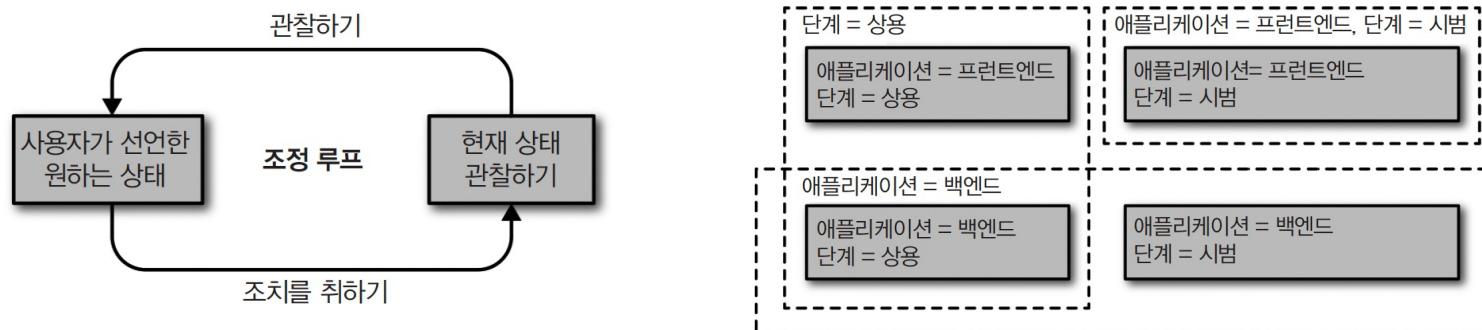


[구조/개념] Kubernetes Architecture

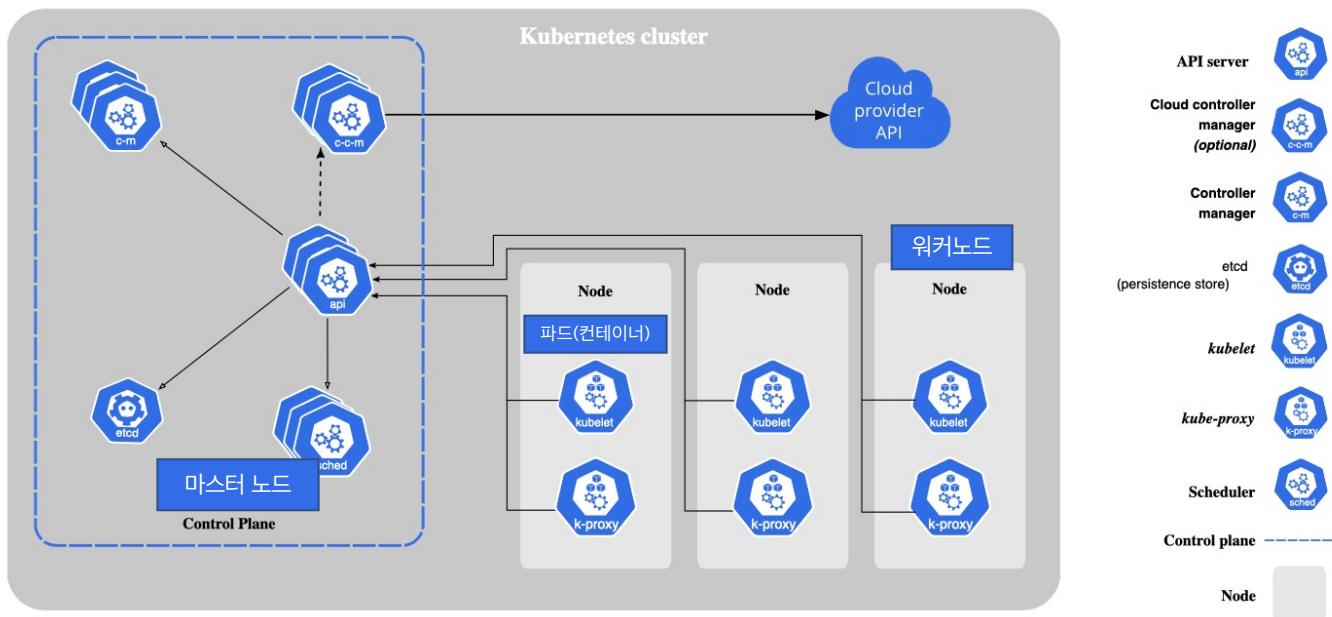
쿠버네티스 설계 사상

- 선언적 구성 : “내 웹서버의 레플리카를 항상 5개씩 실행하고 싶습니다” cf) 명령어 구성 : ”레플리카를 5개 만들어”
 - 제어(조정)루프 : 현재 상태를 관찰하여 사용자가 원하는 상태로 유지 ex) 에어컨
- 컨트롤러 구성 : 각 기능별로 독립적으로 분산되게 설계 ► 유연하고 안정적이지만 복잡함
- 동적 그룹화 : “우리 팀원은 오렌지색 옷을 입은 사람들입니다.” cf) 정적 그룹화 : 우리 팀원은 철수와 영희입니다.
 - 레이블(쿼리 가능) 및 어노테이션(메타데이터용)으로 구성
- API 기반 상호작용 : 쿠버네티스 요소들이 서로 직접 접근 불가, 구성 요소를 대체 구현하기 용이



쿠버네티스 구조 : 클러스터

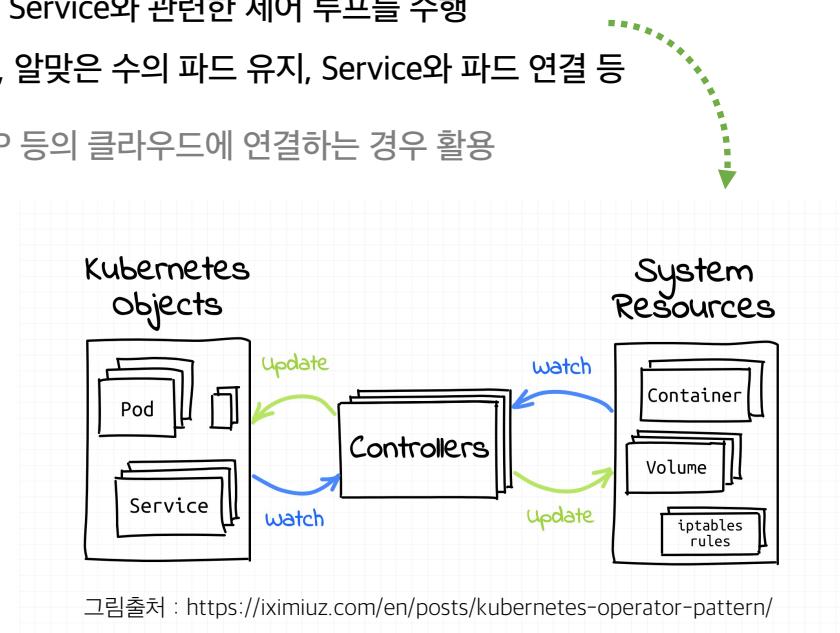
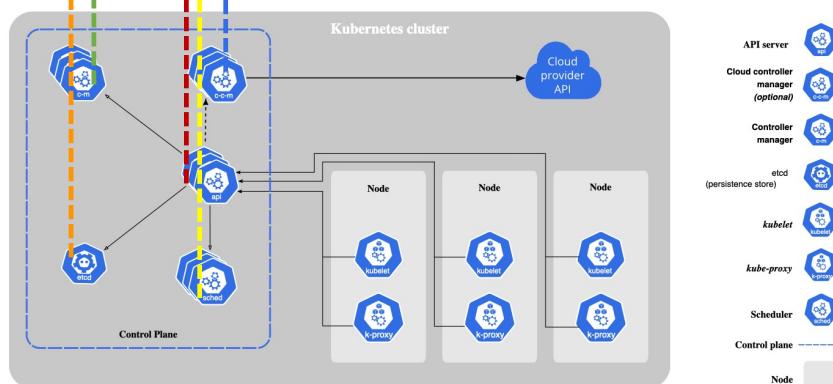
- 클러스터란? 노드라고 하는 워커 머신의 집합
- 노드란? 컨테이너화된 애플리케이션을 실행하는 서버(가상 또는 물리 둘다 가능)
- 워커 노드는 애플리케이션의 구성요소인 파드(⇨ 컨테이너)를 호스트
- 컨트롤 플레인은 워커 노드와 클러스터 내 파드를 관리



쿠버네티스 구조 : 클러스터 구성요소

- 컨트롤 플레인(마스터 노드)

- API서버 : 쿠버네티스 API를 노출, 쿠버네티스 컨트롤 플레인의 프론트 엔드, 수평확장 가능
- etcd : 모든 클러스터 데이터를 담는 쿠버네티스 뒷단의 저장소로 사용되는 일관성·고가용성 키-값 저장소
- 스케줄러 : 노드가 배정되지 않은 새로 생성된 파드를 감지하고, 실행할 노드를 선택
- 컨트롤러 매니저 : ReplicaSets, Deployment, Service와 관련한 제어 루프를 수행
노드가 다운되면 통지/대응, 알맞은 수의 파드 유지, Service와 파드 연결 등
- 클라우드 컨트롤러 매니저 : AWS, Azure, GCP 등의 클라우드에 연결하는 경우 활용

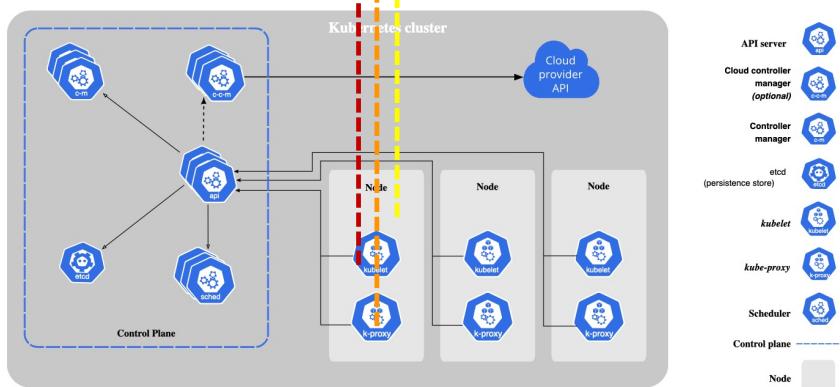


그림출처 : <https://iximiuz.com/en/posts/kubernetes-operator-pattern/>

쿠버네티스 구조 : 클러스터 구성요소

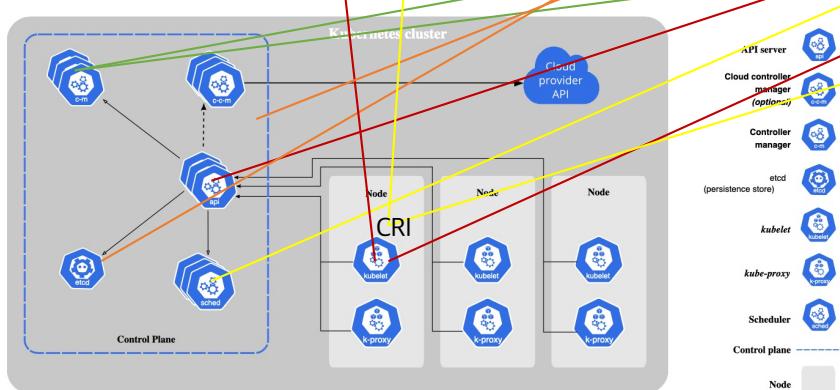
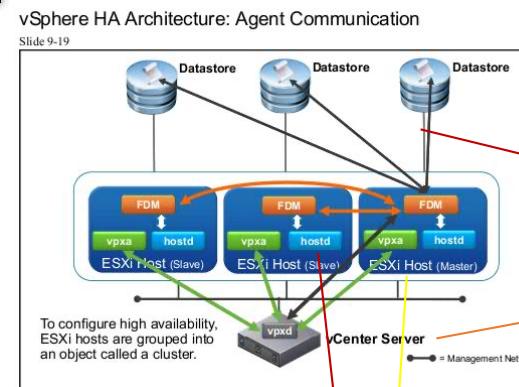
- 워커 노드

- Kubelet(쿠블렛) : 컨테이너가 동작하도록 관리, 쿠버네티스 클러스터와 워커노드의 CPU/Mem/Disk 간을 연결
- Kube-proxy : 쿠버네티스 Service(로드밸런서 리소스)에 맞게 커널의 netfilter(iptables) 등을 관리하는 역할
- 컨테이너 런타임 : 컨테이너 실행을 담당하는 소프트웨어(도커, containerd, CRI-O 등)

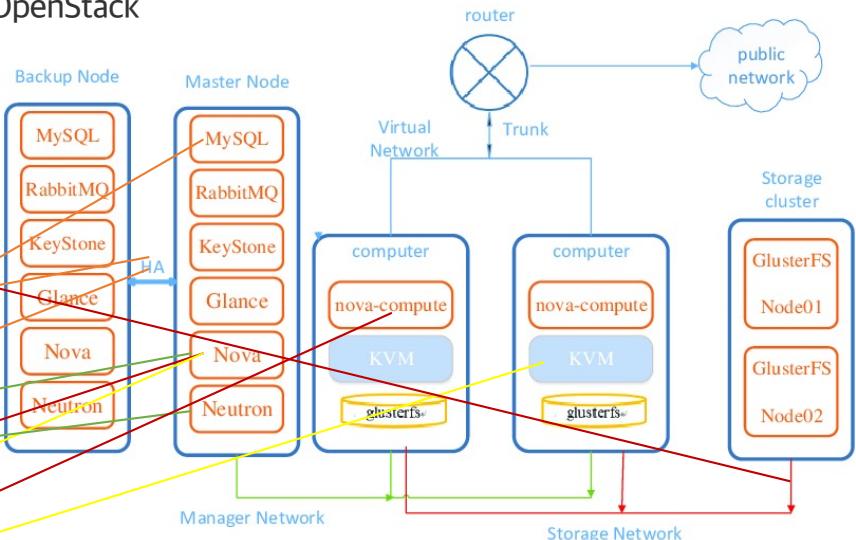


쿠버네티스 구조(번외) : 타 솔루션과 비교

- VMware



- OpenStack

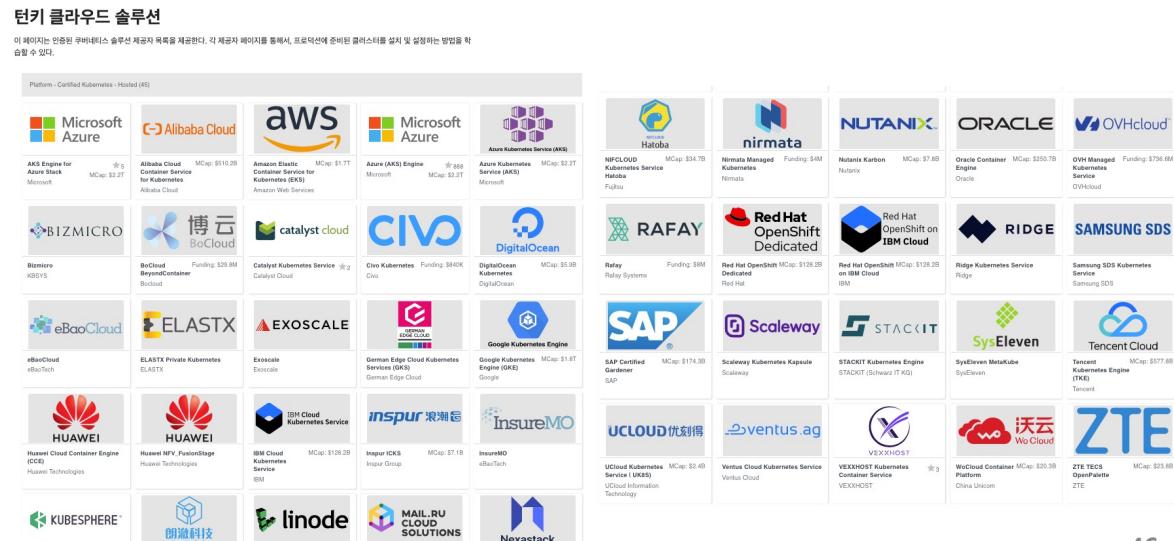


쿠버네티스 설치

- 설치방법(배포도구 활용)
 - Kubeadm, Kops, Kubespray

차이? 공식docs 링크 문서 / <https://github.com/kubernetes-sigs/kubespray/blob/master/docs/comparisons.md>

- 설치방법(턴키솔루션 활용)
 - Redhat Openshift, AWS EKS, MS AKS 등



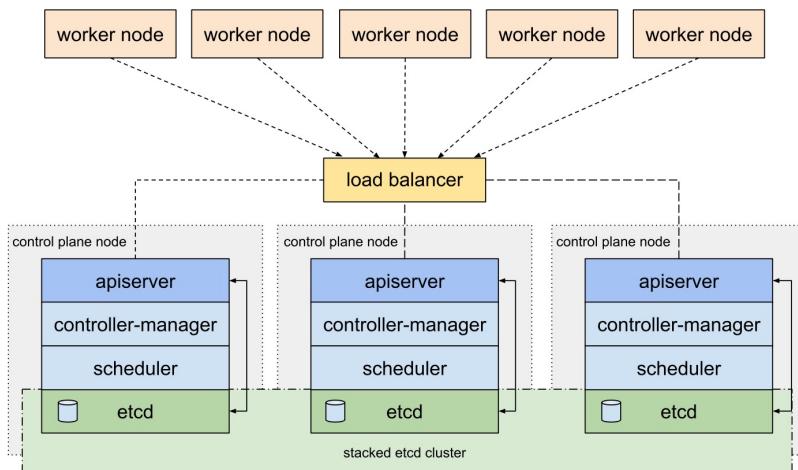
쿠버네티스 설치

- 고가용성 토플로지

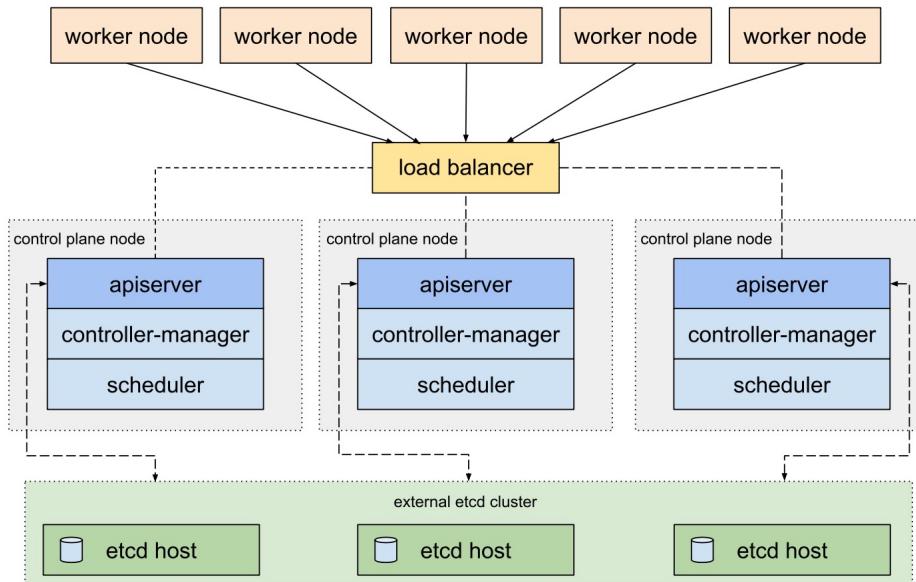
<https://kubernetes.io/ko/docs/setup/production-environment/tools/kubeadm/ha-topology/>

- 중첩된 etcd 토플로지 vs 외부 etcd 토플로지 : etcd 배포 위치 차이

kubeadm HA topology - stacked etcd

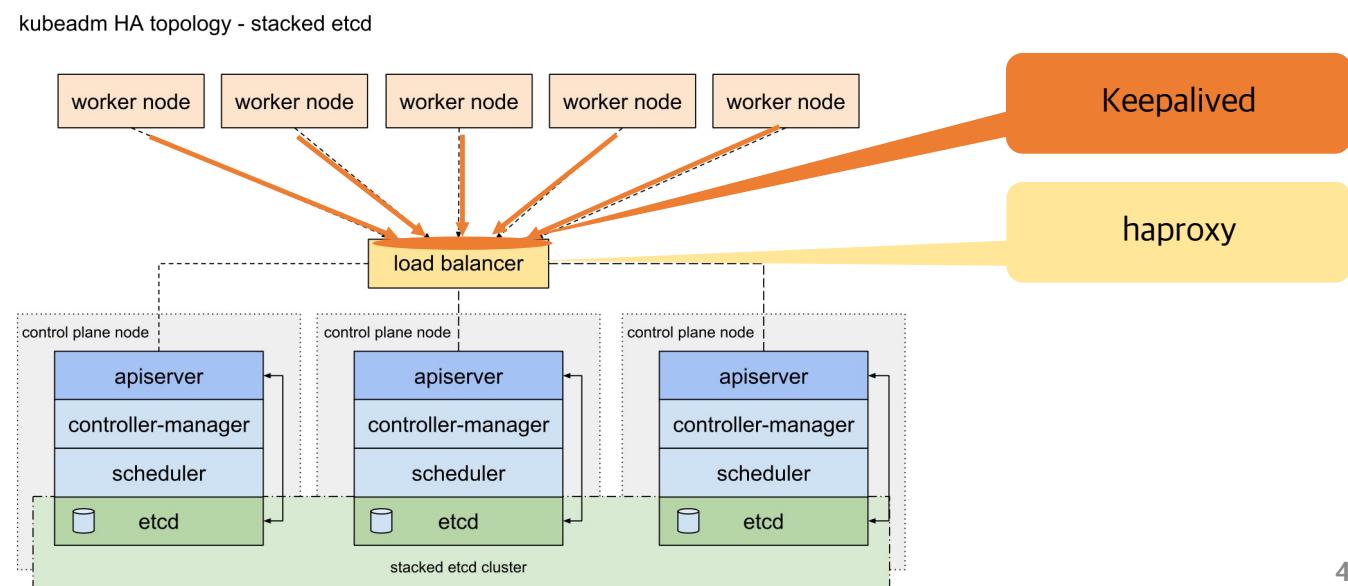


kubeadm HA topology - external etcd



쿠버네티스 설치

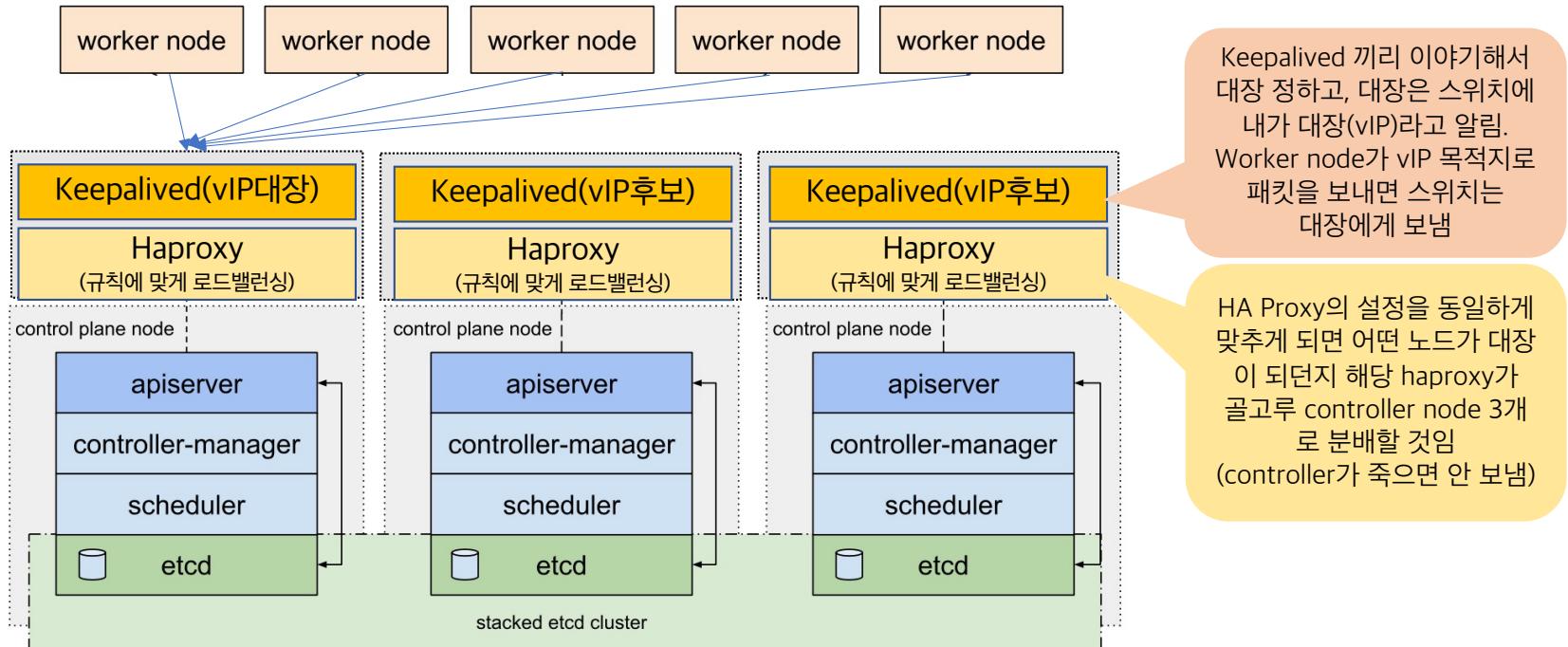
- Kubeadm으로 High Availability 클러스터 설치
<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/>
- 3개의 Master에 연결하기 위한 물리적인 L4스위치 / ha-proxy / nginx / CSP에서 제공하는 Load Balancer 등 다양한 Load Balancer 구성 방식이 있으며, 본 교육에서는 ha-proxy(로드밸런서) 사용



쿠버네티스 설치

- 물리적 구성도

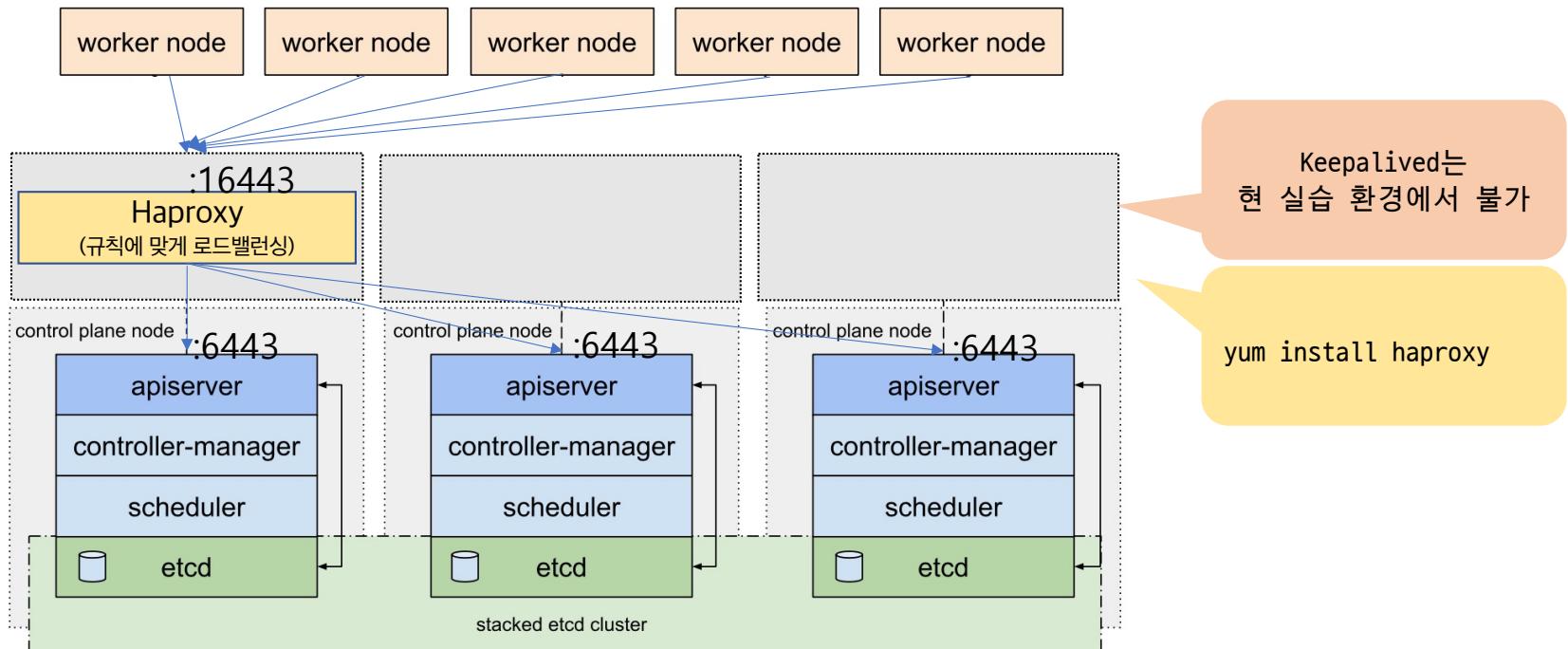
kubeadm HA topology - stacked etcd



쿠버네티스 설치

- 물리적 구성도

kubeadm HA topology - stacked etcd



쿠버네티스 설치

- [Master 1번 노드만] Load balancer인 haproxy 설치, 설정

```
# sudo setenforce 0
# sudo yum install haproxy -y
# sudo vi /etc/haproxy/haproxy.cfg
frontend kubernetes-master-lb
    bind 0.0.0.0:16443
    option tcplog
    mode tcp
    default_backend kubernetes-master-nodes

backend kubernetes-master-nodes
    mode tcp
    balance roundrobin
    option tcp-check
    option tcplog
    server master1 172.30.4.88:6443 check
    server master2 172.30.7.75:6443 check
    server master3 172.30.6.170:6443 check
```

- 16443 포트가 Listen 중인지 확인

```
# netstat -nltp
```

쿠버네티스 설치

- Kubeadm 설치
<https://kubernetes.io/ko/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
- 설치 전,
 - 리눅스 기반. 2GB Ram, 2 CPU 이상의 성능
 - SWAP 미사용 (카카오 기본 이미지에 SWAP 미설정됨)
 - 고유한 hostname, MAC, UUID, 포트 개방, 네트워크 연결

[전체노드] # hostname 으로 hostname 확인 후 /etc/hosts 에 반영 (worker도)
- MAC주소 및 UUID가 고유한지 확인
- 네트워크 어댑터 확인
 - 카카오 내 VM은 1개만 이므로 관계없음



쿠버네티스 설치

- [전체 노드] br_netfilter 모듈 로딩 : 리눅스 노드의 iptables가 브리지된 트래픽을 올바르게 보기 위한 요구 사항

```
# cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf  
br_netfilter  
EOF
```

```
# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
EOF
```

```
# sudo sysctl --system
```

쿠버네티스 설치

- 주요 포트 확인
- 외부에서 쿠버네티스 API를 접근 시 6443 포트 사용

필수 포트 확인

컨트롤 플레인 노드

프로토콜	방향	포트 범위	목적	사용자
TCP	인바운드	6443*	쿠버네티스 API 서버	모두
TCP	인바운드	2379-2380	etcd 서버 클라이언트 API	kube-apiserver, etcd
TCP	인바운드	10250	kubelet API	자체, 컨트롤 플레인
TCP	인바운드	10251	kube-scheduler	자체
TCP	인바운드	10252	kube-controller-manager	자체

워커 노드

프로토콜	방향	포트 범위	목적	사용자
TCP	인바운드	10250	kubelet API	자체, 컨트롤 플레인
TCP	인바운드	30000-32767	NodePort 서비스†	모두

† NodePort 서비스의 기본 포트 범위.

*로 표시된 모든 포트 번호는 재정의할 수 있으므로, 사용자 지정 포트도 열려 있는지 확인해야 한다.

etcd 포트가 컨트롤 플레인 노드에 포함되어 있지만, 외부 또는 사용자 지정 포트에서 자체 etcd 클러스터를 호스팅할 수도 있다.

사용자가 사용하는 파드 네트워크 플러그인(아래 참조)은 특정 포트를 열어야 할 수도 있다. 이것은 각 파드 네트워크 플러그인마다 다르므로, 필요한 포트에 대한 플러그인 문서를 참고한다.

쿠버네티스 설치

- [전체 노드] CRI (Container runtime interface) 설치 필요 : Docker 설치/시작
- ```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```
- ```
# sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```
- ```
sudo yum install docker-ce -y
```
- ```
# sudo vi /usr/lib/systemd/system/docker.service
```



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

- ```
sudo systemctl daemon-reload
```
- ```
# sudo systemctl start docker && sudo systemctl enable docker
```
- ```
sudo docker info | grep -i cgroup
```

## Configuring the container runtime cgroup driver

The [Container runtimes](#) page explains that the `systemd` driver is recommended for kubeadm based setups instead of the `cgroupfs` driver, because kubeadm manages the kubelet as a systemd service.

## Configuring the kubelet cgroup driver

kubeadm allows you to pass a `KubeletConfiguration` structure during `kubeadm init`. This `KubeletConfiguration` can include the `cgroupDriver` field which controls the cgroup driver of the kubelet.

**Note:** In v1.22, if the user is not setting the `cgroupDriver` field under `KubeletConfiguration`, `kubeadm` will default it to `systemd`.

# 쿠버네티스 설치

- [전체 노드] Redhat 배포판 버전으로 따라하기

<https://kubernetes.io/ko/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#kubeadm-kubelet-및-kubectl-설치>

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF
```

- # sudo setenforce 0
- # sudo sed -i 's/^SELINUX=enforcing\$/SELINUX=permissive/' /etc/selinux/config

# 쿠버네티스 설치

- [전체 노드] 단, kubelet / kubeadm / kubectl 은 최신보다 한단계 낮은 버전으로 설치 (향후 교육과정 고려)

```
sudo yum info kubelet --disableexcludes=Kubernetes -y
sudo yum install kubelet-1.21.0 --disableexcludes=kubernetes -y
sudo yum install kubectl-1.21.0 --disableexcludes=kubernetes -y
sudo yum install kubeadm-1.21.0 --disableexcludes=kubernetes -y
sudo systemctl enable --now kubelet
```

- 앞에서 도커의 cgroup driver를 systemd로 설정했기에, 쿠버도 마찬가지로 설정

```
sudo vi /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf 내에
[Service]
Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=systemd --runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice"
```

- daemon reload, kubelet restart 수행. 이제 필요한 패키지는 모두 설치 완료. 클러스터를 만들어야 함

# 쿠버네티스 설치

- [컨트롤 첫번째만] (etcd가 함께 포함된) Stack Control node 설치

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/#stacked-control-plane-and-etcd-nodes>

Stacked control plane and etcd nodes

```
sudo kubeadm init --control-plane-endpoint "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT" --upload-certs
```

- # sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --control-plane-endpoint "172.30.~.~:16443" --upload-certs

---

Calico 활용 예정

HA Proxy가 설치된 master1 및 해당 포트

인증서 전달하여

손쉽게 구성



Your Kubernetes control-plane has initialized successfully!

# 쿠버네티스 설치

- [결과 메세지에 따라 이어서 진행]
- 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
```

Kubeconfig  
인증에 사용되는 쿠베컨피그

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/>

→ <https://docs.projectcalico.org/getting-started/kubernetes/self-managed-onprem/onpremises>  
→ # curl https://docs.projectcalico.org/manifests/calico.yaml -O  
→ # kubectl apply -f calico.yaml

# 쿠버네티스 설치

- [결과 메세지에 따라 진행]

(다른 마스터노드 2,3에서 진행)

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

```
kubeadm join 172.30.5.193:16443 --token 1ecqfl.3grbrnswq5ggwakz \
--discovery-token-ca-cert-hash sha256:6daa1758e52cf26c5ba797700c7fb33fdc89294a3c2fc02a185fdb309907b3ea \
--control-plane --certificate-key 6e924d9d18d75d979eda3599e9d62da107e3b72a9f8c64b71bea4b00d78c01ed
```

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**:

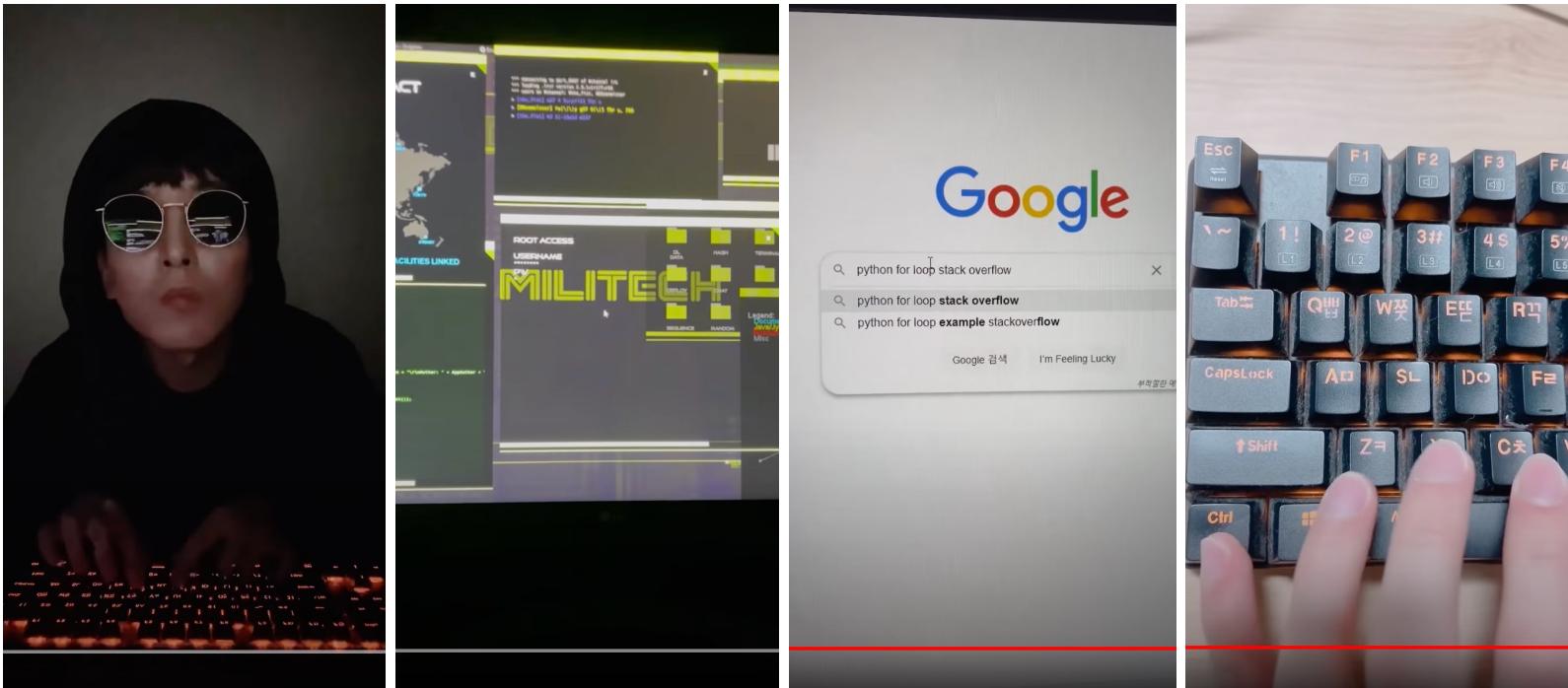
(워커노드1,2에서 진행)

```
kubeadm join 172.30.5.193:16443 --token 1ecqfl.3grbrnswq5ggwakz \
--discovery-token-ca-cert-hash sha256:6daa1758e52cf26c5ba797700c7fb33fdc89294a3c2fc02a185fdb309907b3ea
```

# 쿠버네티스 설치

- 쉬엄쉬엄, 차근차근

<https://www.youtube.com/watch?v=cXB8rCW7lto>



# 쿠버네티스 설치

- 클러스터 전체 구성 완료!

```
[root@osk-master-01 ~]# kubectl get nodes
```

| NAME                                  | STATUS | ROLES                | AGE   | VERSION |
|---------------------------------------|--------|----------------------|-------|---------|
| osk-master-01.kr-central-1.c.internal | Ready  | control-plane,master | 22m   | v1.21.0 |
| osk-master-02.kr-central-1.c.internal | Ready  | control-plane,master | 4m36s | v1.21.0 |
| osk-master-03.kr-central-1.c.internal | Ready  | control-plane,master | 2m29s | v1.21.0 |
| osk-worker-01.kr-central-1.c.internal | Ready  | <none>               | 32s   | v1.21.0 |
| osk-worker-02.kr-central-1.c.internal | Ready  | <none>               | 58s   | v1.21.0 |

```
[root@osk-master-01 ~]#
```



# 설치 완료 후, Kubectl?

- Kubectl 은 쿠버네티스 클러스터를 제어하기 위한 커맨드 라인 도구
  - ✓ kubectl [command] [TYPE] [NAME] [flags]
    - command: 명령을 하려는 ‘동사’. 예: create, get, describe, delete
    - TYPE: 리소스 타입
    - NAME: 리소스 이름
    - flags: 선택적 옵션
  - ✓ kubectl help를 사용하여 command를 확인 가능 (`-- help`)
  - ✓ kubectl에서 자주사용하는 output flag는 `-o wide`, `-o yaml`, `-o json`, `--sort-by=<jsonpath_exp>` 등  
`--dry-run=client -o yaml > filename.yaml`

```
osk@osk-MacBook-Pro ~ % kubectl get nodes -o wide
osk@osk-MacBook-Pro ~ %
osk@osk-MacBook-Pro ~ % kubectl run test --image=nginx --dry-run=client -o yaml > 1.yaml
```

## 설치 완료 후, 인증서

- Preflight-check가 완료되면 kubeadm은 CA(자체 인증) 파일과 키를 생성

```
[root@osk-master-01 pki]# ll /etc/kubernetes/pki
total 56
-rw-r--r--. 1 root root 1326 Aug 14 11:29 apiserver.crt
-rw-r--r--. 1 root root 1155 Aug 14 11:29 apiserver-etcd-client.crt
-rw-----. 1 root root 1675 Aug 14 11:29 apiserver-etcd-client.key
-rw-----. 1 root root 1679 Aug 14 11:29 apiserver.key
-rw-r--r--. 1 root root 1164 Aug 14 11:29 apiserver-kubelet-client.crt
-rw-----. 1 root root 1679 Aug 14 11:29 apiserver-kubelet-client.key
-rw-r--r--. 1 root root 1066 Aug 14 11:29 ca.crt
-rw-----. 1 root root 1675 Aug 14 11:29 ca.key
drwxr-xr-x. 2 root root 162 Aug 14 11:29 etcd
-rw-r--r--. 1 root root 1078 Aug 14 11:29 front-proxy-ca.crt
-rw-----. 1 root root 1679 Aug 14 11:29 front-proxy-ca.key
-rw-r--r--. 1 root root 1119 Aug 14 11:29 front-proxy-client.crt
-rw-----. 1 root root 1679 Aug 14 11:29 front-proxy-client.key
-rw-----. 1 root root 1679 Aug 14 11:29 sa.key
-rw-----. 1 root root 451 Aug 14 11:29 sa.pub
```

→ public key infrastructure

→ .crt : 서버 인증서  
→ .key : 서버 개인키

# 설치 완료 후, 팁

- 자동 완성 설정 : <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-bash-completion>

## ✓ Bash

```
sudo yum install -y bash-completion
source /usr/share/bash-completion/bash_completion # bash-completion 패키지를 먼저 설치 후, bash의 자동 완성 셸에 설정
echo "source <(kubectl completion bash)" >> ~/.bashrc # 자동 완성을 bash 셸에 영구적으로 추가
kubectl completion bash >/etc/bash_completion.d/kubectl # root 권한으로 실행
```

## ✓ zsh(맥북)

```
source <(kubectl completion zsh) # 현재 셸에 zsh의 자동 완성 설정
echo "[[$commands[kubectl]]] && source <(kubectl completion zsh)" >> ~/.zshrc # 자동 완성을 zsh 셸에 영구적으로 추가한다.
```

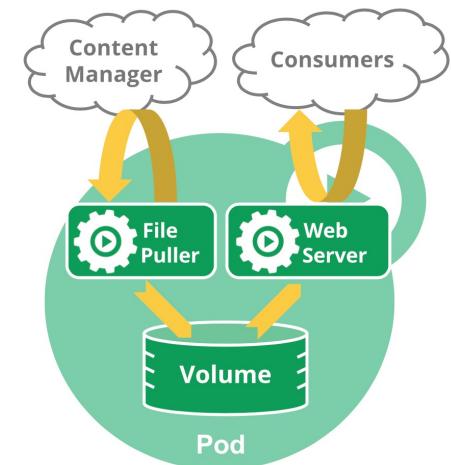
\* 맥에서 complete:13 에러 발생시, .zshrc 내부에 아래 추가

```
autoload -Uz compinit
compinit -i
```

# 쿠ber네티스 리소스 : 워크로드 오브젝트

- POD : (고래 떼를 일컫음. 도커의 고래에서 유래) 하나 이상의 컨테이너로 구성
  - 스케일링의 단위, 어플리케이션에 친숙(환경변수 / 정상 여부 상태검사 정의 등이 용이)
- 1개 파드에 2개 이상의 각각 다른 이미지 가진 컨테이너가 가능함
- 파드 리소스는 노드 IP와 별개로 파드 만의 고유한 IP를 할당받으며 파드 내의 컨테이너들은 IP를 공유함
- 파드 내의 컨테이너들은 동일한 볼륨과 연결이 가능
- 파드는 배포의 최소단위이며 특정 네임스페이스에 실행됨

```
osk@osk-MacBook-Pro lkln-kakao % kubectl run test --image=nginx --dry-run=client -o yaml > 1.yaml
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod
No resources found in default namespace.
osk@osk-MacBook-Pro lkln-kakao % kubectl run test --image=nginx
pod/test created
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod
NAME READY STATUS RESTARTS AGE
test 0/1 ContainerCreating 0 4s
osk@osk-MacBook-Pro lkln-kakao % kubectl delete pod test
pod "test" deleted
osk@osk-MacBook-Pro lkln-kakao % kubectl apply -f 1.yaml
pod/test created
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod
NAME READY STATUS RESTARTS AGE
test 1/1 Running 0 20s
```



1) kubectl 명령어 활용  
2) yaml 파일 생성 후 apply로 적용

# 쿠버네티스 리소스 : 워크로드 오브젝트

- POD 명령어

```
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
test 1/1 Running 0 73m 10.240.32.66 osk-test-lkln-5c439d5-gm5hf <none> <none>
```

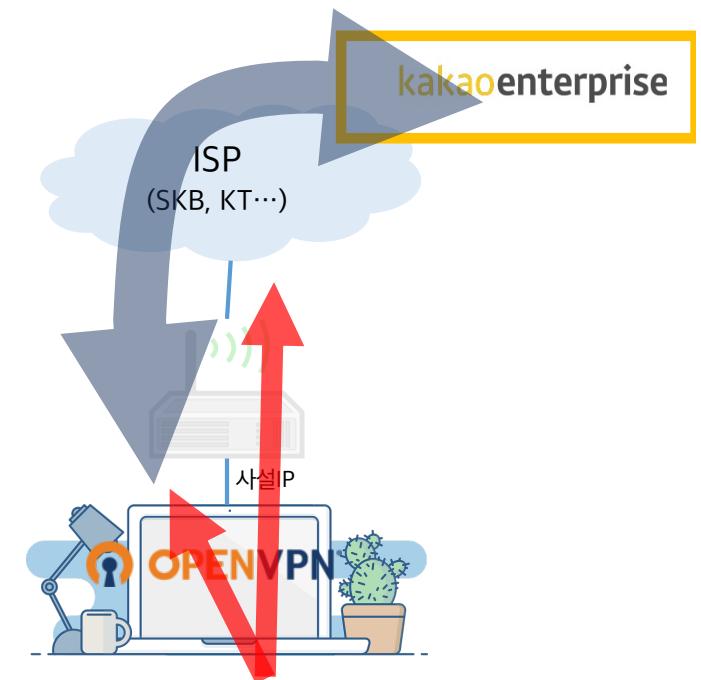
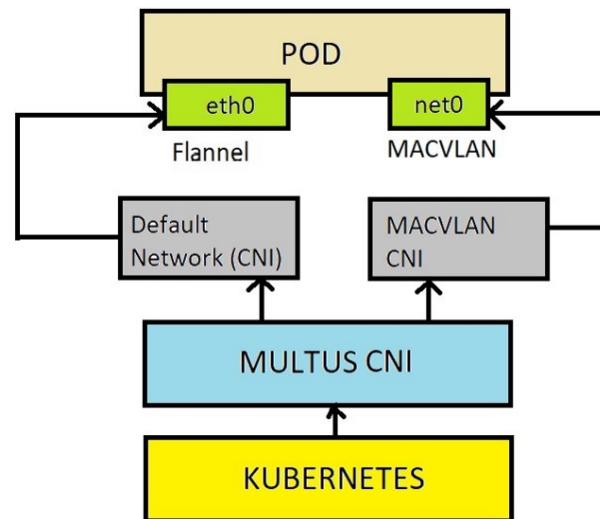
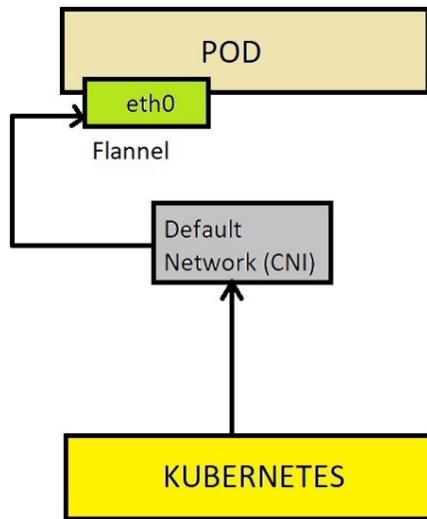
```
osk@osk-MacBook-Pro lkln-kakao % kubectl describe pod test
Name: test
Namespace: default
Priority: 0
Node: osk-test-lkln-5c439d5-gm5hf/172.30.6.194
```

```
osk@osk-MacBook-Pro lkln-kakao % kubectl logs test
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
```

```
osk@osk-MacBook-Pro lkln-kakao % kubectl exec -it test -- /bin/sh
hostname
test
```

# 쿠버네티스 리소스 : 워크로드 오브젝트

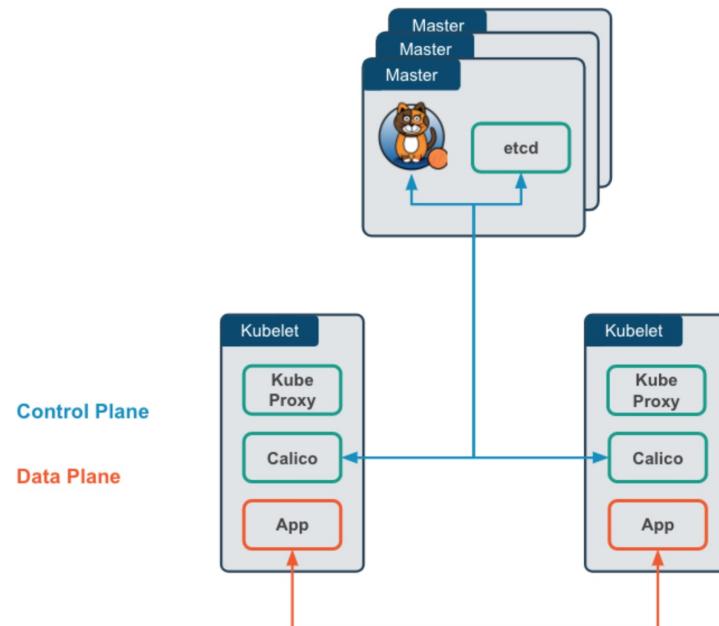
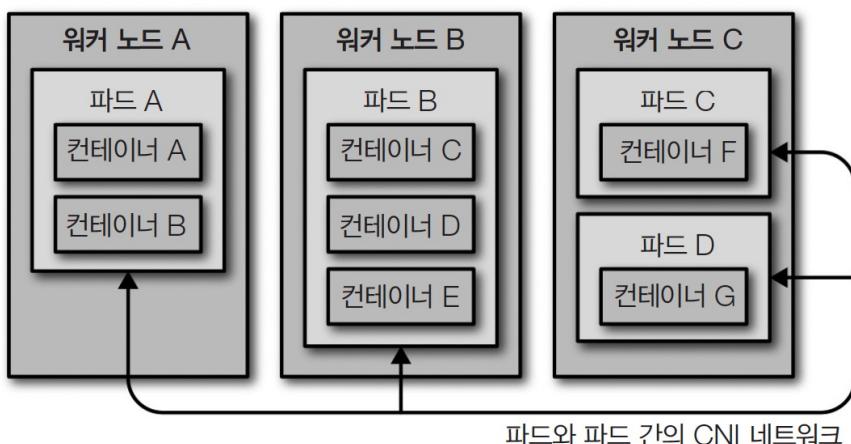
- POD는 1개의 IP만 가짐 (CNI 플러그인이 할당)  
(참고) Multus 를 활용시 POD에 2개 네트워크도 연결 가능
- 2개 인터페이스 활용 사례



그림출처 : <https://medium.com/@nitinbedazzled/introduction-and-setting-up-multus-34ece7ba780b>

# 쿠버네티스 리소스 : 워크로드 오브젝트

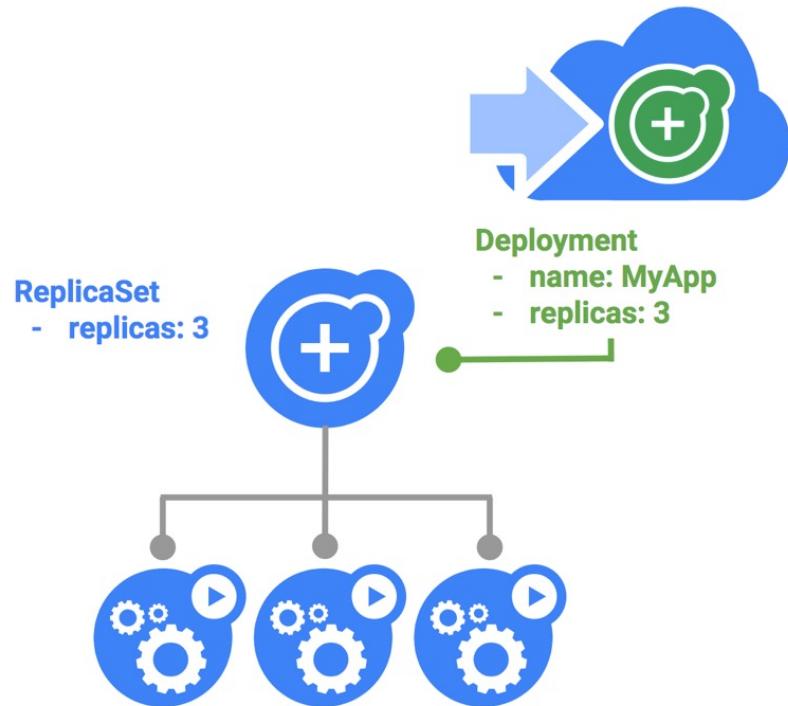
- 컨테이너 네트워크 인터페이스(CNI)  
네트워크로 연결될 파드는 동일 노드에, 다른 노드에 있을 수도 있음. CNI의 역할은 단순히 파드간 연결을 용이하게 만드는 것
- 컨테이너 런타임(예:도커)은 CNI 플러그인 실행파일(예:칼리코)을 호출하여 컨테이너의 네트워킹 네임스페이스에 인터페이스를 추가/제거



그림출처 : <https://www.docker.com/blog/networking-in-docker-enterprise-edition-2-0/>

## 쿠버네티스 리소스 : 컨트롤러 오브젝트

- ReplicaSet : (Node 고장 / Pod 삭제 등 발생 시) 파드 복제하여 개수 유지
- Deployment : 구 버전에서 신 버전으로 복제, 레플리카셋 관리(보통 pod를 일일이 관리하기보다는 deployment 를 서비스 단위로 관리)



그림출처 : <https://www.ianlewis.org/en/bluegreen-deployments-kubernetes>

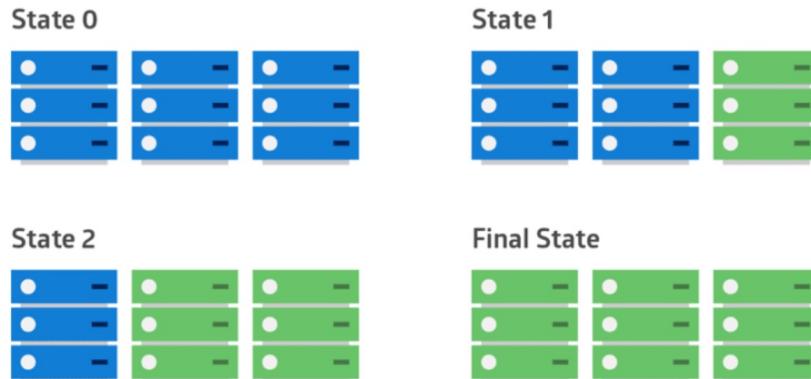
# 쿠버네티스 리소스 : 컨트롤러 오브젝트

- Deployment 생성시 Replica 숫자 지정. 파드를 지워도 ReplicaSet에 의해 다시 재생성

```
osk@osk-MacBook-Pro lkln-kakao % kubectl create deployment deploy-test --image nginx --replicas=3
deployment.apps/deploy-test created
osk@osk-MacBook-Pro lkln-kakao % kubectl get deployments.apps deploy-test -o wide
NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR
deploy-test 1/3 3 1 12s nginx nginx app=deploy-test
osk@osk-MacBook-Pro lkln-kakao %
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod
NAME READY STATUS RESTARTS AGE
deploy-test-76bc8c5457-5xjsm 1/1 Running 0 47s
deploy-test-76bc8c5457-l45sb 1/1 Running 0 47s
deploy-test-76bc8c5457-lppr7 1/1 Running 0 47s
test 1/1 Running 0 12m
osk@osk-MacBook-Pro lkln-kakao %
osk@osk-MacBook-Pro lkln-kakao % kubectl get replicaset.apps
NAME DESIRED CURRENT READY AGE
deploy-test-76bc8c5457 3 3 3 54s
osk@osk-MacBook-Pro lkln-kakao %
osk@osk-MacBook-Pro lkln-kakao % kubectl delete pod deploy-test-76bc8c5457-5xjsm
pod "deploy-test-76bc8c5457-5xjsm" deleted
osk@osk-MacBook-Pro lkln-kakao %
osk@osk-MacBook-Pro lkln-kakao % kubectl get pod
NAME READY STATUS RESTARTS AGE
deploy-test-76bc8c5457-l45sb 1/1 Running 0 7m45s
deploy-test-76bc8c5457-lppr7 1/1 Running 0 7m45s
deploy-test-76bc8c5457-wgpn6 1/1 Running 0 17s
test 1/1 Running 0 19m
```

# 쿠버네티스 리소스 : (참고) App 업데이트 방법

- 룰링 업데이트
  - ✓ 장점 : Down Time 없음. 추가 인프라 투자비 별로 없음.
  - ✓ 단점 : 신/구 버전에 따른 개발 및 검증 사항 고려 필요. 신/구 버전이 같이 활용하는 인프라에서 문제 발생 가능

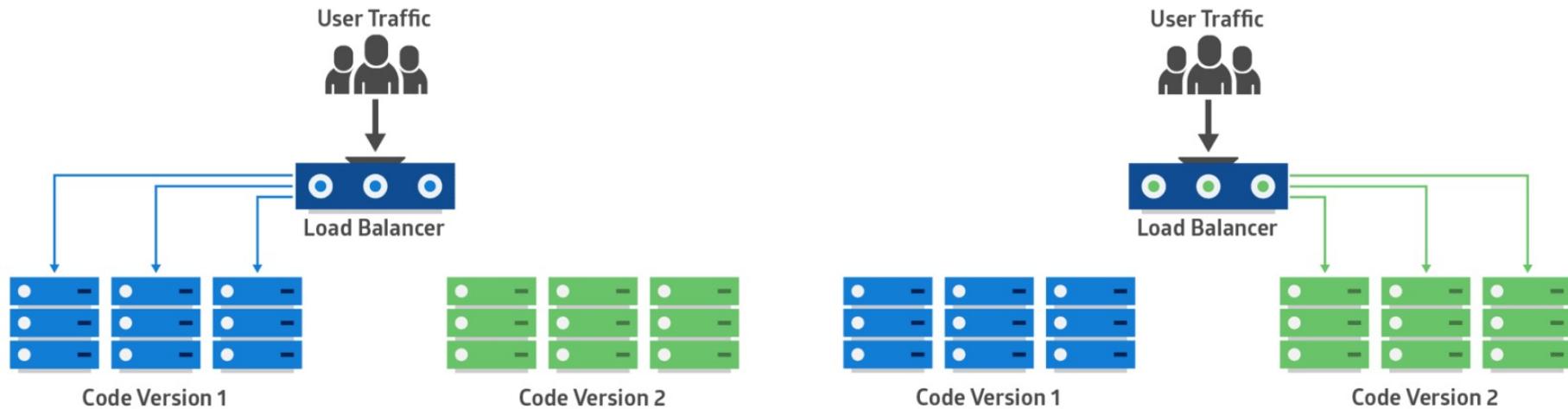


```
osk@osk-MacBook-Pro ~ % kubectl get deploy -n default
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
deploy-test 1 1 1 1 10m

osk@osk-MacBook-Pro ~ % kubectl get deployment.apps deploy-test -n default
Name: deploy-test
Namespace: default
Labels: app=deploy-test
Annotations: deployment.kubernetes.io/revision: 1
Selector: app=deploy-test
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
```

# 쿠버네티스 리소스 : (참고) App 업데이트 방법

- 블루그린 업데이트 방식 ( 레드블랙 업데이트 / AB 배포 )
  - ✓ 장점 : Down Time 없음. 복구가 빠름
  - ✓ 단점 : 인프라 투자가 2배로 필요. 로드 밸런서 추가 필요



그림출처 : dev.to의 Jason Skowronski 73

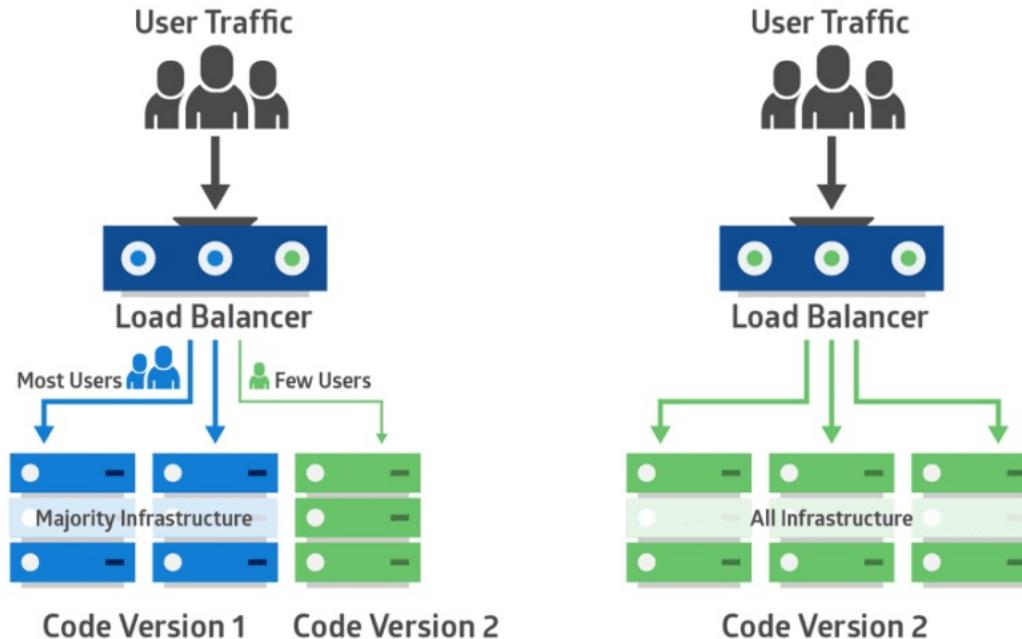
# 쿠버네티스 리소스 : (참고) App 업데이트 방법

- 카나리 업데이트 방식

✓ 장점 : Downtime 없음. 아주 적은 사용자만 새 버전에 유입시키므로 영향도 최소화, 추가 인프라 투자 별로 없음



✓ 단점 : 신/구 버전 공존에 따른 개발 및 검증 사항 고려 필요. 업데이트 시간이 롤링 업데이트보다 더 걸림. 로드 밸런서 추가 필요

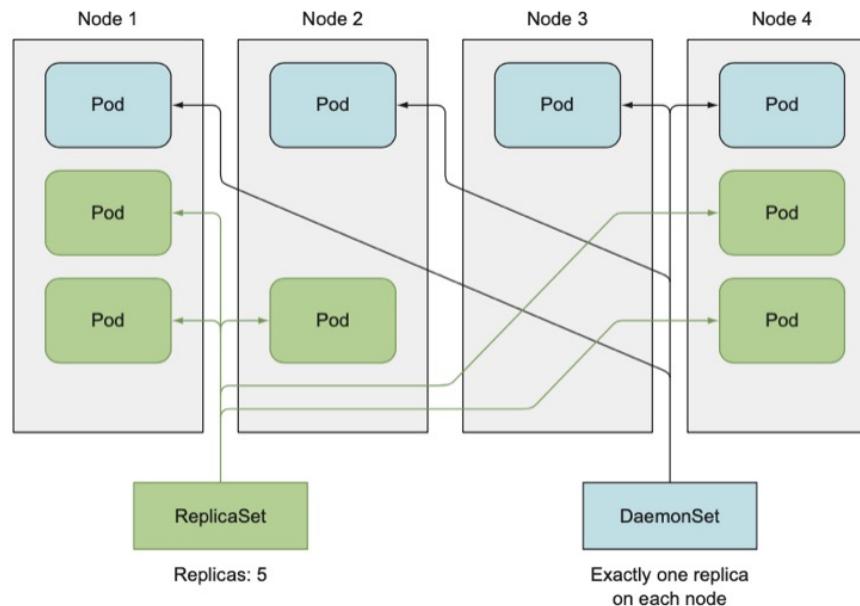


그림출처 : dev.to의 Jason Skowronski 74

# 쿠버네티스 리소스 : 컨트롤러 오브젝트

- DaemonSet

- ✓ 모든 노드에 동일한 파드를 실행시키고 싶은 경우 활용
- ✓ 리소스 모니터링, 로그 수집기 등에 유용. 클러스터에 노드가 추가/삭제 되면 자동으로 파드도 생성/삭제됨



# 쿠버네티스 리소스 : 로드밸런서 오브젝트

- Service : POD 는 임시적인(ephemeral) 오브젝트. TCP/UDP 로드밸런싱을 담당하는 추상적 개념  
파드는 생성시마다 IP가 변하기 때문에, 파드 외부와 통신시 고정 IP를 갖는 'Service'를 통하여 서비스  
kubectl create service 명령어도 서비스 생성이 가능하지만, kubectl expose 명령어를 활용하면 생성 ~ 연결까지 한번에 가능  
- 종류 : Cluster IP(기본값), Node Port, LoadBalancer, ExternalName으로 구분
- Ingress : HTTP 로드밸런서. Service로 라우팅 가능

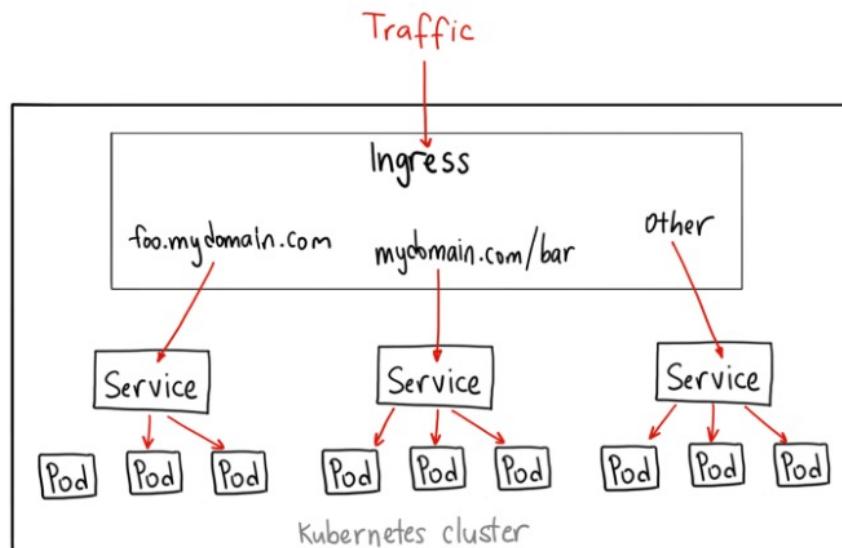


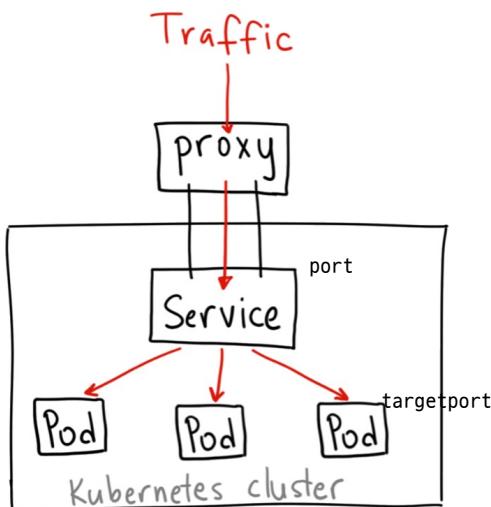
그림 출처 : Medium, Sandeep Dinesh 76

# 쿠버네티스 리소스 : 로드밸런서 오브젝트

- Service 종류 : Cluster IP(기본값), Node Port, LoadBalancer, ExternalName으로 구분

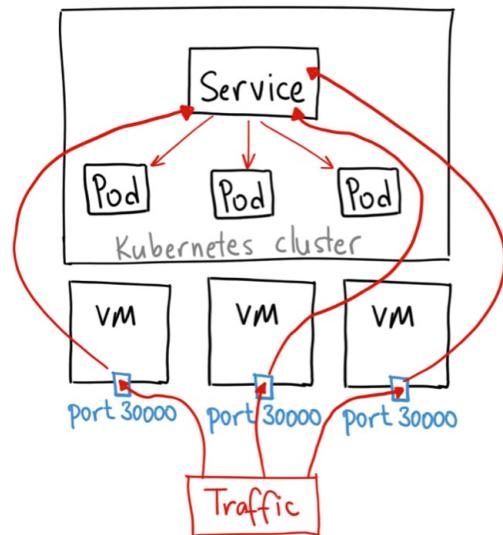
## [Cluster IP]

- 쿠버네티스 클러스터 내에서만 통신 가능
- 외부와 통신이 필요하면 별도 프록시를 두어야함



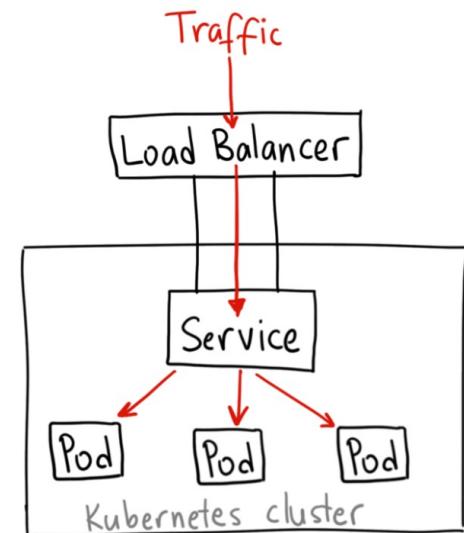
## [Node Port]

- 어떤 노드더라도, '특정 포트'로 들어오게 된다면 서비스가 알맞은 파드로 로드밸런싱 해줌
- 노드에 해당 Pod가 있든 없든 모든 노드 전체에 해당



## [LoadBalancer]

- 각각의 노드에 트래픽 분산처리 가능
- 단일 IP로 여러 노드에 로드 밸런싱 가능  
(현재 카카오 클라우드 내에서는 사설IP로만 가능)

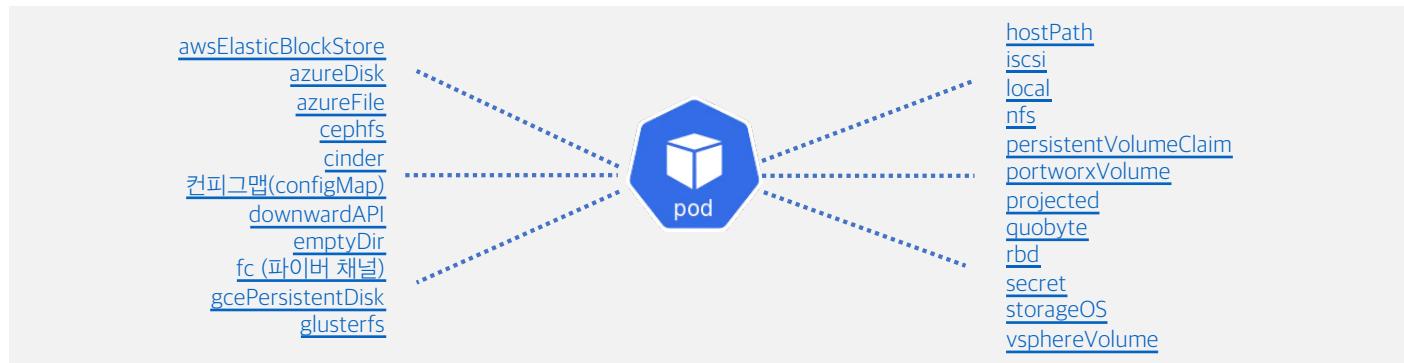


[ExternalName] 외부 서비스를 쿠버네티스 내부에서 호출 시 활용

그림 출처 : Medium, Sandeep Dinesh 77

# 쿠버네티스 리소스 : 스토리지 오브젝트

- 컨테이너 내의 디스크에 있는 파일은 임시적. 퍼시스턴트 볼륨은 지속 존재.



# 쿠버네티스 리소스 : 스토리지 오브젝트(PV, PVC)

- Persistent Volume(PV) : 관리자가 프로비저닝하거나 스토리지 클래스를 사용하여 동적으로 프로비저닝한 클러스터의 스토리지
- Persistent Volume Claim(PVC) : 사용자의 스토리지에 대한 요청. 파드와 비슷.
  - ✓ 파드 : 노드 리소스를 사용 / POD 명세서 내 특정 수준의 리소스(CPU 및 메모리)를 요청
  - ✓ PVC : PV 리소스를 사용 / 클레임 명세서 내 특정 크기 및 접근 모드를 요청(예: ReadWriteOnce, ReadOnlyMany, ReadWriteMany)

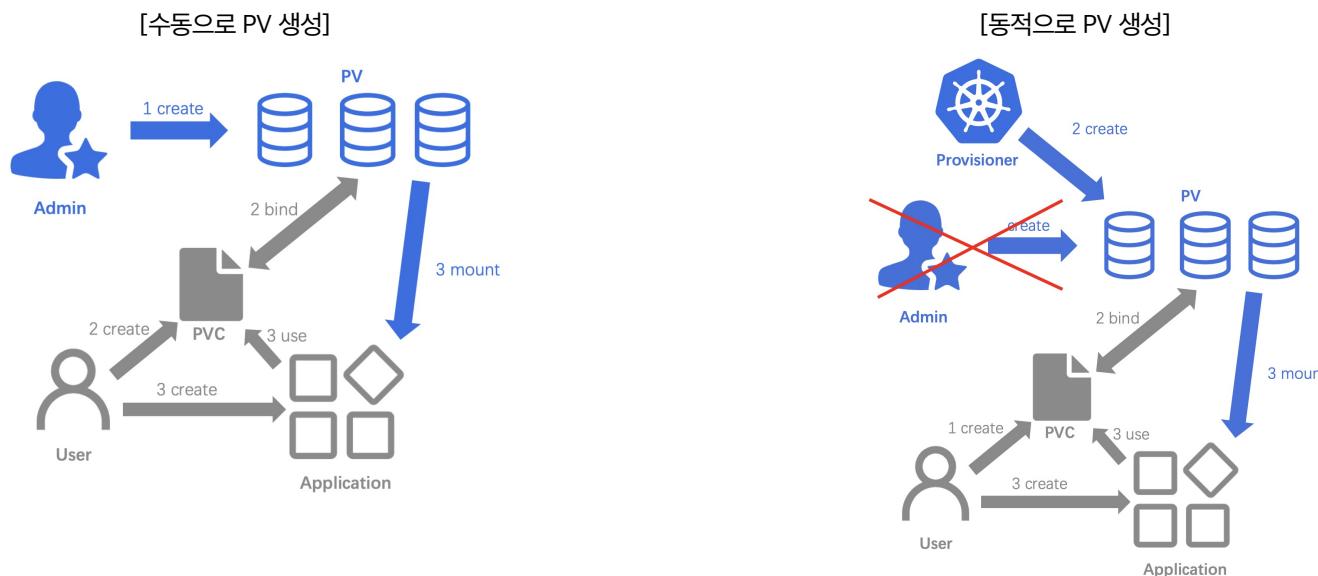


그림 출처 : [https://www.alibabacloud.com/blog/kubernetes-persistent-storage-process\\_596505](https://www.alibabacloud.com/blog/kubernetes-persistent-storage-process_596505) 79