Kubernetes Networking

Agenda

1. Kubernetes - Überblick
    - Aufbau Allgemein
    - Ports und Protokolle
2. Kubernetes - Misc
    - Wann wird podIP vergeben ?
    - Bash completion installieren
    - Remote-Verbindung zu Kubernetes (microk8s) einrichten
    - vim support for yaml
3. Kubernetes - Netzwerk (CNI's) / Mesh
    - Netzwerk Interna
    - Übersicht Netzwerke
    - Calico/Cilium - nginx example NetworkPolicy
    - Beispiele Ingress Egress NetworkPolicy
    - Mesh / istio
    - Kubernetes Ports/Protokolle
    - IPV4/IPV6 Dualstack
    - Ingress controller in microk8s aktivieren
    - DNS - Resolution - Services
4. Kubernetes - Ingress
    - ingress mit ssl absichern
5. Kubernetes - Wartung / Debugging
    - kubectl drain/uncordon
    - Alte manifeste konvertieren mit convert plugin
    - Netzwerkverbindung zu pod testen
    - Curl from pod api-server
6. Kubernetes Praxis API-Objekte
    - Das Tool kubectl (Devs/Ops) - Spickzettel
    - kubectl example with run
    - Bauen einer Applikation mit Resource Objekten
    - kubectl/manifest/deployments
    - Services - Aufbau
    - kubectl/manifest/service
    - DaemonSets (Devs/Ops)
    - Hintergrund Ingress
    - Ingress Controller auf Digitalocean (doks) mit helm installieren
    - Documentation for default ingress nginx
    - Beispiel Ingress
    - Install Ingress On Digitalocean DOKS
    - Beispiel mit Hostnamen
    - Achtung: Ingress mit Helm - annotations
    - Permanente Weiterleitung mit Ingress
    - ConfigMap Example
    - Configmap MariaDB - Example
    - Configmap MariaDB my.cnf
7. Kubernetes Deployment Scenarios
    - Deployment green/blue,canary,rolling update
    - Service Blue/Green
    - Praxis-Übung A/B Deployment
8. Helm (Kubernetes Paketmanager)
    - Helm Grundlagen
    - Helm Warum ?
    - Helm Example
9. Kubernetes - RBAC
    - Nutzer einrichten microk8s ab kubernetes 1.25
    - Tipps&Tricks zu Deploymnent - Rollout
10. Kubernetes QoS
    - Quality of Service - evict pods
11. Kustomize
    - Kustomize Overlay Beispiel
    - Helm mit kustomize verheiraten

12. Kubernetes - Tipps & Tricks
    - Kubernetes Debuggen ClusterIP/PodIP
    - Debugging pods
    - Taints und Tolerations
    - Autoscaling Pods/Deployments
    - pod aus deployment bei config - Änderung neu ausrollen
13. Kubernetes Advanced
    - Curl api-server kubernetes aus pod heraus
14. Kubernetes - Documentation
    - Documentation zu microk8s plugins/addons
    - Shared Volumes - Welche gibt es ?
15. Kubernetes - Hardening
    - Kubernetes Tipps Hardening
    - Kubernetes Security Admission Controller Example
    - Was muss ich bei der Netzwerk-Sicherheit beachten ?
16. Kubernetes Interna / Misc.
    - OCI,Container,Images Standards
    - Geolocation Kubernetes Cluster

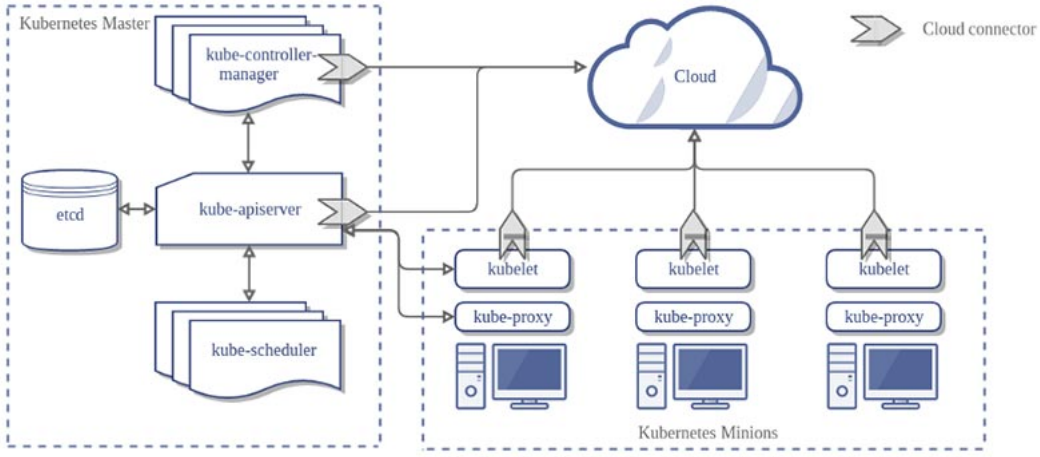
Backlog

1. Kubernetes - Überblick
    - Installation - Welche Komponenten from scratch
2. Kubernetes - microk8s (Installation und Management)
    - kubectl unter windows - Remote-Verbindung zu Kuberenets (microk8s) einrichten
    - Arbeiten mit der Registry
    - Installation Kubernetes Dashboard
3. Kubernetes - RBAC
    - Nutzer einrichten - kubernetes bis 1.24
4. kubectl
    - Tipps&Tricks zu Deploymnent - Rollout
5. Kubernetes - Monitoring (microk8s und vanilla)
    - metrics-server aktivieren (microk8s und vanilla)
6. Kubernetes - Backups
    - Kubernetes Aware Cloud Backup - kasten.io
7. Kubernetes - Tipps & Tricks
    - Assigning Pods to Nodes
8. Kubernetes - Documentation
    - LDAP-Anbindung
    - Helpful to learn - Kubernetes
    - Environment to learn
    - Environment to learn II
    - Youtube Channel
9. Kubernetes - Shared Volumes
    - Shared Volumes with nfs
10. Kubernetes - Hardening
    - Kubernetes Tipps Hardening
11. Kubernetes Probes (Liveness and Readiness)
    - Übung Liveness-Probe
    - Funktionsweise Readiness-Probe vs. Liveness-Probe

Kubernetes - Überblick

Aufbau Allgemein

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

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

Komponenten des Masters

ETCD

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

KUBE-CONTROLLER-MANAGER

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

KUBE-API-SERVER

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

KUBE-SCHEDULER

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

Nodes

- 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

Ports und Protokolle

- https://kubernetes.io/docs/reference/networking/ports-and-protocols/

Kubernetes - Misc

Wann wird podIP vergeben ?

Example (that does work)

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

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

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

Bash completion installieren

Walkthrough

```
## Eventuell, wenn bash-completion nicht installiert ist.
apt install bash-completion
source /usr/share/bash-completion/bash_completion
## is it installed properly
type _init_completion
```

```
## activate for all users
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

## verifizieren - neue login shell
su -

## zum Testen
kubectl g<TAB>
kubectl get
```

Alternative für k als alias für kubectl

```
source <(kubectl completion bash)
complete -F __start_kubectl k
```

Reference

- https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/

Remote-Verbindung zu Kubernetes (microk8s) einrichten

```
## on CLIENT install kubectl
sudo snap install kubectl --classic

## On MASTER -server get config
## als root
cd
microk8s config > /home/kurs/remote_config

## Download (scp config file) and store in .kube - folder
cd ~
mkdir .kube
cd .kube  # Wichtig: config muss nachher im verzeichnis .kube liegen
## scp kurs@master_server:/path/to/remote_config config
## z.B.
scp kurs@192.168.56.102:/home/kurs/remote_config config
## oder benutzer 11trainingdo
scp 11trainingdo@192.168.56.102:/home/11trainingdo/remote_config config

##### Evtl. IP-Adresse in config zum Server aendern

## Ultimative 1. Test auf CLIENT
kubectl cluster-info

## or if using kubectl or alias
kubectl get pods

## if you want to use a different kube config file, you can do like so
kubectl --kubeconfig /home/myuser/.kube/myconfig
```

vim support for yaml

Ubuntu (im Unterverzeichnis /etc/vim/vimrc.local - 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

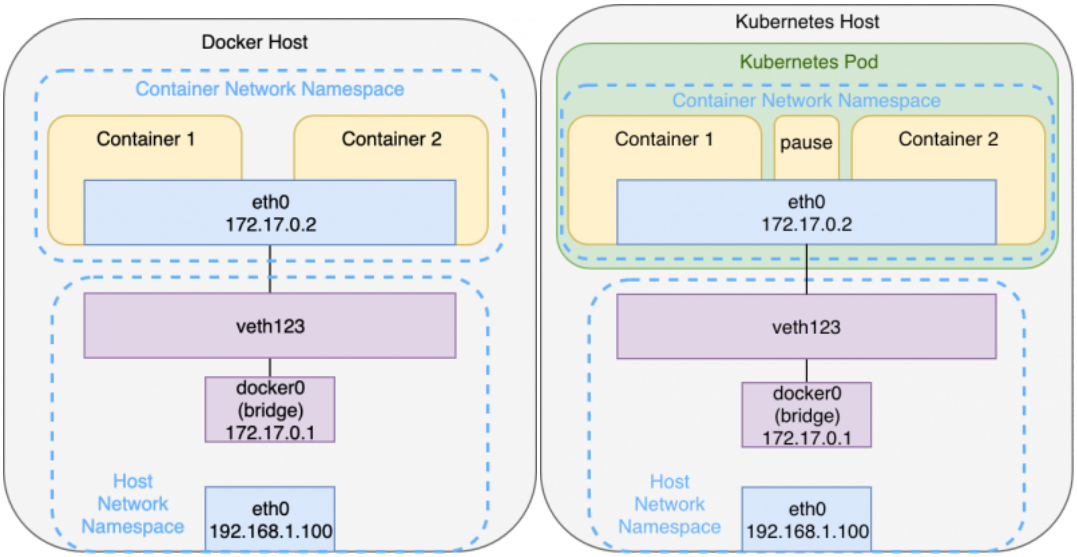
```
vim test.yml
Eigenschaft: <return> # springt eingerückt in die nächste Zeile um 2 spaces eingerückt

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

Kubernetes - Netzwerk (CNI's) / Mesh

Netzwerk Interna

Network Namespace for each pod

Overview



General

- Each pod will have its own network namespace
  - with routing, networkdevices
- Connection to default namespace to host is done through veth - Link to bridge on host network
  - similar like on docker to docker0

```
Each container is connected to the bridge via a veth-pair. This interface pair functions like a virtual point-to-point ethernet connection and connects the network namespaces of the containers with the network namespace of the host
```

- Every container is in the same Network Namespace, so they can communicate through localhost
  - Example with hashicorp/http-echo container 1 and busybox container 2 ?

Pod-To-Pod Communication (across nodes)

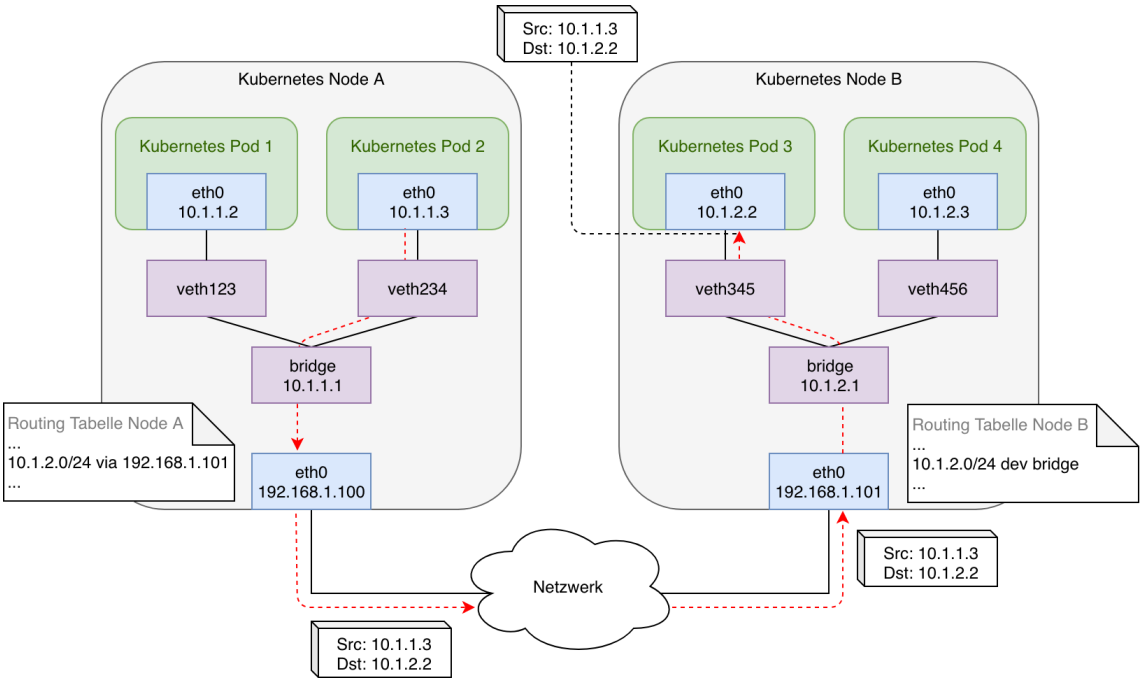
Prerequisites

- pods on a single node as well as pods on a topological remote can establish communication at all times
- Each pod receives a unique IP address, valid anywhere in the cluster. Kubernetes requires this address to not be subject to network address translation (NAT)
- Pods on the same node through virtual bridge (see image above)

General (what needs to be done) - and could be doen manually

- local bridge networks of all nodes need to be connected
- there needs to be an IPAM (IP-Address Managemenet) so addresses are only used once
- The need to be routes so, that each bridge can communicate with the bridge on the other network
- Plus: There needs to be a rule for incoming network
- Also: A tunnel needs to be set up to the outside world.

General - Pod-to-Pod Communiation (across nodes) - what would need to be done



General - Pod-to-Pod Communication (side-note)

- This could of cause be done manually, but it is too complex
- So Kubernetes has created an Interface, which is well defined
  - The interface is called CNI (common network interface)
  - Funtionally is achieved through Network Plugin (which use this interface)
    - e.g. calico / cilium / weave net / flannel

CNI

- CNI only handles network connectivity of container and the cleanup of allocated resources (i.e. IP addresses) after containers have been deleted (garbage collection) and therefore is lightweight and quite easy to implement.
- There are some basic libraries within CNI which do some basic stuff.

Hidden Pause Container

What is for ?

- Holds the network - namespace for the pod
- Gets started first and falls asleep later

- Will still be there, when the other containers die

```
cd
mkdir -p manifests
cd manifests
mkdir pausetest
cd pausetest
nano 01-nginx.yml
```

```
## vi nginx-static.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pausetest
  labels:
    webserver: nginx:1.21
spec:
  containers:
  - name: web
    image: nginx
```

```
kubectl apply -f .

ctr -n k8s.io c list | grep pause
```

References

- https://www.inovex.de/de/blog/kubernetes-networking-part-1-en/
- https://www.inovex.de/de/blog/kubernetes-networking-2-calico-cilium-weavenet/

Übersicht Netzwerke

CNI

- Common Network Interface
- Feste Definition, wie Container mit Netzwerk-Bibliotheken kommunizieren

Docker - Container oder andere

- Container wird hochgefahren -> über CNI -> zieht Netzwerk - IP hoch.
- Container witd runtergahren -> uber CNI -> Netzwerk - IP wird released

Welche gibt es ?

- Flanel
- Canal
- Calico
- Cilium
- Weave Net

Flannel

Overlay - Netzwerk

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

Vorteile

- Guter einfacher Einstieg
- redziert auf eine Binary flanneld

Nachteile

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

Canal

General

- Auch ein Overlay - Netzwerk
- Unterstüzt auch policies

Calico

Generell

- klassische Netzwerk (BGP)

Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

Vorteile

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

Referenz

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

Cilium

Weave Net

- Ähnlich calico
- Verwendet overlay netzwerk
- Sehr stabil bzgl IPV4/IPV6 (Dual Stack)
- Sehr grosses Feature-Set
- mit das älteste Plugin

microk8s Vergleich

- https://microk8s.io/compare

```
snap.microk8s.daemon-flanneld
Flannel is a CNI which gives a subnet to each host for use with container runtimes.

Flanneld runs if ha-cluster is not enabled. If ha-cluster is enabled, calico is run instead.

The flannel daemon is started using the arguments in ${SNAP_DATA}/args/flanneld. For more information on the configuration, see
the flannel documentation.
```

Calico/Cilium - nginx example NetworkPolicy

```
## Schritt 1:
kubectl create ns policy-demo
kubectl create deployment --namespace=policy-demo nginx --image=nginx:1.21
kubectl expose --namespace=policy-demo deployment nginx --port=80
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

```
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: policy-demo
spec:
  podSelector:
    matchLabels: {}
EOF
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

```
## Schritt 3: Zugriff erlauben von pods mit dem Label run=access
kubectl create -f - <<EOF
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-nginx
  namespace: policy-demo
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
    - from:
      - podSelector:
          matchLabels:
            run: access
EOF

## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
## pod hat durch run -> access automatisch das label run:access zugewiesen
kubectl run --namespace=policy-demo access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

```
kubectl run --namespace=policy-demo no-access --rm -ti --image busybox
```

```
## in der shell
wget -q nginx -O -
```

```
kubectl delete ns policy-demo
```

Ref:

- https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic

Beispiele Ingress Egress NetworkPolicy

Links

- https://github.com/ahmetb/kubernetes-network-policy-recipes
- https://k8s-examples.container-solutions.com/examples/NetworkPolicy/NetworkPolicy.html

Example with http (Cilium !!)

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
description: "L7 policy to restrict access to specific HTTP call"
metadata:
  name: "rule1"
spec:
  endpointSelector:
    matchLabels:
      type: l7-test
  ingress:
  - fromEndpoints:
    - matchLabels:
        org: client-pod
    toPorts:
    - ports:
      - port: "8080"
        protocol: TCP
      rules:
        http:
        - method: "GET"
          path: "/discount"
```

Downside egress

- No valid api for anything other than IP's and/or Ports
- If you want more, you have to use CNI-Plugin specific, e.g.

Example egress with ip's

```
## Allow traffic of all pods having the label role:app
## egress only to a specific ip and port
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 10.10.0.0/16
    ports:
    - protocol: TCP
      port: 5432
```

Example Advanced Egress (cni-plugin specific)

Cilium

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

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: "fqdn-pprof"
  # namespace: msp
spec:
  endpointSelector:
    matchLabels:
      webserver: nginx
  egress:
  - toFQDNs:
    - matchPattern: '*.google.com'
  - toPorts:
    - ports:
      - port: "53"
        protocol: ANY
      rules:
        dns:
        - matchPattern: '*'
```

```
kubectl apply -f .
```

Calico

- Only Calico enterprise
  - Calico Enterprise extends Calico's policy model so that domain names (FQDN / DNS) can be used to allow access from a pod or set of pods (via label selector) to external resources outside of your cluster.
  - https://projectcalico.docs.tigera.io/security/calico-enterprise/egress-access-controls

Using isitio as mesh (e.g. with cilium/calico)

Installation of sidecar in calico

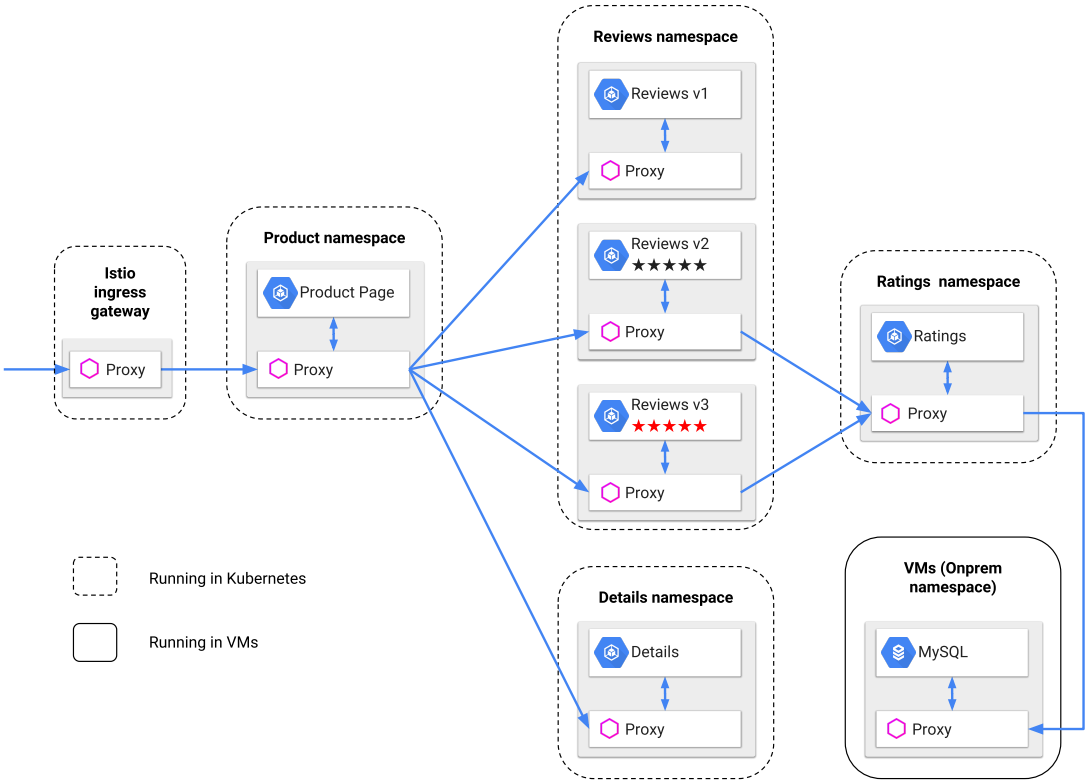
- https://projectcalico.docs.tigera.io/getting-started/kubernetes/hardway/istio-integration

Example

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 10.10.0.0/16
    ports:
    - protocol: TCP
      port: 5432
```

Mesh / istio

Schaubild



Istio

```
## Visualization
## with kiali (included in istio)
https://istio.io/latest/docs/tasks/observability/kiali/kiali-graph.png

## Example
## https://istio.io/latest/docs/examples/bookinfo/
The sidecars are injected in all pods within the namespace by labeling the namespace like so:
kubectl label namespace default istio-injection=enabled

## Gateway (like Ingress in vanilla Kubernetes)
kubectl label namespace default istio-injection=enabled
```

istio tls

- https://istio.io/latest/docs/ops/configuration/traffic-management/tls-configuration/

istio - the next generation without sidecar

- https://istio.io/latest/blog/2022/introducing-ambient-mesh/

Kubernetes Ports/Protokolle

- https://kubernetes.io/docs/reference/networking/ports-and-protocols/

IPV4/IPV6 Dualstack

- https://kubernetes.io/docs/concepts/services-networking/dual-stack/

Ingress controller in microk8s aktivieren

Aktivieren

```
microk8s enable ingress
```

Referenz

- https://microk8s.io/docs/addon-ingress

DNS - Resolution - Services

Kubernetes - Ingress

ingress mit ssl absichern

Kubernetes - Wartung / Debugging

kubectl drain/uncordon

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

## Alle pods von replicasets werden jetzt auf andere nodes verschoben
## Ich kann jetzt wartungsarbeiten durchführen

## Wenn fertig bin:
kubectl uncordon n17

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

Alte manifeste konvertieren mit convert plugin

What is about?

- Plugins needs to be installed seperately on Client (or where you have your manifests)

Walkthrough

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

## Does it work
kubectl convert --help

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

Reference

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

Netzwerkverbindung zu pod testen

Situation

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

Behelf: Eigenen Pod starten mit busybox

```
## laengere Version
kubectl run podtest --rm -ti --image busybox -- /bin/sh
```

```
## kuerzere Version
kubectl run podtest --rm -ti --image busybox
```

Example test connection

```
## wget befehl zum Kopieren
wget -O - http://10.244.0.99
```

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

Curl from pod api-server

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

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops) - Spickzettel

Allgemein

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

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

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

Arbeiten mit manifesten

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

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

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

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

Ausgabeformate

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

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

## get a specific value from the complete json - tree
kubectl get node k8s-nue-jo-ff1p1 -o=jsonpath='{.metadata.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
## 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

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

Referenz

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

kubectl example with run

Example (that does work)

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

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

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

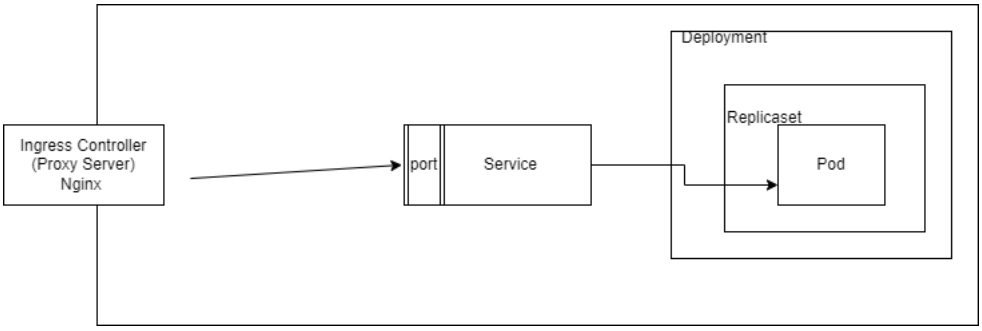
Example (that does not work)

```
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

Bauen einer Applikation mit Resource Objekten



kubectl/manifest/deployments

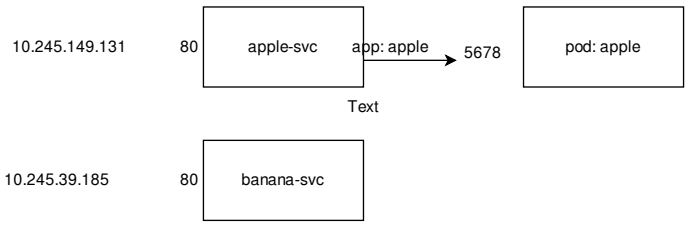
```
cd
mkdir -p manifests
cd manifests
mkdir 03-deploy
cd 03-deploy
nano deploy.yml
```

```

## vi deploy.yml
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 8 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

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

Services - Aufbau



kubectl/manifest/service

Schritt 1: Deployment

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

```
## 01-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 3
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

```
kubectl apply -f .
```

Schritt 2:

```
## 02-svc.yml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
```

```
    svc: nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
```

```
kubectl apply -f .
```

Ref.

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

Hintergrund Ingress

Ref. / Dokumentation

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

Ingress Controller auf Digitalocean (doks) mit helm installieren

Basics

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

Prerequisites

- kubectl muss eingerichtet sein

Walkthrough (Setup Ingress Controller)

```
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 --namespace ingress --create-namespace --set
controller.publishService.enabled=true

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

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

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

Documentation for default ingress nginx

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

Beispiel Ingress

Prerequisits

```
## Ingress Controller muss aktiviert sein
microk8s enable ingress
```

Walkthrough

Schritt 1:

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

```
## apple.yml
## vi apple.yml
kind: Pod
```

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

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

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

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

---

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

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

Schritt 2:

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
        - path: /apple
          backend:
            serviceName: apple-service
            servicePort: 80
        - path: /banana
          backend:
```

```
            serviceName: banana-service
            servicePort: 80
```

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

Reference

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

Find the problem

```
## Hints

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

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

## now we can adjust our config
```

Solution

```
## in kubernetes 1.22.2 - ingress.yml needs to be modified like so.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

Install Ingress On Digitalocean DOKS

Beispiel mit Hostnamen

Prerequisits

```
## Ingress Controller muss aktiviert sein
### Nur der Fall wenn man microk8s zum Einrichten verwendet
### Ubuntu
microk8s enable ingress
```

Walkthrough

Step 1: pods and services

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

```
## apple.yml
## vi apple.yml
kind: Pod
```

```
apiVersion: v1
metadata:
  name: apple-app
  labels:
    app: apple
spec:
  containers:
    - name: apple-app
      image: hashicorp/http-echo
      args:
        - "-text=apple-<dein-name>"
---

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

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

```
## 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-<dein-name>"

---

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

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

Step 2: Ingress

```
## 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
    # otherwice it does not know, which controller to use
    # old version... use ingressClassName instead
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "<euername>.lab<nr>.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-ressources

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

## now we can adjust our config
```

Solution

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

Achtung: Ingress mit Helm - annotations

Permanente Weiterleitung mit Ingress

Example

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

---
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.de
    nginx.ingress.kubernetes.io/permanent-redirect-code: "308"
  creationTimestamp: null
  name: destination-home
  namespace: my-namespace
spec:
  rules:
  - host: web.training.local
    http:
      paths:
      - backend:
          service:
            name: http-svc
            port:
              number: 80
        path: /source
        pathType: ImplementationSpecific
```

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

/etc/hosts
45.23.12.12 web.training.local
```

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

Umbauen zu google ;o)

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

Refs:

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

ConfigMap Example

Schritt 1: configmap vorbereiten

```
cd
mkdir -p manifests
cd manifests
mkdir configmaptests
cd configmaptests
nano 01-configmap.yml
```

```
### 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-configmap
data:
  # als Wertepaare
  database: mongodb
  database_uri: mongodb://localhost:27017
```

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

Schrit 2: Beispiel als Datei

```
nano 02-pod.yml
```

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-mit-configmap

spec:
```

```
  # Add the ConfigMap as a volume to the Pod
  volumes:
    # `name` here must match the name
    # specified in the volume mount
    - name: example-configmap-volume
      # Populate the volume with config map data
      configMap:
        # `name` here must match the name
        # specified in the ConfigMap's YAML
        name: example-configmap

  containers:
    - name: container-configmap
      image: nginx:latest
      # Mount the volume that contains the configuration data
      # into your container filesystem
      volumeMounts:
        # `name` here must match the name
        # from the volumes section of this pod
        - name: example-configmap-volume
          mountPath: /etc/config
```

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

```
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-mit-configmap -- ls -la /etc/config
kubectl exec -it pod-mit-configmap --  bash
## ls -la /etc/config
```

Schritt 3: Beispiel. ConfigMap als env-variablen

```
nano 03-pod-mit-env.yml
```

```
## 03-pod-mit-env.yml
kind: Pod
apiVersion: v1
metadata:
  name: pod-env-var
spec:
  containers:
    - name: env-var-configmap
      image: nginx:latest
      envFrom:
        - configMapRef:
            name: example-configmap
```

```
kubectl apply -f 03-pod-mit-env.yml
```

```
## und wir schauen uns das an
##Jetzt schauen wir uns den Container/Pod mal an
kubectl exec pod-env-var -- env
kubectl exec -it pod-env-var --  bash
## env
```

Reference:

- https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html

Configmap MariaDB - Example

Schritt 1: configmap

```
cd
mkdir -p manifests
cd manifests
mkdir cftest
cd cftest
nano 01-configmap.yml
```

```
### 01-configmap.yml
kind: ConfigMap
apiVersion: v1
metadata:
  name: mariadb-configmap
data:
```

```
  # als Wertepaare
  MARIADB_ROOT_PASSWORD: 11abc432
```

```
kubectl apply -f .
kubectl get cm
kubectl get cm mariadb-configmap -o yaml
```

Schritt 2: Deployment

```
nano 02-deploy.yml
```

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

```
kubectl apply -f .
```

Important Sidenode

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

Configmap MariaDB my.cnf

configmap zu fuss

```
vi mariadb-config2.yml
```

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

```
kubectl apply -f .
```

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

```
    spec:
      containers:
      - name: mariadb-cont
        image: mariadb:latest
        envFrom:
        - configMapRef:
            name: mariadb-configmap

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

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

```
kubectl apply -f .
```

Kubernetes Deployment Scenarios

Deployment green/blue,canary,rolling update

Canary Deployment

```
A small group of the user base will see the new application
(e.g. 1000 out of 100.000), all the others will still see the old version

From: a canary was used to test if the air was good in the mine
(like a test balloon)
```

Blue / Green Deployment

```
The current version is the Blue one
The new version is the Green one

New Version (GREEN) will be tested and if it works
the traffic will be switch completey to the new version (GREEN)

Old version can either be deleted or will function as fallback
```

A/B Deployment/Testing

```
2 Different versions are online, e.g. to test a new design / new feature
You can configure the weight (how much traffic to one or the other)
by the number of pods
```

Example Calculation

```
e.g. Deployment1: 10 pods
Deployment2: 5 pods

Both have a common label,
The service will access them through this label
```

Service Blue/Green

Step 1: Deployment + Service

```
## vi blue.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-version-blue
spec:
  selector:
    matchLabels:
      version: blue
  replicas: 10 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
        version: blue
    spec:
```

```
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
```

```
## vi green.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-version-green
spec:
  selector:
    matchLabels:
      version: green
  replicas: 1 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
        version: green
    spec:
      containers:
      - name: nginx
        image: nginx:1.22
        ports:
        - containerPort: 80
```

```
## svc.yml
apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

Step 2: Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-config
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # old version useClassName instead
    # otherwice it does not know, which controller to use
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "app.lab1.t3isp.de"
    http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: svc-nginx
              port:
                number: 80
```

```
kubectl apply -f .
```

Praxis-Übung A/B Deployment

Walkthrough

```
cd
cd manifests
mkdir ab
cd ab
```

```
## vi 01-cm-version1.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-version-1
data:
  index.html: |
    <html>
    <h1>Welcome to Version 1</h1>
    </br>
    <h1>Hi! This is a configmap Index file Version 1 </h1>
    </html>
```

```
## vi 02-deployment-v1.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy-v1
spec:
  selector:
    matchLabels:
      version: v1
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
        version: v1
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        volumeMounts:
            - name: nginx-index-file
              mountPath: /usr/share/nginx/html/
      volumes:
      - name: nginx-index-file
        configMap:
          name: nginx-version-1
```

```
## vi 03-cm-version2.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-version-2
data:
  index.html: |
    <html>
    <h1>Welcome to Version 2</h1>
    </br>
    <h1>Hi! This is a configmap Index file Version 2 </h1>
    </html>
```

```
## vi 04-deployment-v2.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy-v2
spec:
  selector:
    matchLabels:
      version: v2
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
        version: v2
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        volumeMounts:
```

```
            - name: nginx-index-file
              mountPath: /usr/share/nginx/html/
      volumes:
      - name: nginx-index-file
        configMap:
          name: nginx-version-2
```

```
## vi 05-svc.yml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    svc: nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

```
kubectl apply -f .
## get external ip
kubectl get nodes -o wide
## get port
kubectl get svc my-nginx -o wide
## test it with curl apply it multiple time (at least ten times)
curl <external-ip>:<node-port>
```

Helm (Kubernetes Paketmanager)

Helm Grundlagen

Wo ?

```
artifacts helm
```

- https://artifacthub.io/

Komponenten

```
Chart - beeinhaltet Beschreibung und Komponenten
tar.gz - Format
oder Verzeichnis

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

Installation

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

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

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

## Test
kubectl cluster-info
```

Helm Warum ?

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

Helm Example

Prerequisites

- kubectl needs to be installed and configured to access cluster
- Good: helm works as unprivileged user as well - Good for our setup
- install helm on ubuntu (client) as root: snap install --classic helm

    - this installs helm3
- Please only use: helm3. No server-side components needed (in cluster)
    - Get away from examples using helm2 (hint: helm init) - uses tiller

Simple Walkthrough (Example 0)

```
## Repo hinzufpgen
helm repo add bitnami https://charts.bitnami.com/bitnami
## gecachte Informationen aktualieren
helm repo update

helm search repo bitnami
## helm install release-name bitnami/mysql
helm install my-mysql bitnami/mysql
## Chart runterziehen ohne installieren
## helm pull bitnami/mysql

## Release anzeigen zu lassen
helm list

## Status einer Release / Achtung, heisst nicht unbedingt nicht, dass pod läuft
helm status my-mysql

## weitere release installieren
## helm install neuer-release-name  bitnami/mysql
```

Under the hood

```
## Helm speichert Informationen über die Releases in den Secrets
kubectl get secrets | grep helm
```

Example 1: - To get know the structure

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami
helm repo update
helm pull bitnami/mysql
tar xzvf mysql-9.0.0.tgz
```

Example 2: We will setup mysql without persistent storage (not helpful in production ;o()

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami
helm repo update

helm install my-mysql bitnami/mysql
```

Example 2 - continue - fehlerbehebung

```
helm uninstall my-mysql
## Install with persistentStorage disabled - Setting a specific value
helm install my-mysql --set primary.persistence.enabled=false bitnami/mysql

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

Example 2b: using a values file

```
## mkdir helm-mysql
## cd helm-mysql
## vi values.yml
primary:
  persistence:
    enabled: false
```

```
helm uninstall my-mysql
helm install my-mysql bitnami/mysql -f values.yml
```

Example 3: Install wordpress

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-wordpress \
  --set wordpressUsername=admin \
  --set wordpressPassword=password \
  --set mariadb.auth.rootPassword=secretpassword \
    bitnami/wordpress
```

Example 4: Install Wordpress with values and auth

```
## mkdir helm-mysql
## cd helm-mysql
## vi values.yml
persistence:
  enabled: false



wordpressUsername: admin
wordpressPassword: password
mariadb:
  primary:
    persistence:
      enabled: false

  auth:
    rootPassword: secretpassword
```

```
helm uninstall my-wordpress
helm install my-wordpress bitnami/wordpress -f values
```

Referenced

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

Kubernetes - RBAC

Nutzer einrichten microk8s ab kubernetes 1.25

Enable RBAC in microk8s

```
## This is important, if not enable every user on the system is allowed to do everything
microk8s enable rbac
```

Schritt 1: Nutzer-Account auf Server anlegen und secret anlegen / in Client

```
cd
mkdir -p manifests/rbac
cd manifests/rbac
```

Mini-Schritt 1: Definition für Nutzer

```
## vi service-account.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: training
  namespace: default
```

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

Mini-Schritt 1.5: Secret erstellen

- From Kubernetes 1.25 tokens are not created automatically when creating a service account (sa)
- You have to create them manually with annotation attached
- https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

```
## vi secret.yml
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: trainingtoken
```

```
  annotations:
    kubernetes.io/service-account.name: training
```

```
kubectl apply -f .
```

Mini-Schritt 2: ClusterRolle festlegen - Dies gilt für alle namespaces, muss aber noch zugewiesen werden

```
### Bevor sie zugewiesen ist, funktioniert sie nicht - da sie keinem Nutzer zugewiesen ist

## vi pods-clusterrole.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-clusterrole
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

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

Mini-Schritt 3: Die ClusterRolle den entsprechenden Nutzern über RoleBinding zu ordnen

```
## vi rb-training-ns-default-pods.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rolebinding-ns-default-pods
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pods-clusterrole
subjects:
- kind: ServiceAccount
  name: training
  namespace: default
```

```
kubectl apply -f rb-training-ns-default-pods.yml
```

Mini-Schritt 4: Testen (klappt der Zugang)

```
kubectl auth can-i get pods -n default --as system:serviceaccount:default:training
```

Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

Mini-Schritt 1: kubeconfig setzen

```
kubectl config set-context training-ctx --cluster microk8s-cluster --user training

## extract name of the token from here

TOKEN=`kubectl get secret trainingtoken -o jsonpath='{.data.token}' | base64 --decode`
echo $TOKEN
kubectl config set-credentials training --token=$TOKEN
kubectl config use-context training-ctx

## Hier reichen die Rechte nicht aus
kubectl get deploy
## Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:kube-system:training" cannot list # resource
"pods" in API group "" in the namespace "default"
```

Mini-Schritt 2:

```
kubectl config use-context training-ctx
kubectl get pods
```

Mini-Schritt 3: Zurück zum alten Default-Context

```
kubectl config get-contexts
```

```
CURRENT   NAME           CLUSTER            AUTHINFO    NAMESPACE
          microk8s       microk8s-cluster   admin2
*         training-ctx   microk8s-cluster   training2
```

```
kubectl config use-context microk8s
```

Refs:

- https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm
- https://microk8s.io/docs/multi-user
- https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286

Ref: Create Service Account Token

- https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

Tipps&Tricks zu Deploymnent - Rollout

Warum

```
Rückgängig machen von deploys, Deploys neu unstossen.
(Das sind die wichtigsten Fähigkeiten
```

Beispiele

```
## Deployment nochmal durchführen
## z.B. nach kubectl uncordon n12.training.local
kubectl rollout restart deploy nginx-deployment

## Rollout rückgängig machen
kubectl rollout undo deploy nginx-deployment
```

Kubernetes QoS

Quality of Service - evict pods

Die Class wird auf Basis der Limits und Requests der Container vergeben

```
Request: Definiert wieviel ein Container mindestens braucht (CPU,memory)
Limit: Definiert, was ein Container maximal braucht.

in spec.containers.resources
kubectl explain pod.spec.containers.resources
```

Art der Typen:

- Guaranteed
- Burstable
- BestEffort

Guaranteed

```
Type: Guaranteed:
https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/#create-a-pod-that-gets-assigned-a-qos-class-of-
guaranteed

set when limit equals request
(request: das braucht er,
limit: das braucht er maximal)

Garantied ist die höchste Stufe und diese werden bei fehlenden Ressourcen
als letztes "evicted"

apiVersion: v1

kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"

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

Kustomize

Kustomize Overlay Beispiel

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
cd; mkdir -p manifests/kustomize-example1/base; mkdir -p manifests/kustomize-example1/overlays/prod; cd manifests/kustomize-
example1
```

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

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

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

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

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

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

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

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

```
## 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: 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

Helm mit kustomize verheiraten

Kubernetes - Tipps & Tricks

Kubernetes Debuggen ClusterIP/PodIP

Situation

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

Lösung

```
## Wir starten eine Busybox und fragen per wget und port ab
## busytester ist der name
## long version
kubectl run -it --rm --image=busybox busytester
## wget <pod-ip-des-ziels>
## exit


## quick and dirty
kubectl run -it --rm --image=busybox busytester -- wget <pod-ip-des-ziels>
```

Debugging pods

How ?

1. Which pod is in charge
2. Problems when starting: kubectl describe po mypod
3. Problems while running: kubectl logs mypod

Taints und Tolerations

Taints

```
Taints schliessen auf einer Node alle Pods aus, die nicht bestimmte taints haben:

Möglichkeiten:

o Sie werden nicht gescheduled - NoSchedule
o Sie werden nicht executed - NoExecute
o Sie werden möglichst nicht gescheduled. - PreferNoSchedule
```

Tolerations

```
Tolerations werden auf Pod-Ebene vergeben:
tolerations:

Ein Pod kann (wenn es auf einem Node taints gibt), nur
gescheduled bzw. ausgeführt werden, wenn er die
Labels hat, die auch als
Taints auf dem Node vergeben sind.
```

Walkthrough

Step 1: Cordon the other nodes - scheduling will not be possible there

```
## Cordon nodes n11 and n111
## You will see a taint here
kubectl cordon n11
kubectl cordon n111
kubectl describe n111 | grep -i taint
```

Step 2: Set taint on first node

```
kubectl taint nodes n1 gpu=true:NoSchedule
```

Step 3

```
cd
mkdir -p manifests
cd manifests
mkdir tainttest
cd tainttest
nano 01-no-tolerations.yml
```

```
##vi 01-no-tolerations.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-no-tol
  labels:
    env: test-env
spec:
  containers:
  - name: nginx
    image: nginx:1.21
```

```
kubectl apply -f .
kubectl get po nginx-test-no-tol
kubectl get describe nginx-test-no-tol
```

Step 4:

```
## vi 02-nginx-test-wrong-tol.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-wrong-tol
  labels:
    env: test-env
spec:
  containers:
  - name: nginx
    image: nginx:latest
  tolerations:
  - key: "cpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

```
kubectl apply -f .
kubectl get po nginx-test-wrong-tol
kubectl describe po nginx-test-wrong-tol
```

Step 5:

```
## vi 03-good-tolerations.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-test-good-tol
  labels:
    env: test-env
spec:
  containers:
  - name: nginx
    image: nginx:latest
  tolerations:
  - key: "gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

```
kubectl apply -f .
kubectl get po nginx-test-good-tol
kubectl describe po nginx-test-good-tol
```

Taints rausnehmen

```
kubectl taint nodes n1 gpu:true:NoSchedule-
```

uncordon other nodes

```
kubectl uncordon n11
kubectl uncordon n111
```

References

- Doku Kubernetes Taints and Tolerations
- https://blog.kubecost.com/blog/kubernetes-taints/

Autoscaling Pods/Deployments

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

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 3
  selector:
```

```
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello
        image: k8s.gcr.io/hpa-example
        resources:
          requests:
            cpu: 100m
---
kind: Service
apiVersion: v1
metadata:
  name: hello
spec:
  selector:
    app: hello
  ports:
    - port: 80
      targetPort: 80
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hello
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hello
  minReplicas: 2
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

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

Reference

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

pod aus deployment bei config - Änderung neu ausrollen

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

Kubernetes Advanced

Curl api-server kubernetes aus pod heraus

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

Kubernetes - Documentation

Documentation zu microk8s plugins/addons

- https://microk8s.io/docs/addons

Shared Volumes - Welche gibt es ?

- https://kubernetes.io/docs/concepts/storage/volumes/

Kubernetes - Hardening

Kubernetes Tipps Hardening

PSA (Pod Security Admission)

```
Policies defined by namespace.
e.g. not allowed to run container as root.

Will complain/deny when creating such a pod with that container type
```

Möglichkeiten in Pods und Containern

```
## für die Pods
kubectl explain pod.spec.securityContext
kubectl explain pod.spec.containers.securityContext
```

Example (seccomp / security context)

```
A. seccomp - profile
https://github.com/docker/docker/blob/master/profiles/seccomp/default.json
```

```
apiVersion: v1
kind: Pod
metadata:
  name: audit-pod
  labels:
    app: audit-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json

  containers:

  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

SecurityContext (auf Pod Ebene)

```
kubectl explain pod.spec.containers.securityContext
```

NetworkPolicy

```
## Firewall Kubernetes
```

Kubernetes Security Admission Controller Example

Seit: 1.2.22 Pod Security Admission

- 1.2.22 - ALpha - D.h. ist noch nicht aktiviert und muss als Feature Gate aktiviert (Kind)
- 1.2.23 - Beta -> d.h. aktiviert

Vorgefertigte Regelwerke

- privileges - keinerlei Einschränkungen
- baseline - einige Einschränkungen
- restricted - sehr streng

Praktisches Beispiel für Version ab 1.2.23 - Problemstellung

```
mkdir -p manifests
cd manifests
mkdir psa
cd psa
nano 01-ns.yml
```

```
## Schritt 1: Namespace anlegen
## vi 01-ns.yml

apiVersion: v1
kind: Namespace
metadata:
  name: test-ns1
  labels:
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

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

```
## Schritt 2: Testen mit nginx - pod
## vi 02-nginx.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns1
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
```

```
## a lot of warnings will come up
kubectl apply -f 02-nginx.yml
```

```
## Schritt 3:
## Anpassen der Sicherheitseinstellung (Phase1) im Container

## vi 02-nginx.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns1
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
```

```
kubectl delete -f 02-nginx.yml
kubectl apply -f 02-nginx.yml
kubectl -n test-ns1 get pods
```

```
## Schritt 4:
## Weitere Anpassung runAsNotRoot
## vi 02-nginx.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns<tln>
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
        runAsNonRoot: true
```

```
## pod kann erstellt werden, wird aber nicht gestartet
kubectl delete -f 02-nginx.yml
kubectl apply -f 02-nginx.yml
kubectl -n test-ns1 get pods
kubectl -n test-ns1 describe pods nginx
```

Praktisches Beispiel für Version ab 1.2.23 -Lösung - Container als NICHT-Root laufen lassen

- Wir müssen ein image, dass auch als NICHT-Root laufen kann
- .. oder selbst eines bauen (;o)) o bei nginx ist das bitnami/nginx

```
## vi 03-nginx-bitnami.yml
apiVersion: v1
kind: Pod
metadata:
```

```
  name: bitnami-nginx
  namespace: test-ns1
spec:
  containers:
    - image: bitnami/nginx
      name: bitnami-nginx
      ports:
        - containerPort: 80
      securityContext:
        seccompProfile:
          type: RuntimeDefault
        runAsNonRoot: true
```

```
## und er läuft als nicht root
kubectl apply -f 03_pod-bitnami.yml
kubectl -n test-ns1 get pods
```

Was muss ich bei der Netzwerk-Sicherheit beachten ?

Bereich 1: Kubernetes (Cluster)

```
1. Welche Ports sollten wirklich geöffnet sein ?

für Kubernetes

2. Wer muss den von wo den Kube-Api-Server zugreifen

- den Traffic einschränken
```

Bereich 2: Nodes

```
Alle nicht benötigten fremden Ports sollten geschlossen sein
Wenn offen, nur über vordefinierte Zugangswege (und auch nur bestimmte Nutzer)
```

Pods (Container / Image)

```
## Ingress (NetworkPolicy) - engmaschig stricken
## 1. Wer soll von wo auf welche Pod zugreifen können

## 2. Welche Pod auf welchen anderen Pod (Service)

ä Egress
## Welche Pods dürfen wohin nach draussen
```

Einschränking der Fähigkeien eines Pods

```
kein PrivilegeEscalation
nur notwendige Capabilities
unter einem nicht-root Benutzer laufen lassen
...

### Patching
```

pods -> neuestes images bei security vulnerablities

nodes -> auch neues patches (apt upgrade)

kubernetes cluster -> auf dem neuesten Stand

-> wie ist der Prozess ClusterUpdate, update der manifeste zu neuen API-Versionen

```
### RBAC
```

Nutzer (kubectl, systemnutzer -> pods)

1. Zugriff von den pods

2. Zugriff über helm / kubectl

Wer darf was ? Was muss der Nutzer können

```
### Compliance
```

PSP's / PSA PodSecurityPolicy was deprecated in Kubernetes v1.21, and removed from Kubernetes in v1.25

PSA - Pode Security Admission

```
## Kubernetes Interna / Misc.

### OCI,Container,Images Standards


### Schritt 1:
```

cd mkdir bautest cd bautest

```
### Schritt 2:
```

nano docker-compose.yml

version: "3.8"

services: myubuntu: build: ./myubuntu restart: always

```
### Schritt 3:
```

mkdir myubuntu cd myubuntu

```

```

nano hello.sh

```

```

##!/bin/bash let i=0

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

```

```

nano Dockerfile

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

```
### Schritt 4:
```

cd ../

wichtig, im docker-compose - Ordner seiend

##pwd ##~/bautest docker-compose up -d

wird image gebaut und container gestartet

Bei Veränderung vom Dockerfile, muss man den Parameter --build mitangeben

docker-compose up -d --build

```
### Geolocation Kubernetes Cluster

  * https://learnk8s.io/bite-sized/connecting-multiple-kubernetes-clusters

## Kubernetes - Überblick

### Installation - Welche Komponenten from scratch


### Step 1: Server 1 (manuell installiert -> microk8s)
```

Installation Ubuntu - Server

cloud-init script

s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

Server 1 - manuell

Ubuntu 20.04 LTS - Grundinstallation

minimal Netzwerk - öffentlichen IP

nichts besonderes eingerichtet - Standard Digitalocean

Standard vo Installation microk8s

lo UNKNOWN 127.0.0.1/8 ::1/128

public ip / interne

eth0 UP 164.92.255.234/20 10.19.0.6/16 fe80::c:66ff:fec4:cbce/64

private ip

eth1 UP 10.135.0.3/16 fe80::8081:aaff:feaa:780/64

snap install microk8s --classic

namensaufloesung fuer pods

microk8s enable dns

Funktioniert microk8s

microk8s status

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

Was macht das ?

1. Basisnutzer (11trainingdo) - keine Voraussetzung für microk8s

2. Installation von microk8s

##.>>>>>>> microk8s installiert <<<<<<<<

- snap install --classic microk8s

>>>>>>> Zuordnung zur Gruppe microk8s - notwendig für bestimmte plugins (z.B. helm)

usermod -a -G microk8s root

>>>>>>> Setzen des .kube - Verzeichnisses auf den Nutzer microk8s -> nicht zwingend erforderlich

chown -r -R microk8s ~/.kube

>>>>>>> REQUIRED .. DNS aktivieren, wichtig für Namensauflösungen innerhalb der PODS

>>>>>>> sonst funktioniert das nicht !!!

microk8s enable dns

>>>>>>> kubectl alias gesetzt, damit man nicht immer microk8s kubectl eingeben muss

- echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

cloud-init script

s.u. MITMICROK8S (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

##cloud-config users:

- name: 11trainingdo shell: /bin/bash

runcmd:

- sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
- echo " " >> /etc/ssh/sshd_config
- echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
- echo "AllowUsers root" >> /etc/ssh/sshd_config
- systemctl reload sshd
- sed -i '/11trainingdo/c 11trainingdo:6HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBKl.SYbhS52u70:17476:0:99999:7:::' /etc/shadow
- echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
- chmod 0440 /etc/sudoers.d/11trainingdo
- echo "Installing microk8s"
- snap install --classic microk8s
- usermod -a -G microk8s root
- chown -f -R microk8s ~/.kube
- microk8s enable dns
- echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc

Prüfen ob microk8s - wird automatisch nach Installation gestartet

kann eine Weile dauern

microk8s status

```
### Step 3: Client - Maschine (wir sollten nicht auf control-plane oder cluster - node arbeiten
```

Weiteren Server hochgezogen. Vanilla + BASIS

Installation Ubuntu - Server

cloud-init script

s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

Server 1 - manuell

Ubuntu 20.04 LTS - Grundinstallation

minimal Netzwerk - öffentlichen IP

nichts besonderes eingerichtet - Standard Digitalocean

Standard vo Installation microk8s

lo UNKNOWN 127.0.0.1/8 ::1/128

public ip / interne

eth0 UP 164.92.255.232/20 10.19.0.6/16 fe80::c:66ff:fec4:cbce/64

private ip

eth1 UP 10.135.0.5/16 fe80::8081:aaff:feaa:780/64

Installation von kubectl aus dem snap

NICHT .. keine microk8s - keine control-plane / worker-node

NUR Client zum Arbeiten

snap install kubectl --classic

.kube/config

Damit ein Zugriff auf die kube-server-api möglich

d.h. REST-API Interface, um das Cluster verwalten.

Hier haben uns für den ersten Control-Node entschieden

Alternativ wäre round-robin per dns möglich

Mini-Schritt 1:

Auf dem Server 1: kubeconfig ausspielen

microk8s config > /root/kube-config

auf das Zielsystem gebracht (client 1)

scp /root/kubeconfig 11trainingdo@10.135.0.5:/home/11trainingdo

Mini-Schritt 2:

Auf dem Client 1 (diese Maschine) kubeconfig an die richtige Stelle bringen

Standardmäßig der Client nach eine Konfigurationsdatei sucht in ~/.kube/config

sudo su - cd mkdir .kube cd .kube mv /home/11trainingdo/kube-config config

Verbindungstest gemacht

Damit feststellen ob das funktioniert.

kubectl cluster-info

```
### Schritt 4: Auf allen Servern IP's hinterlegen und richtigen Hostnamen überprüfen
```

Auf jedem Server

hostnamectl

evtl. hostname setzen

z.B. - auf jedem Server eindeutig

hostnamectl set-hostname n1.training.local

Gleiche hosts auf allen server einrichten.

Wichtig, um Traffic zu minimieren verwenden, die interne (private) IP

/etc/hosts 10.135.0.3 n1.training.local n1 10.135.0.4 n2.training.local n2 10.135.0.5 n3.training.local n3

```
### Schritt 5: Cluster aufbauen
```

Mini-Schritt 1:

Server 1: connection - string (token)

microk8s add-node

Zeigt Liste und wir nehmen den Eintrag mit der lokalen / öffentlichen ip

Dieser Token kann nur 1x verwendet werden und wir auf dem ANDEREN node ausgeführt

microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

Mini-Schritt 2:

Dauert eine Weile, bis das durch ist.

Server 2: Den Node hinzufügen durch den JOIN - Befehl

microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

Mini-Schritt 3:

Server 1: token besorgen für node 3

microk8s add-node

Mini-Schritt 4:

Server 3: Den Node hinzufügen durch den JOIN-Befehl

microk8s join 10.135.0.3:25000/09c96e57ec12af45b2752fb45450530c/bcad1949221a

Mini-Schritt 5: Überprüfen ob HA-Cluster läuft

Server 1: (es kann auf jedem der 3 Server überprüft werden, auf einem reicht microk8s status | grep high-availability high-availability: yes

```
### Ergänzend nicht notwendige Scripte
```

cloud-init script

s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

Digitalocean - unter user_data reingepastet beim Einrichten

##cloud-config users:

- name: 11trainingdo shell: /bin/bash

runcmd:

- sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
- echo " " >> /etc/ssh/sshd_config
- echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
- echo "AllowUsers root" >> /etc/ssh/sshd_config
- systemctl reload sshd
- sed -i '/11trainingdo/c 11trainingdo:6HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBKl.SYbhS52u70:17476:0:99999:7:::' /etc/shadow
- echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
- chmod 0440 /etc/sudoers.d/11trainingdo

```
## Kubernetes - microk8s (Installation und Management)

### kubectl unter windows - Remote-Verbindung zu Kuberenets (microk8s) einrichten


### Walkthrough (Installation)
```

Step 1

chocolatry installiert. (powershell als Administrator ausführen)

https://docs.chocolatey.org/en-us/choco/setup

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

Step 2

choco install kubernetes-cli

Step 3

testen: kubectl version --client

Step 4:

powershell als normaler benutzer öffnen

```
### Walkthrough (autocompletion)
```

in powershell (normaler Benutzer) kubectl completion powershell | Out-String | Invoke-Expression

```
### kubectl - config - Struktur vorbereiten
```

in powershell im heimatordner des Benutzers .kube - ordnern anlegen

C:\Users<dein-name>\

mkdir .kube cd .kube

```
### IP von Cluster-Node bekommen
```

auf virtualbox - maschine per ssh einloggen

öffentliche ip herausfinden - z.B. enp0s8 bei HostOnly - Adapter

ip -br a

```
### config für kubectl aus Cluster-Node auslesen (microk8s)
```

auf virtualbox - maschine per ssh einloggen / zum root wechseln

abfragen

microk8s config

Alle Zeilen ins clipboard kopieren

und mit notepad++ in die Datei \Users<dein-name>.kube\config

schreiben

Wichtig: Zeile cluster -> clusters / server

Hier ip von letztem Schritt eintragen:

z.B.

Server: https://192.168.56.106/.....

```
### Testen
```

in powershell

kann ich eine Verbindung zum Cluster aufbauen ?

kubectl cluster-info

```


  * https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/

### Arbeiten mit der Registry

### Installation Kubernetes Dashboard


### Reference:
```

```
  * https://blog.tippybits.com/installing-kubernetes-in-virtualbox-3d49f666b4d6

## Kubernetes - RBAC

### Nutzer einrichten - kubernetes bis 1.24


### Enable RBAC in microk8s
```

This is important, if not enable every user on the system is allowed to do everything

microk8s enable rbac

```
### Schritt 1: Nutzer-Account auf Server anlegen / in Client
```

cd mkdir -p manifests/rbac cd manifests/rbac

```
####  Mini-Schritt 1: Definition für Nutzer
```

vi service-account.yml

apiVersion: v1 kind: ServiceAccount metadata: name: training namespace: default

```

```

kubectl apply -f service-account.yml

```

#### Mini-Schritt 2: ClusterRolle festlegen - Dies gilt für alle namespaces, muss aber noch zugewiesen werden
```

Bevor sie zugewiesen ist, funktioniert sie nicht - da sie keinem Nutzer zugewiesen ist

vi pods-clusterrole.yml

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: pods-clusterrole rules:

- apiGroups: [""] # "" indicates the core API group resources: ["pods"] verbs: ["get", "watch", "list"]

```

```

kubectl apply -f pods-clusterrole.yml

```

#### Mini-Schritt 3: Die ClusterRolle den entsprechenden Nutzern über RoleBinding zu ordnen
```

vi rb-training-ns-default-pods.yml

apiVersion: rbac.authorization.k8s.io/v1 kind: RoleBinding metadata: name: rolebinding-ns-default-pods namespace: default roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: pods-clusterrole subjects:

- kind: ServiceAccount name: training namespace: default

```

```

kubectl apply -f rb-training-ns-default-pods.yml

```

#### Mini-Schritt 4: Testen (klappt der Zugang)
```

kubectl auth can-i get pods -n default --as system:serviceaccount:default:training

```
### Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

#### Mini-Schritt 1: kubeconfig setzen
```

kubectl config set-context training-ctx --cluster microk8s-cluster --user training

extract name of the token from here

TOKEN= `kubectl get secret trainingtoken -o jsonpath='{.data.token}' | base64 --decode` echo $TOKEN kubectl config set-credentials training --token=$TOKEN kubectl config use-context training-ctx

Hier reichen die Rechte nicht aus

kubectl get deploy

Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:kube-system:training" cannot list # resource "pods" in API group "" in the namespace "default"

```
#### Mini-Schritt 2:
```

kubectl config use-context training-ctx kubectl get pods

```
### Refs:

  * https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm
  * https://microk8s.io/docs/multi-user
  * https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286

### Ref: Create Service Account Token

  * https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

## kubectl

### Tipps&Tricks zu Deploymnent - Rollout


### Warum
```

Rückgängig machen von deploys, Deploys neu unstossen. (Das sind die wichtigsten Fähigkeiten

```
### Beispiele
```

Deployment nochmal durchführen

z.B. nach kubectl uncordon n12.training.local

kubectl rollout restart deploy nginx-deployment

Rollout rückgängig machen

kubectl rollout undo deploy nginx-deployment

```
## Kubernetes - Monitoring (microk8s und vanilla)

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


### Warum ? Was macht er ?
```

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

kubectl top pods kubectl top nodes

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

```
### Walktrough
```

Auf einem der Nodes im Cluster (HA-Cluster)

microk8s enable metrics-server

Es dauert jetzt einen Moment bis dieser aktiv ist auch nach der Installation

Auf dem Client

kubectl top nodes kubectl top pods

```
### Kubernetes

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

```
## Kubernetes - Backups

## Kubernetes - Tipps & Tricks

### Assigning Pods to Nodes


### Walkthrough
```

leave n3 as is

kubectl label nodes n7 rechenzentrum=rz1 kubectl label nodes n17 rechenzentrum=rz2 kubectl label nodes n27 rechenzentrum=rz2

kubectl get nodes --show-labels

nginx-deployment

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 9 # tells deployment to run 2 pods matching the template template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 nodeSelector: rechenzentrum: rz2

Let's rewrite that to deployment

apiVersion: v1 kind: Pod metadata: name: nginx labels: env: test spec: containers:

- name: nginx image: nginx imagePullPolicy: IfNotPresent nodeSelector: rechenzentrum=rz2

```
### Ref:

  * https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

## Kubernetes - Documentation

### LDAP-Anbindung

  * https://github.com/apprenda-kismatic/kubernetes-ldap

### Helpful to learn - Kubernetes

  * https://kubernetes.io/docs/tasks/

### Environment to learn

  * https://killercoda.com/killer-shell-cks

### Environment to learn II

  * https://killercoda.com/

### Youtube Channel

  * https://www.youtube.com/watch?v=01qcYSck1c4

## Kubernetes - Shared Volumes

### Shared Volumes with nfs


### Create new server and install nfs-server
```

on Ubuntu 20.04LTS

apt install nfs-kernel-server systemctl status nfs-server

vi /etc/exports

adjust ip's of kubernetes master and nodes

kmaster

/var/nfs/ 192.168.56.101(rw,sync,no_root_squash,no_subtree_check)

knode1

/var/nfs/ 192.168.56.103(rw,sync,no_root_squash,no_subtree_check)

knode 2

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

exportfs -av

```
### On all nodes (needed for production)
```

apt install nfs-common

```
### On all nodes (only for testing)
```

Please do this on all servers (if you have access by ssh)

find out, if connection to nfs works !

for testing

mkdir /mnt/nfs

10.135.0.18 is our nfs-server

mount -t nfs 10.135.0.18:/var/nfs /mnt/nfs ls -la /mnt/nfs umount /mnt/nfs

```
### Persistent Storage-Step 1: Setup PersistentVolume in cluster
```

cd cd manifests mkdir -p nfs cd nfs nano 01-pv.yml

```

```

apiVersion: v1 kind: PersistentVolume metadata:

any PV name

name: pv-nfs-tln labels: volume: nfs-data-volume-tln 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/nginx server: 10.135.0.18 readOnly: false storageClassName: ""

```

```

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

```
### Persistent Storage-Step 2: Create Persistent Volume Claim
```

nano 02-pvc.yml

```

```

vi 02-pvc.yml

now we want to claim space

apiVersion: v1 kind: PersistentVolumeClaim metadata: name: pv-nfs-claim-tln spec: storageClassName: "" volumeName: pv-nfs-tln accessModes:

- ReadWriteMany resources: requests: storage: 1Gi

```

```

kubectl apply -f 02-pvc.yml kubectl get pvc

```
### Persistent Storage-Step 3: Deployment
```

deployment including mount

vi 03-deploy.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 4 # tells deployment to run 4 pods matching the template template: metadata: labels: app: nginx spec:

```
  containers:
  - name: nginx
```

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

    volumeMounts:
      - name: nfsvol
        mountPath: "/usr/share/nginx/html"

  volumes:
  - name: nfsvol
    persistentVolumeClaim:
      claimName: pv-nfs-claim-tln<tln>
```

```
```

kubectl apply -f 03-deploy.yml

```
### Persistent Storage Step 4: service
```

now testing it with a service

cat 04-service.yml

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

- port: 80 protocol: TCP selector: app: nginx

```
```

kubectl apply -f 04-service.yml

```
### Persistent Storage Step 5: write data and test
```

connect to the container and add index.html - data

kubectl exec -it deploy/nginx-deployment -- bash

in container

echo "hello dear friend" > /usr/share/nginx/html/index.html exit

now try to connect

kubectl get svc

connect with ip and port

kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh

curl http://

exit

now destroy deployment

kubectl delete -f 03-deploy.yml

Try again - no connection

kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh

curl http://

exit

```
### Persistent Storage Step 6: retest after redeployment
```

now start deployment again

kubectl apply -f 03-deploy.yml

and try connection again

kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh

curl http://

exit

```
## Kubernetes - Hardening

### Kubernetes Tipps Hardening


### PSA (Pod Security Admission)
```

Policies defined by namespace. e.g. not allowed to run container as root.

Will complain/deny when creating such a pod with that container type

```
### Möglichkeiten in Pods und Containern
```

für die Pods

kubectl explain pod.spec.securityContext kubectl explain pod.spec.containers.securityContext

```
### Example (seccomp / security context)
```

A. seccomp - profile https://github.com/docker/docker/blob/master/profiles/seccomp/default.json

```

```

apiVersion: v1 kind: Pod metadata: name: audit-pod labels: app: audit-pod spec: securityContext: seccompProfile: type: Localhost localhostProfile: profiles/audit.json

containers:

- name: test-container image: hashicorp/http-echo:0.2.3 args:
  - "-text=just made some syscalls!" securityContext: allowPrivilegeEscalation: false

```
### SecurityContext (auf Pod Ebene)
```

kubectl explain pod.spec.containers.securityContext

```


### NetworkPolicy
```

Firewall Kubernetes

```
## Kubernetes Probes (Liveness and Readiness)

### Übung Liveness-Probe


### Übung 1: Liveness (command)
```

What does it do ?

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

cd

mkdir -p manifests/probes

cd manifests/probes

vi 01-pod-liveness-command.yml

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

- name: liveness image: busybox args:
  - /bin/sh
  - -c
  - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600 livenessProbe: exec: command:
    - cat
    - /tmp/healthy initialDelaySeconds: 5 periodSeconds: 5

apply and test

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

cleanup

kubectl delete -f 01-pod-liveness-command.yml

```
### Übung 2: Liveness Probe (HTTP)
```

Step 0: Understanding Prerequisite:

This is how this image works:

after 10 seconds it returns code 500

http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) { duration := time.Now().Sub(started) if duration.Seconds() > 10 { w.WriteHeader(500) w.Write([]byte(fmt.Sprintf("error: %v", duration.Seconds()))) } else { w.WriteHeader(200) w.Write([]byte("ok")) } })

Step 1: Pod - manifest

vi 02-pod-liveness-http.yml

status-code >=200 and < 400 o.k.

else failure

apiVersion: v1 kind: Pod metadata: labels: test: liveness name: liveness-http spec: containers:

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

Step 2: apply and test

kubectl apply -f 02-pod-liveness-http.yml

after 10 seconds port should have been started

sleep 10 kubectl describe pod liveness-http

```
### Reference:
```

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

### Funktionsweise Readiness-Probe vs. Liveness-Probe


### Why / Howto /

  * Readiness checks, if container is ready and if it's not READY
    * SENDS NO TRAFFIC to the container


### Difference to LiveNess

  * They are configured exactly the same, but use another keyword
    * readinessProbe instead of livenessProbe

### Example
```

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

```
### Reference

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