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

Agenda

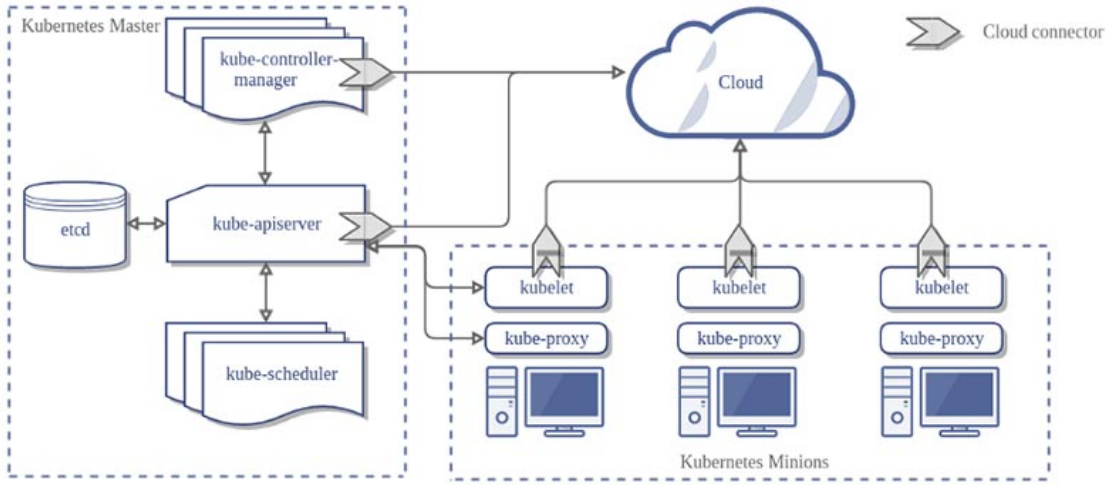
1. Kubernetes (Refresher)
    - Aufbau von Kubernetes
    - Kubernetes und seine Objekte (pods, replicasets, deployments, services, ingress)
    - Verbinde mit kubectl
    - Manifeste ausrollen (im Namespace) (2-3)
    - Arbeiten mit non-root images
2. Kubernetes Praxis API-Objekte
    - Das Tool kubectl (Devs/Ops)
    - kubectl example with run
    - Arbeiten mit manifests (Devs/Ops)
    - Pods (Devs/Ops)
    - kubectl/manifest/pod
    - ReplicaSets (Theorie) - (Devs/Ops)
    - kubectl/manifest/replicaset
    - Deployments (Devs/Ops)
    - kubectl/manifest/deployments
    - Services (Devs/Ops)
    - kubectl/manifest/service
    - DaemonSets (Devs/Ops)
    - IngressController (Devs/Ops)
    - Hintergrund Ingress
    - Documentation for default ingress nginx
    - Beispiel mit Hostnamen
3. Kubernetes Secrets / Sealed Secrets (bitnami)
    - Welche Arten von secrets gibt es?
    - Übung mit secrets
    - Übng mit sealed-secrets
4. Kubernetes Wartung / Fehleranalyse
    - Wartung mit drain / uncordon (Ops)
    - Debugging Ingress
5. Kubernetes Pods Disruption Budget
    - PDB - Uebung
6. Kubernetes PodAffinity/PodAntiAffinity
    - Warum ?
    - Übung
7. Kubernetes - Kustomize
    - Beispiel ConfigMap - Generator
    - Beispiel Overlay und Patching

    - Resources
8. Kubernetes - Storage
    - Praxis. Beispiel. NFS
9. gitlab ci/cd
    - Overview
    - Using the test - template
    - Examples running stages
    - Predefined Vars
    - Rules
    - Example Defining and using artifacts
10. gitlab / Kubernetes (gitops)
    - gitlab Kubernetes Agent with gitops - mode
11. gitlab / Kubernetes (CI/CD - Auto Devops)
    - Was ist Auto DevOps
    - Debugging KUBE_CONTEXT - Community Edition
12. Helm
13. Prometheus
14. Tipps & Tricks
    - Default namespace von kubectl ändern
    - Ingress Controller auf DigitalOcean aufsetzen
    - vi einrückungen für yaml
    - gitlab runner as nonroot
15. RootLess
    - Offizielles RootLess Docker Image für Nginx

Kubernetes (Refresher)

Aufbau von Kubernetes

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

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

Komponenten des Masters

ETCD

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

KUBE-CONTROLLER-MANAGER

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

KUBE-API-SERVER

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

KUBE-SCHEDULER

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

Nodes

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

Pod/Pods

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

Control Plane Node (former: master) - components

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

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

Kube-proxy

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

Referenzen

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

Kubernetes Praxis API-Objekte

Das Tool kubectl (Devs/Ops)

Allgemein

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

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

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

Namespace im context ändern

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

Hauptkommandos

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

namespaces

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

Arbeiten mit manifesten

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

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

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

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

Ausgabeformate / Spezielle Informationen

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

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

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

Zu den Pods

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

```
## Pods anzeigen
kubectl get pods
kubectl get pod

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

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

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

## Status eines Pods anzeigen
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

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

Arbeiten mit namespaces

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

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

Alle Objekte anzeigen

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

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

Logs

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

Referenz

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

kubectl example with run

Example (that does work)

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

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

Example (that does not work)

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


### Ref:

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

### kubectl/manifest/pod


### Walkthrough
```

vi nginx-static.yml

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

- name: web image: bitnami/nginx

```
```

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

show config

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

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

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

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

vi nginx-deployment.yml

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

kubectl apply -f nginx-deployment.yml

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

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

---

apiVersion: v1 kind: Service metadata: name: svc-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

Nur der Fall wenn man microk8s zum Einrichten verwendet

Ubuntu

microk8s enable ingress

```
### Walkthrough
```

mkdir apple-banana-ingress

cd apple-banana-ingress

apple.yml

vi apple.yml

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

---

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

kubectl apply -f apple.yml

banana

vi banana.yml

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

---

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

kubectl apply -f banana.yml

Ingress

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

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

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

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

Reference

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

Find the problem

```
## Hints

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

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

## now we can adjust our config
```

Solution

```
## in kubernetes 1.22.2 - ingress.yml needs to be modified like so.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # otherwice it does not know, which controller to use
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: "app12.lab.t3isp.de"
    http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

Kubernetes Secrets / Sealed Secrets (bitnami)

Welche Arten von secrets gibt es?

Welche Arten von Secrets gibt es ?

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

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

Übung mit secrets

Übung 1 - ENV Variablen aus Secrets setzen

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

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

```
## Schritt 3:
## vi 07-print-envs-complete.yml
apiVersion: v1
kind: Pod
metadata:
  name: print-envs-complete
spec:
  containers:
  - name: env-ref-demo
    image: nginx
    env:
    - name: APP_VERSION
      value: 1.21.1
    - name: APP_FEATURES
      value: "backend,stats,reports"
    - name: APP_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: APP_POD_NODE
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: APP_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: APP_PASSWORD
    - name: APP_EMAIL
      valueFrom:
        secretKeyRef:
          name: mysecret
          key: APP_EMAIL
```

```
    envFrom:
    - configMapRef:
        name: app-config
```

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

Übng 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/

Kubernetes Wartung / Fehleranalyse

Wartung mit drain / uncordon (Ops)

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

## Alle pods von 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
```

Debugging Ingress

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

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

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

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

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

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

```
## -A alle namespaces
kubectl get pods -A | grep -i ingress
## jetzt sollten die pods zu sehen
## Dann logs der Pods anschauen und gucken, ob Anfrage kommt
## Hier steht auch drin, wo sie hin geht (zu welcher PodIP)
## microk8s -> namespace ingress
```

```
## Frage: HTTP_STATUS_CODE welcher ? z.B. 404
kubectl logs -n default <controller-ingress-pod>
```

3. Schritt Pods analyieren, die Anfrage bekommen

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

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

Fehlerfall: ingress correct, aber service und pod nicht da

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

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

## Seite aufrufen über browser

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

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

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

Kubernetes Pods Disruption Budget

PDB - Uebung

Warum ?

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

```
(oder max eine maximale Anzahl nicht verfügbar: maxUnavailable
```

Wann ?

```
Das ganze funktioniert nur fÜr:
Voluntary disruptions

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

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

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

```
## Situation: 3-node-cluster
```

```
## Schritt 1:
## Deployment erstellen
## mkdir pdb-test
## cd pdb-test
## vi 01-pdb-test-deploy.yml
## vi nginx-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test-deployment
## tells deployment to run 2 pods matching the template
spec:
  selector:
    matchLabels:
      app-test: nginx
  replicas: 10
  template:
    metadata:
      labels:
        app-test: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
## Schritt 2:
kubectl apply -f 01-pdb-test-deploy.yml
```

```
## Schritt 3:
## vi 02-pdb-test-budget.yml
## pdb festlegen
```

```
## % oder Zahl möglich
## auch maxUnavailable ist möglich statt minAvailable
kind: List
apiVersion: v1
items:
- apiVersion: policy/v1beta1
  kind: PodDisruptionBudget
  metadata:
    name: pdb-test
  spec:
    minAvailable: 50%
    selector:
      matchLabels:
        app-test: nginx
```

```
## Schritt 4: pdb apply'en
kubectl apply -f 02-pdb-test-budget.yml
```

```
## Schritt 5: Erste node drainen
## hier geht noch alles
kubectl drain <node1>
kubectl get pdb
kubectl describe pdb
```

```
## Schritt 6: 2. Node drainen
## hier geht auch noch alles, aber evtl. bereits meldungen
## von System-Pods
kubectl drain <node2>
kubectl get pdb
kubectl describe pdb
```

```
## Schritt 7: 3. Node drainen
## jetzt kommen meldungen - pod cannot be evicted
## von System-Pods
kubectl drain <node3>
kubectl get pdb
kubectl describe pdb
```

Kubernetes PodAffinity/PodAntiAffinity

Warum ?

PodAffinity - Was ist das ?

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

Es ist auch denkbar für:
o im gleichen Rechenzentrum
```

```
o im gleichen Rack etc.
```

PodAntiAffinity - Was ist das ?

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

Übung

Übung 1: PodAffinity (forced) - auf gleicher Node/Hostname

```
## Schritt 1.
## mkdir pod-affinity-test
## cd pod-affinity-test
## vi 01-busybox-sleep.yml
apiVersion: v1
kind: Pod
metadata:
  name: sleep-busybox
  labels:
    app: backend
spec:
  containers:
    - name: bb
      image: busybox
      command: ["sleep"]
      args: ["999999"]
```

```
## Schritt 2:
kubectl apply -f 01-busybox-sleep.yml
```

```
## Schritt 3:
## Welche Hostnamen gibt es.
## Wichtig, um topologyKey zu verstehen:
kubectl get nodes -o yaml | grep hostname
```

```
## Schritt 4:
## Deployment mit podAffinity festlegen
## vi 02-fronted-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
spec:
  selector:
    matchLabels:
      frontend: nginx
```

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

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

```
## Schritt 5: Prüfen und ausrollen
## auf welcher node läuft busybox
kubectl get pods -l app=backend -o wide

## deployment ausrollen
kubectl apply -f 02-fronted-nginx.yml

## Wo laufen die Deployments ?
kubectl get pods -l frontend=nginx -o wide

## und wir lassen uns nochmal das describe ausgeben
kubectl describe pods nginx-frontend-<key>
```

Übung 2: PodAffinity (forced) - im gleichen Rack

```
## Schritt 1:
## Bei einem cluster, dieser Schritt nur durch trainer
kubectl get nodes --show-labels
kubectl label nodes pool-tg5g9rh4y-cw8mb rack=1
kubectl label nodes pool-tg5g9rh4y-cw8mr rack=1
kubectl label nodes pool-tg5g9rh4y-cw8mw rack=2
```

```
## Schritt 2.
## mkdir pod-affinity-racktest
## cd pod-affinity-racktest
## vi 01-busybox-sleep.yml
apiVersion: v1
kind: Pod
metadata:
  name: sleep-busybox
  labels:
    app: backend
```

```
spec:
  containers:
    - name: bb
      image: busybox
      command: ["sleep"]
      args: ["999999"]
```

```
## Schritt 3:
kubectl apply -f 01-busybox-sleep.yml
```

```
## Schritt 4:
## Welche Hostnamen gibt es.
## Wichtig, um topologyKey zu verstehen:
kubectl get nodes -o yaml | grep hostname
```

```
## Schritt 5:
## Deployment mit podAffinity festlegen
## vi 02-fronted-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
spec:
  selector:
    matchLabels:
      frontend: nginx

  replicas: 10
  template:
    metadata:
      labels:
        frontend: nginx
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchLabels:
                app: backend
            topologyKey: rack

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

```
## Schritt 6: Prüfen und ausrollen
## auf welcher node läuft busybox
kubectl get pods -l app=backend -o wide

## deployment ausrollen
```

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

## Wo laufen die Deployments ?
kubectl get pods -l frontend=nginx -o wide

## und wir lassen uns nochmal das describe ausgeben
kubectl describe pods nginx-frontend-<key>
```

Übung 3: PodAntiAffinity (forced) auf anderem Hosts

```
## Schritt 1.
## mkdir pod-affinity-test
## cd pod-affinity-test
## vi 01-busybox-sleep.yml
apiVersion: v1
kind: Pod
metadata:
  name: sleep-busybox
  labels:
    app: backend
spec:
  containers:
    - name: bb
      image: busybox
      command: ["sleep"]
      args: ["999999"]
```

```
## Schritt 2:
kubectl apply -f 01-busybox-sleep.yml
```

```
## Schritt 3:
## Welche Hostnamen gibt es.
## Wichtig, um topologyKey zu verstehen:
kubectl get nodes -o yaml | grep hostname
```

```
## Schritt 4:
## Deployment mit podAffinity festlegen
## vi 02-fronted-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
spec:
  selector:
    matchLabels:
      frontend: nginx

  replicas: 10
  template:
    metadata:
      labels:
```

```
        frontend: nginx
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchLabels:
                app: backend
            topologyKey: kubernetes.io/hostname

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

```
## Schritt 5: Prüfen und ausrollen
## auf welcher node läuft busybox
kubectl get pods -l app=backend -o wide

## deployment ausrollen
kubectl apply -f 02-fronted-nginx.yml

## Wo laufen die Deployments ? // das muss jetzt auf anderen Hosts sein
kubectl get pods -l frontend=nginx -o wide

## und wir lassen uns nochmal das describe ausgeben
kubectl describe pods nginx-frontend-<key>
```

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

```
## Schritt 1.
## mkdir pod-affinity-preferred
## cd pod-affinity-preferred
## vi 01-busybox-sleep.yml
apiVersion: v1
kind: Pod
metadata:
  name: sleep-busybox
  labels:
    app: backend
spec:
  containers:
    - name: bb
      image: busybox
      command: ["sleep"]
      args: ["999999"]
```

```
## Schritt 2:
kubectl apply -f 01-busybox-sleep.yml
```

```
## Schritt 3:
## Welche Hostnamen gibt es.
## Wichtig, um topologyKey zu verstehen:
kubectl get nodes -o yaml | grep hostname
```

```
## Schritt 4:
## Deployment mit podAffinity festlegen
## vi 02-fronted-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
spec:
  selector:
    matchLabels:
      frontend: nginx

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

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

```
## Schritt 5: Prüfen und ausrollen
## auf welcher node läuft busybox
kubectl get pods -l app=backend -o wide

## deployment ausrollen
kubectl apply -f 02-fronted-nginx.yml

## Wo laufen die Deployments ?
kubectl get pods -l frontend=nginx -o wide

## und wir lassen uns nochmal das describe ausgeben
kubectl describe pods nginx-frontend-<key>
```

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

```
## vi 02-fronted-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-frontend
spec:
  selector:
    matchLabels:
      frontend: nginx

  replicas: 10
  template:
    metadata:
      labels:
        frontend: nginx
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - backend
              topologyKey: kubernetes.io/hostname

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

Kubernetes - Kustomize

Beispiel ConfigMap - Generator

Walkthrough

```
## External source of truth
## Create a application.properties file
## vi application.properties
USER=letterman
ORG=it

## No use the generator
## the name need to be kustomization.yaml
```

```
## kustomization.yaml
configMapGenerator:
- name: example-configmap-1
  files:
  - application.properties
```

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

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

Ref.

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

Beispiel Overlay und Patching

Konzept Overlay

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

Example 1: Walkthrough

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

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

```
## Step 3: Create the service - file
## vi base/service.yml
kind: Service
apiVersion: v1
metadata:
  name: service-app
```

```
spec:
  type: ClusterIP
  selector:
    app: simple-app
  ports:
  - name: http
    port: 80
```

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

```
## Step 4: create the customization file accordingly
##vi overlays/prod/kustomization.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 kustomization overlays/dev

## 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

Resources

Where ?

- Used in base

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

Which ?

- URL
- filename
- Repo (git)

Example:

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

Kubernetes - Storage

Praxis. Beispiel. NFS

Create new server and install nfs-server

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

vi /etc/exports
## adjust ip's of kubernetes master and nodes
## kmaster
/var/nfs/ 192.168.56.101(rw,sync,no_root_squash,no_subtree_check)
## knode1
/var/nfs/ 192.168.56.103(rw,sync,no_root_squash,no_subtree_check)
## knode 2
/var/nfs/ 192.168.56.105(rw,sync,no_root_squash,no_subtree_check)
```

```
exportfs -av
```

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

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

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


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

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

gitlab ci/cd

Overview

Using the test - template

Example Walkthrough

```
## Schritt 1: Neues Repo aufsetzen

## Setup a new repo
## Setting:

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

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


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

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

1x speichern und committen.

## Jetzt wird es in der Pipeline ausgeführt.
```

Examples running stages

Predefined Vars

Example to show them

```
stages:
  - build

show_env:
  stage: build
  scripts:
    - env
    - pwd
```

Reference

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

Rules

Ref:

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

Example Defining and using artifacts

gitlab / Kubernetes (gitops)

gitlab Kubernetes Agent with gitops - mode

gitlab / Kubernetes (CI/CD - Auto Devops)

Was ist Auto DevOps

Debugging KUBE_CONTEXT - Community Edition

Why ?

```
In the community edition, deploy to production does not work
if KUBE_CONFIG isset in Settings -> CI/CD -> Variables

deploy to production fails in pipeline
```

Find out the context

```
## This overwrites auto devops completely
##.gitlab-ci.yml
deploy:
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  script:
    - set
    - kubectl config get-contexts
    - kubectl config use-context dummyhoney/spring-autodevops-tln1:gitlab-devops-tn1
    - kubectl get pods -n gitlab-agent-tln1
```

Fix by setting KUBE_CONFIG

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

Helm

Prometheus

Tipps & Tricks

Default namespace von kubectl ändern

How ?

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

Reference:

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

Ingress Controller auf DigitalOcean aufsetzen

Basics

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

Prerequisites

- kubectl muss eingerichtet sein

Walkthrough (Setup Ingress Controller)

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

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

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

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

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

vi einrückungen für yaml

Ubuntu (im Unterverzeichnis /etc/vim - systemweit)

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

```
cursorcolumn
```

Testen

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

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

gitlab runner as nonroot

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

RootLess

Offizielles RootLess Docker Image für Nginx

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