kubernetes(k8s) HA 구성 설치

LoadBalancer for HA설치 방법

다음 중 하나의 방법을 선택하여 설치(kube-vip 추천)

Keepalived & Haproxy 조합 LoadBalancer

keepalived 설치

설치

```
sudo apt-get install keepalived
```

설정

`keepalived.conf`

- 각 control plane node에 다음 설정 적용

```
sudo tee /etc/keepalived/keepalived.conf <<EOF
global_defs {
    router_id LVS_DEVEL
}
vrrp_script check_apiserver {
  script "/etc/keepalived/check_apiserver.sh"
  interval 3
  weight -2
  fall 10
  rise 2
}

vrrp_instance VI_1 {
    state ${STATE}
    interface ${INTERFACE}
    virtual_router_id ${ROUTER_ID}
    priority ${PRIORITY}
    authentication {
        auth_type PASS
        auth_pass ${AUTH_PASS}
    }
    virtual_ipaddress {
        ${APISERVER_VIP}
    }
    track_script {
```

```
        check_apiserver
    }
}
EOF
```

- `${STATE}` 는 하나의 `MASTER` 와 다수의 `BACKUP` 으로 구성된다. virtual IP는 처음 `MASTER` 에 할당될 것이다.
- `${INTERFACE}` 는 가상 IP(예: eth0)의 협상에 참여하는 네트워크 인터페이스입니다.
- `${ROUTER_ID}` 는 모든 `keepalived` 클러스터 간에 동일해야 하며, 동일한 서브넷의 모든 클러스터 사이에서 유일해야 합니다. 많은 분산에서 51로 설정합니다
- `${PRIORITY}` 는 백업보다 control plane node에서 더 높아야 합니다.
  - control plane: 101
  - backup: 100
- `${AUTH_PASS}` 는 모든 `keepalived` 클러스터에서 같아야 합니다.(예: 42)
- `${APISERVER_VIP}` 는 `keepalived` 클러스터 간에 결정되는 가상 IP 주소 입니다.(예: 192.168.0.80)

`check_apiserver.sh`

- 각 control plane node에 다음 명령어로 bash script 파일 생성

```
sudo tee /etc/keepalived/check_apiserver.sh <<EOF
#!/bin/sh

errorExit() {
    echo "*** $*" 1>&2
    exit 1
}

curl --silent --max-time 2 --insecure
https://localhost:${APISERVER_DEST_PORT}/ -o /dev/null || errorExit
"Error GET https://localhost:${APISERVER_DEST_PORT}/"
if ip addr | grep -q ${APISERVER_VIP}; then
    curl --silent --max-time 2 --insecure
https://${APISERVER_VIP}:${APISERVER_DEST_PORT}/ -o /dev/null ||
errorExit "Error GET https://${APISERVER_VIP}:${APISERVER_DEST_PORT}/"
fi
EOF
```

- `${APISERVER_VIP}` 는 `keepalived` 클러스터 간에 결정되는 가상 IP 주소 입니다.(예: 192.168.0.80)
- `${APIZERVER_DEST_PORT}` Kubernetes가 API Server와 통신할 포트입니다.(예: 36443)

설정 적용

```
sudo systemctl enable keepalived --now
```

haproxy 설치

설치

```
sudo apt-get install haproxy
```

설정

`haproxy.conf`

- 각 control plane node에 다음 설정 적용

```
cat << EOF >> /etc/haproxy/haproxy.conf


frontend apiserver
    bind *:${APISERVER_DEST_PORT}
    mode tcp
    option tcplog
    default_backend apiserverbackend

backend apiserverbackend
    option httpchk GET /healthz
    http-check expect status 200
    mode tcp
    option ssl-hello-chk
    balance     roundrobin
    server ${HOST1_ID} ${HOST1_ADDRESS}:${APISERVER_SRC_PORT} check
    server ${HOST2_ID} ${HOST2_ADDRESS}:${APISERVER_SRC_PORT} check
        ...
EOF
```

- `${APIZERVER_DEST_PORT}` : Kubernetes가 API Server와 통신할 포트입니다. (예: 36443)
- `${APIISERVER_SRC_PORT}` : API Server 인스턴스에서 사용하는 포트 (예: 6443)
- `${HOST1_ID}` : 첫 번째 로드 밸런싱 API Server 호스트의 기호 이름
- `${HOST1_ADDRESS}` : 첫 번째 로드 밸런싱 API Server 호스트의 확인 가능한 주소(DNS 이름, IP 주소)

설정 적용

```
sudo systemctl enable haproxy --now
```

kube-vip LoadBalancer 사용

kube-vip 설치 방법

첫번째 control plane node

1. `kube-vip.yaml` 생성

```
sudo mkdir -p /etc/kubernetes/manifests/

sudo apt-get install jq

export VIP=192.168.0.80
export INTERFACE=<interface> # ip addr 명령어로 확인
KVVERSION=$(curl -sL https://api.github.com/repos/kube-vip/kube-
vip/releases | jq -r ".[0].name")

docker run --network host --rm ghcr.io/kube-vip/kube-vip:$KVVERSION
\
manifest pod \
--interface $INTERFACE \
--vip $VIP \
--port 6443 \
--controlplane \
--arp \
--leaderElection | sudo tee /etc/kubernetes/manifests/kube-vip.yaml
```

2. 이후 `kubeadm init ...` 을 통해 k8s가 실행되면 `/etc/kubernetes/manifests/` 안의 `kube-vip.yaml` 의 설정이 자동 적용되며, kube-vip가 설치 됨

추가 control plane node

1. `kubeadm join ...` 을 통해 추가 control plane 설치
2. `kube-vip.yaml` 생성

```
export VIP=192.168.0.80
export INTERFACE=<interface> # ip addr 명령어로 확인
KVVERSION=$(curl -sL https://api.github.com/repos/kube-vip/kube-
vip/releases | jq -r ".[0].name")

docker run --network host --rm ghcr.io/kube-vip/kube-vip:$KVVERSION
\
manifest pod \
--interface $INTERFACE \
--vip $VIP \
--port 6443 \
--controlplane \
--arp \
--leaderElection | sudo tee /etc/kubernetes/manifests/kube-vip.yaml
```

3. `kube-vip.yaml`이 생성되면 kube-vip 자동으로 설치됨

Docker engine 설치

script 사용 자동 설치

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

수동 설치

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

sudo 없이 docker 실행하도록 설정

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

nvidia-container-toolkit 설치

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-
key add -
curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.list | sudo tee
```

```
/etc/apt/sources.list.d/libnvidia-container.list

sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
```

docker 기본 runtime을 nvidia로

- `/etc/docker/daemon.json` 내용에 `"default-runtime": "nvidia"` 추가

```
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": "/usr/bin/nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
```

cgroup driver 설정

- docker 20.10 버전 이후에서는 cgroup v2를 지원하며, default cgroup driver로 systemd 사용 [1]

현재 사용중인 cgroup 확인

- 확인 후 systemd가 아니면 추가 설정 진행

```
docker info | grep -i cgroup
```

실행결과

```
 Cgroup Driver: systemd
 Cgroup Version: 2
  cgroupns
```

설정 파일 경로 확인

```
sudo systemctl status docker | grep docker.service
```

실행결과

```
● docker.service - Docker Application Container Engine
```

```
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
preset: enabled)
     CGroup: /system.slice/docker.service
```

설정 변경

```
sudo vi /lib/systemd/system/docker.service
```

- ExecStart 구문에 `--exec-opt native.cgroupdriver=systemd` 추가

```
ExecStart=/usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock --exec-opt
native.cgroupdriver=systemd
```

docker service 재시작

```
sudo systemctl restart docker
```

---

1. https://docs.docker.com/config/containers/runmetrics/#running-docker-on-cgroup-v2 ↩

cri-docker 설치

```
DISTRIBUTION=$(lsb_release -cs)

URL=$(curl -s https://api.github.com/repos/Mirantis/cri-
dockerd/releases/latest | grep ubuntu-${DISTRIBUTION}_amd64.deb | grep
browser_download_url | cut -d \" -f 4)

wget -O cri-docker.deb $URL

sudo dpkg -i cri-docker.deb
```

kubeadm, kubelet, kubectl 설치

사전 점검사항

- Unique hostname, MAC address, product_uuid[1]
  - `ip link`로 MAC address 확인

    - `sudo cat /sys/class/dmi/id/product_uuid`로 product_uuid 확인
- 다음 Port가 사용가능한지[2]
    - `nc 127.0.0.1 [port]`명령어로 확인
    - Control plane: 6443, 2379~2380, 10250, 10259, 10257
    - Worker node: 10250, 30000~32767
- swapoff 확인[3]
    - `sudo swapoff -a` 명령어로 임시로 설정 적용
    - `/etc/fstab`에서 swap 관련 부분 주석처리(부팅시 설정 적용)
- docker 설치 확인

설치

```
sudo apt-get update
# apt-transport-https may be a dummy package; if so, you can skip that
package
sudo apt-get install -y apt-transport-https ca-certificates curl

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

---

1. Installing kubeadm | Kubernetes↩
2. Ports and Protocols | Kubernetes↩
3. swapoff↩

k8s HA control plane 설정

k8s HA control plane 생성

Master

```
sudo kubeadm init --control-plane-endpoint
"LOAD_BALANCER_DNS:LOAD_BALANCER_PORT" --upload-certs --cri-socket
unix:///var/run/cri-dockerd.sock
```

```
To start using your cluster, you need to run the following as a regular
user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed
at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of the control-plane node running the
following command on each as root:

  kubeadm join 192.168.0.80:36443 --token t5zrdw.sq1lh4kn2igggh0u \
        --discovery-token-ca-cert-hash
sha256:93a3ea93ec496cc9bad0ad2082aa274c02ebfc25219f4681cd9295d21d55389a
\
        --control-plane --certificate-key
9142302bd4c1d51c842fdb2fcedd7517747ebe72b08440c44fd9cffbb4ded70a

Please note that the certificate-key gives access to cluster sensitive
data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If
necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs
afterward.

Then you can join any number of worker nodes by running the following on
each as root:

kubeadm join 192.168.0.80:36443 --token t5zrdw.sq1lh4kn2igggh0u \
        --discovery-token-ca-cert-hash
sha256:93a3ea93ec496cc9bad0ad2082aa274c02ebfc25219f4681cd9295d21d55389a
```

Slave

```
kubeadm join 192.168.0.80:36443 --token t5zrdw.sq1lh4kn2igggh0u \
        --discovery-token-ca-cert-hash
sha256:93a3ea93ec496cc9bad0ad2082aa274c02ebfc25219f4681cd9295d21d55389a
```

```
\
        --control-plane --certificate-key
9142302bd4c1d51c842fdb2fcedd7517747ebe72b08440c44fd9cffbb4ded70a\
        --cri-socket unix:///var/run/cri-dockerd.sock
```

Helm 설치

Ubuntu

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee
/usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/helm.gpg]
https://baltocdn.com/helm/stable/debian/ all main" | sudo tee
/etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

MacOS

```
brew install helm
```

각 CNI 설치 방법

- 이들 중 하나만 설치하면 됨

flannel 설치

cidr 설정

```
sudo vi /etc/kubernetes/manifests/kube-controller-manager.yaml
# at command ,add
# --allocate-node-cidrs=true
# --cluster-cidr=10.244.0.0/16
```

helm으로 설치

```
kubectl create ns kube-flannel
kubectl label --overwrite ns kube-flannel pod-
security.kubernetes.io/enforce=privileged

helm repo add flannel https://flannel-io.github.io/flannel/
```

```
helm install flannel --set podCidr="10.244.0.0/16" --namespace
kube-flannel flannel/flannel
```

Calico 설치 방법

1. Download

```
curl
https://raw.githubusercontent.com/projectcalico/calico/v3.26
.4/manifests/calico.yaml -O
```

2. 사용할 Pod CIDR 수정(기본값: `192.168.0.0/16`)
3. 기타 값 수정
4. Apply

```
kubectl apply -f calico.yaml
```

local-path-provisioner 설치

- 최신 버전 확인[1]

```
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-
provisioner/v0.0.25/deploy/local-path-storage.yaml
```

---

1. https://github.com/rancher/local-path-provisioner#installation ↩

Error - bad option; for several filesystems (e.g. nfs, cifs) you might need a mount.<type> helper program.

각 노드에 nfs-common 라이브러리 설치 필수

```
sudo apt-get install nfs-common
```

nfs-subdir-external-provisioner 설치

1. Helm 설치
2. nfs-subdir-external-provisioner 설치[1]

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \
--set nfs.server=x.x.x.x \
--set nfs.path=/exported/path

# 기본 storageClass로 설정
kubectl patch storageclass nfs-client -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

---

1. https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner#with-helm ↩

GPU plugin 설치

- 최신 명령어 확인[1]

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.14.3/nvidia-device-plugin.yml
```

---

1. https://github.com/NVIDIA/k8s-device-plugin#enabling-gpu-support-in-kubernetes ↩

too many open files

```
sudo su

cat << EOF >> /etc/sysctl.conf
fs.inotify.max_user_watches = 524288
fs.inotify.max_user_instances = 512
EOF

exit
```

```
sudo sysctl fs.inotify.max_user_watches=524288
sudo sysctl fs.inotify.max_user_instances=512
```

MetalLB 설치 방법

Layer2 mode

1. ARP 설정

```
kubectl edit configmap -n kube-system kube-proxy
```

```
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: "ipvs"
ipvs:
        strictARP: true
```

2. helm을 이용해 설치

```
helm repo add metallb https://metallb.github.io/metallb
helm install -n metallb-system --create-namespace metallb
metallb/metallb
```

BGP mode

- 추후 사용할 일 있으면 추가...

MetalLB 설정 적용 방법

기본

- MetalLB를 설치 후, 서비스 리소스를 `spec.type=LoadBalancer`로 설정하면 자동으로 MetalLB가 적용된다.

특정한 IP 적용

`Service` 리소스에 `spec.loadBalancerIP` 설정

```
kind: Service
spec:
```

```
        loadBalancerIP: 192.168.0.80
```

`Service` 리소스에 `metallb.universe.tf/loadBalancerIPs` annotations 설정

```
kind: Service
metadata:
        name: nginx
        annotations:
                metallb.universe.tf/loadBalancerIPs: 192.168.0.80
```

`Service` 리소스에 `metallb.universe.tf/address-pool` annotations 설정

1. `IPAddressPool` 리소스 생성

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
        name: sandbox
        namespace: metallb-system
spec:
        addresses:
        - 192.168.1.0/24
        - 192.168.0.100-192.168.0.120
```

2. `Service` 리소스 설정

```
kind: Service
metadata:
        name: nginx
        annotations:
                metallb.universe.tf/address-pool: sandbox
```

> - `sandbox`는 `IPAddressPool`로 생성한 리소스의 이름
> - helm을 이용해 설치한 경우 모든 서비스에 `metallb.universe.tf/address-pool: sandbox`가 추가됨[1]
> - 즉, `IPAddressPool` 리소스를 `name: sandbox`로 생성하면 자동으로 서비스에 설정이 적용 됨

---

1. MetalLB, bare metal load-balancer for Kubernetes (universe.tf)↩

Worker node 설치

master node에서 init 성공시 출력되는 join 명령어를 복사하여 worker node에서 실행

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular
user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a Pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  /docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each
node
as root:

  kubeadm join <control-plane-host>:<control-plane-port> --token <token> --
discovery-token-ca-cert-hash sha256:<hash>
```

```
sudo kubeadm join <master node ip>:6443 --token 9x2p2t.md59hbiecwz13vl9  --
discovery-token-ca-cert-hash
sha256:90b50ad5e1c2ea58ee1856960a6b5392dbe8883400846c579e0d0b92e38caf4e --
cri-socket unix:///var/run/cri-dockerd.sock
```

token을 잃어버렸을 시

- control plane에서 실행

```
kubeadm token list
```

```
TOKEN                     TTL  EXPIRES              USAGES
DESCRIPTION            EXTRA GROUPS
8ewj1p.9r9hcjoqgajrj4gi  23h  2018-06-12T02:51:28Z authentication,  The
default bootstrap  system:
                                                  signing
token generated by     bootstrappers:

'kubeadm init'.        kubeadm:
```

```
default-node-token
```

- 기본적으로 token은 24시간 동안만 유효 하므로 이 시간이 지나면 재발급 해야 함

```
kubeadm token create
```

```
5didvk.d09sbcov8ph2amjw
```

`--discovery-token-ca-cert-hash`를 잃어버렸을 시

- control plane에서 실행

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -
outform der 2>/dev/null | \
   openssl dgst -sha256 -hex | sed 's/^.* //'
```

```
8cb2de97839780a412b93877f8507ad6c94f73add17d5d7058e91741c9d5ec78
```

control plane에 작업 할당하기

- 기본적으로 control plane에는 taint가 설정되어 있어 작업할당이 안됨
- traint를 제거함으로써 작업 할당 가능

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```