Introduction

This tutorial will guide you through setting up a Kubernetes cluster using K3S. K3S is a lightweight Kubernetes distribution which is perfectly suited for the small Hetzner VMs like the CX11. Additionally, you will set up Hetzner's cloud load balancer which performs SSL offloading and forwards traffic to your Kubernetes system. Optionally, you will learn how to set up a distributed, replicated file system using GlusterFS. This allows you to move pods between the nodes while still having access to the pods' persistent data.

Prerequisites

This tutorial assumes you have set up Hetzner's CLI utility (`hcloud`) which has access to a Hetzner cloud project in your account.

The following terminology is used in this tutorial:

- Domain: `<example.com>`
- SSH Key: `<your_ssh_key>`
- Random secret token: `<your_secret_token>`
- Hetzner API token: `<hetzner_api_token>`
- IP addresses (IPv4):
  - K3S Master: `10.0.0.2`
  - K3S Node 1: `10.0.0.3`

- K3S Node 2: `10.0.0.4`
- Hetzner Cloud Load Balancer: `10.0.0.254`

Step 1 - Create private network

First, we'll create a private network which is used by our Kubernetes nodes for communicating with each other. We'll use 10.0.0.0/16 as network and subnet.

```
hcloud network create --name network-kubernetes --ip-range 10.0.0.0/16
hcloud network add-subnet network-kubernetes --network-zone eu-central --type
```

Step 2 - Create placement group and servers

Next, we'll create a "spread" placement group for our servers and then the VMs.

Step 2.1 - Create the spread placement group (optional)

The placement group ensures your VMs run on different hosts, so in case on host has a failure, no other VMs are affected.

```
hcloud placement-group create --name group-spread --type spread
```

Step 2.2 - Create the virtual machines

We'll use Debian 10 here. At the time of writing, there seems to be a bug with Debian 11 which prevents K3S from using VXLAN to communicate between the nodes. See: https://github.com/k3s-io/k3s/issues/3863

```
hcloud server create --datacenter nbg1-dc3 --type cx11 --name master-1 --image
hcloud server create --datacenter nbg1-dc3 --type cx11 --name node-1 --image d
hcloud server create --datacenter nbg1-dc3 --type cx11 --name node-2 --image d
```

Step 3 - Create and apply firewall

Now that our servers are up and running, let's create a firewall and restrict ingoing and outgoing traffic. You may need to customize the rules to match your requirements.

Create the firewall:

```
hcloud firewall create --name firewall-kubernetes
```

Allow incoming SSH and ICMP:

```
hcloud firewall add-rule firewall-kubernetes --description "Allow SSH In" --di
hcloud firewall add-rule firewall-kubernetes --description "Allow ICMP In" --d
```

Allow outgoing ICMP, DNS, HTTP, HTTPS and NTP:

```
hcloud firewall add-rule firewall-kubernetes --description "Allow ICMP Out" --
hcloud firewall add-rule firewall-kubernetes --description "Allow DNS TCP Out"
hcloud firewall add-rule firewall-kubernetes --description "Allow DNS UDP Out"
hcloud firewall add-rule firewall-kubernetes --description "Allow HTTP Out" --
hcloud firewall add-rule firewall-kubernetes --description "Allow HTTPS Out" -
hcloud firewall add-rule firewall-kubernetes --description "Allow NTP UDP Out"
```

Apply the firewall rules to all three servers:

```
hcloud firewall apply-to-resource firewall-kubernetes --type server --server m
hcloud firewall apply-to-resource firewall-kubernetes --type server --server n
hcloud firewall apply-to-resource firewall-kubernetes --type server --server n
```

Step 4 - Install K3S

It's showtime for K3S. Before we prepare our master node and agent nodes, first upgrade the system and install AppArmor. SSH into your newly created VMs and run this command on all of then:

```
apt update
apt upgrade -y
apt install apparmor apparmor-utils -y
```

Step 4.1 - Install K3S on master node

SSH into your master node and run the following command to install and start the K3S server:

```
curl -sfL https://get.k3s.io | sh -s - server \
    --disable-cloud-controller \
    --disable metrics-server \
    --write-kubeconfig-mode=644 \
    --disable local-storage \
    --node-name="$(hostname -f)" \
    --cluster-cidr="10.244.0.0/16" \
    --kube-controller-manager-arg="address=0.0.0.0" \
    --kube-controller-manager-arg="bind-address=0.0.0.0" \
    --kube-proxy-arg="metrics-bind-address=0.0.0.0" \
    --kube-scheduler-arg="address=0.0.0.0" \
    --kube-scheduler-arg="bind-address=0.0.0.0" \
    --kubelet-arg="cloud-provider=external" \
    --token="<your_secret_token>" \
    --tls-san="$(hostname -I | awk '{print $2}')" \
    --flannel-iface=ens10
```

You can read more about the applied options in the K3S documentation. In short:

- We disable the integrated cloud controller because we'll install Hetzner's Cloud Controller Manager in the next step.
- We disable the metrics server to save some memory.
- We disable the local storage because we'll use GlusterFS.
- We set the Cluster CIDR to `10.244.0.0/16` .
- We make Kube Controller, Kube Proxy and Kube Scheduler listen on any address (which is not an issue as we've applied firewall rules and the nodes communicate with each other

using the private network).

- We set the shared secret token to `<your_secret_token>` .
- We add the server's private IPv4 (should be `10.0.0.2`) as an additional subject name to the TLS cert.
- We make Flannel use `ens10` , which should be the interface of our private network.

Step 4.2 - Install Hetzner Cloud Controller Manager

Still on your master node, install the Hetzner Cloud Controller Manager:

```
kubectl -n kube-system create secret generic hcloud --from-literal=token=<hetz
kubectl apply -f https://github.com/hetznercloud/hcloud-cloud-controller-manag
```

Step 4.3 - Install System Upgrade Controller (optional)

The System Upgrade Controller performs automatic updates of K3S. If you want to use this feature, install the controller using this command:

```
kubectl apply -f https://github.com/rancher/system-upgrade-controller/releases
```

Step 4.4 - Install K3S on agent nodes

Now that our K3S server is up and running, SSH into your two agent nodes and run the following command to install the K3S agent and connect it to the server:

```
curl -sfL https://get.k3s.io | K3S_URL=https://10.0.0.2:6443 K3S_TOKEN=<your_s
    --node-name="$(hostname -f)" \
    --kubelet-arg="cloud-provider=external" \
    --flannel-iface=ens10
```

You can read more about the applied options in the K3S documentation. In short:

- We disable the integrated cloud controller because we've already installed Hetzner's Cloud Controller Manager.
- We make Flannel use `ens10`, which should be the interface of our private network.

Step 5 - Install GlusterFS (optional)

GlusterFS is a free and open source software scalable network filesystem. You can use it to

replicate files to all your VMs so that your pods can access their persistent storage no matter which node they are running on. Alternatively, you can take a look at Longhorn or OpenEBS.

Step 5.1 - Prepare all nodes

SSH into all three nodes and run the following command to install, enable and start the Gluster server on all machines:

```
wget -O - https://download.gluster.org/pub/gluster/glusterfs/9/rsa.pub | apt-k
echo deb [arch=amd64] https://download.gluster.org/pub/gluster/glusterfs/9/LAT
apt update && apt install -y glusterfs-server
systemctl enable glusterd && systemctl start glusterd
```

Gluster works with so-called "bricks". A brick is a directory which is controlled by Gluster to manage the replicated file system. This file system is then mounted using GlusterFS. Create the necessary directories:

```
mkdir -p /data/glusterfs/k8s/brick1
mkdir -p /mnt/gluster-k8s
```

Step 5.2 - Set up the cluster

Only on your master node, add the two other nodes as peers:

```
gluster peer probe 10.0.0.3
gluster peer probe 10.0.0.4
```

Verify the peer status on the master and agent nodes:

```
gluster peer status
```

Step 5.3 - Create the volume

On your master node, run the following command to create and start a replicated volume:

```
gluster volume create k8s replica 3 \
    10.0.0.2:/data/glusterfs/k8s/brick1/brick \
    10.0.0.3:/data/glusterfs/k8s/brick1/brick \
    10.0.0.4:/data/glusterfs/k8s/brick1/brick \
    force
gluster volume start k8s
gluster volume info
```

This will create and start a replicated volume named "k8s" with three replicas (our three VMs).

Step 5.4 - Mount the GlusterFS volume

On all three nodes, mount the newly created GlusterFS volume:

```
echo "127.0.0.1:/k8s /mnt/gluster-k8s glusterfs defaults,_netdev 0 0" >> /etc/
mount /mnt/gluster-k8s
```

Step 6 - Set up load balancing

We'll use Hetzner's load balancer for SSL offloading and for routing HTTP requests to your K3S setup.

Step 6.1 - Enable proxy protocol in Traefik

To use the proxy protocol, enable it in your K3S' Traefik configuration by setting the cloud load balancer as a trusted IP address:

```
cat <<EOF > /var/lib/rancher/k3s/server/manifests/traefik-config.yaml
apiVersion: helm.cattle.io/v1
kind: HelmChartConfig
metadata:
  name: traefik
  namespace: kube-system
spec:
  valuesContent: |-
    additionalArguments:
      - "--entryPoints.web.proxyProtocol.trustedIPs=10.0.0.254"
      - "--entryPoints.web.forwardedHeaders.trustedIPs=10.0.0.254"
EOF
```

Step 6.2 - Create the load balancer

Create the load balancer and attach it to the private network using the static private IP `10.0.0.254` :

```
hcloud load-balancer create --type lb11 --location nbg1 --name lb-kubernetes
hcloud load-balancer attach-to-network --network network-kubernetes --ip 10.0.
```

Add your three VMs as targets and make sure traffic is routed using the private network:

```
hcloud load-balancer add-target lb-kubernetes --server master-1 --use-private-
hcloud load-balancer add-target lb-kubernetes --server node-1 --use-private-ip
hcloud load-balancer add-target lb-kubernetes --server node-2 --use-private-ip
```

Let Hetzner create a managed Let's Encrypt certificate for `<example.com>` and get the certificate ID `<certificate_id>` :

```
hcloud certificate create --domain <example.com> --type managed --name cert-t1
hcloud certificate list
```

Add the HTTP service for `<example.com>` using proxy protocol and enable the health check:

```
hcloud load-balancer add-service lb-kubernetes --protocol https --http-redirec
hcloud load-balancer update-service lb-kubernetes --listen-port 443 --health-c
```

This will route HTTP requests from Hetzner's load balancer (which performs SSL offloading) to your Kubernetes' Traefik reverse proxy, which in turn routed the request to configured ingress routes. Alternatively, you can route incoming HTTP requests directly from Hetzner's load balancer to an exposed service in your Kubernetes cluster, skipping Traefik. I've chosen to use Traefik as it i.e. allows me to set additional HTTP response headers.

Step 7 - Test your setup (optional)

Your K3S setup is now complete. It's time to test your setup by deploying an nginx pod and publish the HTTP service via K3S' integrated Traefik.

Step 7.1 - Write to GlusterFS volume

Create a static `index.html` file:

```
mkdir /mnt/gluster-k8s/webtest1
echo "Hello World!" > /mnt/gluster-k8s/webtest1/index.html
```

Step 7.2 - Deploy a webserver

Create an nginx deployment, mount the GlusterFS volume for the static content, expose HTTP port 80 using a service and create a Traefik ingress route for your domain `<example.com>` :

```
cat <<"EOF" | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webtest1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webtest1
  template:
    metadata:
      labels:
        app: webtest1
    spec:
      volumes:
        - name: volume-webtest1
          hostPath:
            path: "/mnt/gluster-k8s/webtest1"
      containers:
      - image: nginx
        name: nginx
        ports:
        - name: port-nginx
          containerPort: 80
        volumeMounts:
          - mountPath: "/usr/share/nginx/html"
            name: volume-webtest1
```

```
            readOnly: false
---
apiVersion: v1
kind: Service
metadata:
  name: webtest1
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: webtest1
  type: ClusterIP
---
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: webtest1
spec:
  entryPoints:
    - web
  routes:
  - match: Host(`<example.com>`)
    kind: Rule
    services:
    - name: webtest1
      port: 80
EOF
```

Step 7.3 - Access your website

In Hetzner's cloud console, your load balancer should turn to healthy green and you should be able to access your website:
`https://<example.com>`

Conclusion

You have successfully set up a K3S Kubernetes cluster with one server node and two agent nodes. Hetzner's highly available load balancer delivers traffic to your system and performs HTTPS offloading. You're ready to put some workload on your K3S.