EXPERIMENTAL HA Kubernetes Cluster for SOHO use on UBUNTU + k3sup + k3s base

Project Goal:

1. self hosted HA fault tolerant (multi master) Kubernetes Cluster
2. NO SINGLE POINT OF FAILURE!
3. no “external” managed Services required (no separate database/load balancer)
4. using 3 UBUNTU Server VMs
5. easy and fast to setup
6. One site only
7. 🔧 Fix Me! shared storage/volumes

Kinda important stuff to read and know about:

- ubuntu server 20.04
- ssh passwordless public key authentiction
- virtualbox or kvm virtualisation platform or bare metal if available
- k3s
  - https://rancher.com/docs/k3s/latest/en/installation/airgap/
  - https://rancher.com/blog/2020/k3s-high-availability
  - https://rancher.com/docs/k3s/latest/en/
  - https://rancher.com/docs/k3s/latest/en/installation/ha-embedded/
- k3sup https://github.com/alexellis/k3sup
- Docker registry:
  - https://docs.docker.com/registry/deploying/#considerations-for-air-gapped-registries
  - https://docs.docker.com/registry/deploying/#run-an-externally-accessible-registry

Installation / Setup:

1. Setup 2-3 UBUNTU Server 20.04 VMs who can be reached via ssh. Must have unique hostnames and IP addresses of course. Make sure your IP addresses are FIXED and not dynamic, because later we will require a DNS record to all fixed IP addresses of a cluster.
2. Configure your VMs static IP addresses. In my scenario i will use **.101** for the first master node, **.102** for the second node etc.. :

```
cat << 'EOF' > /etc/netplan/55-fixed-ip.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      addresses:
      - 192.168.0.101/24
```

```
      gateway4: 192.168.0.1
      nameservers:
        addresses:
        - 192.168.0.1
        search:
        - lan
EOF

reboot ; and do test
```

3. Setup passwordless ssh login to those VMs from your admin workstation
4. Optional: Have fqdn available for each vm instead of plain IP addresses
5. On 1st VM (k3s-master1) login as root:

```
# generate a ssh key pair
ssh-keygen -f /root/.ssh/id_rsa -q -N ""

# distribute the ssh public key to any other master VM

# copy to ourselfs too
ssh-copy-id -o StrictHostKeyChecking=no root@k3s-master1.lan

ssh-copy-id -o StrictHostKeyChecking=no  root@k3s-master2.lan
ssh-copy-id -o StrictHostKeyChecking=no  root@k3s-master3.lan
```

6. Prepare data directories (local storage) for **k3s (alias rancher)** and **kubelet** below **/data/** on every node, since we don't want it to fill up our ROOTFS. That's if you mounted another volume under **/data/** of course :

```
for n in 1 2 3 ; do \
  ssh root@k3s-master$n.lan '\
    rm -rv /data/{k3s,kubelet} ; \
    rm -rv /var/lib/{rancher,kubelet} ; \
    mkdir -vp /data/{k3s,kubelet} ; \
    ln -sv /data/kubelet/ /var/lib/kubelet ; \
    ln -sv /data/k3s/ /var/lib/rancher ; \
    # disable all swap ; \
    swapoff --all ; \
    sed -i "/\sswap\s/s/^/#/" /etc/fstab ; \
    ' ; \
done # prepare data dir
```

7. Prepare a directory for placing local copies of git repos and bins on Admin's host or 1st master node:

```
cd
mkdir -p install/k3sup
cd !$
```

8. Clone k3sup git repo localy, so we keep the sources and docs of what we use next:

```
export GITREPO=alexellis/k3sup
version=$(curl -sI https://github.com/$GITREPO/releases/latest | grep -
i "location:" | awk -F"/" '{ printf "%s", $NF }' | tr -d '\r')
git clone https://github.com/$GITREPO/ $version
cd $version

# show latest tag/version available in local copy
git describe --abbrev=0 --tags

# manually save the matching pre-compiled binary with it
wget https://github.com/$GITREPO/releases/download/$version/k3sup
chmod -c u+x ./k3sup

# OPTIONAL: install binary if prefered
cp -v ./k3sup /usr/local/bin/

# show version
k3sup version
```

9. Optional: If you want to save the git repo of k3s which has been used here try this:

```
cd
mkdir -p install/k3s
cd !$

export GITREPO=k3s-io/k3s
version=$(curl -sI https://github.com/$GITREPO/releases/latest | grep -
i "location:" | awk -F"/" '{ printf "%s", $NF }' | tr -d '\r')
git clone https://github.com/$GITREPO/ $version
cd $version


# show latest tag/version available in local copy
git describe --abbrev=0 --tags



# download the required images tar ball that
# matches the version for later "offline" installation
#
wget
https://github.com/k3s-io/k3s/releases/download/$version/k3s-airgap-ima
ges-amd64.tar

#
# manually install required images where k3s expects them:
#
mkdir -p /var/lib/rancher/k3s/agent/images/
cp -v k3s-airgap-images-amd64.tar /var/lib/rancher/k3s/agent/images/
```

```
# manually save the matching pre-compiled binary with it
wget https://github.com/$GITREPO/releases/download/$version/k3s
chmod -c u+x ./k3s

# show version of binary
./k3s --version

# install binary
cp -v ./k3s /usr/local/bin/
```

10. Now let k3sup do its magic and install the 1st master node “locally”. Since our “cluster” is supposed to be accessible by its own fqdn hostname **cloud.lan** i added its future hostname to the **–tls-san** option. That causes k3s to generate self signed tls certificates which also contain a reference to this prefered hostname, so the certificates are valid for this hostname as well:

```
cd

k3sup install \
  --print-command \
  --tls-san cloud.lan \
  --cluster \
  --host $(hostname -f) \
  --host-ip $(hostname -f) \
  --k3s-extra-args '\
    --cluster-domain cloud.lan \
    '
```

11. Lets test if our first master node is up and running:

```
export KUBECONFIG=/root/kubeconfig
kubectl config set-context default
kubectl get node -o wide

#example of expected output:
#
#root@k3s-master1:~# export KUBECONFIG=/root/kubeconfig
#root@k3s-master1:~# kubectl config set-context default
#Context "default" modified.

#root@k3s-master1:~# kubectl get node -o wide
#NAME              STATUS   ROLES         AGE   VERSION
INTERNAL-IP    EXTERNAL-#IP   OS-IMAGE             KERNEL-VERSION
CONTAINER-RUNTIME
#k3s-master1.lan   Ready    etcd,master   94s   v1.19.7+k3s1
192.168.0.60   <none>        #Ubuntu 20.04.2 LTS   5.4.0-65-generic
containerd://1.4.3-k3s1
#root@k3s-master1:~#
#
```

12. For convenience lets install some BASH completion code to use with **kubectl**, which kubectl can

provide to us :

```
# repeat on every node you want it to be available later
# will be available on next login/shell

kubectl completion bash > /etc/bash_completion.d/kubectl_completion
```

13. Install and add more master nodes to existing master/cluster (from 1st master nide!) :

```
k3sup join --server --server-host k3s-master1.lan --host k3s-
master2.lan
k3sup join --server --server-host k3s-master1.lan --host k3s-
master3.lan

# check nodes status with
kubectl get node
```

14. Check “cluster” (control plane) status and cluster IP address:

```
kubectl get rc,services

# example for expected result
#
#NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
#service/kubernetes   ClusterIP   10.43.0.1    <none>        443/TCP
97m
#

# show cluster "endpoints"

kubectl describe endpoints

#
#Name:         kubernetes
#Namespace:    default
#Labels:       endpointslice.kubernetes.io/skip-mirror=true
#Annotations:  <none>
#Subsets:
#  Addresses:          192.168.0.101,192.168.0.102,192.168.0.103
#  NotReadyAddresses:  <none>
#  Ports:
#    Name   Port  Protocol
#    ----   ----  --------
#    https  6443  TCP
#
#Events:  <none>
#

kubectl get node
```

```
#
#NAME                STATUS   ROLES          AGE     VERSION
#k3s-master1.lan     Ready    etcd,master    2m34s   v1.19.7+k3s1
#k3s-master2.lan     Ready    etcd,master    77s     v1.19.7+k3s1
#k3s-master3.lan     Ready    etcd,master    48s     v1.19.7+k3s1
#
```

15. So our Kubernetes Cluster is running now.

Monitoring and navigating our Kubernetes cluster

Task	**Command**
Check CPU and MEMORY (load) usage across the cluster/all nodes:	`kubectl top nodes`

Check CPU and MEMORY usage of PODS:	`# across all namespaces`
	`kubectl top pod -A`
	
	`# in default namespace only`
	`kubectl top pod`

Get an overview of whats going on and whats already installed, on your Kubernetes cluster. Again **-A** means **across ALL namespaces**:	`kubectl get all -A`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`
 :	`FIXME`

kubernetes, k3s, k3sup, high, availability, ha, ubuntu, server, vm, vms