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) - Spickzettel
    - 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)
    - Wichtige Befehle
8. Kubernetes Rechteverwaltung (RBAC)
    - Warum ? (Ops)
    - Rollen und Rollenzuordnung (Ops)
    - Service Accounts (Ops)
    - Praktische Umsetzung anhand eines Beispiels (Ops)
9. Kubernetes Monitoring
    - Ebenen des Loggings
    - Working with kubectl logs
    - Built-In Monitoring tools - kubectl top pods/nodes
    - Protokollieren mit Elasticsearch und Fluentd (Devs/Ops)
    - Long Installation step-by-step - Digitalocean
    - Container Level Monitoring (Devs/Ops)
    - Setting up metrics-server - microk8s
    - Installation prometheus auf DigitalOcean Kubernetes mit helm
    - Prometheus/cAdvisor (Devs/Ops) - Überblick
    - InfluxDB (Ops)
10. Kubernetes Security
    - Grundlagen und Beispiel (Praktisch)
11. Kustomize
    - Kustomize Overlay Beispiel
    - Helm mit kustomize verheiraten
12. Kubernetes CI/CD (Optional)
    - Canary Deployment (Devs/Ops)
    - Blue Green Deployment (Devs/Ops)
    - A/B Testing (Devs/Ops)
13. Tipps & Tricks
    - bash-completion
    - kubectl spickzettel
    - Alte manifests migrieren
    - Pod - Verbindung debuggen
    - Übung mit sealed-secrets
    - Config für nginx mit configmap einhängen
14. Fragen

- Q and A

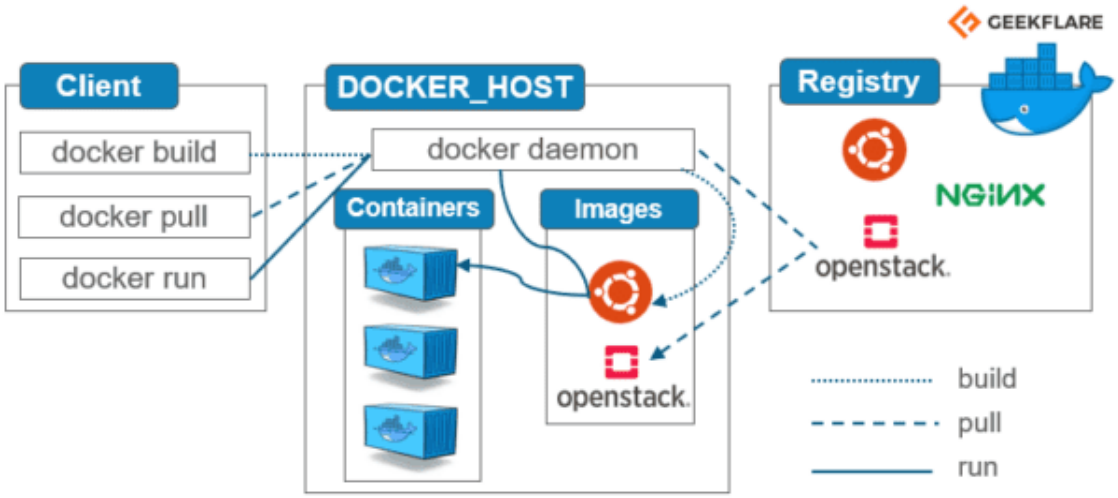
Backlog

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



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

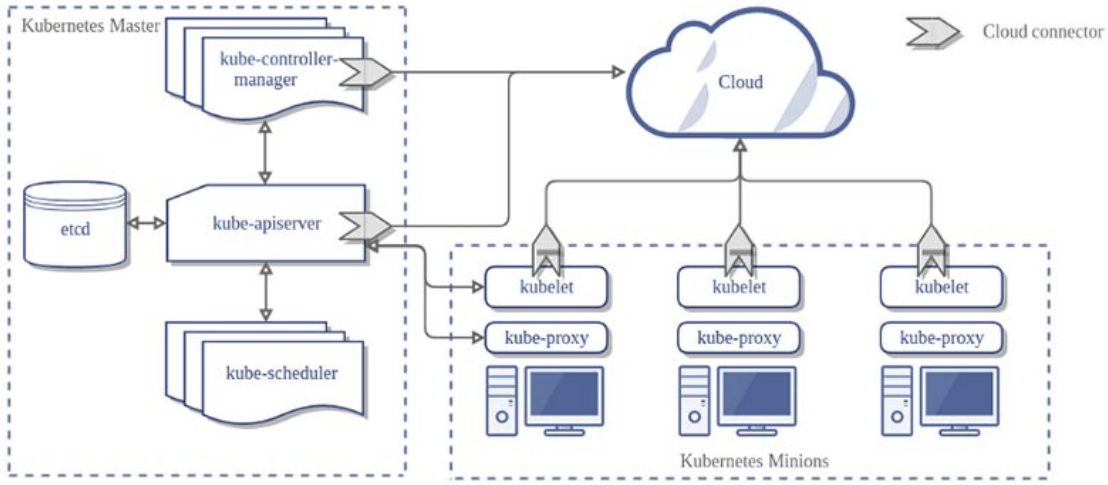
- Virtualisierung von Hardware - 5fache bessere Auslastung
- Google als Ausgangspunkt
- Software 2014 als OpenSource zur Verfügung gestellt
- Optimale Ausnutzung der Hardware, hunderte bis tausende Dienste können auf einigen Maschinen laufen (Cluster)
- Immutable - System
- Selbstheilend

Wozu dient Kubernetes

- Orchestrierung von Containern
- am gebräuchlichsten 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 ich mich strecken als Admin

- Aufbau von Vertrauen in 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 / Rancher Desktop

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

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops) - Spickzettel

Per Default mit namespace namespace1 arbeiten

```
## namespace muss vorhanden sein, evtl anlegen
kubectl create ns namespace1
kubectl config set-context --current --namespace=namespace1
```

Hilfe zu einem bestimmten Befehl

```
## z.B. Hilfe zu config
kubectl help config
```

Hilfe zu Objekten

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

Allgemein

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

## Welche api-resources gibt es ?
kubectl api-resources
kubectl api-resources | grep namespaces
## Gross- und Kleinschreibung egal
kubectl api-resources | grep -i ingress
```

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


### In den Container / Pod reingehen
```

Kommando in pod ausführen

kubectl exec -it nginx -- bash

bei deployment hineinwechseln in einen beliebigen pod

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

```
### Arbeiten mit namespaces
```

Welche namespaces auf dem System

kubectl get ns kubectl get namespaces

Standardmäßig wird immer der default namespace verwendet

wenn man kommandos aufruft

kubectl get deployments

Möchte ich z.B. deployment vom kube-system (installation) aufrufen,

kann ich den namespace angeben

kubectl get deployments --namespace=kube-system kubectl get deployments -n kube-system

```
### Alle Objekte anzeigen
```

Manchen Objekte werden mit all angezeigt

kubectl get all kubectl get all,configmaps

Über alle Namespaces hinweg

kubectl get all -A

```
### Logs
```

kubectl logs kubectl logs

e.g.

kubectl logs -n namespace8 deploy/nginx

with timestamp

kubectl logs --timestamp -n namespace8 deploy/nginx

continously show output

kubectl logs -f

```
### 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
```

cd; mkdir manifests; cd manifests

mkdir web; cd web;

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
```

cd; mkdir -p manifests/rs; cd manifests/rs

nano 01-rs.yml

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

kubectl apply -f . kubectl get rs -o wide kubectl get po -o wide

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

cd; mkdir -p manifests/deploy; cd manifests/deploy

nano 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 get deploy; kubectl get rs; kubectl get po

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

cd; mkdir -p manifests/service; cd manifests/service

nano 01-nginx.yml

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: ClusterIP ports:

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

```
kubectl apply -f .
kubectl get deploy; kubectl get po -o wide; kubectl get service -o wide
```

Ref.

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

IngressController (Devs/Ops)

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 und Verbindung zum Cluster haben (kubectl cluster-info)

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
*.lab3.t3isp.de A 188.166.195.238
```

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

```
## cd; mkdir -p manifests/abc; cd manifests/abc

## 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-tln<tln>"
---

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-tln<tln>"

---

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:
  ingressClassName: nginx
  rules:
  - host: "app<tln>.lab3.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:
  ingressClassName: nginx
  rules:
  - host: "app<tln>.lab3.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
```

Ausblick AutoScaling (Ops)

Example:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
    name: busybox-1
spec:
    scaleTargetRef:
        kind: Deployment
        name: busybox-1
    minReplicas: 3
    maxReplicas: 4
    targetCPUUtilizationPercentage: 80
```

Reference

- https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054

Kubernetes Storage

Praxis. Beispiel (Dev/Ops)

Create new server and install nfs-server (if not existent)

```
## 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 (only on self-hosted kubernetes)

```
#### 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

```
## cd; mkdir -p manifests/nfs; cd manifests/nfs
## vi 01-pv.yml
## Important user
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln<tln>
  labels:
    volume: nfs-data-volume-tln<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<tln>/nginx
    server: 10.135.0.5
    readOnly: false
  storageClassName: ""
```

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

```
## vi 02-pvs.yml
## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim-tln<tln>
spec:
  storageClassName: ""
  volumeName: pv-nfs-tln<tln>
  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-tln<tln>
```

```
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
- Cilium

Flannel

Overlay - Netzwerk

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

Vorteile

- Guter einfacher Einstieg
- reduziert 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 -- sh
```

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

```
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo
## cd; mkdir -p manifests/network; cd manifests/network
## vi 01-policy.yml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: a-simple-policy
  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-<tln>
```

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 Unternehmen/Personen/Institutionen Pakete teilen
können
```

Grundlagen / Aufbau / Verwendung (Dev/Ops)

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.
-> saubere -> .kube/config

## Test
kubectl cluster-info
```

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 (runterladen):

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

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/

Wichtige Befehle

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

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

## Chart herunterladen
helm repo pull bitnami/mysql

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

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

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

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

```
helm show values bitnami-mysql
helm get values my-mysql
```

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<nr>
  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 -n default get serviceaccount training<nr> -o
jsonpath='{.secrets[0].name}'` # nr durch teilnehmer <nr> ersetzen

TOKEN=`kubectl -n default 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

Ebenen des Loggings

- container-level logging
- node-level logging
- Cluster-Ebene (cluster-wide logging)

Working with kubectl logs

Logs

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

Built-In Monitoring tools - kubectl top pods/nodes

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

Protokollieren mit Elasticsearch und Fluentd (Devs/Ops)

Installieren

```
microk8s enable fluentd

## Zum anzeigen von kibana
kubectl port-forward -n kube-system service/kibana-logging 8181:5601
## in anderer Session Verbindung aufbauen mit ssh und port forwarding
ssh -L 8181:127.0.0.1:8181 11trainingdo@167.172.184.80

## Im browser
http://localhost:8181 aufrufen
```

Konfigurieren

```
Discover:
Innerhalb von kibana -> index erstellen
auch nochmal in Grafiken beschreiben (screenshots von kibana)
https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes
```

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

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

Installation prometheus auf DigitalOcean Kubernetes mit helm

Set the custom-values

```
## There are some restrictions on the digitalocean kubernetes cluster
## and we need to take these into account
```

```
## cd; mkdir manifests/helm-prometheus; cd manifests/helm-prometheus
## vi values.yml
## Define persistent storage for Prometheus (PVC)
prometheus:
  prometheusSpec:
    storageSpec:
      volumeClaimTemplate:
        spec:
          accessModes: ["ReadWriteOnce"]
          storageClassName: do-block-storage
          resources:
            requests:
              storage: 5Gi

## Define persistent storage for Grafana (PVC)
grafana:
  # Set password for Grafana admin user
  adminPassword: your_admin_password
  persistence:
    enabled: true
    storageClassName: do-block-storage
    accessModes: ["ReadWriteOnce"]
    size: 5Gi

## Define persistent storage for Alertmanager (PVC)
alertmanager:
  alertmanagerSpec:
    storage:
      volumeClaimTemplate:
        spec:
          accessModes: ["ReadWriteOnce"]
          storageClassName: do-block-storage
          resources:
            requests:
              storage: 5Gi

## Change default node-exporter port
```

```
prometheus-node-exporter:
  service:
    port: 30206
    targetPort: 30206

## Disable Etcd metrics
kubeEtcd:
  enabled: false

## Disable Controller metrics
kubeControllerManager:
  enabled: false

## Disable Scheduler metrics
kubeScheduler:
  enabled: false
```

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

```
## install helm chart
## Unfortunately this does not work with digitalocean managed kubernetes (doks) -
1.22.
## helm install --namespace monitoring --create-namespace -f values.yml prometheus
prometheus-community/kube-prometheus-stack
## Error with EOF

## But you can do this
helm template --namespace monitoring --create-namespace -f values.yml prometheus
prometheus-community/kube-prometheus-stack > all.yml
kubectl apply -f all.yml


## now check, if everything is up and running // takes a couple of minutes
kubectl -n monitoring get all
```

Access Prometheus

```
## What Service is it ?
kubectl -n monitoring get svc prometheus-kube-prometheus-prometheus

## Prometheus: (Version kubectl on your client)
kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090

## in our case kubectl is on remote client, so we need open a tunnel
## Session 1:
kubectl port-forward -n monitoring svc/prometheus-kube-prometheus-prometheus 9090

## Session 2:
## Service Listens on 8000
ssh -L 9090:localhost:9090 tln<tln>@<ip-client>
```

```
## in browser
http://localhost:9090
```

Let us look into prometheus

Access Grafana

```
## Grafana: (Version kubectl on your client)
kubectl port-forward -n monitoring svc/prometheus-grafana 8000:80

## in our case kubectl is on remote client, so we need open a tunnel
## Session 1:
kubectl port-forward -n monitoring svc/prometheus-grafana 8000:80

## Session 2:
## Service Listens on 8000
ssh -L 8000:localhost:8000 tln<tln>@<ip-client>

## In browser:
http://localhost:8000
user: admin
pass: <from-values-files-above>
```

```
## Alternative Quick hack, change service to nodeport
kubectl -n monitoring get svc/prometheus-grafana -o yaml > 01-svc-grafana.yml
## change type -> to -> NodePort
kubectl apply -f 01-svc-grafana.yml
kubectl -n monitoring get svc/prometheus-grafana

## IP of one Node
http://68.183.216.6:30220/
```

Reference

- A bit outdated, be cause the name of the chart has changed
  - https://www.digitalocean.com/community/tutorials/how-to-set-up-digitalocean-kubernetes-cluster-monitoring-with-helm-and-prometheus-operator

Prometheus/cAdvisor (Devs/Ops) - Überblick

What does it do ?

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

Technical

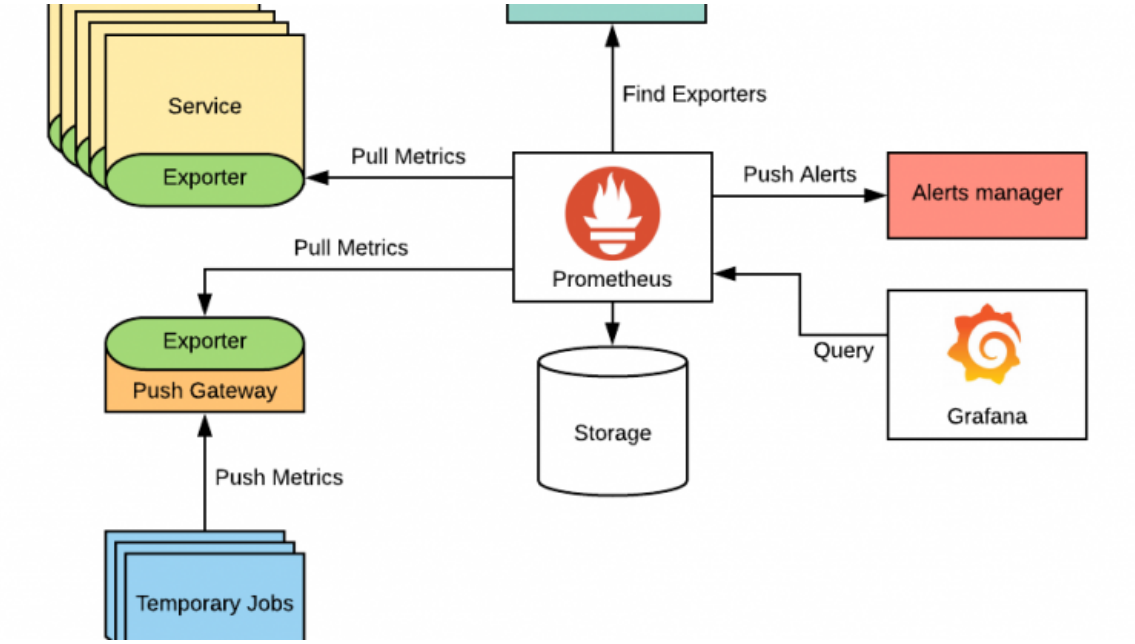
- Prometheus has a TDB (Time Series Database) and is good as storing time series with data

- Prometheus includes a local on-disk time series database, but also optionally integrates with remote storage systems.
- Prometheus's local time series database stores data in a custom, highly efficient format on local storage.
- Ref: https://prometheus.io/docs/prometheus/latest/storage/

What are time series ?

- A time series is a sequence of data points that occur in successive order over some period of time.
- Beispiel:
  - Du willst die täglichen Schlusspreise für eine Aktie für ein Jahr dokumentieren
  - Damit willst Du weitere Analysen machen
  - Du würdest das Paar Datum/Preis dann in der Datumsreihenfolge sortieren und so ausgeben
  - Dies wäre eine "time series"

Kompenenten von Prometheus



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

Prometheus Server

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

Grafana ?

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

Kubernetes Security

Grundlagen und Beispiel (Praktisch)

Geschichte

- Namespaces sind die Grundlage für Container
- LXC - Container

Grundlagen

- letztendlich nur ein oder mehreren laufenden Prozesse im Linux - Systeme

Seit: 1.2.22 Pod Security Admission

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

Vorgefertigte Regelwerke

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

Praktisches Beispiel für Version ab 1.2.23 - Problemstellung

```
## Schritt 1: Namespace anlegen

## mkdir manifests/security
## cd manifests/security
## vi 01-ns.yml

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

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

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

apiVersion: v1
kind: Pod
metadata:
  name: nginx
```

```
  namespace: test-ns<tln>
spec:
  containers:
    - image: nginx
      name: nginx
      ports:
        - containerPort: 80
```

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

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

## vi 02-nginx.yml

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

```
kubectl delete -f 02-nginx.yml
kubectl apply -f 02_pod.yml
kubectl -n test-ns<tln> get pods
```

```
## Schritt 4:
## Weitere Anpassung runAsNotRoot
## vi 02-nginx.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: test-ns12
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_pod.yml
kubectl apply -f 02_pod.yml
kubectl -n test-ns<tln> get pods
kubectl -n test-ns<tln> describe pods nginx
```

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

  * Wir müssen ein image, dass auch als NICHT-Root kaufen 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-ns12
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-ns<tln> get pods
```

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

Option 1: helm chart entpacken und das helm chart patchen

```
helm add repo bitnami ....
helm template --base-directory=base bitnami/mysql
## patchen kustomize
kustomize build overlay/prod
kubectl apply -k overlay/prod
```

Option 2: packe helm chart aus

```
## pull
helm pull
tar xvf mysql-9.0.34.tgz
## templates werden
kubectl kustomize build overlay/prod
helm install mysql-release mysql # 2. mysql wäre das chart-verzeichnis lokal im
filesystem
## Vorteile
## ich kann das ganze auch wieder so installieren
## ich kann ein update durch führen
```

Option 3: helm --post-renderer

```
## erst wird template erstellt und dann dann ein weiteres script ausgeführt
## und dann erst installiert.
helm install --post-renderer=./patch.sh

## im shell-script
## kubectl kustomize
## https://austindewey.com/2020/07/27/patch-any-helm-chart-template-using-a-kustomize-post-renderer/
```

Option 4: kustomize lädt helm - chart

```
## kustomization.yml
https://github.com/kubernetes-sigs/kustomize/blob/master/examples/chart.md
```

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

Per Default mit namespace namespace1 arbeiten

```
## namespace muss vorhanden sein, evtl anlegen
kubectl create ns namespace1
kubectl config set-context --current --namespace=namespace1
```

Hilfe zu einem bestimmten Befehl

```
## z.B. Hilfe zu config
kubectl help config

```

Hilfe zu Objekten

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

Allgemein

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

## Welche api-resources gibt es ?
kubectl api-resources
kubectl api-resources | grep namespaces
## Gross- und Kleinschreibung egal
kubectl api-resources | grep -i ingress

```

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


### In den Container / Pod reingehen

```

```
## Kommando in pod ausführen
kubectl exec -it nginx -- bash
## bei deployment hineinwechseln in einen beliebigen pod
kubectl exec -it deploy/nginx-deployment -- bash

```

Arbeiten mit namespaces

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

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

Alle Objekte anzeigen

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

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


```

Logs

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

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

Pod - Verbindung debuggen

```
## Rahmenbedingungen.
## Gemanaged'er Kubectl - Cluster und ich kann nicht per ssh zugreifen
## Lösung - busybox starten und damit Verbindungen testen.
kubectl run --rm -it busybux --image busybox -- sh

## Jetzt kann ich mich so verhalten als wäre ich im Cluster
## z.B.
## ping <andere-pod-ip>
```

Übung mit sealed-secrets

2 Komponenten

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

Schritt 1: Walkthrough - Client Installation (als root)

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

```

Schritt 2: Walkthrough - Server Installation mit kubectl client

```
## auf dem Client
## cd
## mkdir manifests/seal-controller/ #
## cd manifests/seal-controller
## Neueste Version
wget https://github.com/bitnami-labs/sealed-
```

````
secrets/releases/download/v0.17.5/controller.yaml
kubectl apply -f controller.yaml
```

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

```
kubeseal --fetch-cert

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

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

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

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

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

kubectl apply -f basic-auth-sealed.yaml

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

```

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

```

### Hinweis: Ubuntu snaps

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

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

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

### Ref:

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

### Config für nginx mit configmap einhängen

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  nginx.conf: |
    # jochens personal config map
    user nginx;
    worker_processes 1;
    events {
      worker_connections 10240;
    }
    http {
      server {
        listen   80;
        server_name localhost;
        location / {
          root /usr/share/nginx/html;
        }
      }
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
```
```

```
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
            - name: nginx-conf-vol
              mountPath: /etc/nginx/nginx.conf
              subPath: nginx.conf
              readOnly: true
      volumes:
      - name: nginx-conf-vol
        configMap:
          name: nginx-conf
          items:
            - key: nginx.conf
              path: nginx.conf



```

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/

Wie kann ich die Default Class für Ingress bzw. IngressClass setzten

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
  name: nginx-example
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
```

Wie kann ich bei docker einloggen mit kubernetes bzw. ...

```
## einen pod von einem image aus einer registry mit credentials ziehen.
## klassische / unsichere Lösung / weil credentials in manifest gespeichert werden
(unverschlüsselt)
https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/
```

Sind Änderungen in einer eingehängt config-map sofort im pod sichtbar ?

```
Nein
```

```
Lösung als Workaround: https://github.com/stakater/Reloader
```

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
  * Cilium
```

````
### Flannel

#### Overlay - Netzwerk

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

#### Vorteile

  * Guter einfacher Einstieg
  * reduziert 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/