[edv]schulen
des Landkreises Deggendorf

Technische Dokumentation

für das Projekt

PostgreSQL-Cluster

Installation eines ausfallsicheren PostgreSQL Clusters auf K3s mit Backup-Lösung

Bearbeitungszeitraum:

1. Halbjahr 2024/2025

11.09.2024 - 15.01.2025

Bearbeiter:

Heinrich Thaler

www.it-thaler.de

info@it-thaler.de

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bezeichnung
CLI	Command Line Interface
K8s	Kubernetes
LXC	Linux Container
PVCs	Persistent Volume Claims
PVs	Persistent Volumes
SQL	Structured Query Language
iSCSI	internet Small Computer System Interface
SCSI	Small Computer System Interface
KVM	Kernel-basierte Virtual Machine

1. Einleitung

Abbildung 1: Logo von K3s

Das Ziel des Projektes ist die Implementierung eines hochverfügbaren PostgreSQL-Clusters, das auf K3s (einer minimalen Kubernetes-Distribution) läuft. Es soll sicherstellen, dass die PostgreSQL-Datenbank, die kritische Daten für den Schulalltag wie Stundenpläne und Termine enthält, bei Ausfällen nicht unerreichbar wird. Zudem muss ein Backup eingerichtet werden, um Datenverlust zu verhindern und eine schnelle Wiederherstellung zu ermöglichen.

PostgreSQL ist eine der weltweit führenden Open-Source-Datenbanklösungen, die sich durch ihre Zuverlässigkeit, Skalierbarkeit und Flexibilität auszeichnet. Da die Verwaltung von Datenbanken eine zentrale Rolle in modernen IT-Infrastrukturen spielt, ist es essenziell, deren Verfügbarkeit und Sicherheit zu gewährleisten.

Die Wahl von K3s als Plattform ermöglicht es, eine minimalistische und ressourcenschonende Kubernetes-Umgebung zu nutzen, die optimal für kleinere Infrastrukturen geeignet ist. Gleichzeitig bietet die Backup-Lösung einen Schutz vor Datenverlust, was für den Umgang mit sensiblen und unersetzlichen Daten von entscheidender Bedeutung ist.

2. Ressourcenplanung

Es werden mehrere Computer benötigt, um den Cluster aufzusetzen. Da nicht für jedes Schülerprojekt mehrere Server ausgehändigt werden können, soll das Projekt auf LXC-Containern eingerichtet werden. Es werden keine leistungsstärkeren Rechner benötigt, da K3s sehr Ressourceneffizient ist. Zudem wurde die bestehende Infrastruktur wie z.B. das Netzwerk der Schule benutzt.
Ein weiterer Vorteil ist die ausschließliche Nutzung von Open-Source-Software, wodurch keine zusätzlichen Lizenzkosten anfallen.

3. Implementation

Der PostgreSQL-Cluster wurde auf einer K3s-Kubernetes-Umgebung installiert, um eine robuste, hochverfügbare und ausfallsichere Datenbanklösung zu bieten. Der Cluster besteht aus mehreren Nodes, die in einer verteilten Architektur zusammenarbeiten. Diese Konfiguration ermöglicht Datenreplikation, effiziente Lastverteilung und automatische Failover-Szenarien, um die ständige Verfügbarkeit der Dienste sicherzustellen.

Dieses Diagramm zeigt den Aufbau welcher im folgenden beschrieben wird:

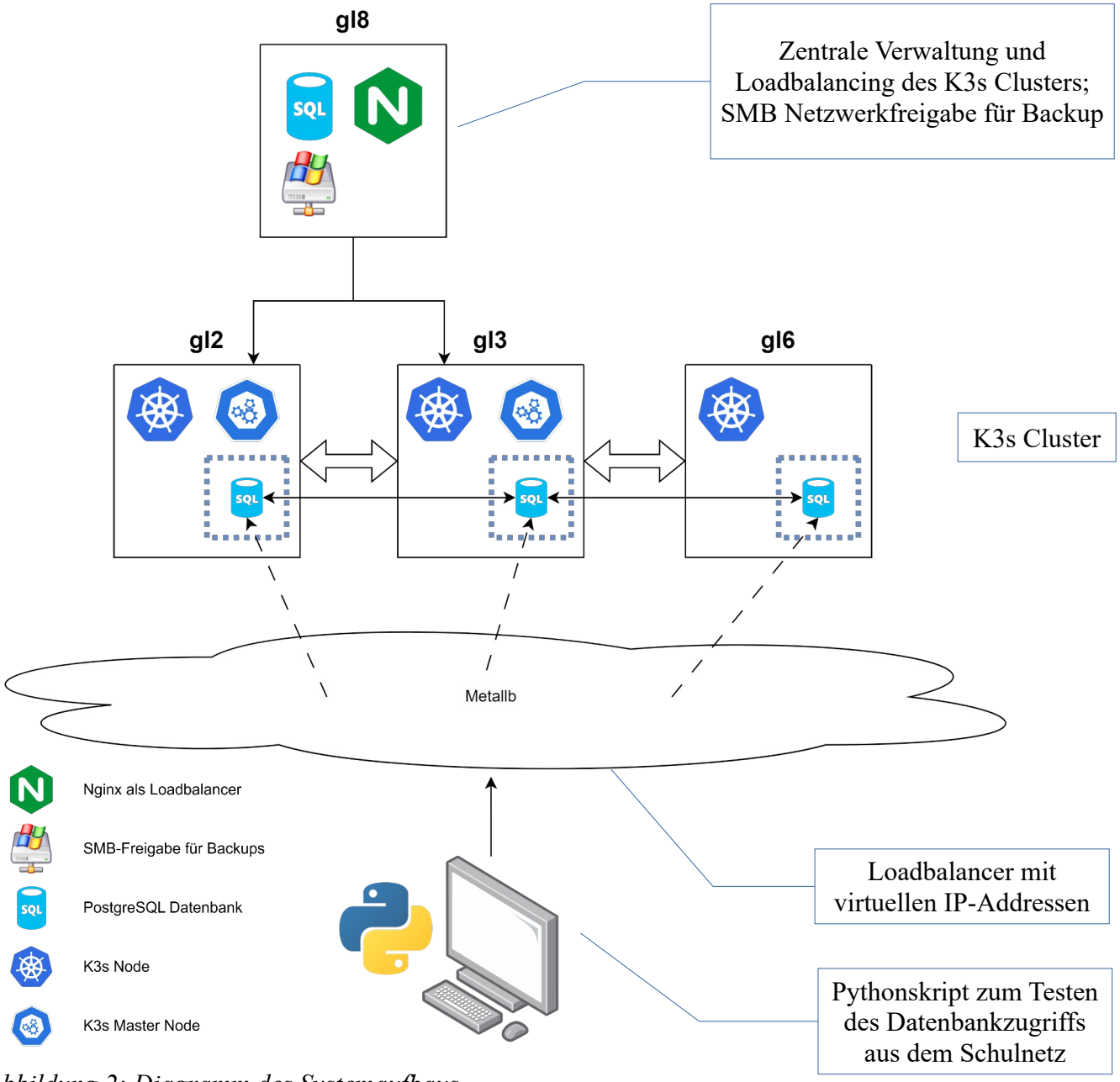

Abbildung 2: Diagramm des Systemaufbaus

3.1. LXC-Container

Es waren bereits Debian Container vorhanden, welche Geklont wurden.

Um eine Warnung vom Sudo-Befehl loszuwerden, musste der Hostname in die Hosts-Datei eingetragen werden. Darin wurden auch die Hostnames der jeweils anderen Container eingetragen, damit die IP-Adressen nicht verwendet werden müssen.

/etc/hosts (alle Container)

```
1 127.0.0.1       localhost
2 ::1             localhost ip6-localhost ip6-loopback
3 ff02::1         ip6-allnodes
4 ff02::2         ip6-allrouters
5
6 10.0.145.110    gl2
7 10.0.145.111    gl3
```

```
8 10.0.145.112   gl6
9 10.0.145.113   gl8
```

3.2. K3s Cluster

Ich habe mich dazu entschieden, eine externe Datenbank für K3s zu verwenden. Falls eine Master-Node ausfallen sollte, kann die andere weiterarbeiten. Dazu habe ich auf gl8 PostgreSQL installiert:

```
 1  sudo apt install postgresql
 2
 3  /etc/postgresql/*/main/postgresql.conf
 4  listen_addresses = '*'
 5
 6  sudo -u postgres psql
 7  create database k3s;
 8  create user k3s_user with encrypted password 'abc123def187';
 9  grant all privileges on database k3s_user to k3s;
10  exit;
11
12  systemctl restart postgresql
```

Um K3s zu installieren, wurde die offizielle Anleitung befolgt.

Hier entstand das erste Problem: K3s konnte nicht Starten und gab folgende Fehlermeldung aus: `/dev/kmsg: no such file or directory`
Der Fehler wurde mithilfe von Herrn Grallinger und diesem Beitrag im Proxmox-Forum behoben: https://forum.proxmox.com/threads/kubernetes-sharing-of-dev-kmsg-with-the-container.61622/#post-291514. Der Container hatte nicht genügend rechte um auf /dev/kmsg zuzugreifen, obwohl es kubelet benötigte. Dies führte zu einem Absturz mit der bereits genannten Fehlermeldung. Um den KMSG-Fehler zu beheben, muss man der LXC-Konfigurationsdatei folgendes anhängen:

```
lxc.apparmor.profile: unconfined
lxc.cap.drop:
lxc.cgroup.devices.allow: a
lxc.mount.auto: proc:rw sys:rw
```

Zwei der drei Nodes wurden als Master installiert, damit bei einem Ausfall der andere übernehmen kann. Der Token muss vom ersten Master ausgelesen und bei den anderen mit angegeben werden. Auch die vorher installierte Datenbank wird übergeben.

Master 1 (gl2):

```
curl -sfL https://get.k3s.io | sh -s - server
--datastore-endpoint="postgres://k3s_user:abc123def187@gl8:5432/k3s"

sudo cat /var/lib/rancher/k3s/server/node-token
```

Master 2 (gl3):

```
curl -sfL https://get.k3s.io | sh -s - server   --
token=K106d1096c6c5ae5b1b24da0b891363fa16519170c6773ac9138f2c51ef4a78f4ff::serve
r:61db40ebd0992283422a834916dbe950
--datastore-endpoint="postgres://k3s_user:abc123def187@gl8:5432/k3s"
```

Node 1 (gl6):

```
curl -sfL https://get.k3s.io |
K3S_TOKEN=K106d1096c6c5ae5b1b24da0b891363fa16519170c6773ac9138f2c51ef4a78f4ff::s
erver:61db40ebd0992283422a834916dbe950 sh -s - agent --server https://gl8:6443
```

Anschließend wird Nginx auf gl8 als Loadbalancer konfiguriert, damit bei einem Ausfall der Cluster noch gesteuert werden kann.

/etc/nginx/nginx.conf (gl8)

```
load_module /usr/lib/nginx/modules/ngx_stream_module.so;
events {}

stream {
  upstream k3s_servers {
    server gl2:6443;
    server gl3:6443;
  }

  server {
    listen 6443;
    proxy_pass k3s_servers;
  }
}
```

Kubectl ist die CLI für Kubernetes. Dieses ist bei K3s enthalten und muss nur noch auf gl8 installiert werden:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
```

Konfigurationsdatei für kubectl auf gl8 kopieren:

```
scp gl2:/etc/rancher/k3s/k3s.yaml ~/.kube/config
```

Ab jetzt kann der Cluster vollständig von gl8 aus gesteuert werden. Mithilfe von `kubectl apply -f` werden alle folgenden .yaml Konfigurationsdateien angewandt.

3.3. Helm

Helm wurde anhand der offiziellen Anleitung installiert. Es wird verwendet, um im nächsten Schritt den PostgreSQL Operator zu installieren.

```
curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -
sudo apt-get install apt-transport-https --yes
echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

3.4. PostgreSQL Operator

Um eine Helm-Chart zu installieren, muss zuerst das jeweilige Repository hinzugefügt werden.

Mithilfe der Befehlen von der Website wurde der Operator und die UI installiert

```
1 helm repo add postgres-operator-charts
https://opensource.zalando.com/postgres-operator/charts/postgres-operator
2 helm install postgres-operator postgres-operator-charts/postgres-operator
3 helm repo add postgres-operator-ui-charts
https://opensource.zalando.com/postgres-operator/charts/postgres-operator-ui
4 helm install postgres-operator-ui postgres-operator-ui-charts/postgres-operator-ui
5
6 # port forwarden zum Testen
7 kubectl port-forward svc/postgres-operator-ui 8081:80 –address='0.0.0.0'
```

Die folgende Konfigurationsdatei wurde mithilfe der PostgreSQL Operator UI erstellt. Es enthält Einstellungen wie den Namen des Clusters, die Standarddatenbank und die Anzahl der Replicas.

wichtige-daten-3.yaml

```yaml
kind: "postgresql"
apiVersion: "acid.zalan.do/v1"

metadata:
  name: "wichtige-daten-3"
  namespace: "default"
  labels:
    team: acid

spec:
  teamId: "acid"
  postgresql:
    version: "16"
  numberOfInstances: 3
  enableMasterLoadBalancer: true
  enableReplicaLoadBalancer: true
  enableConnectionPooler: true
  enableReplicaConnectionPooler: true
  enableMasterPoolerLoadBalancer: true
  enableReplicaPoolerLoadBalancer: true
  maintenanceWindows:
  volume:
    size: "10Gi"
  users:
    bauer: []
  databases:
    firmendaten: bauer
  allowedSourceRanges:
  - 0.0.0.0/0
  resources:
    requests:
      cpu: 100m
```

```
      memory: 100Mi
    limits:
      cpu: 500m
      memory: 500Mi
```

3.5. Metallb

Metallb wurde auch mithilfe von Helm installiert:

```
helm repo add metallb https://metallb.github.io/metallb
helm install metallb metallb/metallb
```

Die folgende Konfiguration wurde für Metallb verwendet. Darin muss definiert werden, welcher IP-Adressen Bereich verwendet werden soll. „layer2“ bedeutet, dass das ARP Protokoll verwendet wird, um die virtuellen IP-Adressen zu veröffentlichen.

metallb-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
        - 10.0.145.130-10.0.145.149
```

3.6. Backup

Regelmäßige Backups werden auf der SMB-Freigabe gespeichert, um Datenverluste zu vermeiden. Die Backups werden automatisiert mittels CronJobs durchgeführt.

Zuerst muss eine SMB Freigabe erstellt werden:

```
sudo apt update && sudo apt install samba
mkdir /home/hans/backup
sudo nano /etc/samba/smb.conf
```

/etc/samba/smb.conf (am Ende anhängen)

```
[backup]
   comment = Backups für PosgreSQL auf K3s
   path = /home/hans/backup
   read only = no
   browsable = yes
```

```
sudo service smbd restart
sudo smbpasswd -a hans
```

Zum Einbinden wird dieses Paket auf allen Nodes benötigt:
sudo apt install cifs-utils -y

PV und PVC erstellen:
smb-backup.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: smb-backup-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: smbbak
  csi:
    driver: smb.csi.k8s.io
    volumeHandle: smb-backup-volume
    volumeAttributes:
      source: "//10.0.145.113/backup"
    nodeStageSecretRef:
      name: smbcreds
      namespace: default
  mountOptions:
    - dir_mode=0775
    - file_mode=0775
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: smb-backup-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  volumeName: smb-backup-pv
  storageClassName: smbbak
```

ConfigMap mit Backupscript erstellen:
postgres-backup-script.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-backup-script
data:
  backup.sh: |
    #!/bin/bash
    PG_HOST=wichtige-daten-3
    PG_USER=postgres
    PG_DB=firmendaten
    BACKUP_PATH=/mnt/smb-backup/backup_$(date +%Y%m%d_%H%M%S).sql
```

```
12
13
PGPASSWORD=jYLfzb4Hskm7wmXBiC9pYQDLPU9k1Vk7rXmRghC3BpDbKtLwOd8x14mSJ
uasISTI pg_dump -h $PG_HOST -U $PG_USER $PG_DB > $BACKUP_PATH
14
15    if [ $? -eq 0 ]; then
16       echo "Backup successful! Saved to $BACKUP_PATH"
17    else
18       echo "Backup failed!"
19    fi
```

Cronjob welcher den Backup ausführt:
postgres-backup.yaml

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: postgres-backup
spec:
  schedule: "0 2 * * *"  # Daily at 2 AM
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: postgres:16
            volumeMounts:
            - name: smb-backup
              mountPath: /mnt/smb-backup
            - name: backup-script
              mountPath: /scripts
            command: ["/bin/bash", "/scripts/backup.sh"]
          restartPolicy: OnFailure
          volumes:
          - name: smb-backup
            persistentVolumeClaim:
              claimName: smb-backup-pvc
          - name: backup-script
            configMap:
              name: postgres-backup-script
```

Der Cronjob führt Täglich um zwei Uhr das Backup-Script aus. Die Dumps werden auf der Samba-Freigabe (gl8) gespeichert.

4. Einbindung

4.1. Datenbankzugriff

Der Zugang zur Datenbank erfolgt über die externe IP-Adresse, die durch MetalLB bereitgestellt wird. Diese kann mithilfe dieses Befehls ausgelesen werden:

```
kubectl get svc
```

Das Standardpasswort für den Benutzer postgres kann so ausgelesen werden:

```
kubectl get secret postgres.wichtige
daten-3.credentials.postgresql.acid.zalan.do -n default -o
jsonpath="{.data.password}" | base6
```

Die Verbindung kann anschließend so getestet werden:

```
psql -h 10.0.145.136 -p 5432 -U bauer -d firmendaten
```

4.2. Backup und Wiederherstellung

Ein Backup kann Manuell durchgeführt werden, indem man einen Job erstellt:

```
kubectl create job --from=cronjob/postgres-backup
manual-backup-job
ls -la backup
kubectl delete job manual-backup-job
```

Anschließend kann der Dump mit pg_restore wieder eingespielt werden:

```
pg_restore -h 10.0.145.133 -p 5432 -U postgres -v -f
"/home/hans/backup/backup_20241108_010001.sql"
```

4.3. Automatisches Failover

Im Idealfall verläuft der Failover-Prozess schnell und nahezu unbemerkt für die Benutzer. Sobald Patroni einen Ausfall der Primary-Instanz erkennt, wird ein Leader-Election-Prozess eingeleitet, um eine neue Primary zu bestimmen. Die verbleibenden Replikas werden synchronisiert, um die Datenintegrität sicherzustellen. Der Loadbalancer MetalLB aktualisiert automatisch seine Routing-Regeln, sodass alle Anfragen nahtlos an die neue Primary-Instanz geleitet werden. Dieser Prozess erfolgt innerhalb von Sekunden oder Millisekunden, sodass Anwendungen wie Webseiten oder Apps ohne Unterbrechung weiterlaufen. Eventuelle minimale Verzögerungen bei laufenden Anfragen bleiben für die Nutzer meist unbemerkt.

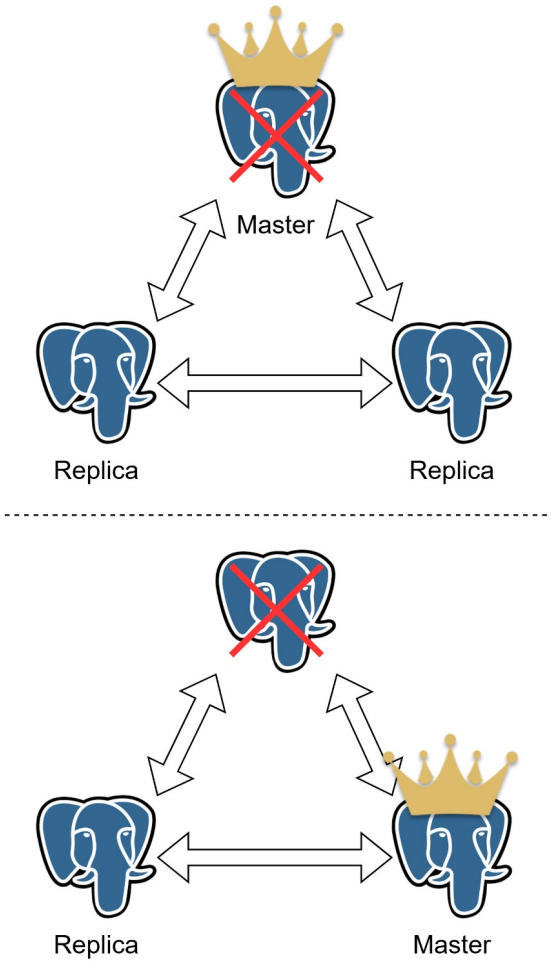

Abbildung 3: PostgreSQL Failover mit Patroni

5. Ausblick

Alle Komponenten wurden mithilfe eines Paketmanagers installiert. Für APT und Helm gibt es jeweils einen Befehl, mit dem die neusten Versionen heruntergeladen werden können. Updates sind also sehr bequem möglich.

Anstatt LXC hätte für dieses Projekt KVM verwendet werden sollen. Manche Module die für K3s oder Longhorn gebraucht werden fehlen bei LXC-Containern. KVM emuliert die Hardware, weshalb die Probleme dann nicht auftreten würden.

Nachdem das Projekt fast vervollständigt war, stieß ich auf CloudNativePG. Dies hat dieselbe Aufgabe als PostgreSQL Operator, jedoch mit etwas mehr Features. Es wäre interessant es in der

Zukunft mal auszuprobieren und zu Vergleichen.
Auch interessant wäre Terraform. Es erlaubt die Definition der Infrastruktur als Code.

Abbildungsverzeichnis

6. Glossar

LXC (Linux Containers)	Virtualisierungstechnik auf Betriebssystemebene, welche isolierte Linux-Systeme auf einem Host ausführt.
KVM (Kernel-basierte Virtual Machine)	Open-Source-Lösung zum Ausführen von virtuellen Maschinen auf der Hardware mithilfe des Linux-Kernels.
K8s (Kubernetes)	Open-Source System zur Verwaltung von Container-Anwendungen. Ermöglicht sehr hohe Skalierung und High Availability.
K3s	Minimalistische, leichtgewichtige Distribution von Kubernetes.
Helm	Ein Paketmanager für Kubernetes.
Helm-Charts	Vorlagen aus dem Internet ermöglichen schnelle Bereitstellung von Anwendungen.
Metallb	Load-Balancer-Lösung für ein K8s Cluster, welches Services mit einer externen IP versorgt.
Patroni	Eine Vorlage für Hochverfügbare PostgreSQL Lösungen.
Cronjobs	CronJobs sind wiederkehrende Aufgaben, die automatisiert auf unixartigen Betriebssystemen ausgeführt werden.
Multithreading	Gleichzeitiges Arbeiten mehrerer Threads innerhalb eines Prozesses, um Aufgaben Parallel abzuarbeiten.
iSCSI	Ein Verfahren, welches die Nutzung des SCSI-Protokolls über TCP ermöglicht.
SCSI	Eine Familie von standardisierten Protokollen und Schnittstellen für die Verbindung und Datenübertragung zwischen Peripheriegeräten und Computern.