Kubernetes Training (3 days)

Agenda

1. Grundlagen
    - Welches System ? (minikube, micro8ks etc.)
    - Aufbau
2. Installation
    - Überblick
    - Linux Client aufsetzen
3. microk8s
    - Installation Ubuntu - snap
    - Patch to next major release - cluster
    - Remote-Verbindung zu Kubernetes (microk8s) einrichten
    - Create a cluster with microk8s
4. kubectl
    - alle Ressourcen (Möglichkeiten) der Api anzeigen
    - pod starten mit beispiel
    - alle pods anzeigen
    - auf welcher Node läuft ein pod
    - pods löschen
5. Kubernetes
    - Deployments
6. Gitlab CI/CD
    - Predefined Variables
7. Examples
    - Kuard pod
    - Ingress Nginx

Grundlagen

Welches System ? (minikube, micro8ks etc.)

General

```
kubernetes itself has not convenient way of doing specific stuff like
creating the kubernetes cluster.

So there are other tools/distri around helping you with that.
```

Kubeadm

General

- The official CNCF (https://www.cncf.io/) tool for provisioning Kubernetes clusters (variety of shapes and forms (e.g. single-node, multi-node, HA, self-hosted))
- Most manual way to create and manage a cluster

microk8s

- Created by Canonical (Ubuntu)
- Runs on Linux
- Runs only as snap
- In the meantime it is also available for Windows/Mac
- HA-Cluster

Advantage

- Easy to create HA-Cluster
- Easy to manage

minikube

k3s

kind

```
Minikube
Minikube can run on Windows and MacOS, because it relies on virtualization (e.g.
Virtualbox) to deploy a kubernetes cluster in a Linux VM. You can also run minikube
directly on linux with or without virtualization. It also has some developer-friendly
features, like add-ons.

Minikube is currently limited to a single-node Kubernetes cluster (for details, see
this issue). Although, it is on their roadmap.

Microk8s
Disclaimer: of all the K8s offerings, I know the least about this one

Microk8s is similar to minikube in that it spins up a single-node Kubernetes cluster
with its own set of add-ons.

Like minikube, microk8s is limited to a single-node Kubernetes cluster, with the added
```

limitation of only running on Linux and only on Linux where snap is installed.

K3s
K3s runs on any Linux distribution without any additional external dependencies or tools. It is marketed by Rancher as a lightweight Kubernetes offering suitable for edge environments, IoT devices, CI pipelines, and even ARM devices, like Raspberry Pi's. K3s achieves its lightweight goal by stripping a bunch of features out of the Kubernetes binaries (e.g. legacy, alpha, and cloud-provider-specific features), replacing docker with containerd, and using sqlite3 as the default DB (instead of etcd). As a result, this lightweight Kubernetes only consumes 512 MB of RAM and 200 MB of disk space. K3s has some nice features, like Helm Chart support out-of-the-box.

Unlike the previous two offerings, K3s can do multiple node Kubernetes cluster. However, due to technical limitations of SQLite, K3s currently does not support High Availability (HA), as in running multiple master nodes. The K3s team plans to address this in the future.

Now, on to some honorary mentions...

Kind
Kind (Kubernetes-in-Docker), as the name implies, runs Kubernetes clusters in Docker containers. This is the official tool used by Kubernetes maintainers for Kubernetes v1.11+ conformance testing. It supports multi-node clusters as well as HA clusters. Because it runs K8s in Docker, kind can run on Windows, Mac, and Linux.

Kind is optimized first and foremost for CI pipelines, so it may not have some of the developer-friendly features of other offerings.

Desktop Docker
Docker for Mac/Windows now ships with a bundled Kubernetes offering.

However:

Kubernetes versions are tightly coupled with the Docker version (i.e. Docker stable channel ships with K8s v1.10. If you want K8s v1.13, you need to switch to Docker edge channel).

Not as easy to destroy and start a new K8s cluster. AFAIK, you would have to disable Kubernetes and re-enable it through the Docker desktop app preferences.

Kubeadm

... And there are plenty more that I do not remember off the top of my head.

So, it all depends on what you want to get out these tools and out of Kubernetes.

Are you a developer who wants a simple K8s cluster and don't need or care about multi-node/HA features?

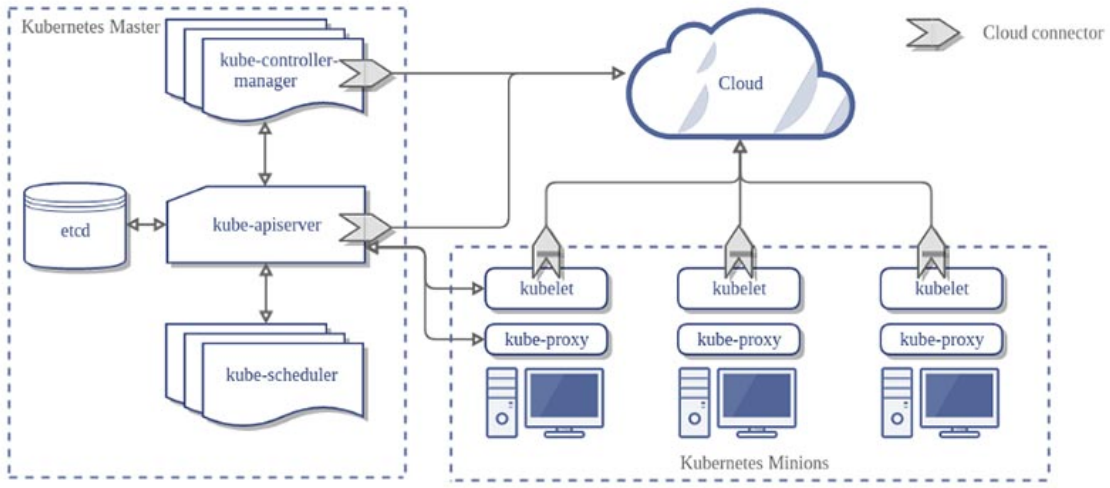
Are you a developer working on a distributed application that runs on Kubernetes and need to test various failure scenarios (e.g. node failure)?

```
Are you trying to learn about Kubernetes from a cluster administrator's perspective?

I hope you find this information helpful.
```

Aufbau

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

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

Installation

Überblick

Linux Client aufsetzen

```
## If you want to use ubuntu as your local kubectl client
## set it up like so (e.g. in virtualbox)


## hostnamectl set-hostname client.training.local
## will install latest version as snap
sudo snap install kubectl --classic
## show other versions
snap info kubectl

kubectl version --client

## vi ~/.bashrc
echo "alias k=kubectl" >> ~./bashrc
source ~/.bashrc
```

microk8s

Installation Ubuntu - snap

```
sudo snap install microk8s --classic
## Important enable dns // otherwice not dns lookup is possible
microk8s enable dns
microk8s status

## Execute kubectl commands like so
microk8s kubectl
microk8s kubectl cluster-info

## Make it easier with an alias
echo "alias kubectl='microk8s kubectl'" >> ~/.bashrc
source ~/.bashrc
kubectl
```

Patch to next major release - cluster

Remote-Verbindung zu Kubernetes (microk8s) einrichten

```
## On master-server get config
microk8s config > remote_config

## Download (scp config file) and store in .kube - folder
cd ~
mkdir .kube
cd .kube
scp master_server:/path/to/remote_config config

## now you can execute all kubectl commands, but they are executed against remote
server
microk8s kubectl get pods
## or if using kubectl or alias
kubectl get pods

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

Create a cluster with microk8s

Walkthrough

```
## auf master (jeweils für jedes node neu ausführen)
microk8s add-node

## dann auf jeweiligem node vorigen Befehl der ausgegeben wurde ausführen
## Kann mehr als 60 sekunden dauern ! Geduld...Geduld..Geduld
```

Ref:

- https://microk8s.io/docs/high-availability

kubectl

alle Ressourcen (Möglichkeiten) der Api anzeigen

```
kubectl api-resources
```

pod starten mit beispiel

Example

```
## 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
```

Ref:

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

alle pods anzeigen

Basics

```
kubectl get pods
## aber auch get pod kann verwendet werden, gleiches Ergebnis
kubectl get pod
```

Erweitert

```
kubectl get pods -o wide
```

auf welcher Node läuft ein pod

```
## Hier hilft die Ausgabe -o wide
kubectl get pods -o wide
```

pods löschen

```
## Variante 1
kubectl delete pods/mypod

## Variante 2
kubectl apply -f pod-manifest.yml
kubectl get pods
## now delete
kubectl delete -f pod-manifest.yml
```

Kubernetes

Deployments

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/

Gitlab CI/CD

Predefined Variables

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

Examples

Kuard pod

kuard-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

kuard-pod apply

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

Ingress Nginx

Walkthrough

```
mkdir 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"
---

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

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

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

---

kind: Service
```

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

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

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
        - path: /apple
          backend:
            serviceName: apple-service
            servicePort: 5678
        - path: /banana
          backend:
            serviceName: banana-service
            servicePort: 5678
```

```
## 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
```