Kubernetes

Alles Rund um Kubernetes

Persistent Volumes

Persistent Volumes auf NFS

Dieses Config File zeigt, wie ein Persisten Volumen in Kubernetes erstellt werden kann, welches auf einem NFS-Share liegt.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pve-nfs
spec:
  capacity:
    storage: 8000Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: nfs
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tank/containers
    server: 172.22.0.200
```

Terminologie & Aufbau

Deployment Management

Erstellung eines Deployments

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster

Erstellung

```
kubectl create deployment <deployment-name> -n <namespace> --image <image-name>
# Beispiel
kubectl create deployment nginx -n nginx --image nginx
```

Überprüfung

```
kubectl get deployment <deployment-name> -n <namespace>
# Beispiel
kubectl get deployment nginx -n nginx
```

Bearbeitung eines Deployments

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Bearbeitung

Öffnen der aktuellen Konfiguration

Achtung: Wenn die änderungen gespeichert werden, werden diese direkt vom Cluster Übernommen.

```
kubectl edit deployment <deployment-name> -n <namespace>
# Beispiel
kubectl edit deployment nginx -n nginx
```

Die Bearbeitung

Passen Sie die Änderungen in dieser neu geöffneten Datei nach Ihrem wunsch an.

Beispielweise wurde der Port hinzugefügt.

Die Änderung

Fügen Sie die Änderung am Richtigen Ort hinzu, beispiel folgt unten.

```
ports:
- containerPort: 80
protocol: TCP
```

Die komplette Datei

Wenn Sie die Datei schliessen werden die Änderungen übernommen. Wenn Sie jedoch eine Änderung an der YAML datei tätigen, welche nicht angepasst werden kann wird sich die Datei erneuert öffnen und anschliessend im Obersten Sektor den Fehler als Kommentar anzeigen.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "2"
  creationTimestamp: "2021-12-23T19:25:25Z"
  generation: 3
  labels:
    app: nginx
  name: nginx
  namespace: nginx
  resourceVersion: "255984"
  uid: c45eed25-d12f-4e32-b1fb-8965f372796b
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
```

```
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 3
  conditions:
  - lastTransitionTime: "2021-12-23T19:25:25Z"
    lastUpdateTime: "2021-12-23T19:27:27Z"
    message: ReplicaSet "nginx-7848d4b86f" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  - lastTransitionTime: "2021-12-23T19:28:47Z"
    lastUpdateTime: "2021-12-23T19:28:47Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```

Überprüfung

```
kubectl get deployment <deployment-name> -n <namespace> -o <output-optionen>
# Beispiel
kubectl get deployment nginx -n nginx -o yaml
```

Deployments veröffentlichen mittels Loadbalancer

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Veröffentlichung

Bei der veröffentlichung wird ein neuer Service vom Typ Loadbalancer erstellt, welcher die Port-Weiterleitung vom Host zum Container ermöglicht. Der Loadbalancer routed den Trafic vom allen nodes über einen Gewissen Port zu einem Host, auf welchem der veröffentlichte Container verfügbar ist.

```
kubectl expose deployment <deployment-name> --port=<Hostport> --target-port=<ContainerPort> --type=<Loadbalancer> -n <namespace>
# Beispiel
kubectl expose deployment nginx --port=8300 --target-port=80 --type=LoadBalancer -n nginx
```

Überprüfung

```
kubectl get service -n <namespace>
# Beispiel
kubectl get service -n nginx
```

Deployment Skalieren

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)

Skalieren

Im falle einer Skalierung werden die Pods skaliert. Dies bedeutet das sämtliche Container in den Pods multipliziert werden.

```
kubectl scale --replicas=<anzahl> deployment <deployment-name> -n <namespace>
# Beispiel
kubectl scale --replicas=5 deployment nginx -n nginx
```

Überprüfung

```
kubectl get deployment <deploymentname> -n <namespace>
# Beispiel
kubectl get deployment nginx -n nginx
```

Erstellung eines Namespaces

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster

Erstellung

```
kubectl create namespace <namespace>
# Beispiel
kubectl create namespace nginx
```

Überprüfung

Wenn der Namespace existiert sollte er mit folgendem Command angezeigt werden können.

```
kubectl get namespaces
```

Deployments veröffentlichen mittels Ingress

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Bereits erstelltes Deployment (How To)
- Bereits vorhandener Loadbalancer (How To)

Erstellung eines Ingress

Ein Ingress ist eine Art V-Host einfach für ein Kubernetes Cluster. Es muss nicht unbedingt ein Loadbalancer dazwischen sein, es kann auch eine Cluster IP oder ähnliches sein.

```
kubectl create ingress <ingress-name> --rule="<cname+directory>=<Loadbalancer:Port>" -n <namespace>
# Beispiel
kubectl create ingress nginx --rule="nginx.voser.local/=nginx:8300" -n nginx
```

Überprüfung

```
kubectl get ingress -n <namespace>
# Beispiel
kubectl get ingress -n nginx
```

Metallb

Was ist Metallb

Kurzbeschrieb

Metallb ist ein Loadbalancer für Baremetal Kubernetes Cluster.

Wie Funktioniert es?

Bei der Installation von Metallb wird ein Master Pod auf dem Cluster erstellt, so wie ein Speaker Pod auf jeder Node. Der Master Pod ist der Controller, welcher die Speakter kontrolliert. Mittels einer Configmap wird dem Metallb Deployment ein IP Bereich Zugewiesen, aus welchem die IPs frei verteilt werden können. Metallb routet danach die Anfragen zu den Richtigen Container / Pods.

Warum Metallb?

Wenn man ein Service welcher auf mehreren Nodes Läuft veröffentlichen möchte, gibt es verschiedene Arten wie man dies tun kann.

1. Cluster IP

Jeder Pod verfügt über eine Cluster IP, jedoch ist die Cluster IP nur intern vom Cluster erreichbar.

2. Host-Port

In diesem Fall öffnet der Pod auf der Node, aufwelcher er aktuell läuft einen Host-Port, mit welchem man direkt auf den Pod zugreifen kann. Nachteil ist, dass dieser nur auf einer Node zur verfügung steht was bedeutet, dass bei einem Ausfall dieses Hosts der Pod nicht mehr erreichbar ist.

3. Node-Port

In diesem Fall öffnet der Pod auf jeder Node einen Gewissen Port, aus der Port range von Kubernetes(30'000 - 32768), sämtliche Nodes leiten allen verkehr, welcher über einen bestimmten Port kommen an den Pod weiter. Nachteil ist, wenn alle Ports verwendet werden ist der ganze Spass vorbei & es werden keine Floating & Virtual IP's unterstützt.

4. Loadbalancer

Metallb ist ein Loadbalancer, dieser unterstützt Floating & virtual IP's, was die perfekte Redundanz ermöglicht. Kubernetes kann mit diesem Loadbalancer die Redundanz auf Layer 2 absichern.

Installation von Metallb

Voraussetzungen

- Funktionsfähiges Kubernetes Cluster
- Verbindung mittels kubectl auf das Cluster
- Offizielle Ressourcen Metallb

Installation

Bitte beachten Sie das es aktuellere Version von Metallb geben wird. Installieren Sie wenn möglich die neuste Version.

Erstellung des Namespace

Führen Sie den folgenden Befehl aus um einenen neuen Namespace für metallb zu erstellen.

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.11.0/manifests/namespace.yaml
```

Anschliessend können Sie mit dem Befehl: `kubectl get namespaces` überprüfen, ob ein neuer Namespace für Metallb erstellt wurde. Dieser hat den Namen "metallb-system"

Anwenden des Manifests

Bei der Installation des Manifests werden der Master Pod sowie die Speaker erstellt.

`kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.11.0/manifests/metallb.yaml`

Anschliessend können Sie mit dem Befehl: `kubectl get all -n metallb-system` überprüfen, ob der Master so wie de Speaker auf laufen.

k3s

k3s ist eine Kubernetes Distribution.

Loadbalancing

Wie funktionierts?

Standard mässig verwendet k3s die Software "klippert" als Loadbalancer, dieser erstellt sich als dämon set auf dem Cluster, wenn ein Service via Loadbalancer veröffentlicht wird.

Verwendung eines anderen Loadbalancer

Wenn ein anderer Loadbalancer wie Beispielsweise Metallb verwendet werden möchte, muss klippert entfernt, bzw. deaktiviert werden.

Klippert deaktivieren

Da k3s kubernetes in einem Binary ist, sollte Klippert beim Start des Servers (Binary) mittels Argument deaktiviert werden. Verwenden Sie dazu das extra Argument "`--disable servicelb`". `Dies können Sie im SystemD-Service File unter "ExecStart" hinzufügen.`

`/etc/systemd/k3s.service`

`Das zusatz Argument finden Sie auf der 10ten Zeile.`

```
[Unit]
Description=Lightweight Kubernetes
Documentation=https://k3s.io
After=network-online.target

[Service]
Type=notify
ExecStartPre=-/sbin/modprobe br_netfilter
ExecStartPre=-/sbin/modprobe overlay
```

```
ExecStart=/usr/local/bin/k3s server --data-dir /var/lib/rancher/k3s --disable servicelb
KillMode=process
Delegate=yes
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=1048576
LimitNPROC=infinity
LimitCORE=infinity
TasksMax=infinity
TimeoutStartSec=0
Restart=always
RestartSec=5s

[Install]
WantedBy=multi-user.target
```

Installation via Ansible

Das k3s Projekt verfügt über ein öffentliches Github-Repository, mit welchem man einfach sein Eigenes Kubernetes Cluster erstellen kann.

Voraussetzungen

- Server mit SSH-Zugriff
- Vorkonfigurierte Ansible Infrastruktur

Installation

Kurzzusammenfassung der Installation

1. Git repo klonen
2. Custom Ansible Inventory
3. Ansible Playbook ausführen
4. Kube-Config auf Client Kopieren

Git Repository klonen

Klonen Sie als erstes das Github-Repository in ihr gewünschter Ordner. Verwenden Sie dazu git (`git clone https://github.com/k3s-io/k3s-ansible.git`). Alternativ können Sie auch das Repository als Zip herunterladen und auspacken. Navigieren Sie anschliessend in das Lokal geklonte Repository(`cd k3s-ansible`).

Custom Ansible Inventory erstellen

Im k3s-ansible Repository existiert bereits ein Beispiel Inventory, dieses wird kopiert und anschliessend bearbeitet.

Kopieren des Beispiel Inventory

In diesem Beispiel wird das Beispiel-Inventory(./inventory/sample) in den Ordner my-cluster(./inventory/my-cluster) kopiert. Verwenden Sie dazu den Befehl "`cp ./inventory/sample/ ./inventory/my-cluster -r`" Sie können auch mehrere Ordner erstellen, falls Sie mehrere Server Gruppen Administrieren.

Ansible-Inventory mit Hosts ergänzen

Bearbeiten Sie die Host-Datei("hosts.ini") im neu erstellten Ordner(./inventory/my-cluster). Bearbeiten Sie die Textdatei mit ihrem gewünschten Text editor. Bsp.: `vim ./inventory/my-cluster/hosts.ini`

Hier ist mein Beispiels Inventory

```
[master]
srvk3s01.voser.local ansible_port=122

[node]
srvk3s02.voser.local ansible_port=222
srvk3s03.voser.local ansible_port=322
srvk3s04.voser.local ansible_port=422
srvk3s21.voser.local ansible_port=2122
srvk3s22.voser.local ansible_port=2222
srvk3s24.voser.local ansible_port=2322

[k3s_cluster:children]
master
node

[k3s_cluster:vars]
ansible_host=home.voser.cloud
ansible_user=ansible-user
```

Ansible-Gruppen Variablen bearbeiten

Der Inventory Ordner beinhaltet einen Unterordner namens: "group_vars", in diesem befindet sich eine Datei namens: "all.yml". Diese Datei beinhaltet allgemeine Informationen zum kubernetes cluster, bearbeiten Sie diese nach Ihrer Infrastruktur. Das einzige was gesetzt werden muss ist die Master IP, sowie der Ansible-User, je nach Infrastruktur möchten Sie jedoch auch die anderen Informationen überarbeiten.

Bsp.: `vim ./inventory/my-cluster/group_vars/all.yml`

```
---
k3s_version: v1.21.7+k3s1
ansible_user: ansible-user
systemd_dir: /etc/systemd/system
master_ip: "172.22.0.101"
#master_ip: "{{ hostvars[groups['master'][0]]['ansible_host'] | default(groups['master'][0]) }}"
extra_server_args: "--disable servicelb"
extra_agent_args: ""
```

Ansible Playbook ausführen

Es existieren zwei Ansible-Playbooks in diesem Repository (site.yml & reset.yml)

Mit dem site.yml wird ein Cluster Installiert, sowie aktualisiert, und mit dem reset.yml wird alles von den Hosts entfernt (Deinstallation).

Führen Sie die Playbooks mit dem normalen Befehl "ansible-playbook", sowie dem extra Parameter "-i <Inventoryfile>" aus. Der Parameter "-i" steht für Inventoryfile.

Bsp. Installation: `ansible-playbook -i ./inventory/my-cluster/hosts.ini site.yml`

Bsp. Deinstallation: `ansible-playbook -i ./inventory/my-cluster/hosts.ini reset.yml`

Kube-Config auf Cleint Kopieren

Wenn das Cluster erfolgreich Installiert ist finden Sie die Kube-Config Datei auf dem Master server unter (/etc/rancher/k3s.yml) kopieren Sie den Inhalt auf Ihren gewünschten Host und bearbeiten Sie die Variable "server:" in der Datei. Anschliessend können Sie ihr Kubernetes Cluster via Kubectl auf Ihrem Computer ansteuern. Wenn Sie das Cluster öffentlich nicht erreichbar machen wollen, können Sie kubectl auch auf dem Master verwenden.

kubectl

Kube cuddle
tipps & tricks

Set default namespace

How to set the default namespace

This is usefull if you work in a special namespace for many commands/tasks

```
kubectl config set-context --current --namespace=<namespace>
# Example
kubectl config set-context --current --namespace=nginx
```

Set default editor

How to set the default editor

If you use the command "kubectl edit" you may not want to use vi as editor, so you can expose your preferred editor with a normal environment variable expose.

```
expose EDITOR=<your prefered editor>
# Example
expose EDITOR=vim
```