

Deployment und Handling von Applikationen mit Kubernetes, Helm, Prometheus und Gitlab

Agenda

1. Kubernetes (Refresher)

- [Aufbau von Kubernetes](#)
- Kubernetes und seine Objekte (pods, replicaset, deployments, services, ingress)
- Verbinde mit kubectl
- Manifeste ausrollen (im Namespace) (2-3)
- Arbeiten mit non-root images

2. Kubernetes Praxis API-Objekte

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

3. Kubernetes Secrets / Sealed Secrets (bitnami)

- [Welche Arten von secrets gibt es?](#)
- [Übung mit secrets](#)
- [Übung mit sealed-secrets](#)

4. Kubernetes Probes (Liveness and Readiness)

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

5. Kubernetes Wartung / Fehleranalyse

- [Wartung mit drain / uncordon \(Ops\)](#)
- [Debugging Ingress](#)

6. Kubernetes Pods Disruption Budget

- [PDB - Uebung](#)

7. Kubernetes PodAffinity/PodAntiAffinity

- [Warum ?](#)
- [Übung](#)

8. Kubernetes - Kustomize

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

9. Kubernetes Paketmanagement (Helm)

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

10. Kubernetes - Storage

- [Praxis. Beispiel. NFS](#)

11. gitlab ci/cd

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

12. gitlab / Kubernetes (gitops)

- [gitlab Kubernetes Agent with gitops - mode](#)

13. gitlab / Kubernetes (CI/CD - old-school mit kubectl)

- [Vorteile gitlab-agent](#)
- [Step 1: Installation gitlab-agent for kubernetes](#)
- [Step 2: Debugging KUBE_CONTEXT - Community Edition](#)
- [Step 3: gitlab-ci.yml setup for deployment and sample manifest](#)

14. gitlab / Kubernetes (CI/CD - Auto Devops)

- [Was ist Auto DevOps](#)
- [Debugging KUBE_CONTEXT - Community Edition](#)

15. Prometheus

- [Prometheus/Grafana Überblick](#)
- [Prometheus Installation / Walkthrough](#)

16. Tipps & Tricks

- [Default namespace von kubectl ändern](#)
- [Ingress Controller auf DigitalOcean aufsetzen](#)
- [vi einrückungen für yaml](#)
- [gitlab runner as nonroot](#)
- [curl zum Überprüfen mit Pod](#)

17. RootLess / Security

- [seccomp-profile-default docker](#)
- [Pod Security Policy](#)
- [Pod Security Policy - Übung](#)
- [RunAsUser Exercise](#)
- [Offizielles RootLess Docker Image für Nginx](#)

18. Rechte

- [RBAC / Rechte allgemein](#)

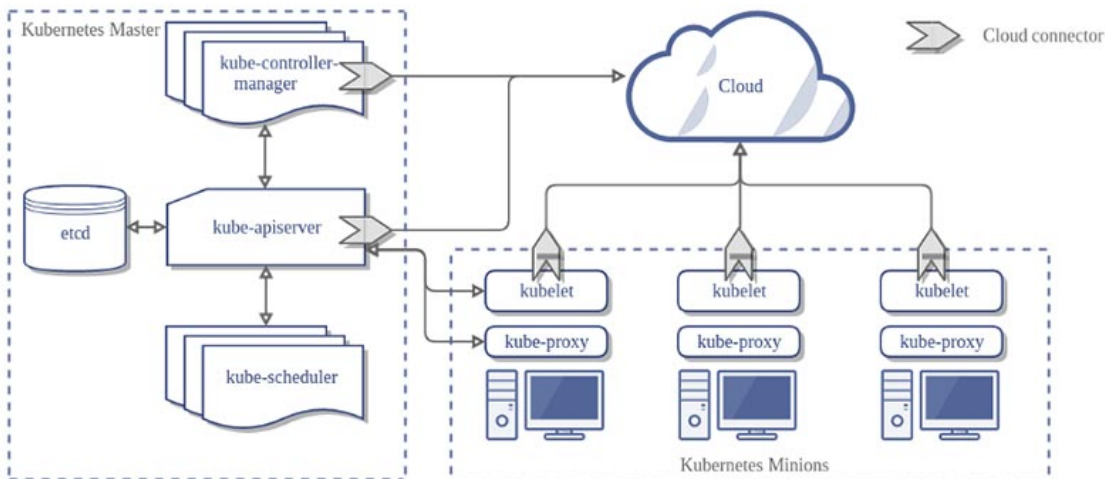
19. Documentation

- [helm dry-run vs. template](#)
- [Marktuebersicht Kubernetes Hosting](#)

Kubernetes (Refresher)

Aufbau von Kubernetes

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

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

Komponenten des Masters

ETCD

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

KUBE-CONTROLLER-MANAGER

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

KUBE-API-SERVER

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

KUBE-SCHEDULER

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

Nodes

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

Pod/Pods

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

Control Plane Node (former: master) - components

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

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

Kube-proxy

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

Referenzen

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

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops)

Allgemein

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

## Welche api-resources gibt es ?
kubectl api-resources
kubectl api-resources | grep namespaces

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

Namespace im context ändern

```
## mein default namespace soll ein anderer sein, z.B eines Projekt
kubectl config set-context --current --namespace=tln2
```

Hauptkommandos

```
kubectl get
kubectl delete
kubectl create
```

namespaces

```
kubectl get ns
kubectl get namespaces
```

Arbeiten mit manifesten

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

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

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

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

Ausgabeformate / Spezielle Informationen

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

## gilt natürluch auch für andere kommandos
kubectl get deploy -o json
kubectl get deploy -o yaml

## Label anzeigen
kubectl get deploy --show-labels
```

Zu den Pods

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

## Pods anzeigen
kubectl get pods
kubectl get pod

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

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

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

## Status eines Pods anzeigen
```

```
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

Arbeiten mit namespaces

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

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

Alle Objekte anzeigen

```
## Manchen Objekte werden mit all angezeigt
kubectl get all
kubectl get all,configmaps

## Über alle Namespaces hinweg
kubectl get all -A
```

Logs

```
kubectl logs <container>
kubectl logs <deployment>
## e.g.
## kubectl logs -n namespace8 deploy/nginx
## with timestamp
kubectl logs --timestamp -n namespace8 deploy/nginx
## continuously show output
kubectl logs -f <container>
```

Referenz

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

kubectl example with run

Example (that does work)

```
## Synopsis (most simplistic example)
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx

kubectl get pods
```

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

Example (that does not work)

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

### Ref:

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

### kubectl/manifest/pod

### Walkthrough
```

vi nginx-static.yml

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

- name: web image: bitnami/nginx

```
kubectl apply -f nginx-static.yml kubectl describe pod nginx-static-web
```

show config

```
kubectl get pod/nginx-static-web -o yaml kubectl get pod/nginx-static-web -o wide
```

```
### kubectl/manifest/replicaset
```

apiVersion: apps/v1 kind: ReplicaSet metadata: name: nginx-replica-set spec: replicas: 2 selector: matchLabels: tier: frontend template: metadata: name: nginx-replicas labels: tier: frontend spec: containers: - name: nginx image: "nginx:latest" ports: - containerPort: 80

```
### kubectl/manifest/deployments
```

vi nginx-deployment.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 2 # tells deployment to run 2 pods matching the template template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80

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

```
### kubectl/manifest/service
```

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

```
apiVersion: v1 kind: Service metadata: name: svc-my-nginx labels: run: svc-my-nginx spec: type: NodePort ports:
```

- port: 80 protocol: TCP selector: run: my-nginx

```
### Ref.

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

### Hintergrund Ingress

### Ref. / Dokumentation

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

### Documentation for default ingress nginx

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

### Beispiel mit Hostnamen

### Prerequisites
```

Ingress Controller muss aktiviert sein

Nur der Fall wenn man microk8s zum Einrichten verwendet

Ubuntu

microk8s enable ingress

```
### Walkthrough
```

mkdir apple-banana-ingress

cd apple-banana-ingress

apple.yml

vi apple.yml

```
kind: Pod apiVersion: v1 metadata: name: apple-app labels: app: apple spec: containers: - name: apple-app image:
hashicorp/http-echo args: - "-text=apple-tln12"
```

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

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

banana

vi banana.yml

```
kind: Pod apiVersion: v1 metadata: name: banana-app labels: app: banana spec: containers: - name: banana-app
image: hashicorp/http-echo args: - "-text=banana-tln12"
```

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

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

Ingress

apiVersion: extensions/v1beta1 kind: Ingress metadata: name: example-ingress annotations:
ingress.kubernetes.io/rewrite-target: / # with the ingress controller from helm, you need to set an annotation #
otherwise it does not know, which controller to use kubernetes.io/ingress.class: nginx spec: rules:

- host: "app12.lab1.t3isp.de" http: paths:

```
- path: /apple
  backend:
    serviceName: apple-service
    servicePort: 80
- path: /banana
  backend:
    serviceName: banana-service
    servicePort: 80
```

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

Reference

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

Find the problem

```
## Hints

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

## 2. Let's see, how the configuration works
kubectl explain --api-version=networking.k8s.io/v1
```

```
ingress.spec.rules.http.paths.backend.service
```

```
## now we can adjust our config
```

Solution

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

Kubernetes Secrets / Sealed Secrets (bitnami)

Welche Arten von secrets gibt es?

Welche Arten von Secrets gibt es ?

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

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

Übung mit secrets

Übung 1 - ENV Variablen aus Secrets setzen

```
## Schritt 1: Secret anlegen.  
## Diesmal noch nicht encoded - base64  
## vi 06-secret-unencoded.yml  
apiVersion: v1  
kind: Secret  
metadata:  
  name: mysecret  
type: Opaque  
stringData:  
  APP_PASSWORD: "s3c3tp@ss"  
  APP_EMAIL: "mail@domain.com"
```

```
## Schritt 2: Apply'en und anschauen  
kubectl apply -f 06-secret-unencoded.yml  
## ist zwar encoded, aber last_applied ist im Klartext  
## das könnte ich nur umgehen, in dem ich es encoded speichere  
kubectl get secret mysecret -o yaml
```

```
## Schritt 3:  
## vi 07-print-envs-complete.yml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: print-envs-complete  
spec:  
  containers:  
  - name: env-ref-demo  
    image: nginx  
    env:  
  - name: APP_VERSION  
    value: 1.21.1  
  - name: APP_PASSWORD  
    valueFrom:  
      secretKeyRef:  
        name: mysecret  
        key: APP_PASSWORD  
  - name: APP_EMAIL  
    valueFrom:  
      secretKeyRef:  
        name: mysecret  
        key: APP_EMAIL
```

```
## Schritt 4:  
kubectl apply -f 07-print-envs-complete.yml  
kubectl exec -it print-envs-complete -- bash  
##env | grep -e APP_ -e MYSQL
```

Übung mit sealed-secrets

2 Komponenten

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

Schritt 1: Walkthrough - Client Installation (als root)

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

Schritt 2: Walkthrough - Server Installation mit kubectl client

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

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

```
kubeseal --fetch-cert

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

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

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

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

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

kubectl apply -f basic-auth-sealed.yaml

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

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

Hinweis: Ubuntu snaps

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

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

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

Ref:

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

Kubernetes Probes (Liveness and Readiness)

Übung Liveness-Probe

Übung 1: Liveness (command)

```
What does it do ?

* At the beginning pod is ready (first 30 seconds)
* Check will be done after 5 seconds of pod being startet
* Check will be done periodically every 5 minutes and will check
  * for /tmp/healthy
  * if not there will return: 0
  * if not there will return: 1
* After 30 seconds container will be killed
* After 35 seconds container will be restarted
```

```
## cd
## mkdir -p manifests/probes
## cd manifests/probes
## vi 01-pod-liveness-command.yml

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
```

```
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
## apply and test
kubectl apply -f 01-pod-liveness-command.yml
kubectl describe -l test=liveness pods
sleep 30
kubectl describe -l test=liveness pods
sleep 5
kubectl describe -l test=liveness pods
```

```
## cleanup
kubectl delete -f 01-pod-liveness-command.yml
```

Übung 2: Liveness Probe (HTTP)

```
## Step 0: Understanding Prerequisite:
This is how this image works:
## after 10 seconds it returns code 500
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([](fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]("ok"))
    }
})
```

```
## Step 1: Pod - manifest
## vi 02-pod-liveness-http.yml
## status-code >=200 and < 400 o.k.
## else failure
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
```

```
image: k8s.gcr.io/liveness
args:
- /server
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
    httpHeaders:
    - name: Custom-Header
      value: Awesome
  initialDelaySeconds: 3
  periodSeconds: 3
```

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

Reference:

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

Funktionsweise Readiness-Probe vs. Liveness-Probe

Why / Howto / Difference to LiveNess

- Readiness checks, if container and if not
 - SENDS NO TRAFFIC to the container
- They are configured exactly the same, but use another keyword
 - readinessProbe instead of livenessProbe

Example

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Reference

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

Kubernetes Wartung / Fehleranalyse

Wartung mit drain / uncordon (Ops)

```
## Achtung, bitte keine pods verwenden, dies können "ge"-drained (ausgetrocknet) werden
kubectl drain <node-name>
z.B.
## Daemonsets ignorieren, da diese nicht gelöscht werden
kubectl drain n17 --ignore-daemonsets

## Alle pods von replicaset werden jetzt auf andere nodes verschoben
```

```
## Ich kann jetzt wartungsarbeiten durchführen

## Wenn fertig bin:
kubectl uncordon n17

## Achtung: deployments werden nicht neu ausgerollt, dass muss ich anstossen.
## z.B.
kubectl rollout restart deploy/webserver
```

Debugging Ingress

1. Schritt, Nginx-Pods finden und was sagen die Logs des controller

```
## -A alle namespaces
kubectl get pods -A | grep -i ingress
## jetzt sollten die pods zu sehen
## Dann logs der Pods anschauen und gucken, ob Anfrage kommt
## Hier steht auch drin, wo sie hin geht (zu welcher PodIP)
## microk8s -> namespace ingress
## Frage: HTTP_STATUS_CODE welcher ? z.B. 404
kubectl logs -n default <controller-ingress-pod>

## FEHLERFALL 1 (in logs):
"Ignoring ingress because of error while validating ingress class"

## Lösung wir haben vergessen, die IngressClass mit anzugeben
## Das funktioniert bei microk8s, aber nicht bei Installation aus helm

## Zeile bei annotations ergänzen
annotations:
  kubernetes.io/ingress.class: nginx
## kubectl apply -f 03-ingress.yml
```

2. Schritt Pods finden, die als Ingress Controller fungieren und log nochmal checken

```
## -A alle namespaces
kubectl get pods -A | grep -i ingress
## jetzt sollten die pods zu sehen
## Dann logs der Pods anschauen und gucken, ob Anfrage kommt
## Hier steht auch drin, wo sie hin geht (zu welcher PodIP)
## microk8s -> namespace ingress
## Frage: HTTP_STATUS_CODE welcher ? z.B. 404
kubectl logs -n default <controller-ingress-pod>
```

3. Schritt Pods analysieren, die Anfrage bekommen

```
## Dann den Pod herausfinden, wo die Anfrage hinging
## anhand der IP
kubectl get pods -o wide

## Den entsprechenden pod abfragen bzgl. der Logs
kubectl logs <pod-name-mit-ziel-ip>
```

Fehlerfall: ingress correct, aber service und pod nicht da


```
## Es kommt beim Aufrufen der Seite - 503 Server temporarily not available

## Teststellung
kubectl delete -f 01-apple.yml

## Seite aufrufen über browser

## Das sagen die logs
## Es taucht hier auch keine Ziel-IP des pods auf.
kubectl logs -n default <controller-ingress-pod>
104.248.254.206 - - [22/May/2022:07:23:28 +0000] (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36" 471 0.000 [tln2-
apple-service-80] [] - - - 6c120f60faa57d2ea4409e87d544b1b0

## Lösung: Hier sollten wir überprüfen, ob
## a) Der Pod an sich erreichbar ist
## b) Der service generell erstmal den pod erreichen kann (intern über clusterIP)

## Wichtig:
## In den Logs von nginx wird nur eine ip angezeigt, wenn sowohl service als auch pod da sind und
erreichbar
## Beispiel: Hier ist er erreichbar !! -> IP 10.224.1.4
## 10.135.0.5 - - [22/May/2022:07:31:17 +0000] (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36" 497 0.007 [tln2-
apple-service-80] [] 10.244.1.4:5678 12 0.004 200 42288726fa35984ccdd07d67aacde8f2
```

Debugging mit Curl

```
kubectl run -it --rm curly --image=curlimages/curl -- sh
## alternativ direkt verwenden
kubectl run -it --rm curly --image=curlimages/curl -- curl 10.14.35.10

## Hiermit dann connection zu services und pods testen
kubectl get svc pods -o wide
## damit ips sehen
```

```
## Kubernetes Pods Disruption Budget

### PDB - Übung

### Warum ?
```

PDB ermöglicht, dass sichergestellt wird, dass immer eine bestimmte Mindestanzahl an Pods für ein bestimmtes Label laufen -> minAvailable

(oder max eine maximale Anzahl nicht verfügbar: maxUnavailable)

```
### Wann ?
```

Das ganze funktioniert nur für: Voluntary disruptions

(D.h. ich beende bewusst durch meine Aktion einen Pod, z.B. durch drain'en)

Für Involuntary disruptions .. z.B. Absturz.. funktioniert das ganze nicht

```
### Übungsbeispiele (nur für Gruppen, wo jeder sein eigenes Cluster hat)
```

Situation: 3-node-cluster

Schritt 1:

Deployment erstellen

mkdir pdb-test

cd pdb-test

vi 01-pdb-test-deploy.yml

vi nginx-deployment.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-test-deployment

tells deployment to run 2 pods matching the template

spec: selector: matchLabels: app-test: nginx replicas: 10 template: metadata: labels: app-test: nginx spec: containers:
- name: nginx image: nginx:latest ports: - containerPort: 80

Schritt 2:

kubectl apply -f 01-pdb-test-deploy.yml

Schritt 3:

vi 02-pdb-test-budget.yml

pdb festlegen

% oder Zahl möglich

auch maxUnavailable ist möglich statt minAvailable

apiVersion: policy/v1 kind: PodDisruptionBudget metadata: name: pdb-test spec: minAvailable: 50% selector:
matchLabels: app-test: nginx

Schritt 4: pdb apply'en

kubectl apply -f 02-pdb-test-budget.yml

Schritt 5: Erste node drainen

hier geht noch alles

kubectl drain kubectl get pdb kubectl describe pdb

Schritt 6: 2. Node drainen

hier geht auch noch alles, aber evtl. bereits meldungen

von System-Pods

kubectl drain kubectl get pdb kubectl describe pdb

Schritt 7: 3. Node drainen

jetzt kommen meldungen - pod cannot be evicted

von System-Pods

kubectl drain kubectl get pdb kubectl describe pdb

```
## Kubernetes PodAffinity/PodAntiAffinity

### Warum ?

### PodAffinity - Was ist das ?
```

Situation: Wir wissen nicht, auf welchem Node ein Pod läuft, aber wir wollen sicherstellen das ein "verwandter Pod" auf dem gleichen Node läuft.

Es ist auch denkbar für: o im gleichen Rechenzentrum o im gleichen Rack etc.

```
### PodAntiAffinity - Was ist das ?
```

Das genaue Gegenteil zu PodAffinity: Ein weitere Pod B, soll eben gerade nicht dort laufen wo Pod A läuft. Im einfachsten Fall gerade nicht auf dem gleichen Host/Node

```
### PodAntiAffinity - auch möglich für pods mit gleichem Label
```

Spezialfall: Ich möchte sicherstellen, dass jeder meiner Pods mit gleich label

auf einem anderen Node/Host

```
### Übung
```

```
### Übung 1: PodAffinity (required) - auf gleicher Node/Hostname
```

Schritt 1.

mkdir pod-affinity-test

cd pod-affinity-test

vi 01-busybox-sleep.yml

```
apiVersion: v1 kind: Pod metadata: name: sleep-busybox labels: app: backend spec: containers: - name: bb image: busybox command: ["sleep"] args: ["999999"]
```

Schritt 2:

```
kubectl apply -f 01-busybox-sleep.yml
```

Schritt 3:

Welche Hostnamen gibt es.

Wichtig, um topologyKey zu verstehen:

```
kubectl get nodes -o yaml | grep hostname
```

Schritt 4:

Deployment mit podAffinity festlegen

vi 02-fronted-nginx.yml

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-frontend spec: selector: matchLabels: frontend: nginx
```

```
replicas: 10 template: metadata: labels: frontend: nginx spec: affinity: podAffinity: requiredDuringSchedulingIgnoredDuringExecution: - labelSelector: matchLabels: app: backend topologyKey: kubernetes.io/hostname
```

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

Schritt 5: Prüfen und ausrollen

auf welcher node läuft busybox

```
kubectl get pods -l app=backend -o wide
```

deployment ausrollen

```
kubectl apply -f 02-fronted-nginx.yml
```

Wo laufen die Deployments ?

```
kubectl get pods -l frontend=nginx -o wide
```

und wir lassen uns nochmal das describe ausgeben

```
kubectl describe pods nginx-frontend-
```

```
### Übung 2: PodAffinity (required) - im gleichen Rack
```

Schritt 1:

Bei einem cluster, dieser Schritt nur durch trainer

```
kubectl get nodes --show-labels kubectl label nodes pool-tg5g9rh4y-cw8mb rack=1 kubectl label nodes pool-tg5g9rh4y-cw8mr rack=1 kubectl label nodes pool-tg5g9rh4y-cw8mw rack=2
```

Schritt 2.

mkdir pod-affinity-racktest

cd pod-affinity-racktest

vi 01-busybox-sleep.yml

```
apiVersion: v1 kind: Pod metadata: name: sleep-busybox labels: app: backend spec: containers: - name: bb image: busybox command: ["sleep"] args: ["999999"]
```

Schritt 3:

```
kubectl apply -f 01-busybox-sleep.yml
```

Schritt 4:

Welche Hostnamen gibt es.

Wichtig, um topologyKey zu verstehen:

```
kubectl get nodes -o yaml | grep hostname
```

Schritt 5:

Deployment mit podAffinity festlegen

vi 02-fronted-nginx.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-frontend spec: selector: matchLabels: frontend: nginx

replicas: 10 template: metadata: labels: frontend: nginx spec: affinity: podAffinity:

requiredDuringSchedulingIgnoredDuringExecution: - labelSelector: matchLabels: app: backend topologyKey: rack

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

Schritt 6: Prüfen und ausrollen

auf welcher node läuft busybox

kubectl get pods -l app=backend -o wide

deployment ausrollen

kubectl apply -f 02-frontend-nginx.yml

Wo laufen die Deployments ?

kubectl get pods -l frontend=nginx -o wide

und wir lassen uns nochmal das describe ausgeben

kubectl describe pods nginx-frontend-

```
### Übung 3: PodAntiAffinity (forced) auf anderem Hosts
```

Schritt 1.

mkdir pod-affinity-test

cd pod-affinity-test

vi 01-busybox-sleep.yml

apiVersion: v1 kind: Pod metadata: name: sleep-busybox labels: app: backend spec: containers: - name: bb image: busybox command: ["sleep"] args: ["999999"]

Schritt 2:

kubectl apply -f 01-busybox-sleep.yml

Schritt 3:

Welche Hostnamen gibt es.

Wichtig, um topologyKey zu verstehen:

```
kubectl get nodes -o yaml | grep hostname
```

Schritt 4:

Deployment mit podAffinity festlegen

vi 02-fronted-nginx.yml

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-frontend spec: selector: matchLabels: frontend: nginx
```

```
replicas: 10 template: metadata: labels: frontend: nginx spec: affinity: podAntiAffinity:
```

```
requiredDuringSchedulingIgnoredDuringExecution: - labelSelector: matchLabels: app: backend topologyKey:
```

```
kubernetes.io/hostname
```

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

Schritt 5: Prüfen und ausrollen

auf welcher node läuft busybox

```
kubectl get pods -l app=backend -o wide
```

deployment ausrollen

```
kubectl apply -f 02-fronted-nginx.yml
```

Wo laufen die Deployments ? // das muss jetzt auf anderen Hosts sein

```
kubectl get pods -l frontend=nginx -o wide
```

und wir lassen uns nochmal das describe ausgeben

```
kubectl describe pods nginx-frontend-
```

```
### Übung 4: PodAffinity (preferred) - auf gleicher Node/Hostname
```

Schritt 1.

mkdir pod-affinity-preferred

cd pod-affinity-preferred

vi 01-busybox-sleep.yml

apiVersion: v1 kind: Pod metadata: name: sleep-busybox labels: app: backend spec: containers: - name: bb image: busybox command: ["sleep"] args: ["999999"]

Schritt 2:

```
kubectl apply -f 01-busybox-sleep.yml
```

Schritt 3:

Welche Hostnamen gibt es.

Wichtig, um topologyKey zu verstehen:

```
kubectl get nodes -o yaml | grep hostname
```

Schritt 4:

Deployment mit podAffinity festlegen

vi 02-fronted-nginx.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-frontend spec: selector: matchLabels: frontend: nginx

replicas: 10 template: metadata: labels: frontend: nginx spec: affinity: podAntiAffinity: preferredDuringSchedulingIgnoredDuringExecution: - weight: 80 podAffinityTerm: labelSelector: matchLabels: app: backend topologyKey: kubernetes.io/hostname

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

Schritt 5: Prüfen und ausrollen

auf welcher node läuft busybox

```
kubectl get pods -l app=backend -o wide
```

deployment ausrollen

```
kubectl apply -f 02-fronted-nginx.yml
```

Wo laufen die Deployments ?

```
kubectl get pods -l frontend=nginx -o wide
```

und wir lassen uns nochmal das describe ausgeben

kubectl describe pods nginx-frontend-

```
### Variante 5: PodAffinity (preferred) - auf gleicher Node/Hostname mit matchExpression
```

vi 02-fronted-nginx.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-frontend spec: selector: matchLabels: frontend: nginx

replicas: 10 template: metadata: labels: frontend: nginx spec: affinity: podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution: - weight: 100 podAffinityTerm: labelSelector: matchExpressions: - key: app operator: In values: - backend topologyKey: kubernetes.io/hostname

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

```
## Kubernetes - Kustomize
```

```
### Beispiel ConfigMap - Generator
```

```
### Walkthrough
```

External source of truth

Create a application.properties file

vi application.properties

USER=letterman ORG=it

No use the generator

the name need to be kustomization.yaml

kustomization.yaml

configMapGenerator:

- name: example-configmap-1 files:
 - application.properties

```
## See the output
kubectl kustomize ./
```

```
## run and apply it
kubectl apply -k .
## configmap/example-configmap-1-k4dmb9cbmb created
```

Ref.

- <https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/>

Beispiel Overlay und Patching

Konzept Overlay

- Base + Overlay = Gepatchtes manifest
- Sachen patchen.
- Die werden drübergelegt.

Example 1: Walkthrough

```
## Step 1:
## Create the structure
## kustomize-example1
## L base
## | - kustomization.yml
## L overlays
##.   L dev
##     - kustomization.yml
##.   L prod
##     - kustomization.yml
mkdir -p kustomize-example1/base
mkdir -p kustomize-example1/overlays/prod
cd kustomize-example1
```

```
## Step 2: base dir with files
## now create the base kustomization file
## vi base/kustomization.yml
resources:
- service.yml
```

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

```
## See how it looks like
kubectl kustomize ./base
```

```
## Step 4: create the customization file accordingly
##vi overlays/prod/kustomization.yml
bases:
- ../../base
patches:
- service-ports.yaml
```

```
## Step 5: create overlay (patch files)
## vi overlays/prod/service-ports.yaml
kind: Service
apiVersion: v1
metadata:
  #Name der zu patchenden Ressource
  name: service-app
spec:
  # Changed to Nodeport
  type: NodePort
  ports: #Die Porteinstellungen werden überschrieben
    - name: https
      port: 443
```

```
## Step 6:
kubectl kustomization overlays/prod

## or apply it directly
kubectl apply -k overlays/prod
```

```
## Step 7:
## mkdir -p overlays/dev
## vi overlays/dev/kustomization.yml
bases:
- ../../base
```

```
## Step 8:
## statt mit der base zu arbeiten
kubectl kustomize overlays/dev
```

Example 2: Advanced Patching with patchesJson6902 (You need to have done example 1 firstly)

```
## Schritt 1:
## Replace overlays/prod/kustomization.yml with the following syntax
bases:
- ../../base
patchesJson6902:
- target:
    version: v1
    kind: Service
    name: service-app
  path: service-patch.yaml
```

```
## Schritt 2:
## vi overlays/prod/service-patch.yaml
- op: replace
  path: /spec/type
  value: NodePort

- op: remove
  path: /spec/ports
  value:
    - name: http
      port: 80

- op: add
```

```
path: /spec/ports
value:
- name: https
  port: 443
```

```
## Schritt 3:
kubectl kustomize overlays/prod
```

Special Use Case: Change the metadata.name

```
## Same as Example 2, but patch-file is a bit different
## vi overlays/prod/service-patch.yaml
- op: remove
  path: /spec/ports
  value:
    - name: http
      port: 80

- op: add
  path: /spec/ports
  value:
    - name: https
      port: 443

- op: replace
  path: /metadata/name
  value: svc-app-test
```

```
kubectl kustomize overlays/prod
```

Ref:

- <https://blog.ordix.de/kubernetes-anwendungen-mit-kustomize>

Resources

Where ?

- Used in base

```
## base/kustomization.yml
## which resources to use
## e.g
resources:
- my-manifest.yml
```

Which ?

- URL
- filename
- Repo (git)

Example:

```
## kustomization.yml
resources:
## a repo with a root level kustomization.yml
- github.com/Liujingfang1/mysql
```

```
## a repo with a root level kustomization.yaml on branch test
- github.com/Liujingfang1/mysql?ref=test
## a subdirectory in a repo on branch repoUrl2
- github.com/Liujingfang1/kustomize/examples/helloWorld?ref=repoUrl2
## a subdirectory in a repo on commit `7050a45134e9848fca214ad7e7007e96e5042c03`
- github.com/Liujingfang1/kustomize/examples/helloWorld?
ref=7050a45134e9848fca214ad7e7007e96e5042c03
```

Kubernetes Paketmanagement (Helm)

Warum ? (Dev/Ops)

Ein Paket für alle Komponenten
Einfaches Installieren und Updaten.
Feststehende Struktur, durch die andere Pakete teilen können

Grundlagen / Aufbau / Verwendung (Dev/Ops)

Wo ?

artifacts helm
<https://artifacthub.io/>

Komponenten

Chart - beinhaltet Beschreibung und Komponenten
tar.gz - Format

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

Installation

```
## Beispiel ubuntu
## snap install --classic helm

## Cluster muss vorhanden, aber nicht notwendig wo helm installiert

## Voraussetzung auf dem Client-Rechner (helm ist nichts als anderes als ein Client-Programm)
Ein lauffähiges kubectl auf dem lokalen System (welches sich mit dem Cluster verbinden.
-> saubere -> .kube/config

## Test
kubectl cluster-info
```

Installation: Ref:

- <https://helm.sh/docs/intro/install/>

Helm - wichtige Befehle

```
## Repos
helm repo add gitlab http://charts.gitlab.io
helm repo list
helm repo remove gitlab
helm repo update
```

```

## Suchen
helm search repo mysql # in allen konfigurierten Repos suchen

## Chart herunterladen
helm repo pull bitnami/mysql

## Releases anzeigen
helm list
## history anzeigen
helm history my-mysql

## Release installieren - my-mysql ist hier hier release-name
helm install my-mysql bitnami-mysql
helm install [name] [chart] --dry-run --debug -f <your_values_file> # dry run
## + verwendete values anzeigen
helm get values

## upgrade, wenn vorhanden, ansonsten install
helm upgrade --install my-mysql bitnami/mysql

## Nur template parsen - ohne an den kube-api-server zu schicken
helm template my-mysql bitnami/mysql > test.yml
## template und hilfeseite aufgaben und vorher alles an den kube-api-server
## zur Validierung schicken
helm install --dry-run my-mysql bitnami/mysql

```

Praktisches Beispiel bitnami/mysql (Dev/Ops)

Prerequisites

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

Example 1: We will setup mysql

```

helm repo add bitnami https://charts.bitnami.com/bitnami
## paketliste aktualisieren
helm repo update
helm search repo bitnami

```

```

## download chart - Optional
## for exercise: to learn how it is structured
helm pull bitnami/mysql
mkdir lookaround
cp -a mysql-*.tgz lookaround
cd lookaround
tar xvf mysql-*.tgz

```

```

helm install my-mysql bitnami/mysql

```

Example 2 - values in der Kommandozeile

```

### Vorbereiten - alte Installation löschen
helm uninstall my-mysql
kubectl delete pvc data-my-mysql-0

## Install with persistentStorage disabled - Setting a specific value
helm install my-mysql --set primary.persistence.enabled=false bitnami/mysql
helm get values my-mysql
## Alternative if already installed

## just as notice
## helm uninstall my-mysql

```

Example 3: values im extra-file (auch mehrere möglich)

```

## Aufräumen
helm uninstall my-mysql

```

```

## vi values.yaml
primary:
  persistence:
    enabled: false

```

```

helm install my-mysql -f values.yaml bitnami/mysql
## hilfe
helm get values --help
## Alle, auch defaults anzeigen
helm get values my-mysql --all # alternativ -a

helm get values my-mysql -o json
helm get values my-mysql # default: yaml ausgabe
helm list
## Allerdings nur 1 Eintrag, bei upgrade sinds mehrere drin
helm history my-mysql

```

Referenced

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

Kubernetes - Storage

Praxis. Beispiel. NFS

Create new server and install nfs-server

```

## on Ubuntu 20.04LTS
apt install nfs-kernel-server
systemctl status nfs-server

vi /etc/exports
## adjust ip's of kubernetes master and nodes
## kmaster
/var/nfs/ 192.168.56.101(rw,sync,no_root_squash,no_subtree_check)
## knode1
/var/nfs/ 192.168.56.103(rw,sync,no_root_squash,no_subtree_check)
## knode 2

```

```
/var/nfs/ 192.168.56.105(rw,sync,no_root_squash,no_subtree_check)

exportfs -av
```

On all nodes (needed for production)

```
##
apt install nfs-common
```

On all nodes (only for testing)

```
#### Please do this on all servers (if you have access by ssh)
### find out, if connection to nfs works !

## for testing
mkdir /mnt/nfs
## 192.168.56.106 is our nfs-server
mount -t nfs 192.168.56.106:/var/nfs /mnt/nfs
ls -la /mnt/nfs
umount /mnt/nfs
```

Setup PersistentVolume and PersistentVolumeClaim in cluster

```
## mkdir -p nfs; cd nfs
## vi 01-pv.yml
## Important user
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln<nr>
  labels:
    volume: nfs-data-volume-tln<nr>
spec:
  capacity:
    # storage size
    storage: 1Gi
  accessModes:
    # ReadWriteMany(RW from multi nodes), ReadWriteOnce(RW from a node), ReadOnlyMany(R from
multi nodes)
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    # retain even if pods terminate
    Retain
  nfs:
    # NFS server's definition
    path: /var/nfs/tln<nr>/nginx
    server: 192.168.56.106
    readOnly: false
    storageClassName: ""
```

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

```
## vi 02-pvs.yml
## now we want to claim space
apiVersion: v1
```



```
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim-tln<nr>
spec:
  storageClassName: ""
  volumeName: pv-nfs-tln<nr>
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

```
kubectl apply -f 02-pvs.yml
```

```
## deployment including mount
## vi 03-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # tells deployment to run 4 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
      volumeMounts:
        - name: nfsvol
          mountPath: "/usr/share/nginx/html"
      volumes:
        - name: nfsvol
          persistentVolumeClaim:
            claimName: pv-nfs-claim-tln1
```

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

```
## now testing it with a service
## cat 04-service.yml
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
  labels:
    run: svc-my-nginx
```

```
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

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

## connect to the container and add index.html - data
kubectl exec -it deploy/nginx-deployment -- bash
## in container
echo "hello dear friend" > /usr/share/nginx/html/index.html
exit

## now try to connect
kubectl get svc

## connect with ip and port
kubectl exec -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>:<port> # port -> > 30000
## exit

## now destroy deployment
kubectl delete -f 03-deploy.yml

## Try again - no connection
kubectl exec -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>:<port> # port -> > 30000
## exit

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

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

gitlab ci/cd

Overview

Pipelines

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

Type of pipelines: Basic Pipeline

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

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

image: alpine

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

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

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

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

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

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

Type of pipelines: DAG (Directed Acyclic Graph) Pipelines

- Image:
- Deploy_a can run, although build_b->test_b is not even ready
- Because gitlab knows the dependencies by keyword: needs:

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

```

image: alpine

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

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

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

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

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

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

```

Type of pipelines: Child- / Parent - Pipelines

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

```

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

trigger_a:

```

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

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

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

image: alpine

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

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

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

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

image: alpine

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

test_b:
  stage: test
  needs: [build_b]
  script:
```

```

    - echo "This job tests something else."

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

```

Type of pipelines: Ref:

- https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html

Stages

- Stages run one after each other
- They default to: build, test, deploy (if you do not define any)
- If you want to have less, you have to define which
- Reference:

Jobs

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

Using the test - template

Example Walkthrough

```

## Schritt 1: Neues Repo aufsetzen

## Setup a new repo
## Setting:

## o Public, dann bekommen wir mehr Rechenzeit
## o No deployment planned
## o No SAST
## o Add README.md

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

## Schritt 2: Ein Standard-Template als Ausgangsbasis holen
## Get default ci-Template
CI-CD -> Pipelines -> Try Test-Template

## Testtemplate wird in file gitlab-ci.yaml angelegt.
## Es erscheint unter: CI-CD -> Editor

lx speichern und committen.

## Jetzt wird es in der Pipeline ausgeführt.

```

Examples running stages

Running default stages

- build, test, deploy are stages set by default

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

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

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

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

only run some

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

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

## unit-test-job wurde gelöscht

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

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

Predefined Vars

Example to show them

```
stages:
  - build

show_env:
  stage: build
  scripts:
```

```
- env
- pwd
```

Reference

- https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

Rules

Ref:

- https://docs.gitlab.com/ee/ci/jobs/job_control.html#specify-when-jobs-run-with-rules

Example Defining and using artifacts

What is it ?

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

Example: Creating an artifact

```
## .gitlab-ci.yml

stages:
  - build

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

Example creating artifacts with wildcards and different name

```
## .gitlab-ci.yml

stages:
  - build

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

Artifakten und Name aus Variable vergeben

- If your branch-name contains forward slashes
 - (for example feature/my-feature)
 - it's advised to use \$CI_COMMIT_REF_SLUG instead of \$CI_COMMIT_REF_NAME

- for proper naming of the artifact.

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

Alle files in einem Verzeichnis recursive

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

Artifakte und Bedingungen

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

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

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

create_txt:
  stage: build
  script:
```

```

- mkdir -p path/my-xyz
- echo "toplevel" > path/you-got-it.txt
- echo "hello" > path/my-xyz/ergebnis.txt
- mkdir -p path/some-xyz
- echo "some" > path/some-xyz/testtext.txt
- env
artifacts:
  paths:
    - path/

rules:
- if: $CI_COMMIT_TAG

```

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

Passing artifacts between stages (enabled by default)

```

image: ubuntu:20.04

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

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

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

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

```

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

```

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

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

build:
  stage: build

```

```

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

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

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

```

Passing artifacts (+omitting test - stage)

- You can decide in which state you need the artifacts

```

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

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

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

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

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

```

Using the gitlab - artifacts api

API - Reference:

- https://docs.gitlab.com/ee/api/job_artifacts.html

Reference:

- https://docs.gitlab.com/ee/ci/pipelines/job_artifacts.html

gitlab / Kubernetes (gitops)

gitlab Kubernetes Agent with gitops - mode

gitlab / Kubernetes (CI/CD - old-school mit kubectI)

Vorteile gitlab-agent

Disadvantage of solution before gitlab agent

- the requirement to open up the cluster to the internet, especially to GitLab
- the need for cluster admin rights to get the benefit of GitLab Managed Clusters
- exclusive support for push-based deployments that might not suit some highly regulated industries

Advantage

- Solved the problem of weaknesses.

Technical

- Connected to Websocket Stream of KAS-Server
- Registered with gitlab - project

Reference:

- <https://about.gitlab.com/blog/2020/09/22/introducing-the-gitlab-kubernetes-agent/>

Step 1: Installation gitlab-agent for kubernetes

Steps

```
### Step 1:
```

```
Create New Repository -  
name: b-tln<nr>
```

```
With  
README.md
```

```
### Step 2: config für agents anlegen
```

```
## .gitlab/agents/gitlab-tln<nr>/config.yaml # Achtung kein .yaml wird sonst nicht erkannt.  
## mit folgendem Inhalt
```

```
ci_access:  
  projects:  
    - id: dummyhoney/b-tln<nr>
```

```
### Step 3:
```

```
## agent registrieren / Cluster connecten
```

```
Infrastruktur > Kubernetes Clusters -> Connect a cluster (Agent)
```

```
Jetzt solltest du den Agent auswählen können und klickt auf Register
```

```
### Step 4:
```

```
## Du erhältst die Anweisungen zum Installieren und wandelst das ein bisschen ab,  
## für das Training:
```

```
## Den token verwendest du aus der Anzeige
## tln1 ersetzt durch jeweils (2x) durch Deine Teilnehmer-Nr.
helm upgrade --install gitlab-agent gitlab/gitlab-agent --namespace tln<nr> --create-namespace
--set config.token=<token-from-screen>
```

Step 2: Debugging KUBE_CONTEXT - Community Edition

Why ?

```
kubectl does not work, because KUBECONFIG is not set properly
```

Find out the context (without setting it)

```
## This overwrites auto devops completely
##.gitlab-ci.yml
deploy:
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  script:
    - set
    - kubectl config get-contexts
```

Test Context

```
## This overwrites auto devops completely
##.gitlab-ci.yml
deploy:
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  script:
    - set
    - kubectl config get-contexts
## this will be the repo and the name of the agent
## Take it from the last block
## you will see it from the pipeline
- kubectl config use-context dummyhoney/tln1:gitlab-tln1
- kubectl config set-context --current --namespace tln1
- kubectl get pods
- ls -la
- id
```

Fix by setting KUBE_CONFIG

```
## This is a problem in the community edition (CE)
## We need to fix it like so.
## Adjust it to your right context
## IN Settings -> CI/CD -> Variables
KUBE_CONFIG dummyhoney/spring-autodevops-tln1:gitlab-devops-tn1
```

Step 3: gitlab-ci.yml setup for deployment and sample manifest

Schritt 1: manifests - Struktur einrichten

```
## vi manifests/prod/01-pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-web2
  labels:
    webserver: nginx
spec:
  containers:
    - name: web
      image: bitnami/nginx
```

Schritt 2: gitlab-ci.yml mit kubectl apply --recursive -f

```
## CI-CD -> Editor oder .gitlab-ci.yml im Wurzelverzeichnis
## only change in stage: build
image:
  name: bitnami/kubectl
  entrypoint: [""]

deploy:
  stage: deploy
  script:
    - set
    - kubectl config get-contexts
    - kubectl config use-context dummyhoney/b-tln1:gitlab-tln1
    - kubectl config set-context --current --namespace tln1
    - ls -la
    - kubectl apply --recursive -f manifests/prod
```

Schritt 3: pipeline anschauen

- War es erfolgreich - kein Fehler ?

Schritt 4: Sichtprüfen mit kubectl über Client (lokaler Rechner/Desktop)

```
kubectl get pods | grep web2
```

gitlab / Kubernetes (CI/CD - Auto Devops)

Was ist Auto DevOps

Debugging KUBE_CONTEXT - Community Edition

Why ?

```
kubectl does not work, because KUBECONFIG is not set properly
```

Find out the context (without setting it)

```
## This overwrites auto devops completely
##.gitlab-ci.yml
deploy:
  image:
    name: bitnami/kubectl:latest
```

```
    entrypoint: [""]
script:
  - set
  - kubectl config get-contexts
```

Test Context

```
## This overwrites auto devops completely
##.gitlab-ci.yml
deploy:
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  script:
    - set
    - kubectl config get-contexts
## this will be the repo and the name of the agent
## Take it from the last block
## you will see it from the pipeline
  - kubectl config use-context dummyhoney/tlnl:gitlab-tlnl
  - kubectl config set-context --current --namespace tlnl
  - kubectl get pods
  - ls -la
  - id
```

Fix by setting KUBE_CONFIG

```
## This is a problem in the community edition (CE)
## We need to fix it like so.
## Adjust it to your right context
## IN Settings -> CI/CD -> Variables
KUBE_CONFIG dummyhoney/spring-autodevops-tlnl:gitlab-devops-tnl
```

Prometheus

Prometheus/Grafana Überblick

What does it do ?

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

Technical

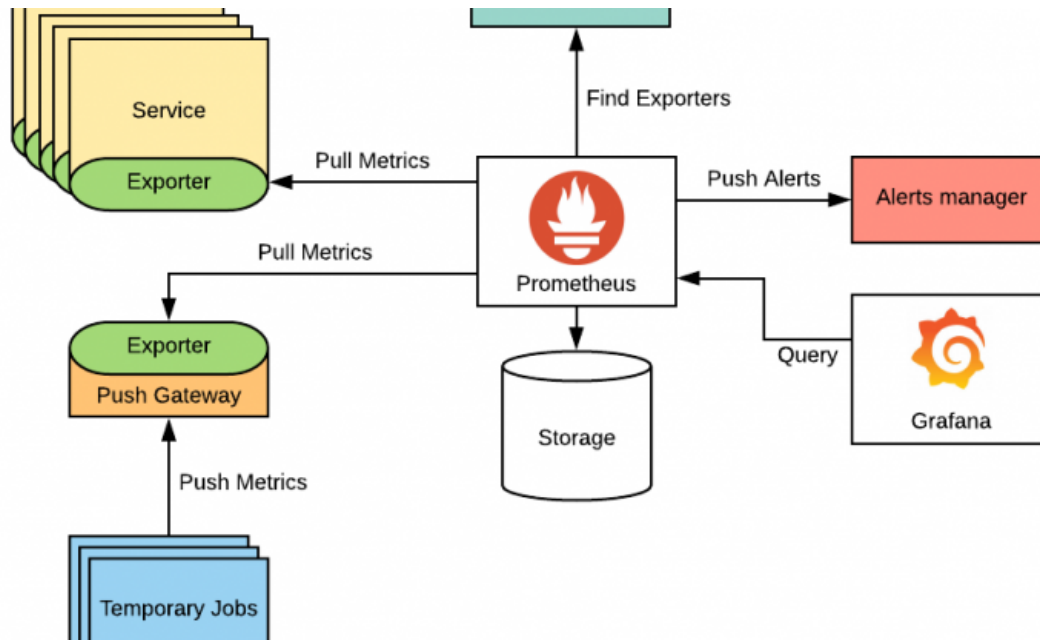
- Prometheus has a TDB (Time Series Database) and is good as storing time series with data
- Prometheus includes a local on-disk time series database, but also optionally integrates with remote storage systems.
- Prometheus's local time series database stores data in a custom, highly efficient format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

What are time series ?

- A time series is a sequence of data points that occur in successive order over some period of time.
- Beispiel:
 - Du willst die täglichen Schlusspreise für eine Aktie für ein Jahr dokumentieren
 - Damit willst Du weitere Analysen machen

- Du würdest das Paar Datum/Preis dann in der Datumsreihenfolge sortieren und so ausgeben
- Dies wäre eine "time series"

Komponenten von Prometheus



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

Prometheus Server

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

Grafana ?

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

Prometheus Installation / Walkthrough

Prerequisites

- Ubuntu 20.04 with running microk8s single cluster
- Works on any other cluster, but installing helm is different

Prepare

```
## Be sure helm is installed on your client
## In our walkthrough, we will do it directly on 1 node,
```



```
## which is not recommended for Production
```

Walkthrough

Step 1: install helm, if not there yet

```
snap install --classic helm
```

Step 2: Rollout prometheus/grafana stack in namespace prometheus

```
## add prometheus repo
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update

## install stack into new prometheus namespace
helm install -n prometheus --create-namespace prometheus prometheus-community/kube-prometheus-stack

## After installation look at the pods
## You should see 3 pods
kubectl --namespace prometheus get pods -l "release=prometheus"

## After a while it should be more pods
kubectl get all -n prometheus
```

Step 3a Let's explain (der Prometheus - Server)

```
## 2 Stateful sets
kubectl get statefulsets -n prometheus
## output
## alertmanager-prometheus-kube-prometheus-alertmanager 1/1 5m14s
## prometheus-prometheus-kube-prometheus-prometheus. 1/1. 5m23s
```

```
## Moving part 1:
## prometheus-prometheus-kube-prometheus-prometheus
## That is the core prometheus server based on the main image

## Let's validate
## schauen wir mal in das File
kubectl get statefulset -n prometheus -o yaml > sts-prometheus-server.yaml

## Und vereinfacht (jetzt sehen wir direkt die beiden verwendeten images)
## 1) prometheus - server
## 2) der dazugehörige config-reloader als Side-Car
kubectl get sts -n prometheus prometheus-prometheus-kube-prometheus-prometheus -o
jsonpath='{.spec.template.spec.containers[*].image}'

## Aber wer managed den server -> managed-by -> kubernetes-operator
kubectl get sts -n prometheus prometheus-prometheus-kube-prometheus-prometheus -o jsonpath="
{.spec.template.metadata.labels}" | jq .

## Wir der sts von helm erstellt ?
## NEIN ;o)
## show us all the template that helm generate to apply them to kube-api-server
helm template prometheus prometheus-community/kube-prometheus-stack > all-prometheus.yaml
## NOPE -> none
cat all-prometheus.yaml | grep -i kind: | grep -i stateful
```

```
## secrets -> configuration von prometheus
## wenn ein eigenschaft Punkte hat, z.B. prometheus.yaml.gz
##
## {"prometheus.yaml.gz":"H4s
## dann muss man escapen, um darauf zuzugreifen -> aus . wird \.
kubectl get -n prometheus secrets prometheus-prometheus-kube-prometheus-prometheus -o
jsonpath='{.data.prometheus\.yaml\.gz}' | base64 -d | gzip -d -
```

Step 3b: Prometheus Operator und Admission Controller -> Hook

```
## The Prometheus Operator for Kubernetes
## provides easy monitoring definitions
## for Kubernetes services and deployment and management of Prometheus instances.

## But how are they created
## After installation new resource-type are introduced
cat all-prometheus.yaml | grep ^kind: | grep -e 'Prometheus' -e 'ServiceM' | uniq
kind: Prometheus
kind: PrometheusRule
kind: ServiceMonitor
```

Step 3c: How are the StatefulSets created

```
## New custom resource definitions are created
## The Prometheus custom resource definition (CRD) declaratively defines a desired Prometheus
setup to run in a Kubernetes cluster. It provides options to # configure replication,
persistent storage, and Alertmanagers to which the deployed Prometheus instances send alerts
to.

## For each Prometheus resource, the Operator deploys a properly configured StatefulSet in the
same namespace. The Prometheus Pods are configured to mount # ca Secret called <prometheus-
name> containing the configuration for Prometheus.
## https://github.com/prometheus-community/helm-charts/blob/main/charts/kube-prometheus-
stack/crds/crd-prometheuses.yaml
```

Step 3d: How are PrometheusRules created

```
## PrometheusRule are manipulated by the MutationHook when they enter the AdmissionController
## The AdmissionController is used after proper authentication in the kube-api-server

cat all-prometheus.yml | grep 'Mutating' -B1 -A32
```

```
## Output
## Ref: https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-
controllers/
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: prometheus-kube-prometheus-admission
  labels:
    app: kube-prometheus-stack-admission
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/instance: prometheus
    app.kubernetes.io/version: "35.4.2"
```

```

app.kubernetes.io/part-of: kube-prometheus-stack
chart: kube-prometheus-stack-35.4.2
release: "prometheus"
heritage: "Helm"
webhooks:
- name: prometheusrulemutate.monitoring.coreos.com
  failurePolicy: Ignore
  rules:
    - apiGroups:
        - monitoring.coreos.com
      apiVersions:
        - "*"
      resources:
        - prometheusrules
      operations:
        - CREATE
        - UPDATE
  clientConfig:
    service:
      namespace: prometheus
      name: prometheus-kube-prometheus-operator
      path: /admission-prometheusrules/mutate
    admissionReviewVersions: ["v1", "v1beta1"]
    sideEffects: None

```

Step 4: Let's look into Deployments

```
kubectl -n prometheus get deploy
```

- What do they do

Step 5: Let's look into DaemonSets

```

kubectl -n prometheus get ds
## node-exporter runs on every node
## connects to server, collects data and exports it
## so it is available for prometheus at the endpoint

```

Helm -> prometheus stack -> What does it do

- Sets up Monitoring Stack
- Configuration for your K8s cluster
 - Worker Nodes monitored
 - K8s components (pods a.s.o) are monitored

Where does configuration come from ?

```

## roundabout 31 configmaps
kubectl -n prometheus get configmaps

## also you have secrets (Grafana, Prometheus, Operator)
kubectl -n prometheus get secrets

```

CRD's were created

```

## custom resource definitions
kubectl -n prometheus crd

```

```
## Sehr lang !
kubectl -n prometheus get crd/prometheuses.monitoring.coreos.com -o yaml
```

Look into the pods to see the image used, how configuration is mounted

```
kubectl -n prometheus get sts
kubectl -n prometheus describe sts/prometheus-prometheus-kube-prometheus-prometheus > prom.yml
kubectl -n prometheus describe sts/alertmanager-prometheus-kube-prometheus-alertmanager > alert.yml

kubectl -n prometheus get deploy
kubectl -n prometheus describe deploy/prometheus-kube-prometheus-operator > operator.yml

## --> das SECRET erstellt der Kubernetes Operator für uns !
## First prom.yml
##. Mounts:
##      /etc/prometheus/config from config (rw)
## -> What endpoints to scrape
## comes from:
kubectl get -n prometheus secrets prometheus-prometheus-kube-prometheus-prometheus -o jsonpath='{.data.prometheus\.yaml\.gz}' | base64 -d | gunzip > config-prom.yml
## vi config-prom.yml
## Look into the scrape_configs
```

Connect to grafana

```
## wie ist der port 3000
kubectl logs prometheus-grafana-776fb976f7-w9nnp grafana
## hier nochmal port und auch, wie das secret heisst
kubectl describe pods prometheus-grafana-776fb976f7-w9nnp | less

## user / pass ?
kubectl get secrets prometheus-grafana -o jsonpath='{.data.admin-password}' | base64 -d
kubectl get secrets prometheus-grafana -o jsonpath='{.data.admin-user}' | base64 -d

## localhost:3000 erreichbarkeit starten -- im Vordergrund
kubectl port-forward deploy/prometheus-grafana 3000

## letzte Schritt: browser aufrufen: http://localhost:3000
```

Reference:

- Techworld with Nana: <https://www.youtube.com/watch?v=QoDqxm7yblc>

Tipps & Tricks

Default namespace von kubectl ändern

How ?

```
kubectl config set-context --current --namespace=<insert-namespace-name-here>
## Validate it
kubectl config view --minify | grep namespace:
```

Reference:

- <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

Ingress Controller auf DigitalOcean aufsetzen

Basics

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

Prerequisites

- kubectl muss eingerichtet sein

Walkthrough (Setup Ingress Controller)

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm show values ingress-nginx/ingress-nginx

## It will be setup with type loadbalancer - so waiting to retrieve an ip from the external
loadbalancer
## This will take a little.
helm install nginx-ingress ingress-nginx/ingress-nginx --set
controller.publishService.enabled=true

## See when the external ip comes available
kubectl --namespace default get services -o wide -w nginx-ingress-ingress-nginx-controller

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

## Now setup wildcard - domain for training purpose
*.lab1.t3isp.de A 157.245.20.222
```

vi einrückungen für yaml

Ubuntu (im Unterverzeichnis /etc/vim - systemweit)

```
hi CursorColumn cterm=NONE ctermbg=lightred ctermfg=white
autocmd FileType y?ml setlocal ts=2 sts=2 sw=2 ai number expandtab cursorline cursorcolumn
```

Testen

```
vim test.yml
Eigenschaft: <return> # springt eingerückt in die nächste Zeile um 2 spaces eingerückt

## evtl funktioniert vi test.yml auf manchen Systemen nicht, weil kein vim (vi improved)
```

gitlab runner as nonroot

- <https://docs.gitlab.com/runner/install/kubernetes.html#running-with-non-root-user>

curl zum Überprüfen mit Pod

Situation

- Kein Zugriff auf die Nodes, zum Testen von Verbindungen zu Pods und Services über die ClusterIP

Lösung

```
## Achtung http:// muss angegeben werden, sonst funktioniert das Kommando möglicherweise nicht
## -L sollte man immer verwenden, leitet um
## --output - gibt es auf stdout (Bildschirm aus)
kubectl run -it --rm --image=curlimages/curl curly -- curl -L --output - http://www.test.de
```

RootLess / Security

seccomp-profile-default docker

- <https://github.com/docker/docker-ce/blob/master/components/engine/profiles/seccomp/default.json>

Pod Security Policy

Welches Objekt ?

- `kubectl api-resources | grep -i podsecuritypolicy`
- short: psp

Namespacedfähig ?

- Nein

Aktivieren (das reicht nicht)

- Der AdmissionController=podSecurityPolicy muss aktiviert sein, dies ist z.B. bei DOKS (Digital Ocean Kubernetes nicht der Fall)
- Wenn er nicht aktiviert ist, greift das angelegte Objekt nicht
- Aktivierung in microk8s

```
## find / -name "kube-apiserver"
## ${SNAP_DATA}/args/kube-apiserver
## --enable-admission-plugins="PodSecurityPolicy"
microk8s stop
microk8s start

## Ref:
## https://microk8s.io/docs/configuring-services
```

Aktivieren (so geht's)

```
Hintergründe:
https://kubernetes.io/docs/concepts/security/pod-security-policy/#troubleshooting
```

Important

- podSecurityPolicy works ClusterWide, so we need to authorize some users
- who are able to edit the policies

```
## There is really a clusterrole that can be called edit and can edit
## So we need some one, who is allowed to do so.
kubectl get clusterrole | grep ^edit
##edit
```

2022-05-08T06:51:30Z

```

## https://kubernetes.io/docs/concepts/policy/pod-security-policy/
---
apiVersion: v1
kind: Namespace
metadata:
  name: pod-security-policy-psp-namespace
---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: pod-security-policy-psp
spec:
  privileged: false # Don't allow privileged pods!
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
    - '*'
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pod-security-policy-user
  namespace: pod-security-policy-psp-namespace
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-security-policy-psp-user-editor
  namespace: pod-security-policy-psp-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
  - kind: ServiceAccount
    name: pod-security-policy-psp-namespace
    namespace: pod-security-policy-psp-namespace
---
apiVersion: v1
kind: Pod
metadata:
  name: pause
  namespace: pod-security-policy-psp-namespace-unprivileged
spec:
  containers:
    - name: pause
      image: k8s.gcr.io/pause
---
apiVersion: v1
kind: Pod
metadata:

```

```

name: pause
namespace: pod-security-policy-psp-namespace-privileged
spec:
  containers:
    - name: pause
      image: k8s.gcr.io/pause
      securityContext:
        privileged: true

```

Ref:

- <https://k8s-examples.container-solutions.com/examples/PodSecurityPolicy/PodSecurityPolicy.html>
- <https://github.com/intelygenz/lab-microk8s-pod-security-policies/blob/master/6.Enable-Pod-Security-Policy.md#test-it>

Pod Security Policy - Übung

Vorbereitung

Die Umgebung muss aufgesetzt, dass PodSecurityPolicy als Admission Controller läuft.
(dazu unter microk8s -> /var/snap/microk8s/currents/args/kubeapi-server die Zeile:
--admission-controller=PodSecurityPolicy hinzugefügt.

In der folgenden Umgebung, ist dies bereits durchgeführt
Es gibt die Möglichkeit dieses Script zu verwenden um in DigitalOcean eine Umgebung aufzusetzen.
Dies kann aber letztendlich auf jedem beliebigen Ubuntu 20.04. LTS Systems erfolgen

```

#!/bin/bash

groupadd sshadmin
USERS="11trainingdo"
for USER in $USERS
do
    echo "Adding user $USER"
    useradd -s /bin/bash $USER
    usermod -aG sshadmin $USER
    echo "$USER:mein-super-geheimes-passwort" | chpasswd
done

## We can sudo with 11trainingdo
usermod -aG sudo 11trainingdo

## Setup ssh stuff
sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
usermod -aG sshadmin root
echo "AllowGroups sshadmin" >> /etc/ssh/sshd_config
systemctl reload sshd

## Now let us do some generic setup
echo "Installing microk8s"

## This only works with 1.23 out of the box
## in 1.24 no secrets for service - accounts are created by default
snap install --classic --channel=1.23/stable microk8s

microk8s enable dns rbac

```



```

echo "alias kubect1='microk8s kubect1'" >> /root/.bashrc
source ~/.bashrc
alias kubect1='microk8s kubect1'

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

git clone https://github.com/jmetzger/lab-microk8s-pod-security-policies lab

## set the correct rolebinding for the service-account
cd /lab

## Create namespace for testing
microk8s kubect1 create namespace sample-psp
microk8s kubect1 apply -f rbac/cluster-role-binding-default-sa-at-kube-system-as-cluster-admin.yml
## Create a developer role in that namespace
microk8s kubect1 apply -f rbac/default-sa-at-example-psp-namespace.yml
## helper - sample psp
helper/create_sa_kubeconfig.sh default sample-psp

## now we need to modify the setting of kube-api-server
## currently in 1.23 no other admission-plugins are activated
echo "--enable-admission-plugins=PodSecurityPolicy" >> /var/snap/microk8s/current/args/kube-apiserver
microk8s stop
microk8s start

```

Übung 1 (nginx erstellen, läuft dieser ?)

```

## mit dem Server per ssh verbinden
## zu Testzwecken verzichten wir hier auf einen extra Client
## sudo -i
cd
mkdir manifests
## vi 01-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.15.4

```

```

kubect1 apply -f 01-nginx.yml
## sind pods gestartet
kubect1 get deploy,rs,pods
## rs - sollte nur einen geben
kubect1 describe rs
## Pod wurde nicht gestartet:
## Error creating: pods "nginx-deployment-8647b96f59-" is forbidden: PodSecurityPolicy: no
providers available to validate pod request
## PodSecurityPolicy ist aktiv, yeah. !! aber es gibt für den user der hier arbeitet keine
podsecurity policy die greift

## Welcher user arbeitet hier - standardmäßig default
kubect1 config current-context

## context: microk8s, user: admin, authentifizierung läuft über token
kubect1 config view

## Alles auf Anfang
kubect1 delete -f 01-pod.yml

```

Übung 2a: Set psp (restrictive), clusterrole, clusterrolebinding and try to deploy nginx

```

## Schritt 1:
## psp erstellen
## mkdir psp-restrictive
## cd psp-restrictive
## vi 01-ppsp.yml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false
  hostNetwork: false
  allowPrivilegeEscalation: false
  defaultAllowPrivilegeEscalation: false
  hostPID: false
  hostIPC: false
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
    - 'configMap'
    - 'downwardAPI'
    - 'emptyDir'
    - 'persistentVolumeClaim'
    - 'secret'
    - 'projected'
  allowedCapabilities:
    - '*'

```

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

```
## psp does not work without
## 1. a role/clusterrole
## 2. a rolebind / clusterrolebinding
## Let's start with 1.
## vi 02-clusterrole.yml
## Decides which policy to use
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-restrictive
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  resourceNames:
  - restrictive
  verbs:
  - use
```

```
kubectl apply -f 02-clusterrole.yml
```

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

```
kubectl apply -f 04-nginx-deploy.yml
## IT works
kubectl get deploy,rs,pods
```

Step 2b: Deploy nginx and WANTING access to host network

- Need Step 2a to be done firstly

```
## vi 05-nginx-host.yml
## With access to host
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-host
  labels:
    app: nginx-host
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-host
  template:
    metadata:
      labels:
        app: nginx-host
    spec:
      containers:
      - name: nginx
```

```
image: nginx:1.15.4
hostNetwork: true
```

```
## to easier see nginx-deployment-host
kubectl delete -f 04-nginx.host.yml
kubectl apply -f 05-nginx-host.yml
## Does not work, why ?
kubectl get deployment,rs.pods
```

```
## HostNetwork is in the way
kubectl describe rs
## See also - Got the point
kubectl get psp restrictive -o yaml
kubectl describe psp restrictive
kubectl describe psp restrictive | grep "Host Network"
```

```
## Cleanup
cd
cd manifests/psp-restrictive
kubectl delete -f .
```

Übung 3: psp (restrictive), clusterrole, rolebinding

```
## Schritt 1: Create permissive psp
cd
cd manifests
mkdir psp-permissive
cd psp-permissive
```

```
## vi 01-psp.yml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: permissive
spec:
  privileged: true
  hostNetwork: true
  hostIPC: true
  hostPID: true
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  hostPorts:
    - min: 0
      max: 65535
  volumes:
    - '*'
```

```
## apply that
kubectl apply -f 01-psp.yml
```

```
kubectl get psp
```

```
## Schritt 2: Create a clusterrole
## create a clusterrole
## vi 02-clusterrole.yml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-permissive
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  resourceNames:
  - permissive
  verbs:
  - use
```

```
kubectl apply -f 02-clusterrole.yml
kubectl get clusterrole
```

```
## Schritt 3: clusterrolebinding anlegen
## create clusterrolebinding but only for use default in kube-system
## vi 03-clusterrolebinding-kube-system.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-sa-at-kube-system-as-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
```

```
kubectl apply -f 03-clusterrolebinding-kube-system.yml
```

```
## Schritt 4: Testing Host Network - Anforderung
## same as in last exercise
## vi 05-nginx-host.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-host
  labels:
    app: nginx-host
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-host
  template:
    metadata:
```

```
  labels:
    app: nginx-host
  spec:
    containers:
    - name: nginx
      image: nginx:1.15.4
      hostNetwork: true
```

```
## Apply but not to kube-system namespace
kubectl -n kube-system apply -f 05-nginx-host.yml
## This work, and we want to figure out, if hostname isset.
kubectl -n kube-system get pods | grep nginx

## See if hostNetwork isset
kubectl -n kube-system -l app=nginx-host get pods -o yaml | grep -i hostNetwork

## Try to look inside
kubectl -n kube-system exec -it deploy/nginx-deployment-host -- bash
## apt update
## apt install -y iproute2 procps
## ip a
## ps aux
## exit

kubectl -n kube-system delete -f 05-nginx-host.yml
kubectl delete -f .
```

RunAsUser Exercise

Hinweis:

Der USER muss auf dem System nicht existieren.
Die Einstellung

```
securityContext:
  runAsuser: 12000
```

überschreibt die Einstellung unter welchem User der Docker - Container läuft.
Directive: USER

Allerdings kommt es zu Problemen, wenn der Docker die Software (ENTRYPOINT) und nachfolgende Software nicht starten kann und der Container stoppt dann

Example 1: (normal mit root)

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 01-privileged.yml
apiVersion: v1
kind: Pod
metadata:
  name: uhsil
spec:
  containers:
  - name: bb
    image: ubuntu
```

```
command: ["/bin/bash"]
tty: true
stdin: true
```

```
## Schritt 2:
## Ausführen
kubectl apply -f 01-privileged.yml
kubectl exec -it ubsi1 -- bash
## id
```

Example 2: (als nobody: 65534)

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 02-nobody-privileged.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi2
spec:
  securityContext:
    runAsUser: 65534
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true
```

```
## Schritt 2:
## Ausführen
kubectl apply -f 01-nobody-privileged.yml
kubectl exec -it ubsi2 -- bash
## id
## touch testfile
## ls -la
```

Example 3: (als 1001 - nutzer existiert nicht)

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 03-user-1001.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi3
spec:
  securityContext:
    runAsUser: 1001
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
```

```
tty: true
stdin: true
```

```
## Schritt 2:
## Ausführen
kubectl apply -f 03-user-1001.yml
kubectl exec -it ubsi3 -- bash
## id
## touch testfile
## ls -la
```

Example 4: inkl. Gruppe

```
## Schritt 1:
## mkdir runtest
## cd runtest
## vi 04-user-group-1001.yml
apiVersion: v1
kind: Pod
metadata:
  name: ubsi4
spec:
  securityContext:
    runAsUser: 1001
    runAsGroup: 1001
  containers:
  - name: bb
    image: ubuntu
    command: ["/bin/bash"]
    tty: true
    stdin: true
```

Offizielles RootLess Docker Image für Nginx

- <https://github.com/nginxinc/docker-nginx-unprivileged>

Rechte

RBAC / Rechte allgemein

Bereich 1: Welche Objekte darf ich als Helm/kubectl - Nutzer erstellen/bearbeiten/ändern

```
kubectl
-> user:
  -> token: identifiziert.

users:
- name: admin
  user:
    token: Q2tJbEsxaUI0eFVDT3haYXJlVGVxYWVsWURHRFlhZ25QWVpNd3lVdi9BST0K

--> über ein Token ->
Hier kann auch ein anderer Context hinzugefügt, der dann als Nutzer verwenden kann
z.B. context restricteduser
kubectl config use-context restricteduser
```



```
Benutzer (User/ServiceAccount) <----->. Rolebinding/Clusterrolebinding <----->
Role/Clusterrole
```

```
## Standard: default -> bei serviceAccount der automatisch eingebunden wird,
alternativ
```

```
kubectl explain deployment.spec.template.spec.serviceAccountName
## <- dann wird dieser ServiceAccountName verwendet.
```

Schritt 1: Authentifizierung -> Du darfst generell erstmal zugreifen...

Schritt 2: Was darfst du ?
anhand von -> rolebinding/clusterrolebinding -> role/clusterrole

Bereich 2: Wie darf einer Nutzer xy einen Pod starten / Vorgabe !!!

```
## Ebene 1: Ein User / Service Account :      hans / nginx-ingress

## Ebene 2: Ein rolebinding / clusterrolebinding
hans -> rolebinding rechte_verkuepfung_hans -> role

## Ebene 3: Rolle (z.B. nicht_admin_rolle)
## In der Rolle steht drin, welche PodSecurityPolicy für ihn gilt

## Ebene 4:
## Definierte PodSecurityPolicy
```

Bereich 3: Welcher Dienst/Controller/Pod -> darf was machen / abfragen

```
## Beispiel, was darf ein Pod -> z.B. nginx in Bezug auf Anfragen an den kube-api-server
## Wenn er bereits im laufenden Betrieb ist.

## Puzzle - Teil 1: Mit welchem Token / Account greift er zu
kubectl run nginx --image=nginx

## Er hängt automatisch den ServiceAccount und das ca-cert und auch den namespace
kubectl describe po nginx
kubectl get po nginx -o yaml

kubectl exec -it nginx -- bash
## cd /var/run/secrets/kubernetes.io/serviceaccount
## ls -la
## cat namespace
## cat token

## So können wir auf den kube-api-server low level zugreifen
## Es ist ein jwt-token, mit Zusatzinformationen wie ServiceAccount
## Lässt sich mit jwt oder auf jwt.io auslesen
TOKEN=$(cat token)
## Esi ist immer diese Domain zum Kube-API-Server
API_URL=https://kubernetes.default.svc.cluster.local

## Api -> v1 abfragen
## curl --cacert ca.crt -H "Authorization: Bearer $TOKEN" $API_URL/api
## Alle anderen apis abfragen (welche gibt es) -> z.B. apps/v1
curl --cacert ca.crt -H "Authorization: Bearer $TOKEN" $API_URL/apis
```

```
## Ist mein Token falsch, bekomme ich forbidden 403
curl --cacert ca.crt -H "Authorization: Bearer $TOKENx" $API_URL/apis
```

```
## Ist mein Token korrekt, aber Pfad falsch auch -> 403 -> apis3 gibt es nicht auf dem Server
curl --cacert ca.crt -H "Authorization: Bearer $TOKEN" $API_URL/apis3
```

```
Ergebnis des Tokens in jwt.io
eyJhbGciOiJSUzI1NiIsImtpZCI6Ikkx1SF1BU1RWNHJ1SlV2T1JxdTdkaElzd01yOGJzTTVZUmpmM3E2VUJiNm8ifQ.eyJhd
y0Z_pEBWqtKhBGDEFZwbdIlWRbfBN0xHrKqmgib9QvBqosqIM4G2Ozm9Dv-
4bEZYjqybq8ylKCMRWVA62LjooS5k7gl1E3ws7qY5G2Jb28MPo-x7Gvgx4MBGCACWsPgfaKLF0kwcbGxHC6VWG7Bgj21di-
_aHeuuBslhkGHeSLHyuWOXwOPi7ne59blrAUKblzmEwNbWqRtGKBjqzelkbAq80GD7-
khK3LxbOaB9XVN6LzvVXeQjOXGSVUr3gkE4SNM-R1zY01raXdD6xJ9cHBbRg_kj3FwpD_dxWDY1G-
VU_5Z16v8SAI1lLvrg
```

```
## payload data
{
  "aud": [
    "https://kubernetes.default.svc"
  ],
  "exp": 1685790779,
  "iat": 1654254779,
  "iss": "https://kubernetes.default.svc",
  "kubernetes.io": {
    "namespace": "default",
    "pod": {
      "name": "nginx",
      "uid": "c1cd9a92-ed8b-4a75-91ac-897ee84f5e7b"
    },
    "serviceaccount": {
      "name": "default",
      "uid": "ff642a63-40b8-4d3d-9d2a-c0620687d7d4"
    },
    "warnafter": 1654258386
  },
  "nbf": 1654254779,
  "sub": "system:serviceaccount:default:default"
}
```

Welche Berechtigungen ?

```
1. Ebene
Welche apiGroups ?
z.B. apps/v1 oder alles *
[''] -> v1
```

```
2. Ebene
Welche Ressourcen -> welches Kind
deployment
* <- für alle
```

```
3. Ebene -> verbs a.k.a (Operationen)
list (kubectl get pods) - Liste -- > items
get (kubectl get pod live-pod) -
create
delete
```

```
watch
update
```

Documentation

helm dry-run vs. template

- <https://jhoog.com/helm-dry-run-install/>

Marktuebersicht Kubernetes Hosting

Google Cloud

- Aktuell (2022/06) -> USD 300
- <https://www.udemy.com/course/certified-kubernetes-security-specialist/learn/lecture/22792893?start=30#learning-tools>

AWS

AZURE

Digital Ocean

- Kubernetes Cluster (Managed) ab \$10 / Monat möglich
- <https://www.digitalocean.com/products/kubernetes>
- Empfehlenswert ab USD 20, aber nur 1 Node, d.h. realistisch mit 3 Nodes, USD 60
- Einfach aufzusetzen, vanilla Kubernetes
- Empfehlenswert
- Rechenzentrum Deutschland möglich.
- Andere Maschinen (Droplet = virtuelle Maschinen) sind zubuchbar.

IONOS

- <https://cloud.ionos.de/preise>
- Preisstruktur schwer durchschaubar, weil es sich aus verschiedenen Komponenten zusammensetzt (CPU .. etc)

OVHCloud

- monatliche Preise anzeigbar, ab Euro 26,- pro Instanz (Stand 05/2022)
- <https://www.ovhcloud.com/de/public-cloud/prices/>
- <https://www.ovhcloud.com/de/public-cloud/kubernetes/#::~:~:text=Der%20Managed%20Kubernetes%20Service%20wird%20kostenfrei%20bereitgestellt.>
- Unklar, ob das gemanaged ist, oder meine ein Cluster aufsetzt, deren zugrundeliegende Maschinen man selber warten muss

Cloudshift

- ab Euro 169,90 / Monat (Stand: 05/2022)
- <https://www.cloudshift.de/cloud-services/kubernetes/#pricing>

minikube (als nicht hosting - alternative)

```
Als Alternative
```

rancherdesktop

```
1 - node - cluster
```

lab - umgebung

- <https://labs.play-with-k8s.com/#>

killercoda

- <https://killercoda.com/playgrounds/scenario/kubernetes>