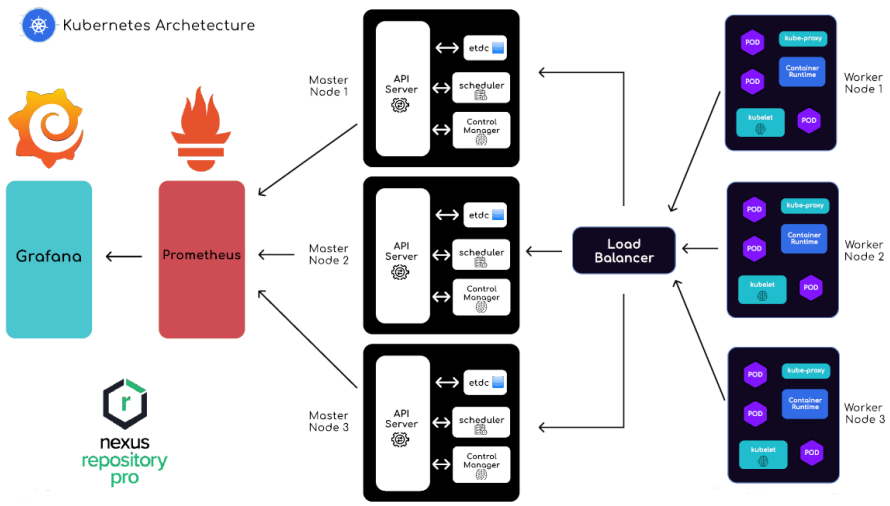
[5조] 내부 etcd Workbook

- Ubuntu `20.04 LTS`
- Docker `20.10.18`
- Kubernetes `v1.24.5`
- memory : `4GB`
- disk : `100GB`
- /dev/sda1 xfs / : `70GB`
- /dev/sda5 swap: : `8GB`
- /dev/sda6 xfs /var/lib/docker : `29372MB`(나머지)

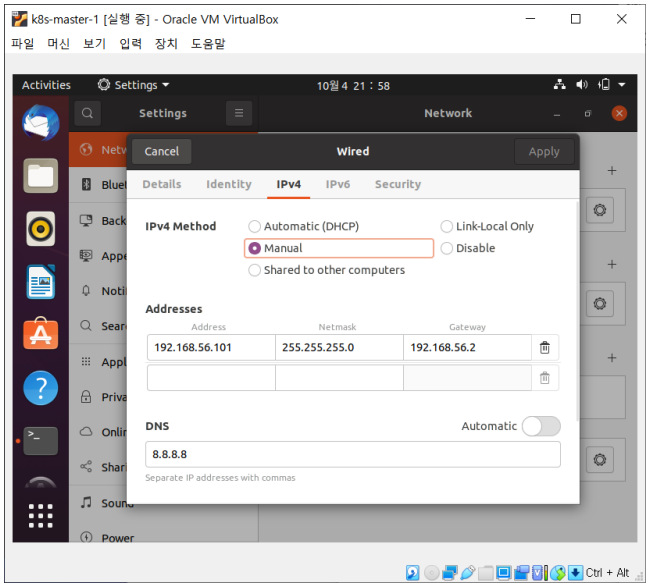
🗺️ Stacked etcd HA Cluster Topology



1️⃣ 서버 셋팅

📌 IP 설정

enp0s8 설정 → 설정 후 enp0s8 한번 껐다 켜주기



📌 기본 설치 및 설정

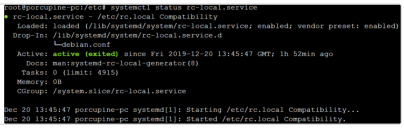
```
# hostname 설정
kevin@master-1:~$ sudo hostnamectl set-hostname master-1

# vim, openssh, net-tools 설치
kevin@master-1:~$ sudo apt -y update
kevin@master-1:~$ sudo apt -y install vim
kevin@master-1:~$ sudo apt -y install openssh-server
kevin@master-1:~$ sudo apt install net-tools
kevin@master-1:~$ sudo reboot

###### putty 연결 ######

# 방화벽 중지
kevin@master-1:~$ sudo apt -y install firewalld
kevin@master-1:~$ sudo systemctl daemon-reload
kevin@master-1:~$ sudo systemctl stop firewalld.service
kevin@master-1:~$ sudo firewall-cmd --reload
FirewallD is not running

# swap off
kevin@master-1:~$ sudo vi /etc/fstab
kevin@master-1:~$ sudo -i
root@master-1:~# swapoff -a
root@master-1:~# echo 0 > /proc/sys/vm/swappiness
root@master-1:~# sed -e '/swap/ s/^#*/#/' -i /etc/fstab
root@master-1:~# exit
logout

# 아래와 같이 swap 주석처리 되어있는지 확인
kevin@master-1:~$sudo vi /etc/fstab
# swap was on /dev/sda5 during installation
#UUID=7d3b72c7-2162-4d28-9210-dae2233515f9 none            swap    sw              0       0

# ntp(network time protocole) - 시간동기화
kevin@master-1:~$ sudo apt -y install ntp
kevin@master-1:~$ sudo systemctl daemon-reload
kevin@master-1:~$ sudo systemctl enable ntp
kevin@master-1:~$ sudo systemctl restart ntp
kevin@master-1:~$ sudo ntpq -p
kevin@master-1:~$ date

# network forward 설정
# why? docker private registry를 이용하기 위해서 적용
kevin@master-1:~$ sudo su -
root@master-1:~# echo '1' > /proc/sys/net/ipv4/ip_forward
root@master-1:~# cat /proc/sys/net/ipv4/ip_forward
1 # 1로 나와야함 !! 0이면 안됨
root@master-1:~# exit
logout
```

💡 **스왑 메모리를 비활성화 해야하는 이유**
- Pod를 할당하고 제어하는 kubelet은 스왑 상황을 처리하도록 설계되지 않음 !
- why? kubernetes에서 가장 기본이 되는 Pod의 컨셉이 필요한 리소스 만큼만 호스트 자원에서 할당 받아 사용한다.
- **따라서,** kubernetes 개발팀이 메모리 스왑을 고려하지 않고 설계했기 때문에 클러스터 노드로 사용할 서버 머신들은 모두 스왑 메모리를 비활성화 해줘야 한다.

💡 **부팅 시 시간 동기화 해결 ::: 부팅 시 자동으로 rdate 명령어 실행**
vi /etc/rc.d/rc.local 파일 아래에 rdate -s time.bora.net 명령을 적어준다.
이 과정을 통해 매번 부팅 시마다 자동으로 시간을 동기화할 수 있다.

😀 만약 rc.local이 존재하지 않는다면(ubuntu 18버전 이후) 아래 블로그를 참조하세요

[리눅스 유용한 팁] 우분투(Ubuntu)에서 rc.local 없을 때, 활성화 시키는 방법!
이번 포스팅에서는 리눅스를 사용하면서 새롭게 부팅될 때 마다 자동으로 먼가를 실행하도록 하고 싶을 때,...
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=sw4r&logNo=221744290073



2️⃣ 로드밸런서 서버 설치

```
kevin@master-1:~$ sudo apt update
kevin@master-1:~$ sudo apt install -y apache2
kevin@master-1:~$ sudo apt install -y haproxy
kevin@master-1:~$ sudo vi /etc/haproxy/haproxy.cfg
global
...

defaults
log     global
        mode    tcp    # https 트래픽도 받을 수 있도록 tcp로 설정
        option  tcplog # https 트래픽도 받을 수 있도록 tcp로 설정
...

frontend proxynode
  bind *:6443
  stats uri /proxystats
  default_backend masters

backend masters
  balance roundrobin
  server master-1 192.168.56.101:6443 check  # master1
  server master-2 192.168.56.102:6443 check  # master2
  server master-3 192.168.56.103:6443 check  # master3

listen stats
  bind :9999
  mode http
  stats enable
  stats hide-version
  stats uri /stats


kevin@master-1:~$ sudo systemctl stop apache2

# haproxy를 restart 하여 haproxy.cfg 설정내용을 적용
kevin@master-1:~$ sudo systemctl restart haproxy
kevin@master-1:~$ sudo systemctl status haproxy
active (running)

kevin@master-1:~$ netstat -nltp | grep 6443
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:6443            0.0.0.0:*               LISTEN      -
```

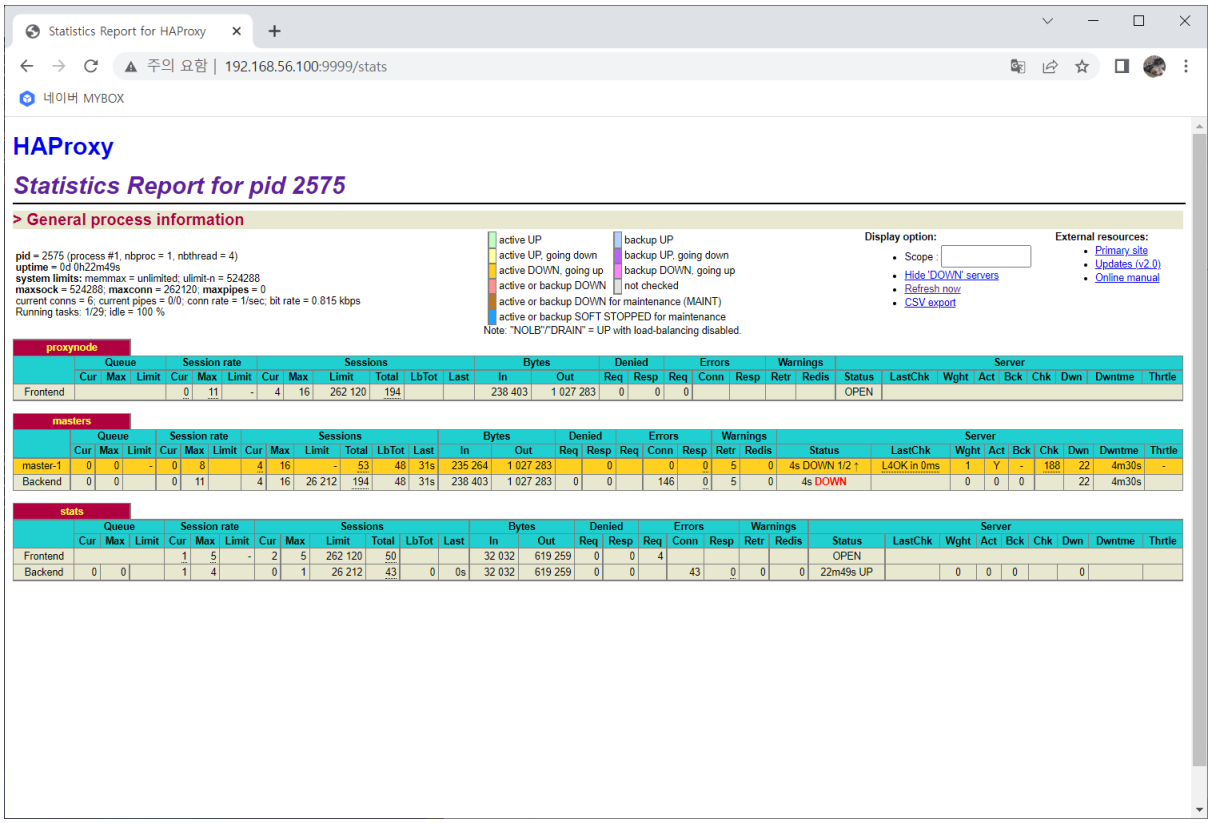
로드밸런서 확인 사항

```
kevin@lb:~$ netstat -nltp | grep 6443
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:6443            0.0.0.0:*               LISTEN      -

kevin@lb:~$ nc -v lb 6443
Connection to lb 6443 port [tcp/*] succeeded!
```

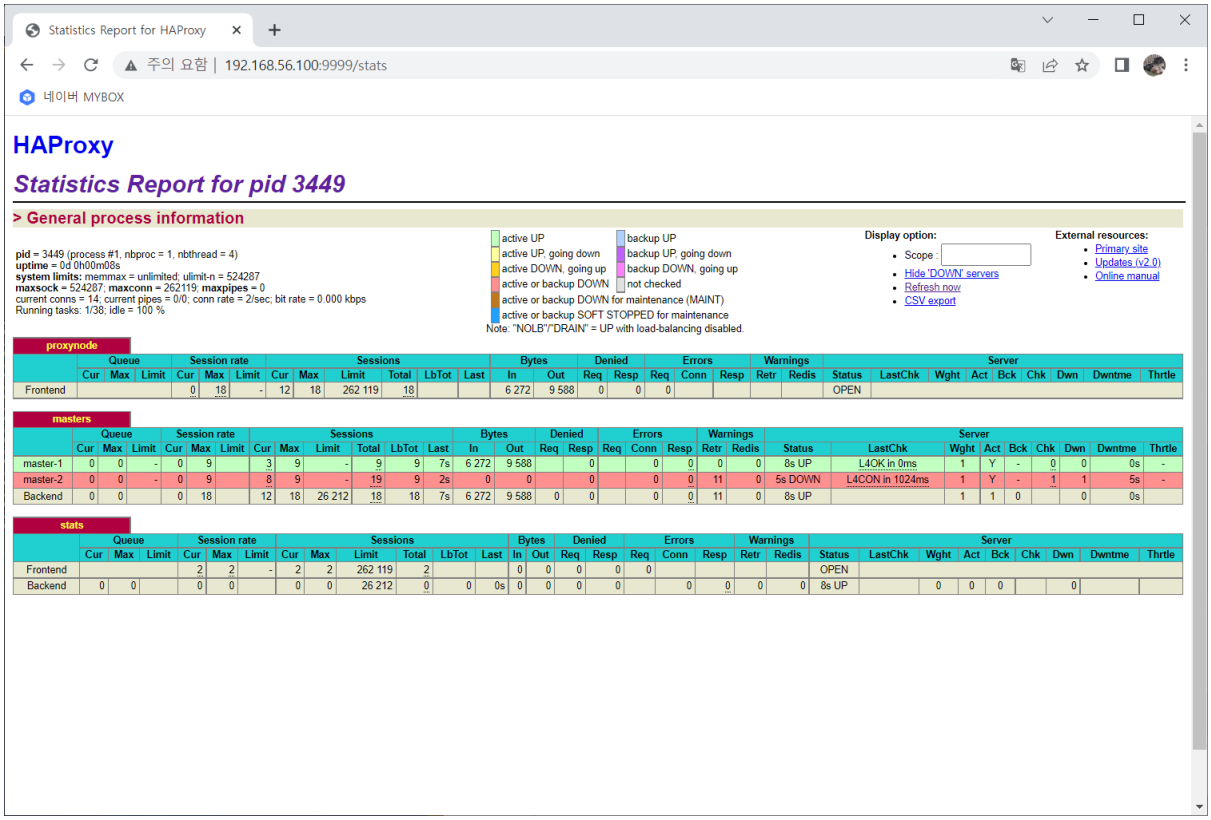
HAproxy에서 마스터 노드 접속 상태 확인하기

로드밸런서의 9999번 포트로 접속하면 HAproxy 동작 상태를 확인할 수 있다.

http://192.168.56.100:9999/stats



master-1 kubaem init 후



master-2 join 후

3️⃣ master node && worker node 공통

📌 도커 설치

```
kevin@master-1:~$ sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg2

# curl 명령어를 통해 gpg key 내려받기
kevin@master-1:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key --keyring /etc/apt/trusted.gpg.d/docker.gpg a
OK
# 패키지 관리 도구에 도커 다운로드 링크 추가
kevin@master-1:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# focal 확인
kevin@master-1:~$ tail /etc/apt/sources.list
deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable

# 패키지 관리 도구 업데이트
kevin@master-1:~$ sudo apt -y update

# 도커 설치
kevin@master-1:~$ sudo apt-get -y install docker-ce

# 설치 확인
kevin@master-1:~$ sudo docker version
kevin@master-1:~$ sudo docker info | grep cgroup
Cgroup Driver: cgroupfs

# Docker 데몬이 사용하는 드라이버를 cgroupfs 대신 systemd를 사용하도록 설정
# why? kubernetes에서 권장하는 Docker 데몬의 드라이버는 systemd 이고,
#      systemd를 사용하면 kubernetes가 클러스터 노드에서 사용 가능한 자원을 쉽게 알 수 있도록 해준다.
kevin@master-1:~$ sudo vi /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
```

```
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
kevin@master-1:~$ sudo mkdir -p /etc/systemd/system/docker.service.d
kevin@master-1:~$ sudo systemctl daemon-reload
kevin@master-1:~$ sudo systemctl enable docker
kevin@master-1:~$ sudo systemctl restart docker

kevin@master-1:~$ sudo systemctl status docker
active (running)
kevin@master-1:~$ sudo docker info | grep -i cgroup
 Cgroup Driver: systemd
 Cgroup Version: 1
```

📌 config.toml 파일명 변경

containerd를 CRI 런타임으로 사용하기 위함

```
kevin@master-1:~$ cd /etc/containerd/
kevin@master-1:/etc/containerd$ sudo mv config.toml config.toml.org
kevin@master-1:/etc/containerd$ ls
config.toml.org

kevin@master-1:/etc/containerd$ cd
kevin@master-1:~$ sudo systemctl restart containerd.service
kevin@master-1:~$ sudo systemctl restart kubelet
```

📌 쿠버네티스 설치

```
kevin@master-1:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
OK
kevin@master-1:~$ cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
> deb https://apt.kubernetes.io/ kubernetes-xenial main
> EOF
deb https://apt.kubernetes.io/ kubernetes-xenial main
kevin@master-1:~$ sudo apt -y update

kevin@master-1:~$ sudo apt -y install kubeadm=1.24.5-00 kubelet=1.24.5-00 kubectl=1.24.5-00
kevin@master-1:~$ sudo apt list | grep kubernetes
...
kubeadm/kubernetes-xenial 1.25.2-00 amd64 [upgradable from: 1.24.5-00]
kubectl/kubernetes-xenial 1.25.2-00 amd64 [upgradable from: 1.24.5-00]
kubelet/kubernetes-xenial 1.25.2-00 amd64 [upgradable from: 1.24.5-00]
...

kevin@master-1:~$ sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

📌 hosts 파일 설정

```
kevin@master-1:~$ sudo vi /etc/hosts
127.0.0.1       localhost
127.0.1.1       master-1        # 복제 후 각각 변경해야 한다.
192.168.56.100  lb
192.168.56.101  master-1
192.168.56.102  master-2
192.168.56.103  master-3
192.168.56.201  worker-1
192.168.56.202  worker-2
192.168.56.203  worker-3
```

📌 복제 후 확인 사항

- `모든 네트워크 어댑터의 새 MAC 주소 생성` , `완전한 복제` 로 복제하기
- IP 및 hostname 변경
- putty 연결
- sudo vi /etc/hosts 수정
- 노드 간 ping 확인
- 방화벽 확인

```
kevin@master-3:~$ sudo systemctl daemon-reload
kevin@master-3:~$ sudo systemctl stop firewalld.service
kevin@master-3:~$ sudo firewall-cmd --reload
FirewallD is not running
```

- date 확인
- 모든 노드끼리 ssh 접근
  - 노드 간 원격접속 되는지 확인

```
kevin@master-1:~$ ssh k8s-node1
kevin@master-1:~$ ssh k8s-node2
...

# 확인
kevin@master-1:~$ cat .ssh/known_hosts
|1|c7ewwIBvjFvAN9lw9c5QiH8H5N8=|g3NBTvsjFvzLQ1TlTRX6ptAsW3Y= ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA
|1|WnwoezAVRflwFh+sYfVZVBMp2DU=|xH7GN6SIDZRWJE1VImN7uzJPkd4= ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA
|1|EUFNLCL5outdMqSqC4qs2AgaYvQ=|vYu2q4Z1uiP75cQy0ZcDhF1r57Q= ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA
|1|ju2Gw1WHYNfcNa7WgSNaXflrowY=|4SaaorFJfjoQkygHoKJli0RCu6Y= ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAA
```

- 쿠버네티스 버전 확인

```
kevin@master-1:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.5", GitCommit:"e979822c185a14537054f15808a118d7fcce1d6e",

kevin@master-1:~$ kubelet --version
Kubernetes v1.24.5
```

4️⃣ kubernetes 초기화 작업

📌 (master-1 에서) kubeadm init

```
kevin@master-1:~$ sudo kubeadm init --control-plane-endpoint=lb:6443 --apiserver-advertise-address=192.168.56.101 --pod-network-cidr=1
...
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:
  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 lb:6443 --token ap62c8.tz1v3xmu1v20pju3 \
    --discovery-token-ca-cert-hash sha256:1d611f9a4ee26405f81cfda578234708a28e6b8c33924a8bad2136131f7ae49b \
    --control-plane --certificate-key 9628886c1cd06f5d1819a14771c36086fd63e660a90baa193e622fa8bfb4c892

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 lb:6443 --token ap62c8.tz1v3xmu1v20pju3 \
    --discovery-token-ca-cert-hash sha256:1d611f9a4ee26405f81cfda578234708a28e6b8c33924a8bad2136131f7ae49b
```

kubeadm init 옵션

- `--control-plane-endpoint`
  - 모든 controle-plane에 대한 공유 엔드 포인트를 설정하는 옵션이다. control-plane이 바라보는 주소를 적는다. 이 옵션을 통해 로드밸런서 기능을 사용할 수 있다. 마스터 노드 클러스터 생성 시 필요하다.
  - 공식 문서에서는 해당 옵션에 ip address를 그대로 사용하지말고, DNS name을 이용하길 권장한다.

> Creating Highly Available Clusters with kubeadm
> This page explains two different approaches to setting up a highly available Kubernetes cluster using kubeadm: With stacked control plane nodes. This approach requires less infrastructure. The etcd members and control plane nodes are co-located. With an external etcd cluster. This approach requires
> https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/

- `--apiserver-advertise-address`
  - control-plane API Server의 IP를 설정한다.
- `--pod-network-cidr`
  - pod의 네트워크를 설정할 때 사용
- `--service-cidr`
  - pod와 service의 네트워크 대역 분리를 위해 사용
- `--upload-certs`
  - MasterNode1의 Certificate가 다른 노드에 자동 배포되도록 한다. 마스터 노크 클러스터 생성 시 필요하다.

```
kevin@master-1:~$ mkdir -p $HOME/.kube
kevin@master-1:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
kevin@master-1:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

# 확인
kevin@master-1:~$ kubectl get no
NAME       STATUS     ROLES    AGE   VERSION
master-1   NotReady   master   10m   v1.19.15
```

📌 (master-2, master-3 에서) control plane join

```
kevin@master-2:~$ sudo kubeadm join lb:6443 --token ap62c8.tz1v3xmu1v20pju3 \
>     --discovery-token-ca-cert-hash sha256:1d611f9a4ee26405f81cfda578234708a28e6b8c33924a8bad2136131f7ae49b \
>     --control-plane --certificate-key 9628886c1cd06f5d1819a14771c36086fd63e660a90baa193e622fa8bfb4c892

...

This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
* The Kubelet was informed of the new secure connection details.
* Control plane (master) label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, 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

Run 'kubectl get nodes' to see this node join the cluster.
```

```
#  kubectl 명령어 사용을 위해 아래 명령어를 실행
kevin@master-2:~$ mkdir -p $HOME/.kube
kevin@master-2:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
kevin@master-2:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

# kubectl get nodes로 클러스터가 잘 생성되었는지 확인
```

📌 worker node join

```
kevin@worker-2:~$ kubeadm join lb:6443 --token ap62c8.tz1v3xmu1v20pju3 \
    --discovery-token-ca-cert-hash sha256:1d611f9a4ee26405f81cfda578234708a28e6b8c33924a8bad2136131f7ae49b
```

5️⃣ k8s에서 서비스 올리기

- `react` : `v16`
- `django` : `v4.1`
- `MySQL` : `v8.0`

> 💡 아래에 해당하는 포트는 제외하고 서비스를 올려합니다.
>
> - 마스터 노드에서 사용하는 포트
> - 6443 포트 : Kubernetes API Server / Used By All
> - 2379~2380 포트 : etcd server client API / Used By kube-apiserver, etcd
> - 10250 포트 : Kubelet API / Used By Self, Control plane
> - 10251 포트 : kube-scheduler / Used By Self
> - 10252 포트 : kube-controller-manager / Used By Self
> - 워커 노드에서 사용하는 포트
> - 10250 포트 : Kubelet API / Used By Self, Control plane
> - 30000~32767 포트 : NodePort Services / Used By All

📌 Dockerfile 생성 후 이미지 빌드하기

DB

```
FROM mysql:8.0

WORKDIR /

COPY db.sql  /docker-entrypoint-initdb.d

ENV MYSQL_ROOT_PASSWORD=password
ENV MYSQL_DATABASE=Nexpot

EXPOSE 3306

CMD ["mysqld"]
```

Backend

```
FROM python:3.9

WORKDIR /

COPY requirements.txt ./
RUN /usr/local/bin/python -m pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

COPY . /project
WORKDIR /project

EXPOSE 8000

CMD [ "python", "manage.py", "runserver","0.0.0.0:8000" ]
```

Frontend

```
FROM node:18

WORKDIR /app
COPY ./front-end .
RUN npm install

EXPOSE 3000

CMD ["npm", "start", "--host", "0.0.0.0"]
```

📌 k8s를 위한 Manifest 파일 작성

db-pv-pvc.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-pv
  labels:
    app: db-pv
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/mysql" ## 여러 가지 기타의 파일 시스템을 마운트 하는 곳
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

db-deploy-svc.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-deploy
  labels:
    app: db
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: db-container
        image: zxcasd3004/app:db1
        ports:
        - containerPort: 3306
        volumeMounts:
        - name: db-vol
          mountPath: /var/lib/mysql
      volumes:
      - name: db-vol
        persistentVolumeClaim:
          claimName: db-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: db-svc
spec:
  selector:
    app: db
  ports:
  - port: 3333
    targetPort: 3306
```

backend-deploy-svc.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: be-deploy
  labels:
    app: be
spec:
  replicas: 3
  selector:
    matchLabels:
      app: be
  template:
    metadata:
      labels:
        app: be
    spec:
      containers:
      - name: be-container
        image: ur2e/be:1.0
        ports:
        - containerPort: 8000
        env:
          - name: DATABASE
            value: db-svc
---
apiVersion: v1  # frontend (react)에게 request보낼 backend 파드의 ip 주소를 알려주기 위해 be-svc 생성
kind: Service
metadata:
  name: be-svc
spec:
  selector:
    app: be
  ports:
  - port: 8888
    targetPort: 8000
```

fe-secret.yaml

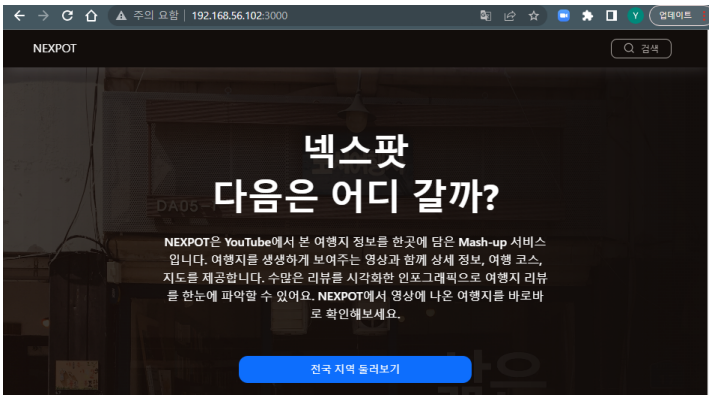
```
# kakao API
apiVersion: v1
kind: Secret
metadata:
  name: fe-secret
data:
  REACT_APP_KAKAOMAP_API_KEY: d413cb24@#$%@#^@#^@#6166
```

frontend-deploy-svc.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fe-deploy
  labels:
    app: fe
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fe
  template:
    metadata:
      labels:
        app: fe
    spec:
      nodeSelector:
        kubernetes.io/hostname: k8s-node2
      containers:
      - name: fe-container
        image: ur2e/fe:4.0
        ports:
        - containerPort: 3000
        env: ## BACKEND와 통신하기 위한 환경변수 설정
        - name: BACKEND
          value: be-svc
        volumeMounts: ## react에게 Kakao API Key를 알려주기위함.
        - name: api-key
          mountPath: "/app/.env"
          readOnly: true
      volumes:
      - name: api-key
        secret:
          secretName: fe-secret

---
apiVersion: v1
kind: Service
metadata:
  name: fe-svc
spec:
  selector:
    app: fe
  ports:
  - port: 3000
    targetPort: 3000
  externalIPs:
  - 192.168.56.102
```

```
# 모든 yaml 파일 apply 후 pod와 service 조회
$ kubectl get po,svc,deploy
NAME                                READY   STATUS    RESTARTS   AGE
pod/be-deploy-6484676968-28kmb   1/1     Running   0          10h
pod/be-deploy-6484676968-kxt2t   1/1     Running   0          10h
pod/be-deploy-6484676968-lg7b5   1/1     Running   0          10h
pod/db-deploy-d6dcb9bdb-vjs68    1/1     Running   0          10h
pod/fe-deploy-5b9b99c65-cd8ff    1/1     Running   0          10h
pod/fe-deploy-5b9b99c65-jng6v    1/1     Running   0          10h
pod/fe-deploy-5b9b99c65-tq5c5    1/1     Running   0          10h
pod/nginx                           1/1     Running   0          13h
pod/nginx-m2                        1/1     Running   0          13h
pod/nginx-m3                        1/1     Running   0          13h

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/be-svc       ClusterIP      20.97.89.246    <none>           8888/TCP         10h
service/db-svc       ClusterIP      20.100.75.230   <none>           3333/TCP         10h
service/fe-svc       ClusterIP      20.98.255.204   192.168.56.202   3000/TCP         10h
service/kubernetes   ClusterIP      20.96.0.1       <none>           443/TCP          4d9h
service/nginx        NodePort       20.97.146.85    <none>           80:30273/TCP     13h
service/nodep-svc    LoadBalancer   20.96.148.127   <pending>        9000:30057/TCP   34h

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/be-deploy   3/3     3            3           10h
deployment.apps/db-deploy   1/1     1            1           10h
deployment.apps/fe-deploy   3/3     3            3           10h
```

📌 service/fe-svc의 External-ip:3000 으로 접속



6️⃣ prometheus + grafana 모니터링

마스터 노드 ⇒ *익스포터* ⇒ *프로메테우스* ⇒ *그라파나* ⇒ *브라우저*

```
mkdir k8s-prometheus && cd $_
git clone https://github.com/brayanlee/k8s-prometheus.git

# Prometheus
kubectl create namespace monitoring
kubectl create -f prometheus/prometheus-ConfigMap.yaml
kubectl create -f prometheus/prometheus-ClusterRoleBinding.yaml
kubectl create -f prometheus/prometheus-ClusterRole.yaml
kubectl create -f prometheus/prometheus-Deployment.yaml
kubectl create -f prometheus/prometheus-Service.yaml
kubectl create -f prometheus/prometheus-DaemonSet-nodeexporter.yaml

# kube-state
kubectl create -f kube-state/kube-state-ClusterRoleBinding.yaml
kubectl create -f kube-state/kube-state-ClusterRole.yaml
kubectl create -f kube-state/kube-state-Service
kubectl create -f kube-state/kube-state-ServiceAccount.yaml
kubectl create -f kube-state/kube-state-Deployment.yaml
kubectl create -f kube-state/kube-state-Service.yaml

# grafana
kubectl create -f grafana/grafana-Deployment.yaml
kubectl create -f grafana/grafana-Service.yaml
```

➕ etcd 데이터 공유

- version
  - `etcdctl`: 3.2.26
  - `API` : 3.2

```
# etcd 설치
sudo apt-get -y update && sudo apt-get -y install etcd-client

# 버전 확인
yji@k8s-master:~$ etcdctl version
etcdctl version: 3.2.26
API version: 3.2

/*
 * 📕 실습 1. etcd에 직접 데이터 생성 후 다른 노드에서 조회하기
 */


🖳 master 1
# ectd3ctl alias 설정
alias etcd3ctl="ETCDCTL_API=3 etcdctl --endpoints=https://192.168.56.101:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kub
```

```
root@k8s-master-1:~# etcd3ctl put key1 value1
OK
root@k8s-master-1:~# etcd3ctl get key1 value1
key1
value1

💻 master 2
# etcd3ctl alias 설정
alias etcd3ctl="ETCDCTL_API=3 etcdctl --endpoints=https://192.168.56.102:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kub

root@k8s-master-1:~# etcd3ctl member list
181ffe2d394f445b, started, k8s-master-1, https://192.168.56.101:2380, https://192.168.56.101:2379
7dcbf96552ea4f10, started, k8s-master-3, https://192.168.56.103:2380, https://192.168.56.103:2379
995e4e0f6e68ae86, started, k8s-master-2, https://192.168.56.102:2380, https://192.168.56.102:2379


# master 2에서 master 1에서 넣은 데이터 조회
root@k8s-master-2:~# etcd3ctl put etcd-test1 I-am-master-1
OK
root@k8s-master-2:~# etcd3ctl get etcd-test1
etcd-test1
I-am-master-1


💻 master 3
alias etcd3ctl="ETCDCTL_API=3 etcdctl --endpoints=https://192.168.56.103:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kub
# master 3에서 master 1에서 넣은 데이터 조회
root@k8s-master-3:~# etcd3ctl put etcd-test1 I-am-master-1
OK
root@k8s-master-3:~# etcd3ctl get etcd-test1
etcd-test1
I-am-master-1

-----------------------------------------------------------------------------------------------------------------
/*
 * 📕 실습 2. pod 생성 후 etcd에 저장된 파드 확인하기
 */


💻 master 1
# nginx-m2라는 이름의 pod 생성
$ kubectl run nginx --image=nginx-m2

# etcd를 조회하기위해 관리자 모드로 변경
$ su -

# etcd에 저장된 데이터들을 알아보기위해 etcd3ctl get 명령을 json 확장자 파일로 redirect
$ etcd3ctl get / --prefix=true -w json > etcd_data.json

# jq 패키지를 사용해서 정렬되지 않은 json 파일을 포맷팅 한다.
$ jq '.' etcd_data.json

    - jq 패키지 실행 전 etcd_data.json
    {"header":{"cluster_id":15516845868796936735,"member_id":11051356363797606022,"revision":517184,"raft_term":12},"kvs":[{"key":"L3J
    json...}}}}


    - jq 패키지 실행 후 etcd_data.json // etcd는 key-value 형태의 스토리지임을 확인할 수 있음
    {
      "header": {
        "cluster_id": 15516845868796936000,
        "member_id": 11051356363797606000,
        "revision": 517184,
        "raft_term": 12
      },
      "kvs": [
        {
          "key": "L3JlZ2lzdHJ5L2FwaWV4dGVuc2lvbnMuazhzLmlvL2N1c3RvbXJlc291cmNlZGVmaW5pdGlvbnMvYmdwY29uZmlndXJhdGlvbnMuY3JkLnByb2plY3Rj
          "create_revision": 7146,
          "mod_revision": 7150,
          "version": 3,
          "value": "eyJraW5kIjoiQ3VzdG9tUmVzb3VyY2VEZWZpbml0aW9uIiwiYXBpVmVyc2lvbiI6ImFwaWV4dGVuc2lvbnMuazhzLmlvL3YxYmV0YT

# key 값 디코딩하여 etcd에 저장된 데이터를 확인
  # jq '.kvs[].key' : jq로 원하는 kvs키의 value 값
  # cut -d '"' -f2: "(쌍따옴표)를 구분자로 2째 필드값을 자르기
  #  base64 --decode: base64-decoding
$ for k in $(cat etcd_data.json | jq '.kvs[].key' | cut -d '"' -f2); do echo $k | base64 --decode; echo; done > etcd-data.txt

# 디코딩된 값 조회
$ vi etcd-data.txt
/registry/csinodes/k8s-master-1
/registry/csinodes/k8s-master-2
/registry/csinodes/k8s-master-3
/registry/daemonsets/kube-system/calico-node
/registry/daemonsets/kube-system/kube-proxy
```

```
/registry/daemonsets/monitoring/node-exporter
/registry/pods/default/nginx-m1
/registry/pods/default/nginx-m2   # 아까 생성한 파드 nginx-m2가 저장


# 아까 생성한 pod를 etcdctl 명령으로 조회하여 etcd가 저장한 pod의 정보를 조회
# 문자열 데이터가 아닌 부분은 깨져있다.
$ etcd3ctl get /registry/pods/default/nginx-m2 > etcd-nginx-m2
^Bv1^R^Rmetadata.namespace^P¤^C^R<91>^A
^H🟣nginx-m2^R^Enginx*^@B^@JL
^Ukube-api-access-hvvvg^P^A^Z-/var/run/secrets/kubernetes.io/serviceaccount"^@2^@j^T/dev/termination-logr^FAlways<80>^A^@<88>^A^@<90>^
^G🟢Running^R#
^KInitialized^R^DTrue^Z^@"^H^Hâã<93><9a>^F^P^@*^@2^@^R^]
^EReady^R^DTrue^Z^@"^H^Héã<93><9a>^F^P^@*^@2^@^R'
^OContainersReady^R^DTrue^Z^@"^H^Héã<93><9a>^F^P^@*^@2^@^R$
^LPodScheduled^R^DTrue^Z^@"^H^H<81>õ<93><9a>^F^P^@*^@2^@^Z^@"^@*^N192.168.56.2032^L10.98.69.216:^H^Hâã<93><9a>^F^P^@Bð^A
^Hnginx-m2^R^L^R

^H^Héã<93><9a>^F^P^@^Z^@ ^A(^@2^^🔴docker.io/library/nginx:latest:_docker.io/library/nginx@sha256:2f770d2fe27bc85f68fd7fe6a63900ef7076
BestEffortZ^@b^N
^L🟡10.98.69.216^Z^@"^@

# 아까 생성한 nginx-m pod의 상세 정보 조회 > etcd가 저장한 pod의 정보와 유사한 부분들을 확인할 수 있음
$ kubectl describe po nginx-m2
Name:         🟣nginx-m2
Namespace:    default
Status:       🟢Running
IP:           10.98.69.216
IPs:
  IP:  🟡10.98.69.216
Containers:
  nginx-m2:
    Container ID:   ●containerd://6ff1306a4f878d831b6530f3a789ed0e4f00ec50962c698814fbffd12b841376
    Image:          nginx
    Image ID:       🔴docker.io/library/nginx@sha256:2f770d2fe27bc85f68fd7fe6a63900ef7076bc703022fe81b980377fe3d27b70
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 11 Oct 2022 13:32:41 +0900
    Ready:          True
```