loop](#)."

- Automating Tasks for CRDs
- E.g. Checks and monitors states, runs scans on a continuous basis
- An example operator
 - Some of the things that you can use an operator to automate include:
 - deploying an application on demand
 - taking and restoring backups of that application's state
 - handling upgrades of the application code alongside related changes such as database schemas or extra configuration settings
 - publishing a Service to applications that don't support Kubernetes APIs to discover them
 - simulating failure in all or part of your cluster to test its resilience
 - choosing a leader for a distributed application without an internal member election process

<https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>

Ausblick

Ausblick

- Cluster Services (monitoring, logging)
- Kubernetes Tools (stern, helm / kustomize)
- Service Mesh
- Harbor
- Security
- CI / CD (devsecops)
- GitOps (argocd, ansible, gitlabci)

Security

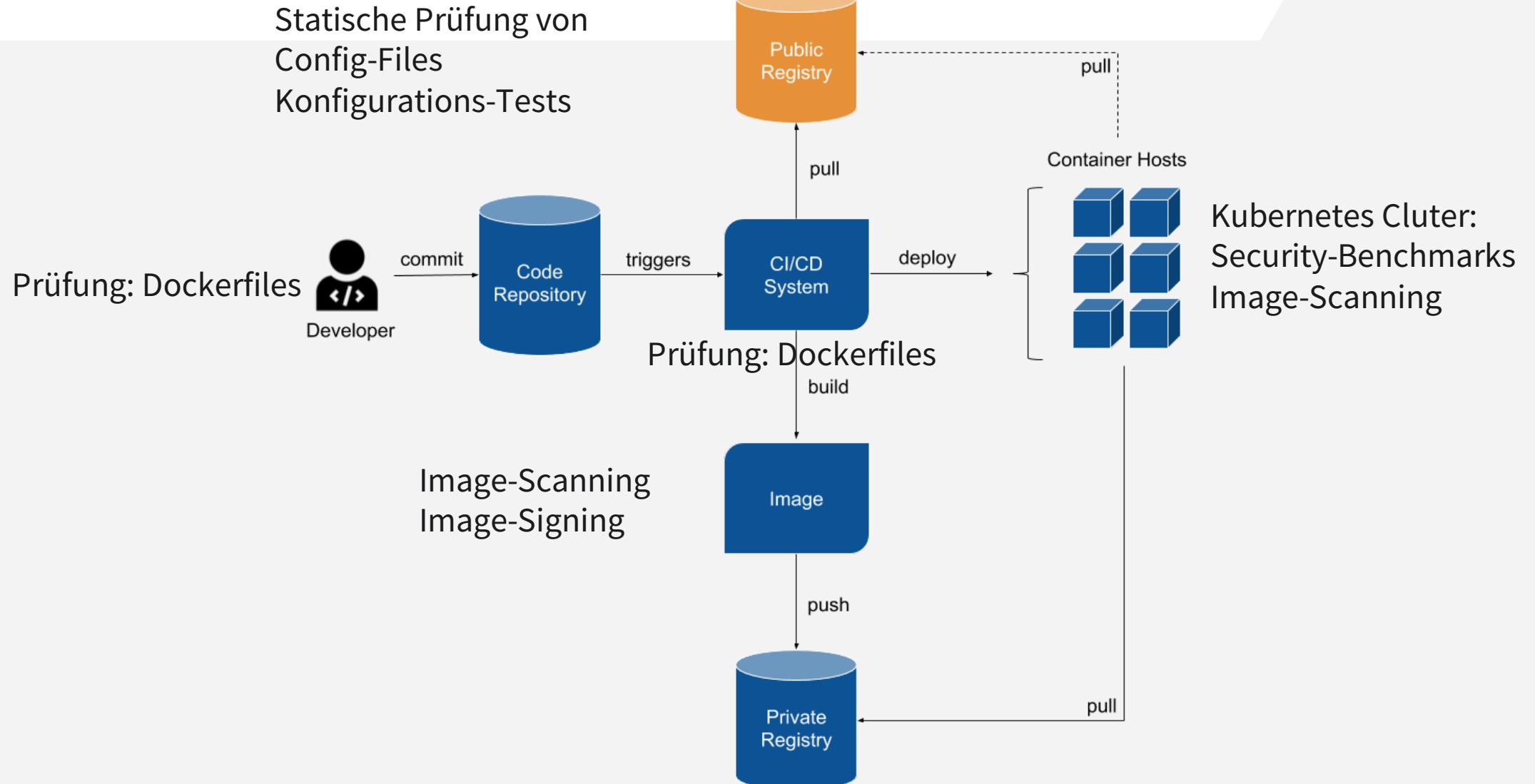
- Development Prozess (CI / CD)
 - Docker
 - git push ⇒ CI ⇒ Docker build ⇒ Scan the created image ⇒ Report
 - Image signing
 - Linting
 - Testen
 - Helm
 - Ggf. Helm Chart erstellen, linten, deployen, testen, scannen, pushen, ...
 - Automatische Schwachstellen, Überblick der laufenden Umgebung:
 - Liste aller laufenden Container mit Vulnerabilities
 - Applikation
 - Best Practices (Dockerfile, User-Rechte, ...)
 - Network Policies nutzen (Whitelisting!)
 - Ggf. eigene PSPs

Security

- Security Richtlinien für Container:
 - BSI SYS 1.6 Container
 - Benchmarks (CIS)
 - NIST
- Infrastruktur
 - Container-, Hostsystem- und Kubernetes-Hardening
 - Auditlogs
 - Forensic readiness
 - Default PSPs nutzen

CI/CD

Whitelist-Filter
Enforce Image-Signoff



Tools

- K9s (kurz aufrufen!)
- Stern
- kubectx + kubens - [Switch faster between clusters and namespaces in kubectl](#)

Toolsammlungen:

- <https://collabnix.github.io/kubetools/>
- <https://kubernetes.io/de/docs/reference/tools/>
- <https://phoenixnap.com/blog/kubernetes-tools>

Learning

- youtube
 - Kanal von Anais Urlich: youtube.com/c/AnaisUrlich
 - Kanal "TechWorld with Nana": youtube.com/c/TechWorldwithNana
 - Kubernetes Tutorial for Beginners [FULL COURSE in 4 Hours]: youtube.com/watch?v=X48VuDVv0do
 - Docker Tutorial for Beginners [FULL COURSE in 3 Hours]: youtube.com/watch?v=3c-iBn73dDE
 - [Kubernetes Course - Full Beginners Tutorial \(Containerize Your Apps!\) \[2:58\]](#)
- Udemy
 - [Certified Kubernetes Administrator \(CKA\) with Practice Tests](#)

Offene Fragen?

Feedback and Review

<https://forms.office.com/r/PYEkWPz6EZ>

Wir bauen Ihre Cloud Native Zukunft.

Vielen Dank für Ihre Aufmerksamkeit.