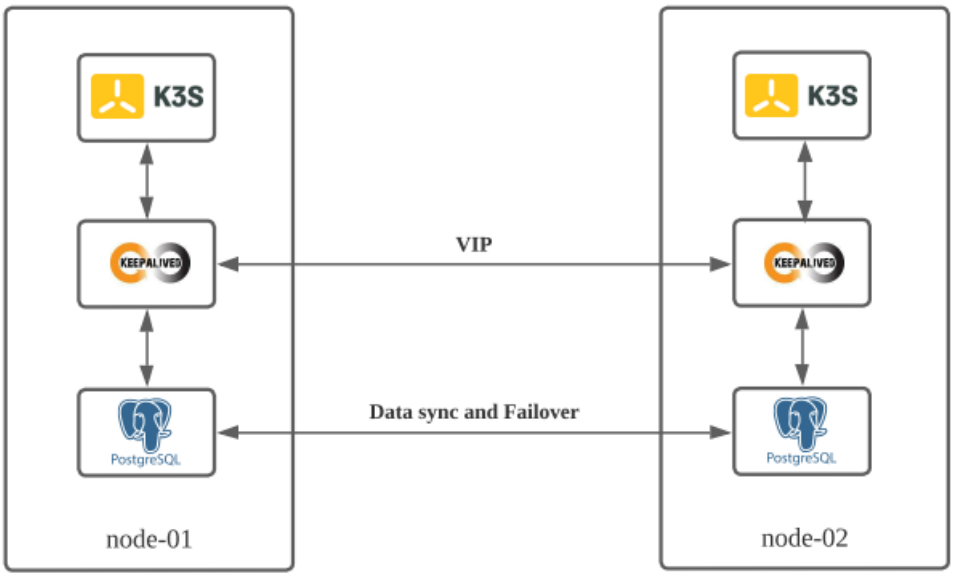
# K3S HA Setup – K3S HA with External etcd (Host Failover Database)

## 1. Setup Requirements

- Two Nodes (Machines)
- Docker installed
- Open Port 443 for k3s
- Open Port 5432 for PostgreSQL
- Rancher (optional)

## 2. High level setup architecture Overview




This setup will be covering K3S HA with External etcd (Host Failover Database).
Two machines node-01 and node-1 will be installed with k3s, keepalived and Postgres. Keepalived will be responsible for creating and maintaining a VIP between the two nodes while Postgres will act as the datastore for k3s.

## 3. Install Postgres database (master/slave)

The database instance will be deployed as a container using docker. The installed databases will be synchronized and able to perform automatic active-passive failover. The database image that we are going to use provides configuration to allow multiple databases across different nodes to be grouped in one cluster and have primary and secondary roles. node-01 will be installed with the master database, while node-02 will be installed as the secondary database.

Create directory for persistent storage on both nodes (docker volume)

```
mkdir -p /var/lib/rancher/postgres
chown 1001:1001 /var/lib/rancher/postgres
```

Install Primary database on node 1:

```
docker run --detach --name pg-0 \
  --publish 5432:5432 \
  --add-host pg-0:<node01-ip-address> \
  --add-host pg-1:<node02-ip-address> \
  --env REPMGR_PARTNER_NODES="pg-0,pg-1" \
  --env REPMGR_NODE_NAME=pg-0 \
  --env REPMGR_NODE_NETWORK_NAME=pg-0 \
  --env REPMGR_PRIMARY_HOST=pg-0 \
  --env REPMGR_PASSWORD=password \
  --env POSTGRESQL_PASSWORD=password \
  --volume /var/lib/rancher/postgres:/bitnami/postgresql/data \
  --restart always \
  bitnami/postgresql-repmgr:latest
```

Verify that container is running smoothly using docker logs pg-0

```
root@node-01:/home/user# docker logs -f --tail 5 pg-0
postgresql-repmgr 01:41:13.03 INFO  ==> ** Starting repmgrd **
[2022-04-05 01:41:13] [NOTICE] repmgrd (repmgrd 5.3.1) starting up
INFO:  set_repmgrd_pid(): provided pidfile is /opt/bitnami/repmgr/tmp/repmgr.pid
[2022-04-05 01:41:13] [NOTICE] starting monitoring of node "pg-0" (ID: 1000)
[2022-04-05 01:41:13] [NOTICE] monitoring cluster primary "pg-0" (ID: 1000)
```

Install backup database on node2:

```
docker run --detach --name pg-1 \
  --publish 5432:5432 \
  --add-host pg-0:<node-01-ip-address> \
  --add-host pg-1:<node-02-ip-address> \
  --env REPMGR_PARTNER_NODES="pg-0,pg-1" \
  --env REPMGR_NODE_NAME=pg-1 \
  --env REPMGR_NODE_NETWORK_NAME=pg-1 \
  --env REPMGR_PRIMARY_HOST=pg-0 \
  --env REPMGR_PASSWORD=password \
  --env POSTGRESQL_PASSWORD=password \
  --restart always \
  --volume /var/lib/rancher/postgres:/bitnami/postgresql/data \
  bitnami/postgresql-repmgr:latest
```

Verify that container is running smoothly using docker logs pg-1

```
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS
abf014a3b9bf   bitnami/postgresql-repmgr:latest   "/opt/bitnami/script…"   3 hours ago   Up 3 hours
```

```
root@node-02:/home/user# docker logs -f --tail 10 pg-1
postgresql-repmgr 01:45:47.79 INFO  ==> Validating settings in POSTGRESQL_* env vars..
postgresql-repmgr 01:45:47.79 INFO  ==> Querying all partner nodes for common upstream node...
postgresql-repmgr 01:45:47.86 INFO  ==> Auto-detected primary node: 'pg-0:5432'
postgresql-repmgr 01:45:47.87 INFO  ==> Preparing PostgreSQL configuration...
postgresql-repmgr 01:45:47.87 INFO  ==> postgresql.conf file not detected. Generating it...
postgresql-repmgr 01:45:47.97 INFO  ==> Preparing repmgr configuration...
postgresql-repmgr 01:45:47.98 INFO  ==> Initializing Repmgr...
postgresql-repmgr 01:45:47.99 INFO  ==> Waiting for primary node...
postgresql-repmgr 01:45:48.03 INFO  ==> Rejoining node...
postgresql-repmgr 01:45:48.03 INFO  ==> Cloning data from primary node...
```

# 4. Install keepalived (Virtual IP address)

For this installation, node-01 will have the master configuration.

**Example of the keepalived configuration (Master):**

```
docker run -d --name keepalived --restart=always \
  --cap-add=NET_ADMIN --cap-add=NET_BROADCAST --cap-add=NET_RAW --net=host \
  -e KEEPALIVED_UNICAST_PEERS="#PYTHON2BASH:['<node01-ipaddress>', '<node02-ipaddress>']" \
  -e KEEPALIVED_VIRTUAL_IPS=<virtual-ip-address> \
  -e KEEPALIVED_STATE="MASTER"  \
  -e KEEPALIVED_INTERFACE="<interface>"\
  -e KEEPALIVED_PRIORITY=200\
  osixia/keepalived:2.0.20
```

```
root@node-01:/home/user# docker run -d --name keepalived --restart=always \
>    --cap-add=NET_ADMIN --cap-add=NET_BROADCAST --cap-add=NET_RAW --net=host \
>    -e KEEPALIVED_UNICAST_PEERS="#PYTHON2BASH:['192.168.20.2', '192.168.20.3']" \
>    -e KEEPALIVED_VIRTUAL_IPS=192.168.20.20 \
>    -e KEEPALIVED_STATE="MASTER"  \
>    -e KEEPALIVED_INTERFACE="enp0s8"\
>    -e KEEPALIVED_PRIORITY=200\
>    osixia/keepalived:2.0.20
```

Node-02 will have the worker configuration.

**Example of the keepalived configuration (worker)**

```
docker run -d --name keepalived --restart=always \
  --cap-add=NET_ADMIN --cap-add=NET_BROADCAST --cap-add=NET_RAW --net=host \
  -e KEEPALIVED_UNICAST_PEERS="#PYTHON2BASH:['<node01-ipaddress>', 'node-02-ipaddress>']" \
  -e KEEPALIVED_VIRTUAL_IPS=<virtual-ip-address> \
  -e KEEPALIVED_PRIORITY=100\
  -e KEEPALIVED_STATE="WORKER"\
  -e KEEPALIVED_INTERFACE="<interface>"\
  osixia/keepalived:2.0.20
```

```
root@node-02:/home/user# docker run -d --name keepalived --restart=always \
>    --cap-add=NET_ADMIN --cap-add=NET_BROADCAST --cap-add=NET_RAW --net=host \
>    -e KEEPALIVED_UNICAST_PEERS="#PYTHON2BASH:['192.168.20.2', '192.168.20.3']" \
>    -e KEEPALIVED_VIRTUAL_IPS=192.168.20.20 \
>    -e KEEPALIVED_PRIORITY=100\
>    -e KEEPALIVED_STATE="WORKER"\
>    -e KEEPALIVED_INTERFACE="enp0s8"\
>    osixia/keepalived:2.0.20
```

Ping the VIP address to verify that it is up.

```
root@node-02:/home/user# ping 192.168.20.20
PING 192.168.20.20 (192.168.20.20) 56(84) bytes of data.
64 bytes from 192.168.20.20: icmp_seq=1 ttl=64 time=1.31 ms
64 bytes from 192.168.20.20: icmp_seq=2 ttl=64 time=1.50 ms
64 bytes from 192.168.20.20: icmp_seq=3 ttl=64 time=2.25 ms
^C
--- 192.168.20.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.313/1.686/2.246/0.402 ms
```

# 5. Launch Kubernetes Master nodes

This setup requires k3s master nodes to be installed in two machines. It is important to include the correct datastore endpoint to allow the k3s to know how to connect to the datastore. The token parameter is optional and will be randomly generated if not specified. The token will be required to join any additional nodes.

Command to install k3s master node with external database:

Run in node-01 machine

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=<node01-ip-address> --flannel-iface=<interface>" sh -s - server \
  --token=SECRET \
  --datastore-endpoint="postgres://postgres:password@<virtualip>:5432/kine?sslmode=disable"
```

```
root@node-01:/home/user# curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=192.168.20.2 --flannel-iface=enp0s8" sh -s - server \
>   --token=SECRET \
>   --datastore-endpoint="postgres://postgres:password@192.168.20.20:5432/kine?sslmode=disable"
[INFO]  Finding release for channel stable
[INFO]  Using v1.22.7+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.22.7+k3s1/sha256sum-amd64.txt
[INFO]  Skipping binary downloaded, installed k3s matches hash
[INFO]  Skipping installation of SELinux RPM
[INFO]  Skipping /usr/local/bin/kubectl symlink to k3s, already exists
[INFO]  Skipping /usr/local/bin/crictl symlink to k3s, already exists
[INFO]  Skipping /usr/local/bin/ctr symlink to k3s, command exists in PATH at /usr/bin/ctr
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
```

Run in node-02 machine

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=<node02-ip-address> --flannel-iface=<interface>" sh -s - server \
  --token=SECRET \
  --datastore-endpoint="postgres://postgres:password@<virtualip>:5432/kine?sslmode=disable"
```

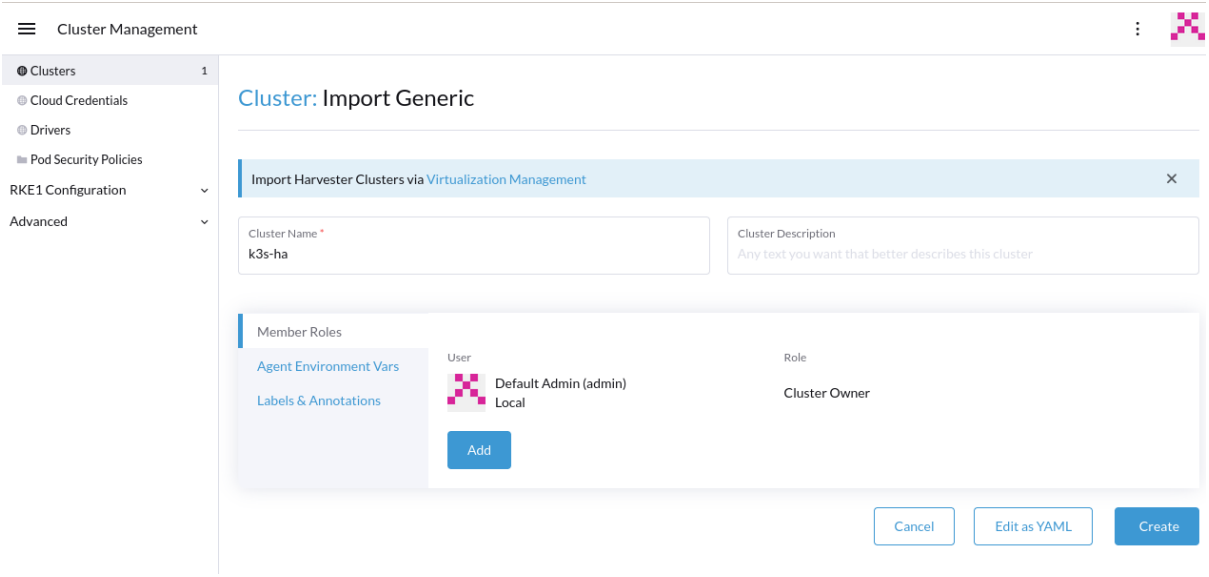
```
root@node-02:/home/user# curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=192.168.20.3 --flannel-iface=enp0s8" sh -s - server \
>   --token=SECRET \
>   --datastore-endpoint="postgres://postgres:password@192.168.20.20:5432/kine?sslmode=disable"
[INFO]  Finding release for channel stable
[INFO]  Using v1.22.7+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.22.7+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.22.7+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Skipping installation of SELinux RPM
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Skipping /usr/local/bin/ctr symlink to k3s, command exists in PATH at /usr/bin/ctr
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
```

Verify that k3s cluster is successfully installed on both nodes:

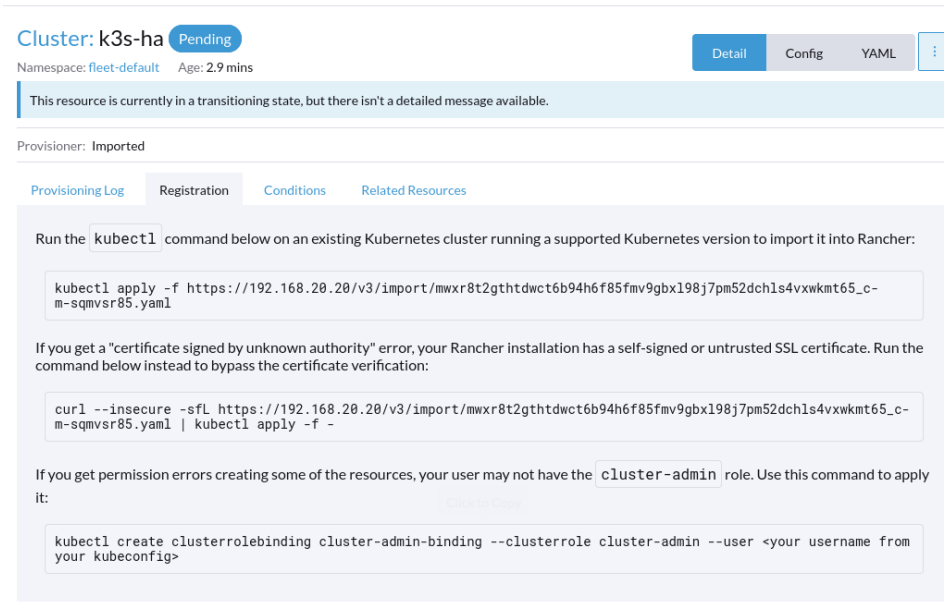
```
root@node-02:/home/user# kubectl get node
NAME      STATUS   ROLES                  AGE     VERSION
node-01   Ready    control-plane,master   7m28s   v1.22.7+k3s1
node-02   Ready    control-plane,master   5m36s   v1.22.7+k3s1
```

# 6. Optional (Import to Rancher)

If Rancher is installed, to import the existing cluster to Rancher, go to Cluster Management -> Import Existing -> Generic to import existing cluster to Rancher




After creation of cluster on Rancher run the kubectl command on node-01



# 7. Create Stateless application Deployment, Service, and Ingress for testing

In this step, we will be creating a Kubernetes deployment, service, and ingress to test the HA failover. The image used in this example is only compatible with AMD devices. Create hello-world.yaml file in node-01 with the following configuration.

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: hello-world-service
            port:
              number: 80
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world-service
spec:
  ports:
    - port: 80
      protocol: TCP
  selector:
    app:  hello-world

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-rancher
spec:
  selector:
    matchLabels:
      app: hello-world
  replicas: 1
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-world-rancher
        image: rancher/hello-world
        ports:
        - containerPort: 80
```

Apply configuration using the following command in the same directory as the hello-world.yaml file.

```
kubectl apply -f hello-world.yaml
```

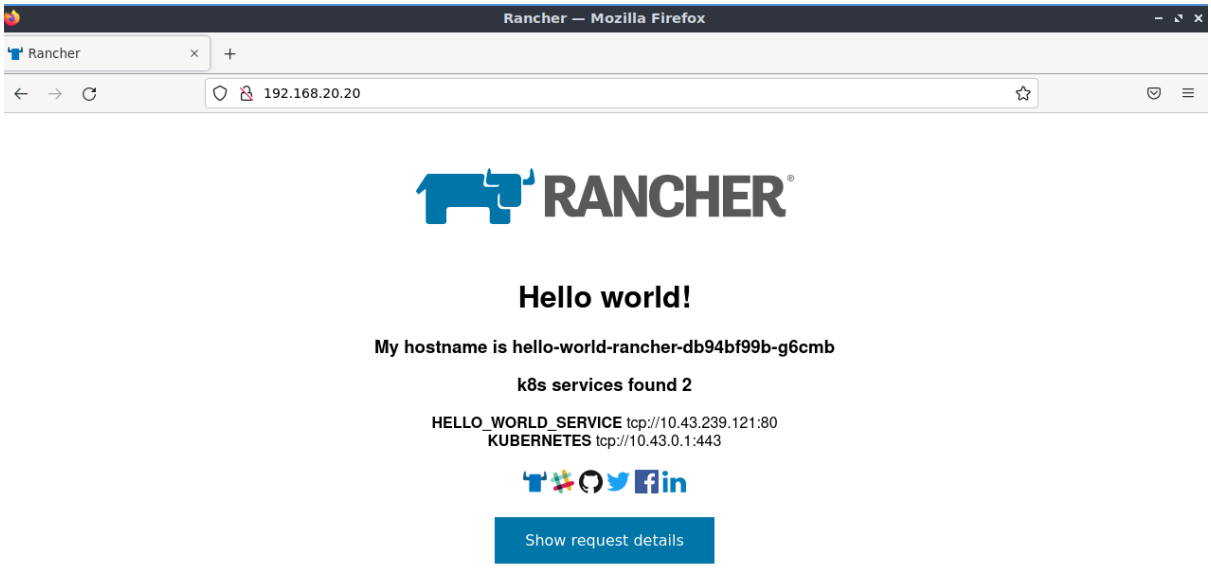
```
root@node-01:/home/user# kubectl apply -f hello-world.yaml
ingress.networking.k8s.io/hello-world-ingress created
service/hello-world-service unchanged
deployment.apps/hello-world-rancher created
```

Verify ingress, service and deployment is working well

```
kubectl get deployment -o wide
kubectl get ingress
kubectl get svc
```

```
root@node-01:/home/user# kubectl get deployment -o wide
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS           IMAGES                SELECTOR
hello-world-rancher   1/1     1            1           6m21s   hello-world-rancher   rancher/hello-world   app=hello-world
root@node-01:/home/user# kubectl get ingress
NAME                 CLASS    HOSTS   ADDRESS                     PORTS   AGE
hello-world-ingress   <none>   *       192.168.20.2,192.168.20.3   80      6m25s
root@node-01:/home/user# kubectl get svc -o wide
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE    SELECTOR
kubernetes           ClusterIP   10.43.0.1       <none>        443/TCP   19m    <none>
hello-world-service   ClusterIP   10.43.239.121   <none>        80/TCP    8m4s   app=hello-world
```

Access the deployed web service in the Kubernetes cluster using the virtual ip address through a browser from a client in the same network.




## 8. High Availability Testing and Recovery

Check which node the workload is being deployed

```
kubectl get pod -o wide
```

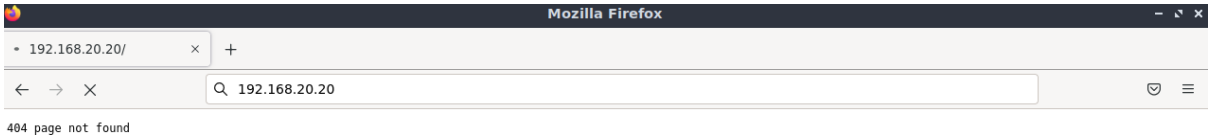
```
root@node-01:/home/user# kubectl get pod -o wide
NAME                                     READY   STATUS    RESTARTS   AGE   IP          NODE
hello-world-rancher-db94bf99b-g6cmb   1/1     Running   0          27m   10.42.1.4   node-02
```

In this scenario, the workload is deployed in node-02. Thus, we will be shutting down the k3s service in node-02 to simulate a machine failure.

```
systemctl stop k3s
```

```
root@node-02:/home/user# systemctl stop k3s
```

Unable to access web service after k3s is shut down.



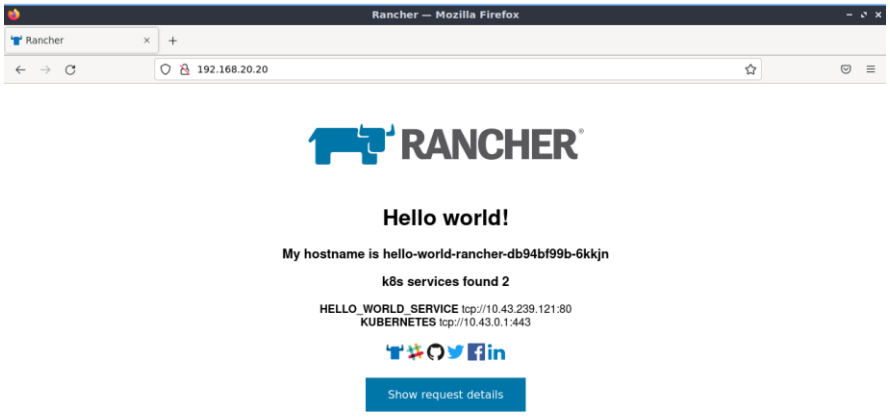

After around 10 minutes, after the Kubernetes control plane will detect that node-02 is down, it will automatically recreate the pod in node-01. We can continue to access the deployed web service after the pod is recreated in node-01

```
kubectl get node
kubectl get pod -o wide
```

```
root@node-01:/home/user# kubectl get node
NAME      STATUS     ROLES                  AGE   VERSION
node-02   NotReady   control-plane,master   48m   v1.22.7+k3s1
node-01   Ready      control-plane,master   50m   v1.22.7+k3s1
```

```
root@node-01:/home/user# kubectl get pod -o wide
NAME                                     READY   STATUS        RESTARTS   AGE     IP          NODE
hello-world-rancher-db94bf99b-g6cmb   1/1     Terminating   0          32m     10.42.1.4   node-02
hello-world-rancher-db94bf99b-6kkjn   1/1     Running       0          3m32s   10.42.0.9   node-01
```

Access the deployed web service in the Kubernetes cluster once again



Lastly we will resume the k3s service on node-02 to simulate the recovery of the machine and check the status of the Kubernetes cluster.

```
systemctl start k3s
```

node-02 will recover and be up on the Kubernetes cluster again.

```
kubectl get node
```

```
root@node-02:/home/user# kubectl get node
NAME      STATUS   ROLES                  AGE   VERSION
node-01   Ready    control-plane,master   51m   v1.22.7+k3s1
node-02   Ready    control-plane,master   49m   v1.22.7+k3s1
```