Kubernetes und Docker Administration und Orchestrierung

Agenda

1. Kubernetes Grundlagen
    - Allgemeine Einführung in Container (Dev/Ops)
    - Warum Kubernetes ? (Devs/Ops)
    - Die Struktur von Kubernetes mit seinen Komponenten (Devs/Ops)
    - Umdenken in der Administration (feste Server vs. Dienste im Cluster) (Ops)
    - Api Versionierung Lifetime
2. Kubernetes Kickoff
    - Orchestrierung (Warum und wozu ?) (Devs/Ops)
    - Microservices (Warum ? Wie ?) (Devs/Ops)
    - Hochverfügbarkeit (Wie funktioniert das ?) (Ops)
    - Vorstellung Management - Tools zum Aufsetzen eines Cluster (microk8s,kubeadm,Rancher) (Ops)
3. Kubernetes Praxis API-Objekte
    - Das Tool kubectl (Devs/Ops)
    - kubectl example with run
    - Arbeiten mit manifests (Devs/Ops)
    - Pods (Devs/Ops)
    - kubectl/manifest/pod
    - ReplicaSets (Theorie) - (Devs/Ops)
    - kubectl/manifest/replicaset
    - Deployments (Devs/Ops)
    - kubectl/manifest/deployments
    - Services (Devs/Ops)
    - kubectl/manifest/service
    - DaemonSets (Devs/Ops)
    - IngressController (Devs/Ops)
    - Hintergrund Ingress
    - Documentation for default ingress nginx
    - Beispiel mit Hostnamen
4. Kubernetes Praxis Scaling/Rolling Updates/Wartung
    - Rolling Updates (Devs/Ops)
    - Scaling von Deployments (Devs/Ops)
    - Wartung mit drain / uncordon (Ops)
    - Ausblick AutoScaling (Ops)
5. Kubernetes Storage
    - Grundlagen (Dev/Ops)
    - Objekte PersistantVolume / PersistantVolumeClaim (Dev/Ops)
    - Praxis. Beispiel (Dev/Ops)
6. Kubernetes Networking

    - Überblick
    - Pod to Pod
    - Webbasierte Dienste (Ingress)
    - IP per Pod
    - Inter Pod Communication ClusterDNS
    - Beispiel NetworkPolicies
7. Kubernetes Paketmanagement (Helm)
    - Warum ? (Dev/Ops)
    - Grundlagen / Aufbau / Verwendung (Dev/Ops)
    - Praktisches Beispiel bitnami/mysql (Dev/Ops)
8. Kubernetes Rechteverwaltung (RBAC)
    - Warum ? (Ops)
    - Rollen und Rollenzuordnung (Ops)
    - Service Accounts (Ops)
    - Praktische Umsetzung anhand eines Beispiels (Ops)
9. Kubernetes Monitoring
    - Protokollieren mit Elasticsearch und Fluentd (Devs/Ops)
    - Long Installation step-by-step - Digitalocean
    - Container Level Monitoring (Devs/Ops)
    - Working with kubectl logs
    - Setting up metrics-server - microk8s
    - Prometheus/cAdvisor (Devs/Ops)
    - InfluxDB (Ops)
10. Kubernetes CI/CD (Optional)
    - Canary Deployment (Devs/Ops)
    - Blue Green Deployment (Devs/Ops)
    - A/B Testing (Devs/Ops)
11. Tipps & Tricks
    - bash-completion
    - kubectl spickzettel
    - Alte manifests migrieren
12. Fragen
    - Q and A

Backlog

1. Kubernetes - microk8s (Installation und Management)
    - Patch to next major release - cluster
    - Installation Kuberenetes Dashboard
2. Kubernetes - API - Objekte
    - Was sind Deployments
    - Service - Objekt und IP
3. Kubernetes - Netzwerk (CNI's)

- Übersicht Netzwerke
- Callico - nginx example
- Callico - client-backend-ui-example

4. kubectl
    - Tipps&Tricks zu Deploymnent - Rollout
5. kubectl - manifest - examples
    - 05 Ingress mit Permanent Redirect
6. Kubernetes - Monitoring (microk8s und vanilla)
    - metrics-server aktivieren (microk8s und vanilla)
7. Kubernetes - Tipps & Tricks
    - Assigning Pods to Nodes
8. Linux und Docker Tipps & Tricks allgemein
    - vim einrückung für yaml-dateien
    - YAML Linter Online

Kubernetes Grundlagen

Allgemeine Einführung in Container (Dev/Ops)

Architektur

 Docker Architecture - copyright geekflare

Was sind Docker Images

- Docker Image benötigt, um zur Laufzeit Container-Instanzen zu erzeugen
- Bei Docker werden Docker Images zu Docker Containern, wenn Sie auf einer Docker Engine als Prozess ausgeführt
- Man kann sich ein Docker Image als Kopiervorlage vorstellen.
    - Diese wird genutzt, um damit einen Docker Container als Kopie zu erstellen

Was sind Docker Container ?

```
- vereint in sich Software
- Bibliotheken
- Tools
- Konfigurationsdateien
- keinen eigenen Kernel
- gut zum Ausführen von Anwendungen auf verschiedenen Umgebungen

### Weil :
- Container sind entkoppelt
- Container sind voneinander unabhängig
- Können über wohldefinierte Kommunikationskanäle untereinander Informationen
austauschen

- Durch Entkopplung von Containern:
  o Unverträglichkeiten von Bibliotheken, Tools oder Datenbank können umgangen werden,
wenn diese von den Applikationen in unterschiedlichen Versionen benötigt werden.
```

Container vs. VM

```
VM's virtualisieren Hardware
Container virtualisieren Betriebssystem
```

Dockerfile

- Textdatei, die Linux - Kommandos enthält
    - die man auch auf der Kommandozeile ausführen könnte
    - Diese erledigen alle Aufgaben, die nötig sind, um ein Image zusammenzustellen
    - mit docker build wird dieses image erstellt

Einfaches Beispiel eines Dockerfiles

FROM nginx:latest COPY html /usr/share/nginx/html

Komplexeres Beispiel eines Dockerfiles

- https://github.com/StefanScherer/whoami/blob/main/Dockerfile

Warum Kubernetes ? (Devs/Ops)

Ausgangslage

- Ich habe jetzt einen Haufen Container, aber:
  - Wie bekomme ich die auf die Systeme.
  - Und wie halte ich den Verwaltungsaufwand in Grenzen.
- Lösung: Kubernetes -> ein Orchestrierungstool

Hintergründe

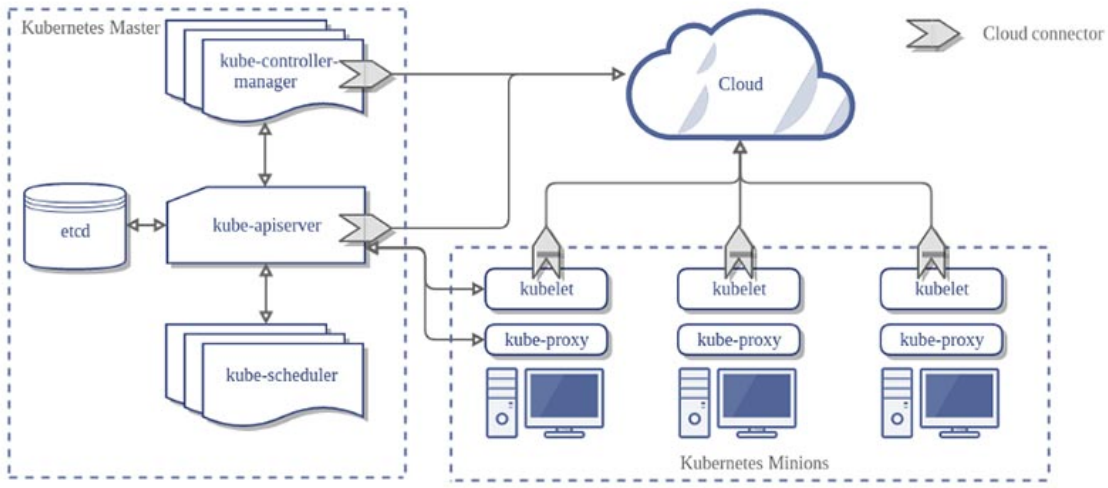
- Virtualisierung von Hardware - 5fache bessere Auslastung
- Google als Ausgangspunkt
- Software 2014 als OpenSource zur Verfügung gestellt
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbstheilend

Wozu dient Kubernetes

- Orchestrierung von Containern
- am gebräuchlisten aktuell Docker

Die Struktur von Kubernetes mit seinen Komponenten (Devs/Ops)

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

Umdenken in der Administration (feste Server vs. Dienste im Cluster) (Ops)

Vorher (old-school) - Imperativ

- Ich setze Server auf
- Auf dem Server läuft eine Anwendung

Jetzt (Kubernetes) - Declarative

- Ich definiere, wieviele Nginx - Server laufen sollen (Beispiel)
- Überlasse Kubernetes, wo diese laufen
- Kubernetes entscheidend anhand der Ressourcen
- Ich kann aber contraints festlegen (d.h. ich sage, nur in bestimmten Rechenzentrum)

Was ist anders ?

- Ich weiss nicht genau, auf welchem node ein container(pod) läuft

Was ist die Konsequenz

- Logs müssen anders ausgewertet werden (Logs sammeln)
  - innerhalb der Container
  - cluster-wide
  - einzelne Nodes
- Backup (Wie lasse ich backups laufen bzw. führe diese durch)
  - Kubernetes aware backup (solution), e.g. kasten.io
- Havarie (wie setze ich das ganze wieder auf - worst case)

Wo muss mich strecken als Admin

- Aufbau von Vertrauen auf Kubernetes (Vertrauen darauf, dass Kubernetes das in meinem Sinne macht)

Sicherheitsaspekt (Server vs. Kubernetes)

- Komplexität und Durchschaubarkeit steigt, weil
  - kann keine einfachen Firewall - Regeln mehr machen
  - Was macht Kubernetes auf ? (Port)
  - Läuft vielleicht ein Ingress-Objekct, was mein System aufmacht (ungewollt)

Api Versionierung Lifetime

Wie ist die deprecation policy ?

- https://kubernetes.io/docs/reference/using-api/deprecation-policy/

Was ist wann deprecated ?

- https://kubernetes.io/docs/reference/using-api/deprecation-guide/

Reference:

- https://kubernetes.io/docs/reference/using-api/

Kubernetes Kickoff

Orchestrierung (Warum und wozu ?) (Devs/Ops)

Was ist das ?

- Ein System, was mir hilft mein Container zu verwalten (Kubernetes - Cluster)
- Beschreibt, welche Container/Pods in welcher Zahl wie laufen sollen ?

Beispiel eines einfaches Orchestrierungstool

- docker-compose
- Als Configuration für mehere Container - 1 yaml-file docker-compose.yml

Vorteile

- Weniger Adminsitrationsaufwand
- Verläßliche Bereitsstellung (Selbstheilung)
- Gute Skalierbarkeit (rein konfigurativ)

Nachteile (Kubernetes)

- Ich habe es weniger in der Hand, was genau passiert.

Microservices (Warum ? Wie ?) (Devs/Ops)

Was soll das ?

```
Ein mini-dienst, soll das minimale leisten, d.h. nur das wofür er da ist.

-> z.B. Webserver
oder Datenbank-Server
oder Dienst, der nur reports erstellt
```

Wie erfolgt die Zusammenarbeit

```
Otchestrierung (im Rahmen der Orchestierung über vorgefertigte Schnittstellen, d.h.
auch feststehende Benamung)
- Label
```

Vorteile

```
##
Leichtere Updates von Microservices, weil sie nur einen kleinere Funktionalität
```

Nachteile

```
* Komplexität
  * z.B. in Bezug auf Debugging
  * Logging / Backups
```

Hochverfügbarkeit (Wie funktioniert das ?) (Ops)

Administration - Elemente, die hochverfügbar gemacht werden können.

- Control-Plane
- etcd (mehrmals)

Applikationen

- Weil sie auf mehreren Nodes laufen
- Weil kubernetes sie verschiebt, wenn ein node ausfällt.

Vorstellung Management - Tools zum Aufsetzen eines Cluster (microk8s,kubeadm,Rancher) (Ops)

Hintergrund

- Um ein Cluster einzurichten, muss ich mich für ein Tool entscheiden.

kubeadm

- Most vanilla - Tool (d.h. am komplexesten)
- Das erste Tool, dass es zum Einrichten eines Clusters gab.

microk8s (ubuntu)

- Einfach zu bedienen
- Reines commandline - tool (microk8s status)
- Bietet plugins, die bestimmte features an und abschalten (Ingress,Metrics-Server) (microk8s enable ingress)
    - Dieses Features, die im Hintergrund manifeste ausführen, können so einfach konfiguriert werden
- Es läßt sich sehr einfach ein Cluster aufbauen (3 Server hochziehen mit microk8s und verheiraten (node2, node3)

Rancher

- Von der 3 Varianten das komfortabelste Tool
- Bietet einen gui um ein Cluster einzurichten

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops)

Allgemein

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

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

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

namespaces

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

Arbeiten mit manifesten

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

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

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

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

Ausgabeformate / Spezielle Informationen

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

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

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

Zu den Pods

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

## Pods anzeigen
kubectl get pods
kubectl get pod

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

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

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

## Status eines Pods anzeigen
```

```
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

Arbeiten mit namespaces

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

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

Alle Objekte anzeigen

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

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

Referenz

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

kubectl example with run

Example (that does work)

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

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

Example (that does not work)

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


### Ref:

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

### kubectl/manifest/pod


### Walkthrough
```

vi nginx-static.yml

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

- name: web image: nginx

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

show config

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

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

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

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

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

vi nginx-deployment.yml

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

kubectl apply -f nginx-deployment.yml

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

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

---

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

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

```
### Ref.

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

### Hintergrund Ingress




### Ref. / Dokumentation

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

### Documentation for default ingress nginx

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

### Beispiel mit Hostnamen


### Prerequisits
```

Ingress Controller muss aktiviert sein

microk8s enable ingress

```
### Walkthrough
```

mkdir apple-banana-ingress

cd apple-banana-ingress

apple.yml

vi apple.yml

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

---

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

kubectl apply -f apple.yml

banana

vi banana.yml

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

---

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

kubectl apply -f banana.yml

Ingress

apiVersion: extensions/v1beta1 kind: Ingress metadata: name: example-ingress annotations: ingress.kubernetes.io/rewrite-target: / spec: rules:

- host: "app12.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: /
spec:
  rules:
  - host: "app12.lab.t3isp.de"
    http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

Kubernetes Praxis Scaling/Rolling Updates/Wartung

Wartung mit drain / uncordon (Ops)

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

## Alle pods von 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
```

Kubernetes Storage

Praxis. Beispiel (Dev/Ops)

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 clients

```
#### Please do this on all servers

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

Setup PersistentVolume and PersistentVolumeClaim in cluster

```
## mkdir -p nfs; cd nfs
## vi 01-pv.yml
## Important user
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln1
  labels:
    volume: nfs-data-volume-tln1
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/tln1/nginx
    server: 192.168.56.106
    readOnly: false
  storageClassName: ""
```

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

```
## vi 02-pvs.yml
## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim-tln1
spec:
  storageClassName: ""
  volumeName: pv-nfs-tln1
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

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

```
## deployment including mount
## vi 03-deploy.yml
```

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

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

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

      volumes:
      - name: nfsvol
        persistentVolumeClaim:
          claimName: pv-nfs-claim-tln1
```

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

```
## now testing it with a service
## cat 04-service.yml
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
  labels:
    run: svc-my-nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

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

## connect to the container and add index.html - data
```

```
kubectl exec -it deploy/nginx-deployment -- bash
## in container
echo "hello dear friend" > /usr/share/nginx/html/index.html
exit

## now try to connect
kubectl get svc

## connect with ip and port
curl http://<cluster-ip>:<port> # port -> > 30000

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

## Try again - no connection
curl http://<cluster-ip>:<port> # port -> > 30000

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

## and try connection again
curl http://<cluster-ip>:<port> # port -> > 30000
```

Kubernetes Networking

Überblick

CNI

- Common Network Interface
- Fest 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

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

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.
```

Beispiel NetworkPolicies

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

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

```
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo
## mkdir network; cd network
## vi 01-policy.yml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
```

```
metadata:
  name: default-deny
  namespace: policy-demo-<tln>
spec:
  podSelector:
    matchLabels: {}
```

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

```
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo access --rm -ti --image busybox /bin/sh
```

```
## innerhalb der shell
## kein Zugriff möglich
wget -q nginx -O -
```

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

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

```
## 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<tln> access --rm -ti --image busybox /bin/sh
```

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

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

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

```

kubectl delete ns policy-demo
```

Ref:

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

Kubernetes Paketmanagement (Helm)

Warum ? (Dev/Ops)

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

Praktisches Beispiel bitnami/mysql (Dev/Ops)

Prerequisites

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

Example 1: 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 1 - continue - fehlerbehebung

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

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

Referenced

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

Kubernetes Rechteverwaltung (RBAC)

Praktische Umsetzung anhand eines Beispiels (Ops)

Enable RBAC in microk8s

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

Wichtig:

```
Jeder verwendet seine eigene teilnehmer-nr z.B.
training1
training2
usw. ;o)
```

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<nr> # <nr> entsprechend eintragen
  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-<nr> # für <nr> teilnehmer - nr eintragen
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-<nr> # <nr> durch teilnehmer nr ersetzen
subjects:
- kind: ServiceAccount
  name: training<nr> # nr durch teilnehmer - nr ersetzen
  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<nr>
# nr durch teilnehmer - nr ersetzen
```

Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen

Mini-Schritt 1: kubeconfig setzen

```
kubectl config set-context training-ctx --cluster microk8s-cluster --user training<nr>
# <nr> durch teilnehmer - nr ersetzen

## extract name of the token from here
TOKEN_NAME=`kubectl get serviceaccount training<nr> -o jsonpath='{.secrets[0].name}'`
# nr durch teilnehmer <nr> ersetzen

TOKEN=`kubectl get secret $TOKEN_NAME -o jsonpath='{.data.token}' | base64 --decode`
echo $TOKEN
kubectl config set-credentials training<nr> --token=$TOKEN # <nr> druch teilnehmer -
nr ersetzen
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

Kubernetes Monitoring

Long Installation step-by-step - Digitalocean

- https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes

Working with kubectl logs

Setting up metrics-server - microk8s

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 CI/CD (Optional)

Tipps & Tricks

bash-completion

Walkthrough

```
apt install bash-completion
source /usr/share/bash-completion/bash_completion
## is it installed properly
type _init_completion

## activate for all users
```

```
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

## verifizieren - neue login shell
su -

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

Alternative für k als alias für kubectl

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

Reference

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

kubectl spickzettel

Allgemein

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

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

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

namespaces

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

Arbeiten mit manifesten

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

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

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

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

Ausgabeformate / Spezielle Informationen

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

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

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

Zu den Pods

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

## Pods anzeigen
kubectl get pods
kubectl get pod

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

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

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

## Status eines Pods anzeigen
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

Arbeiten mit namespaces

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

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

Alle Objekte anzeigen

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

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

Referenz

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

Alte manifests migrieren

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

Fragen

Q and A

Wieviele Replicaset beim Deployment zurückbehalt / Löschen von Replicaset

```
kubectl explain deployment.spec.revisionHistoryLimit

apiVersion: apps/v1
kind: Deployment
## ...
spec:
  # ...
  revisionHistoryLimit: 0 # Default to 10 if not specified
  # ...
```

Wo dokumentieren, z.B. aus welchem Repo / git

```
Labels can be used to select objects and to find collections of objects that satisfy
certain conditions. In contrast, annotations are not used to identify and select
objects.
```

- https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/
- https://kubernetes.io/docs/reference/labels-annotations-taints/

Kubernetes - microk8s (Installation und Management)

Patch to next major release - cluster

Installation Kuberenetes Dashboard

Reference:

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

Kubernetes - API - Objekte

Was sind Deployments

Hierarchy

```
deployment
  replicaset
    pod


Deployment :: create a new replicaset, when needed (e.g. new version of image comes
out)
Replicaset :: manage the state - take care, that the are always x-pods running (e.g.
3)
Pod :: create the containers
```

What are deployments

- Help to manage updates of pods / replicaset (rolling update)

Example

```
## Deploy a sample from k8s.io
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
```

Refs:

- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Service - Objekt und IP

Was ?

```
Stellt eine Netzwerkverbindung zu verschiedenen Pods her,
auf Basis eines Labels
```

Warum ?

```
service (-controller) überprüft welche Nodes mit entsprechenden
Label zur Verfügung stehen und übernimmt das Routing

standardmäßig: round robin
```

What are services ?

- Services help you to connect to the pods seemlessly
- Service knows which pods are available

service - types

```
The type defines how the connection is done (what kind of network/ip/port is provided
to connect to the service

ClusterIP
NodePort
LoadBalancer - an external balancer is used (that is mainly the case in
```

Reference:

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

Kubernetes - Netzwerk (CNI's)

Übersicht Netzwerke

CNI

- Common Network Interface
- Fest 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

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

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.
```

Callico - nginx example

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

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

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

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

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

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

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

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

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

```

kubectl delete ns policy-demo
```

Ref:

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

Callico - client-backend-ui-example

Walkthrough

```
cd
mkdir -p manifests/callico/example1
cd manifests/callico/example1
```

```
### Step 1: Create containers

kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/00-namespace.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/01-management-ui.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/02-backend.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/03-frontend.yaml
kubectl create -f https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/manifests/04-client.yaml

kubectl get pods --all-namespaces --watch
kubectl get ns
```

```
### Step 2: Check connections in the browser (ui)
### Use IP of one of your nodes here
http://164.92.255.234:30002/
```

```
### Step 3: Download default-deny rules
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-demo/policies/default-deny.yaml
### Let us have look into it
### Deny all pods
cat default-deny.yaml
### Apply this for 2 namespaces created in Step 1
kubectl -n client apply -f default-deny.yaml
kubectl -n stars apply -f default-deny.yaml
```

```
### Step 4: Refresh UI and see, that there are no connections possilbe
http://164.92.255.234:30002/
```

```
### Step 5:
### Allow traffic by policy
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-
demo/policies/allow-ui.yaml
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-
demo/policies/allow-ui-client.yaml
### Let us look into this:
cat allow-ui.yaml
cat allow-ui-client.yaml
kubectl apply -f allow-ui.yaml
kubectl apply -f allow-ui-client.yaml
```

```
### Step 6:
### Refresh management ui
### Now all traffic is allowed
http://164.92.255.234:30002/
```

```
### Step 7:
### Restrict traffic to backend
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-
demo/policies/backend-policy.yaml
cat backend-policy.yaml
kubectl apply -f backend-policy.yaml
```

```
### Step 8:
### Refresh
## The frontend can now access the backend (on TCP port 6379 only).
## The backend cannot access the frontend at all.
## The client cannot access the frontend, nor can it access the backend
http://164.92.255.234:30002/
```

```
### Step 9:
wget https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-
demo/policies/frontend-policy.yaml
cat frontend-policy.yaml
kubectl apply -f frontend-policy.yaml
```

```
### Step 10:
## Refresh ui
## Client can now access Frontend
http://164.92.255.234:30002/
```

```
## Alles wieder löschen
kubectl delete ns client stars management-ui
```

Reference

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

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
```

kubectl - manifest - examples

05 Ingress mit Permanent Redirect

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
-

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 - 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/

Linux und Docker Tipps & Tricks allgemein

vim einrückung für yaml-dateien

Ubuntu (im Unterverzeichnis /etc/vim - systemweit)

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

Testen

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

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

YAML Linter Online

- http://www.yamllint.com/