

# Installation

- [Kubernetes Cluster installieren \(Baremetal\)](#)
- [Installation von MetallLB \(Lokaler LoadBalancer\)](#)
- [K3s Cluster installieren](#)
- [Traefik-Reverseproxy vom K3s-Cluster entfernen](#)
- [Metrics-Server vom K3s-Cluster entfernen](#)

# Kubernetes Cluster installieren (Baremetal)

## Einleitung

In diesem Artikel geht es darum, wie wir einen Kubernetes Cluster aufsetzen können, um von den Funktionalitäten von Kubernetes zu profitieren. Kubernetes ist mittlerweile eine weitverbreitete Orchestrationsplattform für Container.

## Voraussetzungen

Um ein Kubernetes Cluster zu installieren und dieser Anleitung zu folgen, müssen die folgenden Voraussetzungen erfüllt sein:

- Mindestens Server mit einem installierten **Debian 11** oder **Debian 12**
- Pro Server mindestens **2 virtuelle CPU-Kerne**
- Pro Server mindestens **2 GB Arbeitsspeicher**
- Pro Server mindestens **20 GB freier Festplattenplatz**
- Administrationsbenutzer
- Stabile Internetverbindung

## Installation

### Netzwerkdesign

In dieser Anleitung werden wir 2 Server betreiben. Einer wird als Master Node fungieren, und der zweite Server als Worker Node. Der Master Node ist für die Verwaltung des Kubernetes Clusters zuständig. Der Worker Node führt nur die sogenannten Pods auf.

Hostbeschreibung	Hostname	IP-Adresse
Master Node	srv-kub-master	192.168.10.200
Worker Node	srv-kub-worker1	192.168.10.201

### Installation

Im ersten Schritt installieren wir ein paar benötigte Pakete und deaktivieren auf jedem Node den Swap-Speicher. Dies ist zwar nicht zwingend erforderlich, aber funktioniert in der Regel besser.

```
apt install -y sudo curl socat -y
sudo swapoff -a
sudo sed -i '/ swap / s/^(.*)$/#\1/g' /etc/fstab
```

Im nächsten Schritt installieren wir die containerd Laufzeitumgebung und stellen ein paar Dinge ein. Dazu führen wir die folgenden Befehle aus. Zuerst stellen wir ein paar Kernel Parameter ein:

```
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

Um die Änderungen jetzt zu übernehmen für wir den folgenden Befehl aus:

```
sudo sysctl --system
```

Jetzt installieren wir das Paket containerd und stellen wieder ein paar Dinge ein.

```
sudo apt update
sudo apt -y install containerd
containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
```

Im nächsten Schritt setzen wir den "*cgroupdriver*" auf allen Nodes.

```
sudo nano /etc/containerd/config.toml
```

Dort müssen wir in dem Pfad

`[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]` die Option `SystemdCgroup` auf `true` verändern.

Danach starten wir den Dienst von containerd einmal neu.

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```

Im nächsten Schritt fügen wir das Kubernetes Apt Repository hinzu.

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee
```

```
/etc/apt/sources.list.d/kubernetes.list  
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o  
/etc/apt/keysrings/kubernetes-apt-keyring.gpg
```

Jetzt installieren wir die Kubernetes Tools auf unseren Servern.

```
sudo apt update  
sudo apt install kubelet kubeadm kubectl -y  
sudo apt-mark hold kubelet kubeadm kubectl
```

Zum Testen, ob alles geklappt hat, können wir einmal `kubectl version` ausführen.

## Kubernetes Konfiguration

Jetzt konfigurieren wir unseren Kubernetes Cluster und verbinden die einzelnen Hosts miteinander, um die grundlegenden Funktionen von Kubernetes zu erhalten.

Wir erstellen im ersten Schritt eine yaml Datei, welches die Konfiguration von unserem Cluster enthält und passen diese Datei unserem Belieben an.

**Dieser Schritt muss nur auf dem Master Node durchgeführt werden!**

```
nano kubelet.yaml
```

```
apiVersion: kubeadm.k8s.io/v1beta3  
kind: InitConfiguration  
---  
apiVersion: kubeadm.k8s.io/v1beta3  
kind: ClusterConfiguration  
kubernetesVersion: "1.28.0" # Ersetzen mit deiner eingesetzten Version  
controlPlaneEndpoint: "k8s-master"  
---  
apiVersion: kubelet.config.k8s.io/v1beta1  
kind: KubeletConfiguration
```

Nachdem wir die Datei erstellt haben, initialisieren wir jetzt unser Kubernetes Cluster.

```
sudo kubeadm init --config kubelet.yaml
```

Wenn alles geklappt hat, sollte eine Meldung auftauchen, dass das **Control-Plane** erfolgreich initialisiert wurde. Wenn dies der Fall ist, sehen wir auch die entsprechenden Befehle damit die anderen **Worker Nodes** als **Worker** oder als **Master** beitreten können.

Diese Befehle können wir bei Bedarf auch neu erstellen. Für das erste die Befehle zur Seite kopieren.

Wir müssen im Anschluss noch einmal die Befehle ausführen, die benötigt werden, damit mit dem Control-Plane kommuniziert werden kann.

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Um zu testen, ob der Cluster richtig hochgefahren wurde, können wir die folgenden Befehle ausführen.

```
kubectl get nodes  
kubectl cluster-info
```

Jetzt können wir auf den Worker Nodes die Befehle ausführen, um dem Kubernetes Cluster beizutreten. Wenn die Nodes beigetreten sind, sollten diese mit dem Befehl `kubectl get nodes` ersichtlich sein. Damit die Nodes im Status hochgefahren werden, brauchen wir sogenannte Netzwerk Add-ons. Wir verwenden hier Calico.

Um Calico zu installieren, führen wir den folgenden Befehl aus:

```
kubectl apply -f  
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/calico.yaml
```

Um zu überprüfen, ob die Calico Pods laufen, führen wir den folgenden Befehl aus:

```
kubectl get pods -n kube-system
```

## Kubernetes Cluster testen

Um das Kubernetes Cluster zu testen, führen wir den folgenden Befehl auf dem Master Node aus:

```
kubectl create deployment nginx-app --image=nginx --replicas 2  
kubectl expose deployment nginx-app --name=nginx-web-svc --type NodePort --port 80  
kubectl describe svc nginx-web-svc
```

Jetzt sollten wir, wenn alles geklappt hat, eine Ausgabe mit dem entsprechenden Port erhalten. Das heißt, wenn wir eine Webanfrage auf die externe IP mit dem angegebenen Port starten, sollte uns die "NGINX Welcome Page" begrüßen.

Mit dem folgenden Befehl können wir das gestartete Deployment wieder stoppen und löschen:

```
kubectl delete deployment nginx-app
```

# Installation von MetallLB (Lokaler LoadBalancer)

## Einleitung

In dieser kurzen Anleitung beschreibe ich kurz, wie wir mit der Hilfe von MetallLB einen lokalen LoadBalancer betreiben können. In der Regel verwendet man einen LoadBalancer innerhalb eines Cloud-Providers und dieser stellt dann einen kostenpflichtigen LoadBalancer zur Verfügung. Sobald wir aber im LAN einen solchen LoadBalancer über die Service-Konfiguration anfordern, sollte die Anfrage auf `"pending"` stehen bleiben und keine IP erhalten. Mit diesem LoadBalancer sind die internen Anwendungen dann erreichbar über eine dedizierte IP-Adresse.

## Durchführung

Im ersten Schritt führen wir den folgenden Befehl auf unserem Kubernetes Master Node aus:

```
kubectl apply -f  
https://raw.githubusercontent.com/metallb/metallb/v0.14.8/config/manifests/metallb-native.yaml
```

Damit wird dann die Konfiguration für den LoadBalancer Pod heruntergeladen und gestartet. Um zu überprüfen, ob die Pods erfolgreich gestartet wurden, können wir den folgenden Befehl ausführen:

```
kubectl get pods --namespace metallb-system
```

Wenn hier bei allen Containern der Status auf `Running` steht, sollten die Pods ordnungsgemäß hochgefahren sein. Wir müssen dann im Anschluss eine neue YAML-Datei anlegen, in dem wir den IPv4-Bereich definieren, welcher vom LoadBalancer verwendet werden darf. Die Datei sieht folgendermaßen aus:

```
apiVersion: metallb.io/v1beta1  
kind: IPAddressPool  
metadata:  
  name: <pool-name>  
  namespace: metallb-system  
spec:  
  addresses:  
    - <ip-von_ip-bis>
```

Jetzt müssen wir eine weitere YAML-Datei erstellen. Diese enthält das "L2Advertisement" und enthält den entsprechenden IP-Pool, der zur Vergabe der IP-Adressen verwendet werden darf. Die

Datei sieht wie folgt aus:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2metallb
  namespace: metallb-system
spec:
  ipAddressPools:
    - <pool-name>
```

Diese beiden Konfigurationen werden dann auch wieder einmal über `kubectl apply -f <dateiname>` aktiviert. Wenn alles geklappt hat, können wir in der Service-Konfiguration den Typen auf LoadBalancer setzen. Wenn wir jetzt die Service-Konfiguration aktualisieren, sollten wir mit dem folgenden Befehl dann die IP-Adresse unseres LoadBalancers sehen können.

```
kubectl get services --all-namespaces
```



# K3s Cluster installieren

## Einleitung

In dieser kurzen Anleitung beschreibe ich, wie wir einen K3s-Cluster installieren können. Auf diesem Wege können wir viel schneller ein laufendes Kubernetes-Cluster auf die Beine stellen. K3s wird als fertiges Skript bereitgestellt, welches auf Ressourcensparenden Betrieb ausgelegt ist.

## Installation

Um das Cluster zu installieren, benötigen wir im ersten Schritt 3 eigenständige Linux-Server. In meinem Fall sind das alle Debian 12 Server mit jeweils einer eigenen IP-Adresse.

Um den Master-Node zu installieren, führen wir den folgenden Befehl aus:

```
apt install curl && curl -sL https://get.k3s.io | sh -
```

Sobald das Skript durchgelaufen ist, läuft unser Control-Plane-Server unseres Kubernetes Cluster. Dieses können wir überprüfen, indem wir den folgenden Befehl ausführen, um alle momentan verfügbaren Nodes unseres Clusters einzusehen:

```
kubectl get nodes
```

Jetzt wollen wir im nächsten Schritt unsere Worker-Nodes dem Cluster hinzufügen. Dazu benötigen wir im ersten Schritt den Cluster-Token. Diesen finden unter: `/var/lib/rancher/k3s/server/node-token`. Der Inhalt der Datei sieht wie folgt aus:

```
K10b02c9d094b28ce8099ca6bbded97e68f7734af130f8a19103cdc7dfc8bf89cda::server:  
4f03364535d090536b282a9d2d22681a
```

Es wird hier aber nur das gelb hinterlegte (Ab `server:`) benötigt. Das ist der Cluster-Token. Jetzt verbinden wir uns auf einen Worker-Node und führen den folgenden Befehl aus:

```
curl -sL https://get.k3s.io | K3S_URL=https://<ip-master-node>:6443 K3S_TOKEN=<cluster-token>  
sh -
```

**Info:** Es muss hier noch die IP-Adresse oder DNS-Name des Master-Nodes und der Cluster-Token eingetragen werden.

Sobald das Skript durchgelaufen ist, sollte der Node dem Cluster beigetreten sein. Dies können wir überprüfen, indem wir auf dem Master wieder den `kubectl get nodes` ausführen. Wenn hier der Worker Node auftaucht, hat alles wie gewünscht geklappt. Diese Schritte führen wir dann für

weitere Worker Nodes aus. So können wir unser Cluster nach Belieben erweitern.

# Traefik-Reverseproxy vom K3s-Cluster entfernen

## Einleitung

In diesem Artikel geht es kurz darum, wie wir in unserem K3s Cluster den Traefik-Reverseproxy entfernen können, welcher standardgemäß immer mitinstalliert wird.

## Durchführung

Um Traefik zu entfernen, müssen wir die service-Datei des K3s-Dienstes anpassen. Dazu verbinden wir uns auf unseren Master-Node und öffnen die folgende Datei:

```
nano /etc/systemd/system/k3s.service
```

Dort fügen wir unter `ExecStart` noch `--disable=traefik` ein. Das sollte dann wie folgt aussehen:

```
ExecStart=/usr/local/bin/k3s \  
    server \  
    --disable=traefik \  
    --
```

Im Anschluss erstellen wir noch eine andere Datei mit dem folgenden Befehl:

```
touch /var/lib/rancher/k3s/server/manifests/traefik.yaml.skip
```

Zum Schluss starten wir jetzt einmal alle Server neu. Dann sollte der Traefik-Pod nicht mehr gestartet werden.

# Metrics-Server vom K3s-Cluster entfernen

## Einleitung

In diesem Artikel geht es kurz darum, wie wir in unserem K3s Cluster den Metrics-Server entfernen können, welcher standardgemäß immer mitinstalliert wird.

## Durchführung

Um den Metrics-Server zu entfernen, müssen wir die service-Datei des K3s-Dienstes anpassen. Dazu verbinden wir uns auf unseren Master-Node und öffnen die folgende Datei:

```
nano /etc/systemd/system/k3s.service
```

Dort fügen wir unter `ExecStart` noch `--disable=traefik` ein. Das sollte dann wie folgt aussehen:

```
ExecStart=/usr/local/bin/k3s \  
server \  
--disable=metrics-server \  

```

Im Anschluss erstellen wir noch eine andere Datei mit dem folgenden Befehl:

```
touch /var/lib/rancher/k3s/server/manifests/traefik.yaml.skip
```

Zum Schluss starten wir jetzt einmal alle Server neu. Dann sollte der Traefik-Pod nicht mehr gestartet werden.