Kubernetes Advanced

Agenda

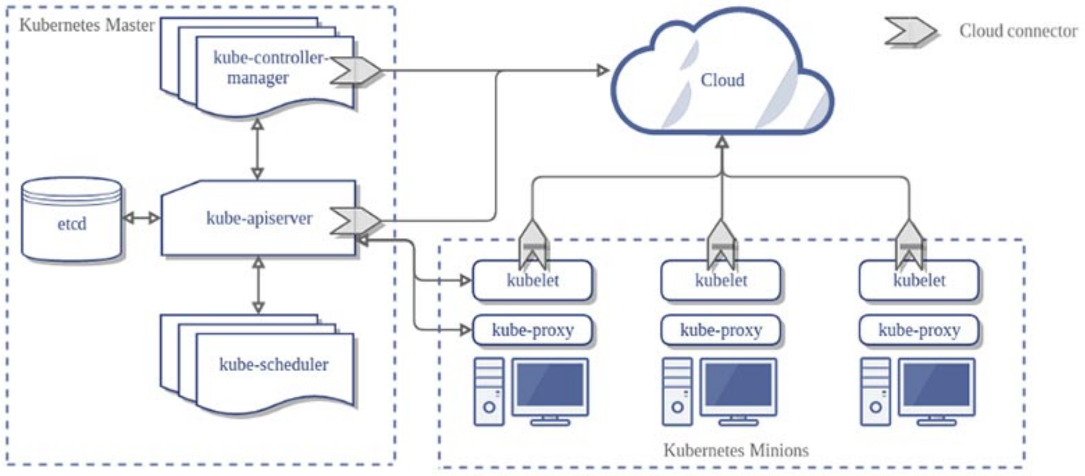
1. Kubernetes - Überblick
   - Aufbau Allgemein
   - Structure Kubernetes Deep Dive
   - Ports und Protokolle
   - kubelet garbage collection
2. Kubernetes Controlplane
   - Renew Certificate
   - HA-Cluster
3. Installation
   - Kubernetes mit der Cluster API aufsetzen
   - Kubernetes mit kubadm aufsetzen (calico)
4. Kubernetes Praxis API-Objekte
   - Das Tool kubectl (Devs/Ops) - Spickzettel
   - Bauen einer Applikation mit Resource Objekten
   - kubectl/manifest/deployments
   - Services - Aufbau
   - kubectl/manifest/service
   - DNS - Resolution - Services
   - DaemonSets (Devs/Ops)
   - Hintergrund Ingress
   - Beispiel mit Hostnamen
   - Configmap MariaDB - Example
5. Kubernetes - Probes
   - Überblick Probes
6. Kubernetes - Wartung / Debugging
   - Netzwerkverbindung zu pod testen
7. Kubernetes Backup
   - Backups mit Velero
8. Kubernetes Upgrade
   - Upgrade von tanzu (Cluster API)
9. Monitoring with Prometheus / Grafana
   - Overview
   - Setup prometheus/Grafana with helm
   - exporters mongodb
   - Good Kubernetes Board for Grafana
10. Kubernetes Tipps & Tricks
    - kubectl kubeconfig mergen
11. Kubernetes Certificates (Control Plane) / Security
    - vmware - cluster api
    - Pod Security Admission (PSA)
    - Pod Security Policy (PSP)
12. Kubernetes Network / Firewall
    - Calico/Cilium - nginx example NetworkPolicy
    - Egress / Ingress Examples with Exercise
    - Mesh / istio
13. Kubernetes Probes (Liveness and Readiness)
    - Übung Liveness-Probe
    - Übung Liveness http aus nginx
    - Funktionsweise Readiness-Probe vs. Liveness-Probe
    - Manueller Check readyz endpoint kubernetes api server aus pod
14. Kubernetes QoS / Limits / Requests
    - Quality of Service - evict pods
    - Tools to identify LimitRange and Requests

Backlog

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

Structure Kubernetes Deep Dive

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

Ports und Protokolle

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

kubelet garbage collection

What is do ?

- Deletes unused containers after 1 minutes
- and unused images after 5 minutes

Reference:

- https://kubernetes.io/docs/concepts/architecture/garbage-collection/#containers-images

list images with ctr

- ctr is the cli tool for containerd

```
## from client
kubectl run nginx --image nginx

## on worker - node
ctr images list | grep nginx
```

Kubernetes Controlplane

Renew Certificate

Zertifikate überprüfen

```
kubeadm certs check-expiration
```

```
. Wo werden Zertifikate benötigt ?

- zum kube-apiserver hin von den einzelnen Komponenten
- zum
usw.
```

Sonderrolle

```
b. Sonderrolle kubelet

Macht ein automatisches Renew the certifikate über die
Zertifikat api. Schritte:


Es erfolgt ein automatisches Approval des Signing Requests
über den Controller Manager

Diese muss aktiviert sein:

https://kubernetes.io/docs/tasks/tls/certificate-rotation/
--rotate-certificates
```

```
root@worker1:/var/lib/kubelet# grep -r "rotate" config.yaml
rotateCertificates: true
```

Zertifikatserneuerung

Schritt 1:

```
c. Wir erneuern wir Zertifikate ?

Wichtig: Das muss auf allen Control-Nodes passieren, wenn sie kurz vor dem ablaufen sind.

auf dem controlplane (bspw. api-server)
kubeadm certs renew apiserver
```

Schritt 2:

```
## nochmal gucken, welches Zertfikat genommen
echo | openssl s_client -showcerts -connect 64.226.76.200:6443 -servername api 2>/dev/null | openssl x509 -noout -enddate

### Wichtig, kein kubectl delete po verwenden .
## command output may be misleading in describing static pods: even if it shows that the static pod restarted recently, the correspondent pod containers were not restarted.

## dann das manifests wegschieben
cd /etc/kubernetes/manifests/
mv kube-apiserver.yaml /tmp

## will not work anymore, because apiserver is not running
kubectl -n kube-system get pods
```

Schritt 3: mit low-level tools checken pod noch läuft / weieder läuft

```
export CONTAINER_RUNTIME_ENDPOINT=unix:///var/run/containerd/containerd.sock
## taucht nicht mehr auf -> apiservre
crictl pods
```

```
mv /tmp/kube-apiserver.yaml .
crictl pods | grep api


kubectl get nodes

kubeadm certs check-expiration | grep "apiserver "
apiserver                  Jan 23, 2025 04:09 UTC   364d            ca                      no
```

```
echo | openssl s_client -showcerts -connect  64.226.76.200:6443 -servername api 2>/dev/null | openssl x509 -noout -enddate
notAfter=Jan 23 04:09:04 2025 GMT
```

Zertifikate ohne Downtime

Das wird nur funktionieren, wenn mir eine HA-Cluster haben. Dort gibt es mehrere Controlplanes und wir haben einen LoadBalancer davor. -> hier vielleicht noch ein Schaubild zeigen.

Ansonsten muss immer der kube-api-server neu gestartet werden und die einzelnen Komponenten, hier haben wir immer eine kurze Downtime.

Dies wird durch ein HA-Cluster vermieden. Dort ist ein LoadBalancer davorgeschaltet.

```

### HA-Cluster


### Übersicht

![image](https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/9f791d15-8c97-4f07-862b-cc2bf6035dc0)

### Aufsetzen eines HA-Clusters (auf vm's oder Metall)

  * https://kubesphere.io/docs/v3.4/installing-on-linux/high-availability-configurations/set-up-ha-cluster-using-keepalived-haproxy/
  * https://mvallim.github.io/kubernetes-under-the-hood/documentation/haproxy-cluster.html
  * https://www.lisenet.com/2021/install-and-configure-a-multi-master-ha-kubernetes-cluster-with-kubeadm-haproxy-and-keepalived-on-centos-7/

### Aufsetzen eines HA-Cluster (Internal)

  * https://github.com/kubesphere/kubekey/blob/master/docs/ha-mode.md

### Varianten den LoadBalancer zu platzieren

```

```
  * https://github.com/kubernetes/kubeadm/blob/main/docs/ha-considerations.md


## Installation

### Kubernetes mit der Cluster API aufsetzen


### Prerequisites

  * You need to have a Kubernetes Cluster running (this will be the management cluster)
    * Within that you will you have cluster api
    * This could be something like kind, rancherdesktop.io
    * And of course also a cluster on premise

### Step 1: Create Management Cluster

#### Step 1a: Install clusterctl
```

Install rancherdesktop.io

You are able to use it on windows (that's what we do now, and install

Install clusterctl in wsl -> Ubuntu

sudo su - cd /usr/src curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v1.4.2/clusterctl-linux-amd64 -o clusterctl sudo install -o root -g root -m 0755 clusterctl /usr/local/bin/clusterctl clusterctl version

```

  * Reference gist: https://gist.github.com/vfarcic/d8113b6f149583e1cf1614d76f2a4182
  * https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl

#### Step 1b: Set env variables for digitalocean
```

export DIGITALOCEAN_ACCESS_TOKEN=[...] # Replace with your token here

```


#### Step 1c: Create kubernetes snapshot to be used for Kubernetes Control Plane and workers
```

can be done as unprivileged user !!!

export PATH=$PATH:~/.local/bin sudo apt update apt install -y jq zip sudo git clone https://github.com/kubernetes-sigs/image-builder cd image-builder/images/capi cat Makefile

Size of machine will always be 1gb and 1vcpu created in NYC1

make build-do-ubuntu-2004

```

#### Step 1d: Add Snapshot to Region FRA1
```

-> Add to Region FRA1 -> under Manage -> Images -> Snapshots Please do this through the web-interface of DigitalOcean

IF YOU DO NOT DO THIS... Droplets cannot be created because they are in NYC1

```

#### Step 1e: Install doctl (optional)
```

works in most cases on wsl, but only if snap is working properly

snap install doctl

if not do -> this

cd ~ wget https://github.com/digitalocean/doctl/releases/download/v1.94.0/doctl-1.94.0-linux-amd64.tar.gz tar xf ~/doctl-1.94.0-linux-amd64.tar.gz sudo mv ~/doctl /usr/local/bin

now authenticate

doctl auth init --access-token ${DIGITALOCEAN_ACCESS_TOKEN}

```
#### Step 1f: Set env for to create worker cluster with controlplane and workers
```

control the datacenter - default nyc1

export DO_REGION=fra1

control size of machines

default 1vcpu-1gb

export DO_CONTROL_PLANE_MACHINE_TYPE=s-2vcpu-2gb export DO_NODE_MACHINE_TYPE=s-2vcpu-2gb

needed to set up the api provider

export DO_B64ENCODED_CREDENTIALS="$(
echo -n "$DIGITALOCEAN_ACCESS_TOKEN"
| base64
| tr -d '\n')"

get the snapshot id / get the right id

doctl compute image list-user

e.g.

132627725

export DO_CONTROL_PLANE_MACHINE_IMAGE=132627725 export DO_NODE_MACHINE_IMAGE=132627725

```
#### Step 1g: Setup cluster and api-provider
```

In our case it sets up the management cluster on rancher

to be used for kubernetes

cd ../../../

clusterctl init
--infrastructure digitalocean

```
#### Step 1h: Generate the yaml scripts for both control plane and workers
```

it looks there will be a fingerprint to be used, which chooses the ssh-key to be used

to connect to the machines

look for all the ssh-key like so:

doctl compute ssh-key list

So we choose one from the list

export DO_SSH_KEY_FINGERPRINT=[...]

Check the variables

Show use the necessary env-variables.

clusterctl generate cluster devops-toolkit
--infrastructure digitalocean
--target-namespace infra
--kubernetes-version v1.24.11
--control-plane-machine-count 3
--worker-machine-count 3
--list-variables

Kuberentes must be the same version as you created the snapshots for do

to be used for digitalocean -> creating a cluster there

clusterctl generate cluster devops-toolkit
--infrastructure digitalocean
--target-namespace infra
--kubernetes-version v1.24.11
--control-plane-machine-count 3
--worker-machine-count 3
| tee cluster.yaml

kubectl create namespace infra

kubectl apply --filename cluster.yaml

```
#### Step 1i: Wait till the control plane is initialized + install calico
```

kubectl get kubeadmcontrolplane

When initialized get kubeconfig and install calicao

clusterctl --namespace infra2
get kubeconfig devops-toolkit
| tee kubeconfig.yaml

kubectl --kubeconfig kubeconfig.yaml get ns

you will see control plane is not ready because of network missing

kubectl --kubeconfig kubeconfig.yaml get nodes

kubectl --kubeconfig kubeconfig.yaml apply -f https://docs.projectcalico.org/v3.25/manifests/calico.yaml

```
#### Step 1j: READY it is (says Yoda)
```

Wait a while, now you will see, the nodes are ready

kubectl --kubeconfig kubeconfig.yaml get nodes

```
### Kubernetes mit kubadm aufsetzen (calico)


### Version

  * Ubuntu 20.04 LTS

### Done for you

  * Servers are setup:
    * ssh-running
    * kubeadm, kubelet, kubectl installed
    * containerd - runtime installed

  * Installed on all nodes (with cloud-init)
```

##!/bin/bash

groupadd sshadmin USERS="mysupersecretuser" SUDO_USER="mysupersecretuser" PASS="yoursupersecretpass" for USER in $USERS do echo "Adding user $USER" useradd -s /bin/bash --create-home $USER usermod -aG sshadmin $USER echo "$USER:$PASS | chpasswd done

We can sudo with $SUDO_USER

usermod -aG sudo $SUDO_USER

20.04 and 22.04 this will be in the subfolder

if [-f /etc/ssh/sshd_config.d/50-cloud-init.conf] then sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config.d/50-cloud-init.conf fi

both is needed

sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config

usermod -aG sshadmin root

TBD - Delete AllowUsers Entries with sed

otherwice we cannot login by group

echo "AllowGroups sshadmin" >> /etc/ssh/sshd_config systemctl reload sshd

Now let us do some generic setup

echo "Installing kubeadm kubelet kubectl"

A lot of stuff needs to be done here

https://www.linuxtechi.com/install-kubernetes-on-ubuntu-22-04/

1. no swap please

swapoff -a sudo sed -i '/ swap / s/^(.*)$/#\1/g' /etc/fstab

2. Loading necessary modules

echo "overlay" >> /etc/modules-load.d/containerd.conf echo "br_netfilter" >> /etc /modules-load.d/containerd.conf modprobe overlay modprobe br_netfilter

3. necessary kernel settings

echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.d/kubernetes.conf sysctl --system

4. Update the meta-information

apt-get -y update

5. Installing container runtime

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" apt-get install -y containerd.io

6. Configure containerd

containerd config default > /etc/containerd/config.toml sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml systemctl restart containerd systemctl enable containerd

7. Add Kubernetes Repository for Kubernetes

mkdir -m 755 /etc/apt/keyrings apt-get install -y apt-transport-https ca-certificates curl gpg curl -fsSL https://pkgs.k8s.io/core:/stable:/$K8S_VERSION/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/$K8S_VERSI # 8. Install kubectl kubeadm kubectl apt-get -y update apt-get install -y kubelet kubeadm kubectl apt-mark hold -y kubelet kubeadm kubectl

9. Install helm

snap install helm --classic

Installing nfs-common

apt-get -y install nfs-common

```
### Prerequisites

  * 4 Servers setup and reachable through ssh.
  * user: 11trainingdo
  * pass: PLEASE ask your instructor
```

Important - Servers are not reachable through

Domain !! Only IP.

controlplane.tln.t3isp.de worker1.tln.do.t3isp.de worker2.tln.do.t3isp.de worker3.tln.do.t3isp.de

```
### Step 1: Setup controlnode (login through ssh)
```

This CIDR is the recommendation for calico

Other CNI's might be different

CLUSTER_CIDR="192.168.0.0/16"

kubeadm init --pod-network-cidr=$CLUSTER_CIDR &&
mkdir -p /root/.kube &&
cp -i /etc/kubernetes/admin.conf /root/.kube/config &&
chown $(id -u):$(id -g) /root/.kube/config &&
cp -i /root/.kube/config /tmp/config.kubeadm &&
chmod o+r /tmp/config.kubeadm

Copy output of join (needed for workers)

e.g.

kubeadm join 159.89.99.35:6443 --token rpylp0.rdphpzbavdyx3llz
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```
### Step 2: Setup worker1 - node (login through ssh)
```

use join command from Step 1:

kubeadm join 159.89.99.35:6443 --token rpylp0.rdphpzbavdyx3llz
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```
### Step 3: Setup worker2 - node (login through ssh)
```

use join command from Step 1:

kubeadm join 159.89.99.35:6443 --token rpylp0.rdphpzbavdyx3llz
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```
### Step 4: Setup worker3 - node (login through ssh)
```

use join command from Step 1:

kubeadm join 159.89.99.35:6443 --token rpylp0.rdphpzbavdyx3llz
--discovery-token-ca-cert-hash sha256:05d42f2c051a974a27577270e09c77602eeec85523b1815378b815b64cb99932

```
### Step 5: CNI-Setup (calico) on controlnode (login through ssh)
```

kubectl get nodes

Output

root@controlplane:~# kubectl get nodes NAME STATUS ROLES AGE VERSION controlplane NotReady control-plane 6m27s v1.28.6 worker1 NotReady 3m18s v1.28.6 worker2 NotReady 2m10s v1.28.6 worker3 NotReady 60s v1.28.6

Installing calico CNI

kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/tigera-operator.yaml kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/custom-resources.yaml kubectl get ns kubectl -n calico-system get all kubectl -n calico-system get pods -o wide -w

After if all pods are up and running -> CTRL + C

```
kubectl -n calico-system get pods -o wide
```

all nodes should be ready now

```
kubectl get nodes -o wide
```

Output

```
root@controlplane:~# kubectl get nodes NAME STATUS ROLES AGE VERSION controlplane Ready control-plane 14m v1.28.6 worker1 Ready 11m v1.28.6 worker2
Ready 10m v1.28.6 worker3 Ready 9m9s v1.28.6
```

```
## 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

```


### Referenz

  * https://kubernetes.io/de/docs/reference/kubectl/cheatsheet/
```

```
### Bauen einer Applikation mit Resource Objekten


![Bauen einer Webanwendung](images/WebApp.drawio.png)

### 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


![Services Aufbau](/images/kubernetes-services.drawio.svg)

### 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 .

```
### Schritt 2b: NodePort
```

02-svc.yml

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

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

kubectl apply -f .

```


### Ref.

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

### DNS - Resolution - Services


### How does it work
```

3 Variants:

svc-name or: svc-name. or: svc-name..svc.cluster.local

```


### Example
```

kubectl run podtest --rm -ti --image busybox If you don't see a command prompt, try pressing enter. / # wget -O - http://apple-service.jochen Connecting to apple-service.jochen (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout / # wget -O - http://apple-service.jochen.svc.cluster.local Connecting to apple-service.jochen.svc.cluster.local (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout / # wget -O - http://apple-service Connecting to apple-service (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout

```

### Hintergrund Ingress




### Ref. / Dokumentation

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

### 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-"

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

```
### 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: ".lab.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

```
### 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


## Kubernetes - Probes

### Überblick Probes


### Welche Probes gibt es ?

  * startup (probe)
  * liveness (probe)
  * readiness (probe)

### Wo werden die Probes definiert ?

  * Die Probes werden immer auf Container-Ebene definiert

#### Liveness Probe

##### Was ist das Standardverhalten (wenn keine Liveness Probe existiert)

  * Es muss ein Prozess mit der id 1 laufen (das ist tatsächlich alles)
```

```
#### Readiness Probe

##### Was ist das Standardverhalten (Es muss ein Prozess mit der id


### Wann brauche ich die start

## Kubernetes - Wartung / Debugging

### 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

```

## Kubernetes Backup

### Backups mit Velero


### Schaubild

![image](https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/c1e1172b-e57f-4c50-a372-ba2f1452ed26)

### Walkthrough in digitalocean

  * https://www.digitalocean.com/community/tutorials/how-to-back-up-and-restore-a-kubernetes-cluster-on-digitalocean-using-velero

## Kubernetes Upgrade

### Upgrade von tanzu (Cluster API)


### Step 1: Upgrade Tanzu Kubernetes Grid (Cluster Api)

  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.6/vmware-tanzu-kubernetes-grid-16/GUID-upgrade-tkg-index.html

### Step 2: Upgrade Management Cluster

  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.6/vmware-tanzu-kubernetes-grid-16/GUID-upgrade-tkg-management-cluster.html

### Step 3: Variante 1: Workload Cluster aktualisieren.

```

```
  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.6/vmware-tanzu-kubernetes-grid-16/GUID-upgrade-tkg-workload-
clusters.html

### Step 3: Variante 2: Neues Cluster hochziehen, ausrollen und altes abschalten

  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.6/vmware-tanzu-kubernetes-grid-16/GUID-tanzu-k8s-clusters-
index.html

## Monitoring with Prometheus / Grafana

### Overview


### 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"

### Kompenenten von Prometheus

![Prometheus Schaubild](https://www.devopsschool.com/blog/wp-content/uploads/2021/01/What-is-Prometheus-Architecutre-
components1-740x414.png)

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

#### Prometheus Server

1. Retrieval (Sammeln)
   * Data Retrieval Worker
     * pull metrics data
1. Storage
   * Time Series Database (TDB)
     * stores metrics data
1. 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

### Setup prometheus/Grafana with helm


### 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.yml
```

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.yml

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 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-w9nrp grafana

hier nochmal port und auch, wie das secret heisst

kubectl describe pods prometheus-grafana-776fb976f7-w9nrp | less

user / pass ?

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

localhost:3000 erreichbarkeit starten -- im Vordergrund

kubectl port-forward deploy/prometheus-grafana 3000

if on remote - system do a ssh-tunnel

ssh -L 3000:127.0.0.1:3000 user@remote-ip

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

```

### Reference:

  * Techworld with Nana: [https://www.youtube.com/watch?v=QoDqxm7ybLc](https://youtu.be/QoDqxm7ybLc?t=190)

### exporters mongodb


### prometheus - export

 * https://github.com/prometheus-community/helm-charts/tree/main/charts/prometheus-mongodb-exporter


### Step 1: mongodb - deployment in mongodb namespace
```

vi mongo-db-deploy.yml

apiVersion: apps/v1 kind: Deployment metadata: name: mongodb-deployment labels: app: mongodb spec: replicas: 1 selector: matchLabels: app: mongodb template: metadata: labels: app: mongodb spec: containers: - name: mongodb image: mongo ports: - containerPort: 27017

apiVersion: v1 kind: Service metadata: name: mongodb-service spec: selector: app: mongodb ports: - protocol: TCP port: 27017 targetPort: 27017

```

```

kubectl apply -f mongo-db-deploy.yml

```


### Step 2: Install prometheus - mongodb - export
```

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts helm repo update helm show values prometheus-community/prometheus-mongodb-exporter > values.yml

adjust so it looks like so:

vi values.yml

[mongodb[+srv]://][user:pass@]host1[:port1][,host2[:port2],...][/database][?options]

mongodb-service is the service name

mongodb: uri: "mongodb://mongodb-service:27017"

serviceMonitor: additionalLabels: release: prometheus

```

```

helm install mongodb-exporter prometheus-community/prometheus-mongodb-exporter -f values.yml

```

### Step 3: Helm -> template -> What does it do ?
```

helm template mongodb-exporter prometheus-community/prometheus-mongodb-exporter -f values.yml

```

### Good Kubernetes Board for Grafana


  * https://github.com/dotdc/grafana-dashboards-kubernetes
  * https://medium.com/@dotdc/a-set-of-modern-grafana-dashboards-for-kubernetes-4b989c72a4b2

## Kubernetes Tipps & Tricks

### kubectl kubeconfig mergen


### So funktioniert es auch bereits:
```

hier werden mehrere kubeconfigs durchsucht

export KUBECONFIG=~/.kube/config:/path/cluster1:/path/cluster2

```

### Jetzt alles in eine Datei
```

cd ~/.kube kubectl config view --flatten > all-in-one-kubeconfig.yaml mv config config.old mv all-in-one-kubeconfig.yaml config

```

### Contexts jeweils anzeigen
```

kubectl config kubectl config use-context mycontext

```

## Kubernetes Certificates (Control Plane) / Security

### vmware - cluster api

  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.6/vmware-tanzu-kubernetes-grid-16/GUID-cluster-lifecycle-secrets.html

### Pod Security Admission (PSA)



### 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. evtl. aktiviert
```

```
### Vorgefertigte Regelwerke

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

  * Reference: https://kubernetes.io/docs/concepts/security/pod-security-standards/

### 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: # soft version - running but showing complaints # pod-security.kubernetes.io/enforce: baseline pod-security.kubernetes.io/enforce: restricted 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

because this image runs as root !! (by default)

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 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

Schritt 4:

Anpassen der Sicherheitseinstellung (Phase1) im Container

apiVersion: v1 kind: Pod metadata: name: nginx namespace: test-ns1 spec: containers: - image: nginx name: nginx ports: - containerPort: 80 securityContext: seccompProfile: type: RuntimeDefault runAsNonRoot: true allowPrivilegeEscalation: false capabilities: drop: ["ALL"]

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

```
### 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

```
### Pod Security Policy (PSP)


### General

  * PodSecurity is an eine Rolle gebunden (clusterrole)
  * Deprecated in 1.21 removed in 1.25
  * From 1.25 on please use PSA (Pod Security Admission) instead

### Prerequisites

  * We should have a running Cluster of 1.22/1.23


### Walkthrough

#### Step 1: Create Digitalocean microk8s 1-node - cluster, with this cloud-init-script

  * cloud-init (ubuntu 20.04 LTS, 8 GB Ram)
```

##!/bin/bash

groupadd sshadmin USERS="11trainingdo" echo $USERS for USER in $USERS do echo "Adding user $USER" useradd -s /bin/bash --create-home $USER usermod -aG sshadmin $USER echo "$USER:deinsehrgeheimespasswort" | chpasswd done

We can sudo with 11trainingdo

usermod -aG sudo 11trainingdo

20.04 and 22.04 this will be in the subfolder

if [-f /etc/ssh/sshd_config.d/50-cloud-init.conf] then sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config.d/50-cloud-init.conf fi

both is needed

sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config

usermod -aG sshadmin root

TBD - Delete AllowUsers Entries with sed

otherwice we cannot login by group

echo "AllowGroups sshadmin" >> /etc/ssh/sshd_config systemctl reload sshd

echo "Installing microk8s" snap install --classic --channel=1.23/stable microk8s microk8s enable dns rbac echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc source ~/.bashrc alias kubectl='microk8s kubectl'

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

```
### Step 2:
```

Setup .kube/config from content

microk8s config

```
### Step 3
```

rbac.yaml

vi service-account.yml

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

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

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

```
### Step 4: Secret aus secrets rauskopiert
```

kubectl get secrets | grep training-token TOKEN=$(kubectl get secrets training-token- -o jsonpath='{.data.token}' | base64 -d)

z.B. TOKEN=$(kubectl get secrets training-token-kjl5m -o jsonpath='{.data.token}' | base64 -d)

```

```

echo $TOKEN kubectl config set-context training-ctx --cluster microk8s-cluster --user training kubectl config set-credentials training --token=$TOKEN

```
### Step 5: Apply yaml-manifests for psp - stuff (as admin)
```

vi setup.yaml

apiVersion: policy/v1beta1 kind: PodSecurityPolicy metadata: name: norootcontainers spec: allowPrivilegeEscalation: false allowedHostPaths: - pathPrefix: /dev/null readOnly: true fsGroup: rule: RunAsAny hostPorts: - max: 65535 min: 0 runAsUser: rule: MustRunAsNonRoot seLinux: rule: RunAsAny supplementalGroups: rule: RunAsAny volumes: - '*'

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: norootcontainers-psp-role rules: - apiGroups: - policy resourceNames: - norootcontainers resources: - podsecuritypolicies verbs: - use

kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata: name: norootcontainers-psp-role:training namespace: default roleRef: kind: ClusterRole name: norootcontainers-psp-role apiGroup: rbac.authorization.k8s.io subjects:

- kind: ServiceAccount name: training namespace: default

```
### Step 5: Change to training-ctx and apply
```

kubectl config use-context training-ctx

vi demopod.yaml

apiVersion: v1 kind: Pod metadata: name: demopod spec: containers: - name: demopod image: nginx

kubectl apply -f demopod.yaml kubectl get pods ## expecting kubectl describe pods demopod

```
### Reference

  * https://docs.mirantis.com/mke/3.4/ops/deploy-apps-k8s/pod-security-policies/psp-examples.html

## Kubernetes Network / Firewall

### 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 -it --image busybox

innerhalb der shell

Verbindung möglich

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

keine Verbindung mehr möglich, weil policy greift

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

### Egress / Ingress Examples with Exercise


### 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 (NetworkPolicy - not ciliumnetworkpolicy)

  * 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 (Exercise)
```

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 . kubectl exec -it nginx-static-web -- bash

im pod

does work

curl -I https://www.google.com

does not work

curl -I https://www.google.de

does not work

curl -I https://www.heise.de

```

#### 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 Schaubild](https://istio.io/latest/docs/examples/virtual-machines/vm-bookinfo.svg)

### 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 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/

### Übung Liveness http aus nginx

### 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

### Manueller Check readyz endpoint kubernetes api server aus pod


### Walkthrough
```

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

im pod

um zu sehen mit welchem Port wir uns verbinden können

env | grep -i kubernetes

kubernetes liegt als service vor

wget -O - https://kubernetes:443/readyz?verbose

```

### Reference:

  * https://kubernetes.io/docs/reference/using-api/health-checks/

## Kubernetes QoS / Limits / Requests

### Quality of Service - evict pods


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

#### Request
```

Request: Definiert wieviel ein Container mindestens braucht (CPU,memory)

```

#### Limit
```

Limit: Definiert, was ein Container maximal braucht.

```

#### Wo ?
```

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"

```

### Referenz

 * https://home.robusta.dev/blog/kubernetes-memory-limit




### Tools to identify LimitRange and Requests


### VPA (Vertical Pod Autoscaler) / goldilocks
```

Please only repo updateMode: "off" will do this

Do not use automatic adjustment

Example VPA configuration apiVersion: autoscaling.k8s.io/v1 kind: VerticalPodAutoscaler metadata: name: my-app-vpa spec: targetRef: apiVersion: "apps/v1" kind: Deployment name: my-app updatePolicy: updateMode: "off"

```

  * goldilocks will now make visible instead of kubectl describe vpa
  * https://github.com/FairwindsOps/goldilocks
  * als Basis: https://github.com/kubernetes/autoscaler/
  * https://www.fairwinds.com/goldilocks


## Kubernetes Autoscaling

### Autoscaling Pods/Deployments


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

#### Prerequisites

  * Metrics-Server needs to be running
```

Test with

kubectl top pods

```

```

Install

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

after that at will be available in kube-system namespace as pod

kubectl -n kube-system get pods | grep -i metrics

```

#### Step 1: deploy app
```

cd mkdir -p manifests cd manifests mkdir hpa cd hpa vi 01-deploy.yaml

```

```

---

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

```
### Step 2: Load Generator
```

vi 02-loadgenerator.yml

apiVersion: apps/v1 kind: Deployment metadata: name: load-generator labels: app: load-generator spec: replicas: 100 selector: matchLabels: app: load-generator template: metadata: name: load-generator labels: app: load-generator spec: containers: - name: load-generator image: busybox command: - /bin/sh - -c - "while true; do wget -q -O- http://hello.default.svc.cluster.local; done"

```
### Downscaling

   * Downscalinng will happen after 5 minutes o
```

Adjust down to 1 minute

apiVersion: autoscaling/v2 kind: HorizontalPodAutoscaler metadata: name: hello spec:

change to 60 secs here

behavior: scaleDown: stabilizationWindowSeconds: 60

end of behaviour change

scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: hello minReplicas: 2 maxReplicas: 20 metrics:

- type: Resource resource: name: cpu target: type: Utilization averageUtilization: 80

For scaling down the stabilization window is 300 seconds (or the value of the --horizontal-pod-autoscaler-downscale-stabilization flag if provided)

```
### Reference

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

## 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: |

Welcome to Version 1

Hi! This is a configmap Index file Version 1

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: |

Welcome to Version 2

Hi! This is a configmap Index file Version 2

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 :

```
## Kubernetes Istio

### Istio vs. Ingress Überblick


![Schaubild](/images/Istio-vs-Ingress-Istio-vs.-IngressController.drawio.png)

### Istio installieren und Addons bereitsstellen


### On the client (where you also use kubectl)

#### Steps 1: Download install and run
```

as tlnx - user

find a decent where to run the installation

not perfect, but better than to put it in home-folder

cd mkdir -p manifests/istio cd manifests/istio

```
```

now download the install an run the shell

curl -L https://istio.io/downloadIstio | sh -

```

### Step 2: Run istioctl - commands (version-check, precheck and install)
```

This istioctl will be under istio-1.20.2/bin

but TRAINER has already installed it under /usr/bin/istioctl

So we can use that one !!

```
```

cd istio-1.20.2/bin

istioctl version istioctl x precheck istioctl install --set profile=demo -y

```

### Step 3: Install the addons
```

Install Add-Ons

kubectl apply -f istio-1.20.2/samples/addons/

```

### Step 4: Check if all the corresponding container (from istio and addons) are running
```

kubectl -n istio-system get pods

```

### Istion Überblick - egress und ingress - gateway


![image](https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/c02c7154-cb9a-4253-8232-6cd125f2862c)

### Istio - Deployment of simple application


### Overview (what we want to do)

![image](https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/285fc65a-57ec-425f-bcd7-729777f79a7d)

  * Catalog Service is reachable through api

### Step 1: Vorbereitung - repo mit beispielen klonen
```

cd git clone https://github.com/jmetzger/istio-exercises/ cd istio-exercises

```

### Step 2: Eigenen Namespace erstellen
```

Jeder Teilnehmer erstellt seinen eigenen Namespace

z.B. istioapp-tlnx

d.h. für Teilnehmer 5 (tln5) -> istioapp-tln5

kubectl create ns istioapp-tln5

Context so einstellen, dass dieser namespace verwendet

kubectl config set-context --current --namespace istioapp-tln5

```
### Step 3: Anwendung untersuchen / istioctl kube-inject

  * Ihr könnt unten direkt den Pfad nehmen, das ist einfacher ;o)
```

apiVersion: v1 kind: ServiceAccount metadata: name: catalog

apiVersion: v1 kind: Service metadata: labels: app: catalog name: catalog spec: ports: - name: http port: 80 protocol: TCP targetPort: 3000 selector: app: catalog

apiVersion: apps/v1 kind: Deployment metadata: labels: app: catalog version: v1 name: catalog spec: replicas: 1 selector: matchLabels: app: catalog version: v1 template: metadata: labels: app: catalog version: v1 spec: serviceAccountName: catalog containers: - env: - name: KUBERNETES_NAMESPACE valueFrom: fieldRef: fieldPath: metadata.namespace image: istioinaction/catalog:latest imagePullPolicy: IfNotPresent name: catalog ports: - containerPort: 3000 name: http protocol: TCP securityContext: privileged: false

```
```

schauen wir uns das mal mit injection an

istioctl kube-inject -f services/catalog/kubernetes/catalog.yaml | less

```
### Step 4: Automatische Injection einrichten.
```

kubectl label namespace istioapp-tlnx istio-injection=enabled

z.B

kubectl label namespace istioapp-tln1 istio-injection=enabled

```
### Step 5: catalog ausrollen
```

kubectl apply -f services/catalog/kubernetes/catalog.yaml

```
```

Prüfen, ob wirklich 2 container in einem pod laufen,

dann funktioniert die Injection

WORKS, Yeah !

kubectl get pods

```
### Step 6: Wir wollen den Catalog jetzt erreichen
```

do it from your namespace, e.g. tlnx

z.B.

kubectl -n tln1 run -it --rm curly --image=curlimages/curl -- sh

```
```

within shell of that pod

catalog.yourappnamespace/items/1

curl http://catalog.istioapp-tln1/items/1 exit

```
### Step 7: Jetzt deployen wir die webapp
```

Wir schauen uns das manifest für die webapp an

und ändern die env-variablen CATALOG_SERVICE_HOST

tlnx durch Eure Teilnehmernummer ersetzen

catalog.istioapp-tlnx

kubectl apply -f services/webapp/kubernetes/webapp.yaml kubectl get pod

```
### Step 8: Verbindung zu webapp testen
```

tlnx

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

z.B.

kubectl -n tln5 run -it --rm curly --image=curlimages/curl -- sh

Within shell connect to webapp

curl -s http://webapp.istioapp-tln1/api/catalog/items/1 exit

Wir können es aber auch visualisieren

kubectl port-forward deploy/webapp 8001:8080

z.B. Teilnehmer tln1 -> 8001:8080

WICHTIG Jeder Teilneher sollte hier einen abweichenden Port nehmen

Jetzt lokal noch einen Tunnel aufbauen

s. Anleitung Putty

Source Port: 8080 # das ist der auf dem Rechner

Destination: localhost:8001

Add

Achtung -> danach noch Session speichern

Jetzt im Browser http://localhost:8080

aufrufen

```
### Step 9: Ingress - Gateway konfigurieren (ähnlich wie Ingress-Objekt)
```

wir schauen uns das vorher mal an

namespace - fähig, d.h. ein Gateway mit gleichem Namen pro Namespace möglich

cat ingress-virtualservice/ingress-gateway.yaml

hier bitte bei Hosts hostname eintragen, der für t3isp.de verwendet, und zwar

jeder Teilnehmer eine eigene Subdomain: z.B. jochen.istio.t3isp.de

kubectl apply -f ingress-virtualservice/ingress-gateway.yaml

```
### Step 10: Reach it from outside
```

We need to find the loadbalancer IP

kubectl -n istio-system get svc

in unserem Fall

146.190.177.12

Das trägt Jochen dns t3isp.de ein.

Wir können jetzt also das System von extern erreichen

vomn client aus, oder direkt über den Browser

##curl -i 146.190.177.12/api/catalog/items/1

Hier hostname statt ip einträgen

curl -i http://tlnx.istio.t3isp.de/api/catalog/items/1

Wir können auch über istioctl direkt überprüfen, ob es einen Routen-Config gibt

istioctl proxy-config routes deploy/istio-ingressgateway.istio-system

Falls das nicht funktioniert, können wir auch überprüfen ob ein gateway und ein virtualservice installiert wurde

kubectl get gateway kubectl get virtualservice

Kurzform des Services reicht, weil im gleichen namespace

Wo soll es hingehen -> == -> Upstream

route -> destination -> host -> webapp

kubectl get virtualservice -o yaml

Wichtiger Hinweis, auf beiden Seiten ingressgateway und vor dem Pod des Dienstes Webapp

Sitzt ein envoy-proxy und kann Telemetrie-Daten und Insight sammeln was zwischen den

applicationen passiert -> das passiert über ein sidecar in jeder Applikation

Wichtig: Das passiert alles ausserhalb der Applikation

Nicht wie früher z.B. bei Netflix innerhalb z.B. für die Sprache Java

```
### Istio - Grafana Dashboard


### Status

  * Wir haben bereits mit den Addons Grafana ausgerollt,
  * Dieses wollen wir jetzt aktivieren

### Schritt 1: Dashboard aktivieren -> achtung jeder nimmt seinen eigenen Port
```

um Grunde macht das auch nur ein port - forward

Das macht der Trainer nur 1x, dann können alle dort zugreifen

istioctl dashboard grafana --port=3000 --browser=false

Jetzt über den Browser öffnen

http://localhost:3000

Dann Dashboard -> istio -> istio services

Lass uns mal Traffic hinschicken vom Client aus

ip vom ingressgateway from loadBalancer

while true; do curl http://jochen.istio.t3isp.de/api/catalog; sleep .5; done

Und das das Dashboard nochmal refreshend

##-> General ausklappen

```

## 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 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: # 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

![Overview](https://www.inovex.de/wp-content/uploads/2020/05/Container-to-Container-Networking_2_neu-400x401.png)
![Overview Kubernetes Networking](https://www.inovex.de/wp-content/uploads/2020/05/Container-to-Container-Networking_3_neu-
400x412.png)

#### 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

![pod to pod across nodes](https://www.inovex.de/wp-content/uploads/2020/05/Pod-to-Pod-Networking.png)


#### 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.

```
### 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

## 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 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

### Curl from pod api-server


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

## Kubernetes Praxis API-Objekte

### 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

### 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

### 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 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 .

```
## 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: ClusterRole 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", "create"]

```

```

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 (ab Kubernetes-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

```
## 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

exit

quick and dirty

kubectl run -it --rm --image=busybox busytester -- wget

```

### Debugging pods


### How ?

  1. Which pod is in charge
  1. Problems when starting: kubectl describe po mypod
  1. 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://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/)
  * https://blog.kubecost.com/blog/kubernetes-taints/

### 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. evtl. aktiviert

### Vorgefertigte Regelwerke

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

  * Reference: https://kubernetes.io/docs/concepts/security/pod-security-standards/

### 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: # soft version - running but showing complaints # pod-security.kubernetes.io/enforce: baseline pod-security.kubernetes.io/enforce: restricted 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

because this image runs as root !! (by default)

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 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

Schritt 4:

Anpassen der Sicherheitseinstellung (Phase1) im Container

apiVersion: v1 kind: Pod metadata: name: nginx namespace: test-ns1 spec: containers: - image: nginx name: nginx ports: - containerPort: 80 securityContext: seccompProfile: type: RuntimeDefault runAsNonRoot: true allowPrivilegeEscalation: false capabilities: drop: ["ALL"]

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

```

### 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)

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:$6$HeLUJW3a$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:$6$HeLUJW3a$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