# 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.tuxnet.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=1.31.1-00 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 (`control1.tuxnet.lab`) wird der Cluster initialisiert 2. die Control-Plane des Clusters wird um zwei Knoten erweitert (`control2.tuxnet.lab`, `control3.tuxnet.lab`) 3. dem Cluster werden zwei Worker-Knoten hinzugefügt (`node1.tuxnet.lab`, `node2.tuxnet.lab`)

## Initialisierung des Clusters

### kube-vip.yml

```
---
# task: install kube-vip for HA Kubernetes Controlplane
# hint: see
https://kube-vip.io/docs/installation/static/#example-arp-manifest
#
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-vip
  namespace: kube-system
spec:
  containers:
  - args:
    - manager
    env:
    - name: address
      value: need-to-be-set                          # use your own kubevip
    - name: port
      value: "6443"
    - name: vip_arp
      value: "true"
    - name: vip_interface                            # use your own network
interface
      value: ens3                                    #    e.g. ens3, eth0
    - name: vip_cidr
      value: "32"
    - name: cp_enable
      value: "true"
    - name: cp_namespace
      value: kube-system
    - name: vip_ddns
      value: "false"
    - name: svc_enable
      value: "false"
    - name: vip_leaderelection
      value: "true"
    - name: vip_leaseduration
      value: "5"
    - name: vip_renewdeadline
      value: "3"
    - name: vip_retryperiod
      value: "1"
    image: ghcr.io/kube-vip/kube-vip:v0.8.5     # change: to internal
registry, if needed
    imagePullPolicy: Always
```

```
    name: kube-vip
    resources: {}
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
        - NET_RAW
        - SYS_TIME
    volumeMounts:
    - mountPath: /etc/kubernetes/admin.conf
      name: kubeconfig
  hostAliases:
  - hostnames:
    - kubernetes
    ip: 127.0.0.1
  hostNetwork: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/super-admin.conf          # for kubernetes
>v1.29.x use super-admin.conf
    name: kubeconfig
```

Die kube-vip.yaml muss auf alle controlplan Nodes unter `/etc/kubernetes/manifests/` kopiert werden. Desweiteren muss **nach dem Initialisieren** und **vor dem joinen** der anderen control plane Nodes in den Cluster die `/etc/kubernetes/super-admin.conf` auf die anderen anderen control plane Nodes kopiert werden

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

```
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: 1.31.1
controlPlaneEndpoint: kubeapi.tuxnet.lab:6443
networking:
  podSubnet: 100.73.0.0/16
  serviceSubnet: 100.74.0.0/16
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
serverTLSBootstrap: true
```

Die beiden Subnetze können angepasst werden und sollten sich nicht mit 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 (`control2.tuxnet.lab`, `control3.tuxnet.lab`) auf insgesamt drei Knoten erweitert werden. Dazu wird der erste `kubeadm join`-Befehl **mit** der Option `--control-plane` verwendet:

```
kubeadm join kubeapi.tuxnet.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.tuxnet.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/<user>/bin`) installiert:

```
mkdir bin
source .profile
wget -O bin/kubectl https://dl.k8s.io/release/v1.31.1/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@control1.tuxnet.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
control1   NotReady   control-plane   97s   v1.31.1
control2   NotReady   control-plane   57s   v1.31.1
control3   NotReady   control-plane   55s   v1.31.1
node1      NotReady   <none>          5s    v1.31.1
node2      NotReady   <none>          5s    v1.31.1
```

# 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/v3.27.2/manifests/cal
ico.yaml
```

Nach einer Weile sind die Knoten Ready:

```
user0@jumphost:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
control1   Ready    control-plane   12m   v1.31.1
control2   Ready    control-plane   12m   v1.31.1
control3   Ready    control-plane   12m   v1.31.1
node1      Ready    <none>          11m   v1.31.1
node2      Ready    <none>          11m   v1.31.1
```

## 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/v1.31.1/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)
alias k=kubectl
complete -F __start_kubectl k
```

`$ source .bash_aliases`

### oder

```
mkdir -p ~/.local/share/bash-completion/completions/
kubectl completion bash > ~/.local/share/bash-completion/completions/kubectl
```

Für Helm würde selbiges funktionieren

```
mkdir -p ~/.local/share/bash-completion/completions/
helm completion bash > ~/.local/share/bash-completion/completions/helm
```

## Installation Krew PlugIn Manager

```
(   set -x; cd "$(mktemp -d)" &&   OS="$(uname | tr '[:upper:]'
```

```
'[:lower:]')" &&   ARCH="$(uname -m | sed -e 's/x86_64/amd64/' -e
's/\(arm\)\(64\)\?.*/\1\2/' -e 's/aarch64$/arm64/')" &&   KREW="krew-
${OS}_${ARCH}" &&   curl -fsSLO
"https://github.com/kubernetes-sigs/krew/releases/latest/download/${KREW}.ta
r.gz" &&   tar zxvf "${KREW}.tar.gz" &&   ./"${KREW}" install krew; )
export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"
echo 'export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"' >> ~/.bashrc
kubectl krew
kubectl krew update
kubectl krew install ctx
kubectl krew install neat
```