Kubernetes Installation

Manuelle Installation der Kubernetes-Binaries

Kernel-Module in die `/etc/modules` eintragen:

```
br_netfilter
overlay
```

Kernel-Module laden:

```
modprobe br_netfilter
modprobe overlay
```

System-Konfiguration anpassen (`/etc/sysctl.conf`):

```
net.ipv4.ip_forward=1
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-ip6tables=1
```

System-Konfiguration laden:

```
sysctl -p /etc/sysctl.conf
```

Docker-Repository-Schlüssel vertrauen:

```
apt install gpg
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor --
output /etc/apt/trusted.gpg.d/docker-keyring.gpg
```

Docker-Repository hinzufügen (`/etc/apt/sources.list.d/docker.list`):

```
deb https://download.docker.com/linux/debian/ bullseye stable
```

Repositories aktuallisieren:

```
apt update
```

containerd installieren:

```
apt install 'containerd.io'
```

containerd-Standard-Konfiguration speichern: <ode bash> containerd config default > /etc/containerd/config.toml </code>

`containerd-Konfiguration /etc/containerd/config.toml anpassen:`

```
sandbox_image = "registry.k8s.io/pause:3.9"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
  SystemdCgroup = true
```

Bei einer eigenen Registry muss dies noch zusätzlich als Mirror eingetragen werden.

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
  endpoint = ["https://registry.training.lab:5000"]
```

containerd neustarten:

```
systemctl restart containerd.service
```

/etc/crictl.yaml anpassen: code bash> runtime-endpoint: unix:*/run/containerd/containerd.sock image-endpoint: unix:*/run/containerd/containerd.sock </code>

Kubernetes-Repository-Schlüssel vertrauen:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | gpg --dearmor --output /etc/apt/trusted.gpg.d/kubernetes-keyring.gpg
```

Kubernetes-Repository hinzufügen (/etc/apt/sources.list.d/kubernetes.list):

```
deb https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
```

Repositories aktuallisieren:

```
apt update
```

Kubernetes-Binaries installieren:

```
apt install kubeadm=1.31.1-00 kubectl=1.25.4-00 kubelet=1.31.1-00
```

Automatische Upgrades verhindern:

```
apt-mark hold kubeadm kubectl kubelet
```

Installation des Clusters

Das Setup des Clusters erfolgt in drei Schritten: 1. auf dem ersten Knoten (`k8s-control1.tuxnet.lan`) wird der Cluster initialisiert 2. die Control-Plane des Clusters wird um zwei Knoten erweitert (`k8s-control2.tuxnet.lan`, `k8s-control3.tuxnet.lan`) 3. dem Cluster werden zwei Worker-Knoten hinzugefügt (`k8s-worker1.tuxnet.lan`, `k8s-worker2.tuxnet.lan`)

Initialisierung des Clusters

Für die Initialisierung des Clusters wird auf dem ersten Knoten (`k8s-control1.tuxnet.lan`) eine Datei mit der Cluster-Konfiguration benötigt (`init.yml`):

```
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: 1.25.4
controlPlaneEndpoint: kubeapi.tuxnet.lan:6443
networking:
  podSubnet: 10.30.0.0/16
  serviceSubnet: 10.40.0.0/16
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
serverTLSBootstrap: true
```

Die beiden Subnetze können angepasst werden und sollten sich nicht der Netzwerkkonfiguration des Hosts überschneiden.

Mit der Cluster-Konfiguration wird der Cluster initialisiert:

```
kubeadm init --config=init.yml --upload-certs
```

Die Option `–upload-certs` bewirkt, dass die Cluster-Zertifikate temporär und verschlüsselt in der Cluster-Datenbank gespeichert werden. Dies vereinfacht die Erweiterung der Control-Plane, da die Zertifikate nicht händisch kopiert werden müssen.

Hat alles funktioniert, werden am Ende zwei `kubeadm join`-Befehle angezeigt. Die Ausgabe sollte gespeichert werden, da die angezeigten Befehle auf den anderen Knoten benötigt werden.

Erweiterung der Control-Plane

Wenn die Cluster-Initialisierung abgeschlossen ist, kann die Control-Plane um zwei Knoten (`master2-X.training.lab`, `master3-X.training.lab`) auf insgesamt drei Knoten erweitert werden. Dazu wird der erste `kubeadm join`-Befehl **mit** der Option `–control-plane` verwendet:

```
kubeadm join kubeapi-X.training.lab:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash> --control-plane --certificate-key <key>
```

Der Certificate-Key ist zwei Stunden gültig. Falls seit der Cluster-Initialisierung mehr als zwei Stunden vergangen sind, kann ein neuer Schlüssel erzeugt werden (die Doppelung im Befehl ist notwendig):

```
kubeadm init phase upload-certs --upload-certs
```

Hinzufügen der Worker-Knoten

Ist die Control-Plane vollständig, können dem Cluster die Worker-Knoten hinzugefügt werden. Dazu wird der zweite `kubeadm join`-Befehl **ohne** die Option `–control-plane` verwendet:

```
kubeadm join kubeapi-X.training.lab:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

Der Token ist 24 Stunden gültig. Soll nach Ablauf dieser Zeit ein Knoten hinzugefügt werden, muss ein neuer Token erzeugt werden:

```
kubeadm token create --print-join-command
```

kubectl auf dem Jumphost einrichten

Das Cluster ist jetzt vollständig. Die weitere Administration des Clusters soll vom Jumphost aus erfolgen.

Dazu wird zunächst `kubectl` im `bin`-Verzeichnis des Benutzers (`/home/userX/bin`) installiert:

```
mkdir bin
source .profile
wget -O bin/kubectl
https://dl.k8s.io/release/v1.25.4/bin/linux/amd64/kubectl
chmod +x bin/kubectl
```

Die Konfiguration für `kubectl` wird von einem Control-Plane-Knoten in die _Datei_ `.kube/config` kopiert:

```
mkdir -m 700 .kube
scp root@master1-X.training.lab:/etc/kubernetes/admin.conf .kube/config
```

Jetzt ist der Zugriff auf Clusters vom Jumphost aus möglich:

```
user0@jumphost:~$ kubectl get nodes
NAME        STATUS   ROLES           AGE     VERSION
master1-0   NotReady   control-plane   97s   v1.25.4
master2-0   NotReady   control-plane   57s   v1.25.4
master3-0   NotReady   control-plane   55s   v1.25.4
worker1-0   NotReady   <none>          5s    v1.25.4
worker2-0   NotReady   <none>          5s    v1.25.4
```

Abschluss der Cluster-Installation

Calico installieren

Bisher werden die Knoten als `NotReady` angezeigt. Das ist an dieser Stelle das erwartete Verhalten. Die Ursache dafür ist, dass noch kein Netzwerk-Plugin installiert ist.

Daher wird jetzt `Calico` in das Cluster installiert:

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/cal
ico.yaml
```

Nach einer Weile sind die Knoten Ready:

```
user0@jumphost:~$ kubectl get nodes
NAME        STATUS   ROLES           AGE   VERSION
master1-0   Ready    control-plane   12m   v1.25.4
master2-0   Ready    control-plane   12m   v1.25.4
master3-0   Ready    control-plane   12m   v1.25.4
worker1-0   Ready    <none>          11m   v1.25.4
worker2-0   Ready    <none>          11m   v1.25.4
```

Kubelet-Zertifikate signieren

Als letzter Schritt müssen die Kubelet-Zertifikate signiert werden. Dazu werden zunächst die offenen Zertifikatsanfragen angezeigt:

```
kubectl get certificatesigningrequests | grep -i pending
```

Die jeweils jüngste Anfrage eines Knotens wird signiert:

```
kubectl certificate approve csr-xxxxx csr-yyyyy csr-zzzzz
```

Alternativ `kubectl get csr kubectl delete csr –all kubectl certificate approve <csr>`

An dieser Stelle ist das Setup des Cluster abgeschlossen.

kubectl von Adminworkstation

https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

```
mkdir bin
source .profile
wget https://dl.k8s.io/release/v1.24.3/bin/linux/amd64/kubectl
chmod +x kubectl
mv kubectl bin
mkdir .kube
```

```
scp root@master-x:/etc/kubernetes/admin.conf .kube/config
```

Bash Anpassungen

vim ~/.bash_aliases

```
# enable bash completion
source <( kubectl completion bash | sed 's#kubectl$#kubectl k#g' )
alias k=kubectl
```

$ source .bash_aliases