

# NFS-Freigabe als PVC-Speicher im Kubernetes Pod verwenden

## Einleitung

In diesem kurzen Artikel geht es darum, wie wir eine **NFS-Freigabe** als **persistenten (PVC) Speicher** verwenden können. Dies verwenden wir z.B. wenn wir keinen Cloud Kubernetes Cluster betreiben, sondern diesen Cluster bei uns im lokalen LAN betreiben.

Über den NFS-Share stellen wir dann sicher, dass jeder Pod Zugriff auf dieselben Daten hat. Ansonsten müssten wir auf einem anderen Weg sicherstellen, dass auf jedem Node die Daten vorhanden sind.

**Info:** Stelle bitte sicher das das Paket `nfs-common` bereits auf allen Nodes installiert ist!

## Durchführung

### Helm Installation

Im ersten Schritt müssen wir auf unserem **Master Node** einmal den **Helm Paketmanager** installieren. Dazu führen wir den folgenden Befehl aus:

```
curl -L https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

Um zu überprüfen, ob die Installation geklappt hat, kannst du den folgenden Befehl ausführen:

```
helm version
```

### Installation vom nfs-provisioner

Im nächsten Schritt installieren wir den **nfs-subdir-external-provisioner**. Dies ist eine Speicherklasse, um den **NFS-Speicher** in das **Kubernetes Cluster** einzubinden. Kubernetes hat von Haus aus keine Möglichkeit, **NFS-Speicher** anzubinden. Um die Installation durchzuführen und einen dedizierten Namespaces zu erstellen, führe die folgenden Befehle aus:

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
kubectl create namespace nfs-provisioner
```

## Starten des nfs-provisioner Pods

In diesem Schritt starten wir den Pod für den **NFS-Provisioner**. Dazu passen wir noch die Serveradresse und den Pfad unserer NFS-Freigabe an.

```
helm install nfs-client nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=<server-
adresse> --set nfs.path=<freigabe-pfad> --namespaces <namespace-name>
```

## Anlegen der Manifest-Dateien für den Storage

Im nächsten Schritt erstellen wir eine **YAML-Datei** für das "*PersistentVolume*". Diese enthält den folgenden Inhalt:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
  namespace: nfs-storage
spec:
  capacity:
    storage: <größe> (z.B. 450Gi)
  accessModes:
    - ReadWriteMany
  nfs:
    server: <server-adresse>
    path: <freigabe-pfad>
```

Jetzt legen wir die **YAML-Datei** für das "*PersistentVolumeClaim*" an. Dies erhält die entsprechende Größe, welches unsere Anwendung benötigt. Die Datei sieht wie folgendermaßen aus:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
  namespace: nfs-storage
spec:
  accessModes:
    - ReadWriteMany
```

```
resources:

requests:

storage: <größe>
```

**Info zu PV und PVC:** Ein PV wird vom Administrator definiert und präsentiert das Speichermedium (Speicher Backend) und das PVC stellt die Anfrage vom Pod an den PV um die entsprechende Größe für die App vom Volume zu erhalten. In der Regel erstellt man einen PV pro Speichermedium und einen PVC pro Applikation / Pod.

Im nächsten Schritt müssen wir beide Dateien wie gewohnt mit `kubectl apply -f <dateipfad>` aktivieren.

## Pod Konfiguration anpassen

Um jetzt den Speicher in dem Pod verfügbar zu machen, müssen wir in der **Deployment-Datei** unseres Pods im spec Segment die **volumes** definieren.

```
volumes:
- name: nfs-volum-app
  nfs:
    server: <server-adresse>
    path: <freigabe-pfad>
```

Dann können wir jetzt im Container Segment in **volumeMounts** definieren:

```
volumeMounts:
- mountPath: <container pfad>
  name: <name-des-volumes>
```

Wenn wir jetzt im Anschluss den Pod starten, und die Berechtigungen stimmen, dann sollten die Daten geschrieben werden und damit sind jetzt die Daten persistent verfügbar.

## Beispiel Pod-Konfiguration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nextcloud
  namespace: nextcloud
annotations:
  author: Phillip
```

labels:

app: nextcloud

spec:

replicas: 3

selector:

matchLabels:

app: nextcloud

template:

metadata:

labels:

app: nextcloud

spec:

containers:

- name: nextcloud-app

image: lscr.io/linuxserver/nextcloud:latest

env:

- name: PUID

value: "1000"

- name: GUID

value: "1000"

ports:

- name: http

containerPort: 80

resources:

requests:

cpu: "250m"

memory: "256Mi"

limits:

cpu: "2000m"

memory: "4096Mi"

volumeMounts:

- mountPath: /config

name: nfs-volume-nextcloud-config

- mountPath: /data

name: nfs-volume-nextcloud-data

volumes:

- name: nfs-volume-nextcloud-config

nfs:

server: 192.168.5.4

path: /mnt/Kubernetes/Daten/nextcloud-config

```
- name: nfs-volume-nextcloud-data
nfs:
  server: 192.168.5.4
  path: /mnt/Kubernetes/Daten/nextcloud-data
```

---

```
apiVersion: v1
kind: Service
metadata:
  name: nextcloud-service
  namespace: nextcloud
spec:
  selector:
    app: nextcloud
  ports:
    - port: 80
```

---

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nextcloud
  namespace: nextcloud
spec:
  ingressClassName: nginx
  rules:
    - host: nextcloud.k8s.domain.de
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: nextcloud-service
                port:
                  number: 80
```

---

Revision #1

Created 24 October 2024 20:00:06 by Phillip U.

Updated 29 October 2024 10:14:49 by Phillip U.