

Kubernetes 기본

First Last name | 김 범 택
Location | 한국정보교육원

서버환경에서 쿠버네티스 클러스터 설치

사전 준비 사항

- 서버의 시간 동기화 (NTP)
- MAC 주소
- 모든 서버는 2GB 메모리, 2 CPU 이상
- “# swapoff -a” 이용한 메모리 스왑(Swap) 비활성화
- /etc/hosts 에 master, worker 의 이름과 IP 주소 작성

서버환경에서 쿠버네티스 클러스터 설치

Kubeadm 으로 쿠버네티스 설치

1. 쿠버네티스 저장소 추가(모든 노드)

```
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -  
# cat <<EOF > /etc/apt/sources.list.d/Kubernetes.list  
deb http://apt.kubernetes.io/ kubernetes-xenial main  
EOF
```

서버환경에서 쿠버네티스 클러스터 설치

Kubeadm 으로 쿠버네티스 설치(Cont.)

2. kubeadm 설치(모든 노드)

본 예에서 쿠버네티스는 도커 컨테이너를 사용하므로 모든 노드에서 도커를 먼저 설치한다.

```
# wget -qO- get.docker.com | sh
```

모든 노드에서 쿠버네티스에 필요한 패키지를 내려받는다

```
# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

서버환경에서 쿠버네티스 클러스터 설치

Kubeadm 으로 쿠버네티스 설치(Cont.)

3. 쿠버네티스 클러스터 초기화(master 노드)

```
# kubeadm init --apiserver-advertise-address [master노드ip] \  
--pod-network-cidr=192.168.0.0/16
```

```
mkdir -p $HOME/.kube  
sudo cp ...  
sudo chown...
```

master 노드에
붙여넣기

```
kubeadm join [master노드ip]:6443 --token ...
```

worker 노드에
붙여넣기

서버환경에서 쿠버네티스 클러스터 설치

Kubeadm 으로 쿠버네티스 설치(Cont.)

4. 컨테이너 네트워크 애드온 설치(master 노드)

쿠버네티스의 컨테이너 간 통신을 위해 flannel, weaveNet 등 오버레이 네트워크를 사용할 수 있지만, 여기에서는 calico 를 기준으로 설정. calico 네트워크 플러그인 설치

```
# kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml
```

```
# kubectl get pods --namespace kube-system
```

```
# kubectl get nodes
```

```
# kubectl reset // 설치된 쿠버네티스는 각 노드에서 reset 이용하여 삭제할 수 있다
```

쿠버네티스 시작하기

k8s의 특성

- 모든 리소스는 오브젝트 형태로 관리
- 명령어의 사용보다, YAML 파일의 사용빈도가 높다
- 여러 개의 컴포넌트로 구성되어 있다



kubernetes

모든 리소스는 오브젝트 형태로 관리

- 오브젝트는 추상화된 집합
- 도커 스웸모드의 서비스도 컨테이너 리소스의 집합을 정의한 것이므로 일종의 오브젝트
- 컨테이너의 집합(Pods), 컨테이너의 집합관리 컨트롤러(Replica Set), 사용자(Account), 노드(Node) 등을 하나의 오브젝트로 사용할 수 있다
- # `kubectl api-resources` 를 통해 오브젝트 확인가능

YAML 파일의 사용빈도가 높다

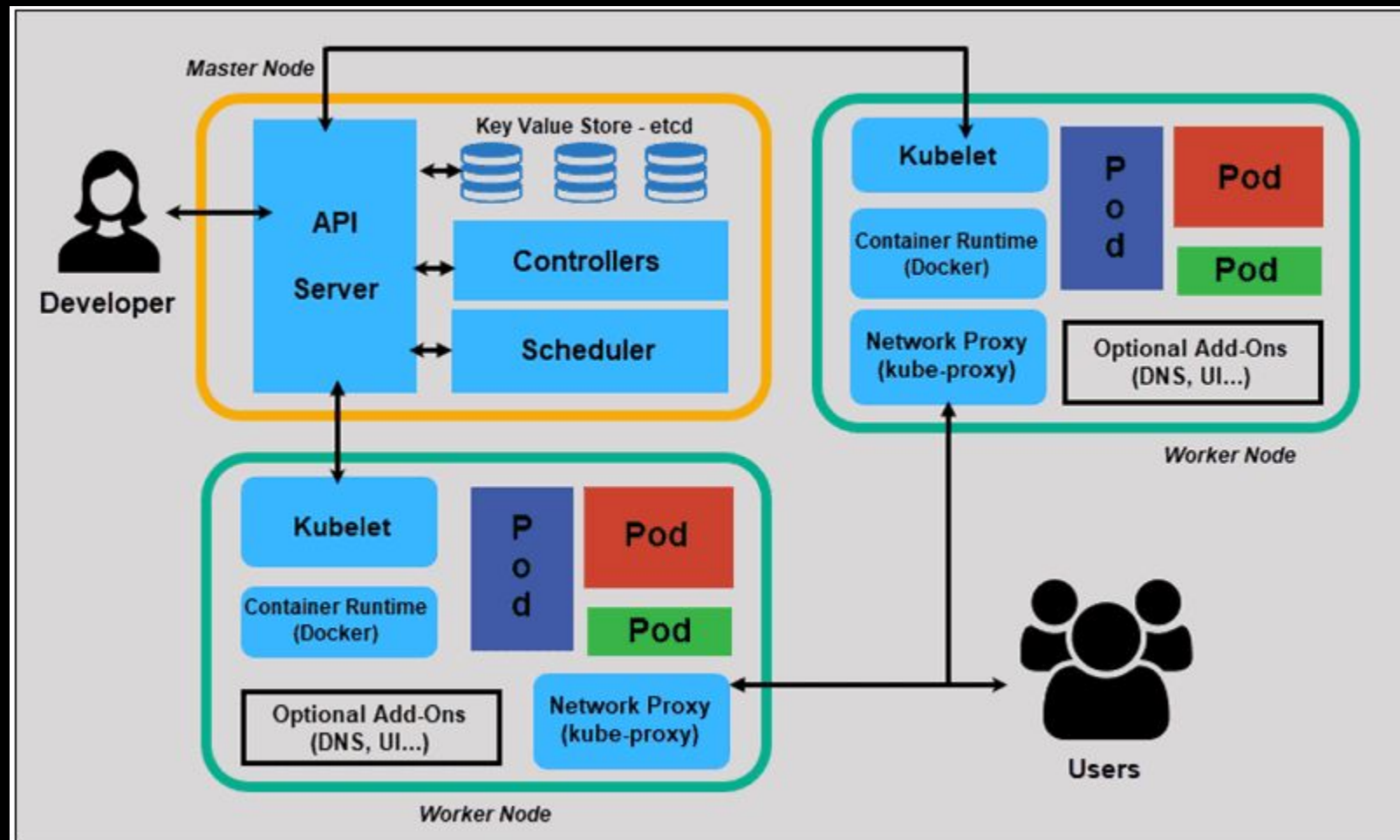
- kubectl 명령어로 쿠버네티스 관리 가능
- docker stack 과 달리 대부분의 리소스 오브젝트들에 대하여 yaml 사용 가능
- yaml 파일을 정의하여 쿠버네티스에 적용시키는 방식으로 배포

여러 개의 컴포넌트로 구성되어 있다

- 마스터 노드 - api 서버(kube-apiserver), 컨트롤러 매니저(kube-controller-manager), 스케줄러(kube-scheduler), DNS서버(coreDNS) 등이 실행
- 모든 노드 - 오버레이 네트워크 구성을 위한 프락시(kube-proxy)와 네트워크 플러그인(calico, flannel 등), 클러스터 구성을 위한 kubelet 에이전트(컨테이너의 생성, 삭제, 마스터와 워커 노드 간의 통신 역할을 담당)
- 컴포넌트들은 기본적으로 도커 컨테이너로서 실행되며 docker container ls 를 통해 확인 가능
- 쿠버네티스의 입장에서 도커 데몬 또한 하나의 컴포넌트. 쿠버네티스에서는 반드시 도커를 사용해야 하는 것은 아니며, OCI(Open Container Initiative)라는 런타임 표준을 구현한 CRI(Container Runtime Interface)를 갖추고 있다면 어떠한 컨테이너를 써도 문제없다

"kubelet 에이전트가 모든 노드에서 실행, 마스터에서 API서버 등이 컨테이너로 실행"

k8s 아키텍처



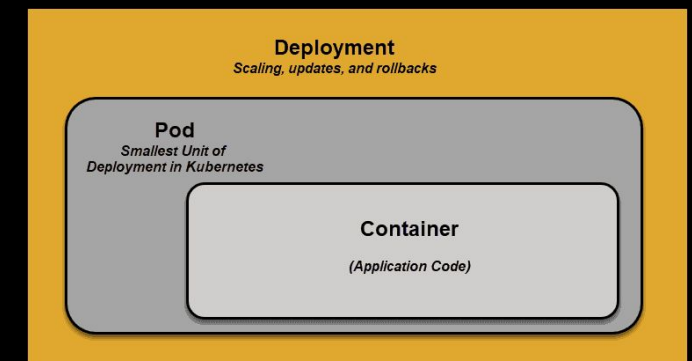
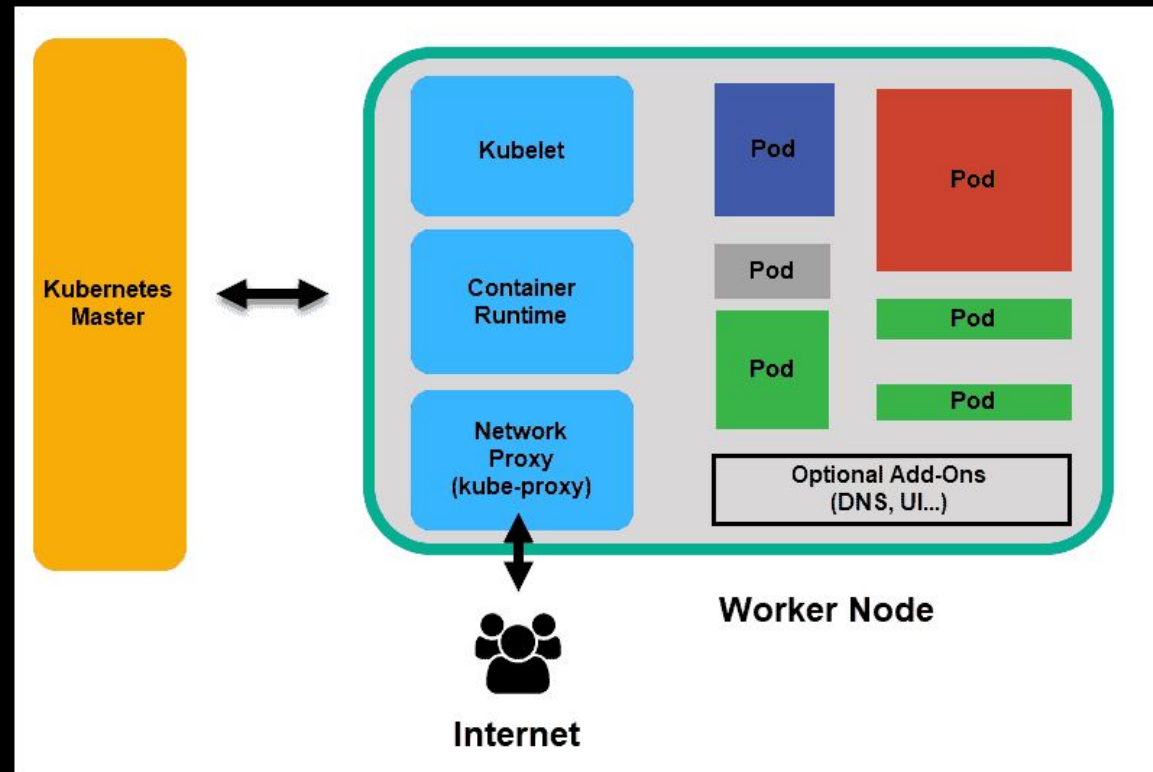
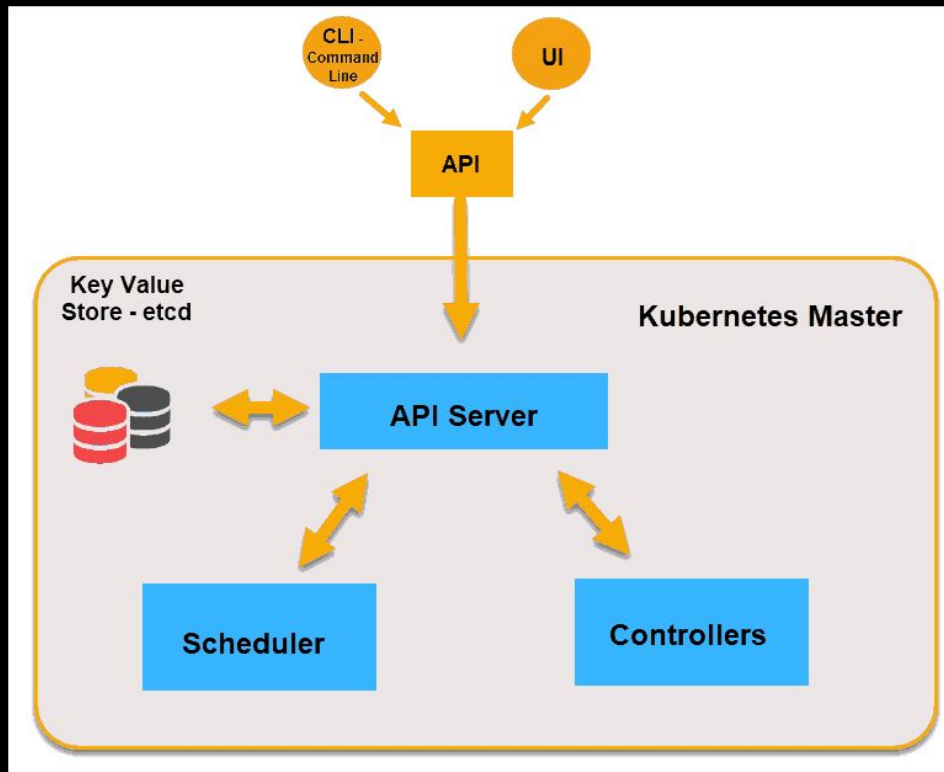
master node

- api-server
- controller
- scheduler
- key-value store(etcd)

worker node

- kubelet
- runtime
- network proxy

k8s 아키텍처



포드(Pod) : 컨테이너를 다루는 기본단위

- 컨테이너 애플리케이션의 기본 단위를 포드(Pod)라고 부르며, 포드는 1개 이상의 컨테이너로 구성된 컨테이너의 집합
- 기본단위
 - 도커 엔진 : 도커 컨테이너
 - 도커 스웜 : 서비스(여러 개의 컨테이너로 구성)

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
spec:
  containers:
    - name: my-nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
          protocol: TCP
```

YAML 파일에서 정의한 오브젝트의 API 버전. 오브젝트의 종류 및 개발 성숙도에 따라 **apiVersion**의 설정값이 달라질 수 있다

라벨, 주석, 이름 등과 같은 리소스의 부가 정보들을 입력

포드(Pod) : 컨테이너를 다루는 기본단위(Cont.)

- yaml 파일은 `kubectl apply -f` 명령어로 쿠버네티스에 생성
 - # `kubectl apply -f nginx-pod.yaml`
 - # `kubectl get pods`
 - # `kubectl describe pod my-nginx-pod`
 - # `kubectl exec -it my-nginx-pod bash`
 - # `kubectl logs my-nginx-pod`
 - # `kubectl delete -f nginx-pod.yaml`
 - # `kubectl delete pods my-nginx-pod`
- 여러 리눅스 네임스페이스(namespace)를 공유하는 여러 컨테이너들을 추상화된 집합으로 사용가능

레플리카셋(Replica Set) : 일정 개수의 포드를 유지하는 컨트롤러

- 정해진 수의 동일한 포드가 항상 실행되도록 관리
- 노드 장애 등의 이유로 포드를 실행할 수 없다면, 다른 노드에서 포드를 다시 생성

```
apiVersion: v1
kind: ReplicaSet
metadata:
  name: replicaset-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-nginx-pods-label
  template:
    metadata:
      name: my-nginx-pod
    labels:
      app: my-nginx-pods-label
    spec:
```

레플리카셋 정의

포드 정의

```
containers:
- name: nginx
  image: nginx:latest
  ports:
  - containerPort: 80
```

디플로이먼트(Deployment) : 레플리카셋, 포드의 배포를 관리

- 디플로이먼트는 레플리카셋의 상위 오브젝트이기 때문에 디플로이먼트를 생성하면 해당 디플로이먼트에 대응하는 레플리카셋도 함께 생성됨

```
# kubectl apply -f deployment-nginx.yaml  
# kubectl get deployment  
# kubectl get rs ; kubectl get pods  
# kubectl delete deploy my-nginx-deployment
```

- 애플리케이션 업데이트할 때 레플리카셋의 변경 사항을 저장하는 리비전(revision)을 남겨 롤백을 가능하게 해 주고, 무중단 서비스를 위해 포드의 롤링 업데이트의 전략을 지정할 수도 있다.

디플로이먼트(Deployment) : 레플리카셋, 포드의 배포를 관리(Cont.)

- 애플리케이션의 버전 업데이트 및 롤백

```
# kubectl apply -f deployment-nginx.yaml --record
# kubectl set image deployment my-nginx-deployment nginx=nginx:1.11 --record
(set 명령 대신, yaml 파일에서 기존의 버전을 1.11 로 변경한 다음 apply 해도 동일함)
# kubectl get pods ; kubectl get rs
# kubectl rollout history deployment my-nginx-deployment
# kubectl rollout undo deployment my-nginx-deployment --to-revision=1
# kubectl get rs
# kubectl describe deploy my-nginx-deployment
# kubectl delete deployment,pod,rs --all
```

서비스(Service) : 포드를 연결하고 외부에 노출

- 쿠버네티스에서는 디플로이먼트를 생성할 때 포드를 외부로 노출하지 않으며, 디플로이먼트의 YAML 파일에는 단지 포드의 애플리케이션이 사용할 내부 포트만 정의
- 포드의 포트를 외부로 노출하여 외부접속을 허용하거나 다른 디플로이먼트의 포드들이 내부적으로 접근하려면 서비스(Service)를 생성해야 한다
 - 여러 개의 포드에 쉽게 접근할 수 있도록 고유한 도메인 이름을 부여
 - 여러 개의 포드에 접근할 때, 요청을 분산하는 로드 밸런서 기능을 수행
 - 클라우드 플랫폼의 로드 밸런서, 클러스터 노드의 포트 등을 통해 포드를 외부로 노출

서비스(Service) : 포드를 연결하고 외부에 노출(Cont.)

- 서비스 타입

- **ClusterIP** : 쿠버네티스 내부에서만 포드들에 접근가능. 외부로 포드를 노출하지 않기 때문에 쿠버네티스 클러스터 내부에서만 사용하는 포드에 적합
- **NodePort** : 포드에 접근할 수 있는 포트를 클러스터의 모든 노드에 동일하게 개방. 따라서 외부에서 포드에 접근할 수 있는 서비스 타입. 접근할 수 있는 포트는 랜덤으로 정해지지만, 특정 포트로 접근할 수 있도록 설정할 수 있다
- **LoadBalancer** : 클라우드 플랫폼에서 제공하는 로드 밸런서를 동적으로 프로비저닝해 포드에 연결. 외부에서 포드에 접근할 수 있는 서비스 타입. 일반적으로 AWS, GCP 등과 같은 클라우드 플랫폼 환경에서만 사용할 수 있다

서비스(Service) : 포드를 연결하고 외부에 노출(Cont.) ClusterIP

■ hostname-svc-clusterip.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-svc-clusterip
spec:
  ports:
    - name: web-port
      port: 8080
      targetPort: 80
  selector:
    app: webserver
  type: ClusterIP
```

- **spec.selector** : 이 서비스에서 어떠한 라벨을 가지는 포드에 접근할 수 있게 만들 것인지 결정
- **spec.ports.port** : 생성된 서비스는 클러스터 내부에서만 사용할 수 있는 고유한 IP(ClusterIP)를 할당받는다. **port** 항목에는 서비스의 IP에 접근할 때 사용할 포트 설정
- **spec.ports.targetPort** : **selector** 항목에서 정의한 라벨에 의해 접근대상이 된 포드들이 내부적으로 사용하고 있는 포트를 입력

서비스(Service) : 포드를 연결하고 외부에 노출(Cont.) NodePort

- 클러스터 외부에서 접근가능. 모든 노드(Node)의 특정 포트(Port)를 개방해 서비스에 접근하는 방법

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-svc-nodeport
spec:
  ports:
    - name: web-port
      port: 8080
      targetPort: 80
      nodePort: 31000
  selector:
    app: webserver
  type: NodePort
```

```
# kubectl apply -f hostname-svc-nodeport.yaml
# kubectl get nodes -o wide
```

- NodePort 서비스자체를 외부로 연결하기 보다는 인그레스(Ingress) 에서 간접적으로 사용되는 경우가 많음

서비스(Service) : 포드를 연결하고 외부에 노출(Cont.) NodePort

- GKE에서 k8s 를 사용하고 있다면 각 노드의 랜덤한 포트에 접근하기 위해 별도로 방화벽 설정을 추가해 주어야 한다. AWS 에서도 SG 에 별도의 inbound 규칙을 추가하여 해당 포트로의 접속을 허용해 주어야 한다.

```
$ gcloud compute firewall-rules create test-nodeport-permit --allow=tcp:31000#규칙추가 $ gcloud compute  
firewall-rules delete test-nodeport-permit #규칙삭제
```

서비스(Service) : 포드를 연결하고 외부에 노출(Cont.) LoadBalancer

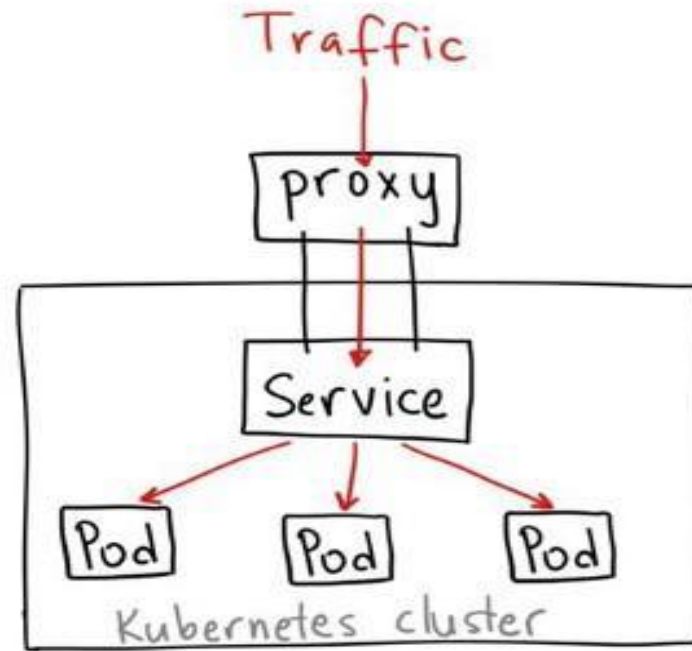
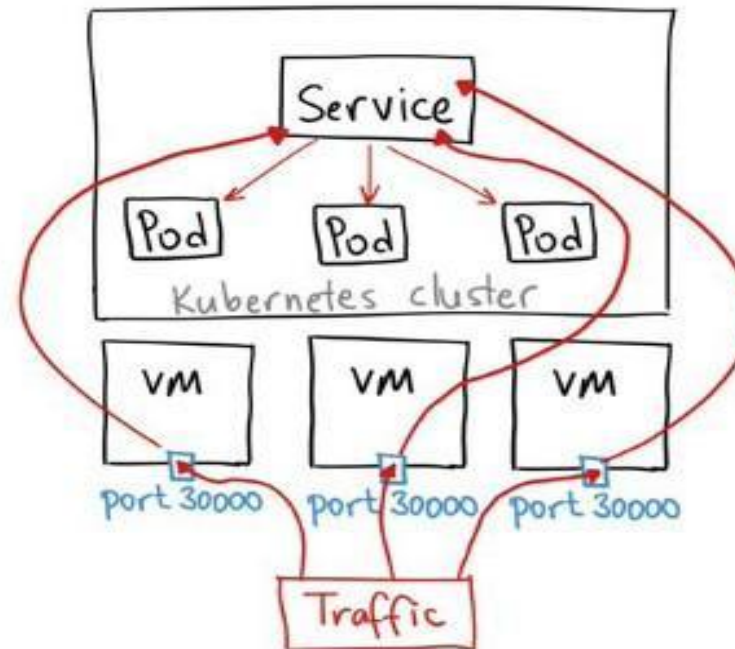
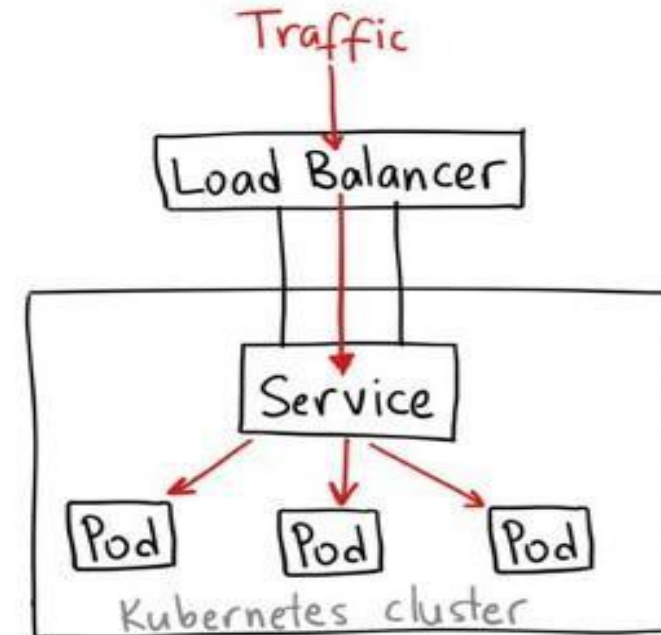
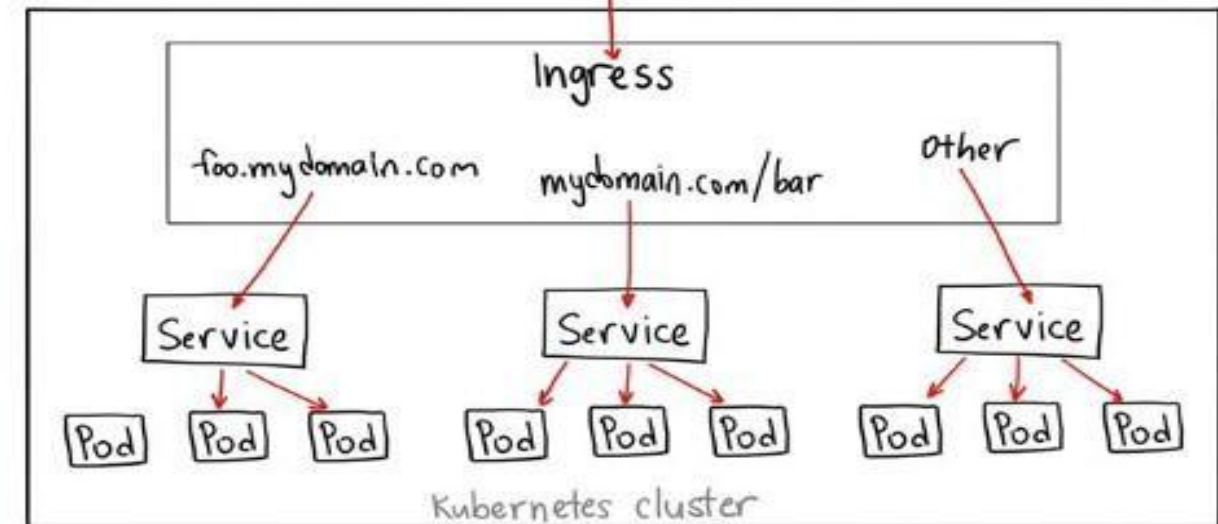
- 서비스 생성과 동시에 로드 밸런서를 새롭게 생성해 포드와 연결
- 클라우드 플랫폼으로 부터 도메인 이름,IP를 할당받기 때문에 NodePort 보다 쉽게 포드에 접근가능

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-svc-lb
spec:
  ports:
    - name: web-port
      port: 80
      targetPort: 80
  selector:
    app: webserver
  type: LoadBalancer
```

서비스(Service)

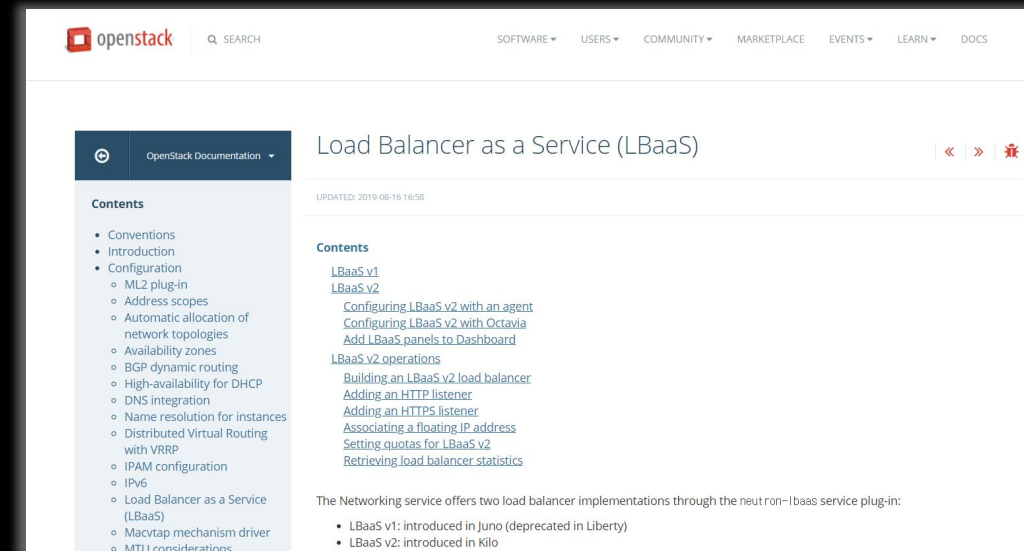
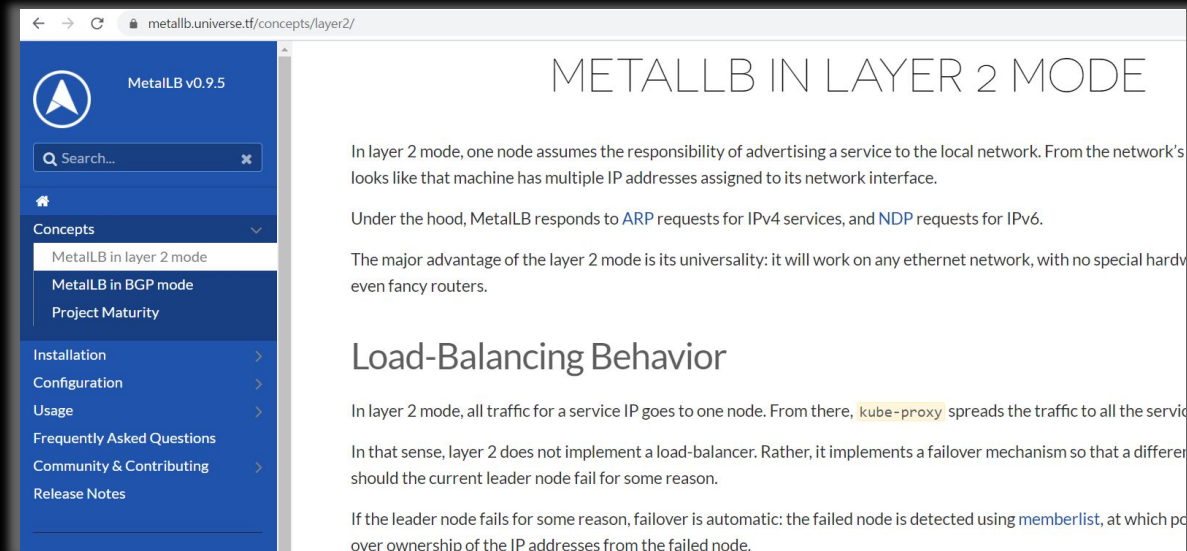
Services

- ClusterIP
- NodePort
- LB
- Ingress

ClusterIP**LoadBalancer****NodePort****Traffic****Ingress**

온프레미스 환경에서 LoadBalancer 서비스의 사용

- 쿠버네티스 자체에서 LoadBalancer 를 제공하지는 않는다
- MetalLB, 오픈스택의 LBaaS 등의 오픈소스 프로젝트를 사용하면 LB 서비스를 사용할 수 있음



쿠버네티스 리소스의 관리와 설정

네임스페이스(Namespace) : 리소스의 구분

- 용도에 따라 컨테이너와 그에 관련된 리소스를 구분짓는 그룹의 역할
- 포드, 레플리카셋, 디플로이먼트, 서비스 등과 쿠버네티스 리소스들이 묶여 있는 가상공간

```
root@lab-k8s:~# kubectl get namespace
NAME                STATUS    AGE
default             Active    20h
kube-node-lease     Active    20h
kube-public         Active    20h
kube-system         Active    20h
root@lab-k8s:~#
```

- 기본적으로 위와 같은 네임스페이스가 존재하며 “# kubectl get pods --namespace default” 와 같은 방법으로 네임스페이스별 동작하는 pod 를 확인할 수 있음.
- --namespace 없을 경우 기본적으로 “default” 네임스페이스를 의미함
- kube-system 는 쿠버네티스 클러스터 구성에 필수적인 컴포넌트들과 설정값 등이 존재
- 노드(nodes)는 쿠버네티스의 오브젝트 이지만, 네임스페이스에는 속하지 않는 오브젝트

네임스페이스(Namespace) : 리소스의 구분

- **kube-system** 네임스페이스에는 쿠버네티스의 포드, 서비스 등을 이름으로 찾을 수 있도록 하는 DNS 서버의 서비스가 미리 생성되어 있다

```
root@lab-k8s:~# kubectl get svc -n kube-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default-http-backend	NodePort	10.96.11.226	<none>	80:32528/TCP	38h
kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP,53/TCP	38h
metrics-server	ClusterIP	10.96.2.31	<none>	443/TCP	38h

- 쿠버네티스 클러스터를 여러 명이 동시에 사용해야 한다면 사용자마다 네임스페이스를 별도로 생성해 사용하도록 설정할 수 있다.
- 용도에 따라 네임스페이스를 여러 개 만들어 특정 목적의 디플로이먼트, 서비스들은 특정 네임스페이스에서만 존재하도록 만들 수도 있다
- 네임스페이스의 리소스들은 논리적으로 구분되어 있는 것이며 물리적으로 격리된 것이 아니므로 서로다른 네임스페이스에서 생성된 포드가 같은 노드에 존재할 수 있음

네임스페이스(Namespace) : 라벨과 네임스페이스

- `# kubectl get pods -l app=testwebserver` 와 같이 특정 라벨이 부착된 포드 출력가능
- 네임스페이스는 라벨보다 넓은 의미로 사용됨
 - **ResourceQuota** 오브젝트 사용하여 특정 네임스페이스에서 생성된 포드의 자원 사용량 제한
 - 애드미션 컨트롤러라는 기능을 이용하여 특정 네임스페이스에서 생성되는 포드에는 항상 사이드카 컨테이너를 붙이도록 할 수 있다
 - 포드, 서비스 등의 리소스를 격리함으로써 편리하게 구분
- 네임스페이스는 컨테이너의 격리된 공간을 생성하기 위해 리눅스 커널 자체를 사용
- 일반적으로 네트워크, 마운트, 프로세스 네임스페이스등을 의미하며 리눅스 네임스페이스와는 별개

네임스페이스(Namespace) : 사용하기

- yaml 파일 작성하기

```
root@lab-k8s:~/btstore/kube/test# cat namespace1.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production
root@lab-k8s:~/btstore/kube/test# kubectl apply -f namespace1.yaml
namespace/production created
root@lab-k8s:~/btstore/kube/test# kubectl create namespace btnamespace
namespace/btnamespace created
root@lab-k8s:~/btstore/kube/test# kubectl get ns | grep production
production          Active    90s
root@lab-k8s:~/btstore/kube/test# kubectl get ns | grep btnamespace
btnamespace         Active   44s
root@lab-k8s:~/btstore/kube/test#
```

네임스페이스(Namespace) : 사용하기

- 특정 네임스페이스에 리소스를 생성
- 앞서 작성한 deployment 에 “namespace: ...” 를 추가

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-webtest-deployment
  namespace: btnamespace
spec:
  replicas: 3
  selector:
```

- 배포와 확인

```
root@lab-k8s:~/btstore/kube/test# kubectl apply -f deployment-webtest-blue.yaml
deployment.apps/my-webtest-deployment created
root@lab-k8s:~/btstore/kube/test#
root@lab-k8s:~/btstore/kube/test# kubectl get pods -n btnamespace
NAME                                READY   STATUS    RESTARTS   AGE
my-webtest-deployment-68546bcb5-fxswf 1/1     Running   0           38s
my-webtest-deployment-68546bcb5-gvghz 1/1     Running   0           38s
my-webtest-deployment-68546bcb5-mr74q 1/1     Running   0           38s
root@lab-k8s:~/btstore/kube/test#
```

컨피그맵(Configmap), 시크릿(Secret)

설정값을 포드에 전달하기

컨피그맵(Configmap)과 시크릿(Secret)의 이해

- 도커이미지는 빌드 후 불변의 상태를 갖기때문에 설정옵션을 유연하게 변경할 수 없음
- YAML 파일과 설정값을 분리할 수 있는 것이 Configmap, Secret
 - Configmap : 설정값
 - Secret : 비밀키 등
- 사용비교

```
spec:
  containers:
  - name: my-srv
    env
      - name:
        LOG_LEVEL
          value: INFO
    inage: ...
```

```
spec:
  containers:
  - name: my-srv
    env
      - name:
        LOG_LEVEL
          value: DEBUG
    inage: ...
```

VS

```
spec:
  containers:
  - name: my-srv
    env
      valueFrom:
        configMapKeyRef:
          name: log-level-config
          key: LOG_LEVEL
```

 LOG_LEVEL : INFO LOG_LEVEL :
DEBUG

컨피그맵(Configmap) : 사용방법의 이해

- 컨피그맵의 생성
 - yaml 파일 이용하기
 - “kubectl create configmap <컨피그맵 이름> <각종 설정값 들>” 명령어 이용하기

```
root@master1: ~  
File Edit View Search Terminal Help  
root@master1:~# kubectl create configmap testmap --from-literal k8s=kubernetes --from-literal con  
tainer=docker  
configmap/testmap created  
root@master1:~#  
root@master1:~# kubectl get configmap  
NAME          DATA  AGE  
kube-root-ca.crt  1     34m  
log-level-cfgmap  1     22m  
testmap         2     15s  
root@master1:~#
```

```
root@master1: ~  
File Edit View Search Terminal Help  
root@master1:~# kubectl describe configmap log-level-cfgmap  
Name:          log-level-cfgmap  
Namespace:     default  
Labels:        <none>  
Annotations:   <none>  
  
Data  
====  
LOG_LEVEL:  
----  
DEBUG  
Events: <none>  
root@master1:~#
```

컨피그맵(Configmap) : 사용방법의 이해

- 컨피그맵을 포드에서 사용하기
 - 컨피그맵의 값을 컨테이너의 환경 변수로 사용하기
컨피그맵에 저장된 **key-value** 데이터가 컨테이너의 환경 변수 **key-value** 로 사용되기 때문에 셸에서 `echo $LOG_LEVEL` 과 같은 방법으로 값을 확인할 수 있다
 - 컨피그맵의 값을 포드 내부의 파일로 마운트하여 사용하기
`LOG_LEVEL=INFO` 라는 값을 가지는 컨피그맵을 `/etc/config/log_level` 이라는 파일로 마운트하면 `log_level` 파일에는 `INFO` 라는 값이 설정된다. 이때 파일이 위치할 경로는 별도로 설정할 수 있다. `nginx.conf` 등의 파일을 통해 설정값을 읽어 들인다면 이 방법을 사용할 수 있다.

컨피그맵(Configmap) : 사용방법의 이해

- 컨피그맵의 데이터를 컨테이너의 환경 변수로 가져오기

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat env-configmap.yml
apiVersion: v1
kind: Pod
metadata:
  name: container-env-test
spec:
  containers:
  - name: my-container
    image: busybox
    args: ['tail', '-f', '/dev/null']
    envFrom:
    - configMapRef:
        name: log-level-cfgmap
    - configMapRef:
        name: testmap
root@master1:~/k8s# kubectl apply -f env-configmap.yml
pod/container-env-test created
root@master1:~/k8s#
```

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# kubectl exec container-env-test env
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=container-env-test
LOG_LEVEL=DEBUG
container=docker
k8s=kubernetes
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
```


컨피그맵(Configmap) : 사용방법의 이해

- 컨피그맵의 내용을 파일로 포드 내부에 마운트 하기
 - testmap 컨피그맵에 존재하는 모든 key-value 쌍을 /etc/config 디렉토리에 위치시키기

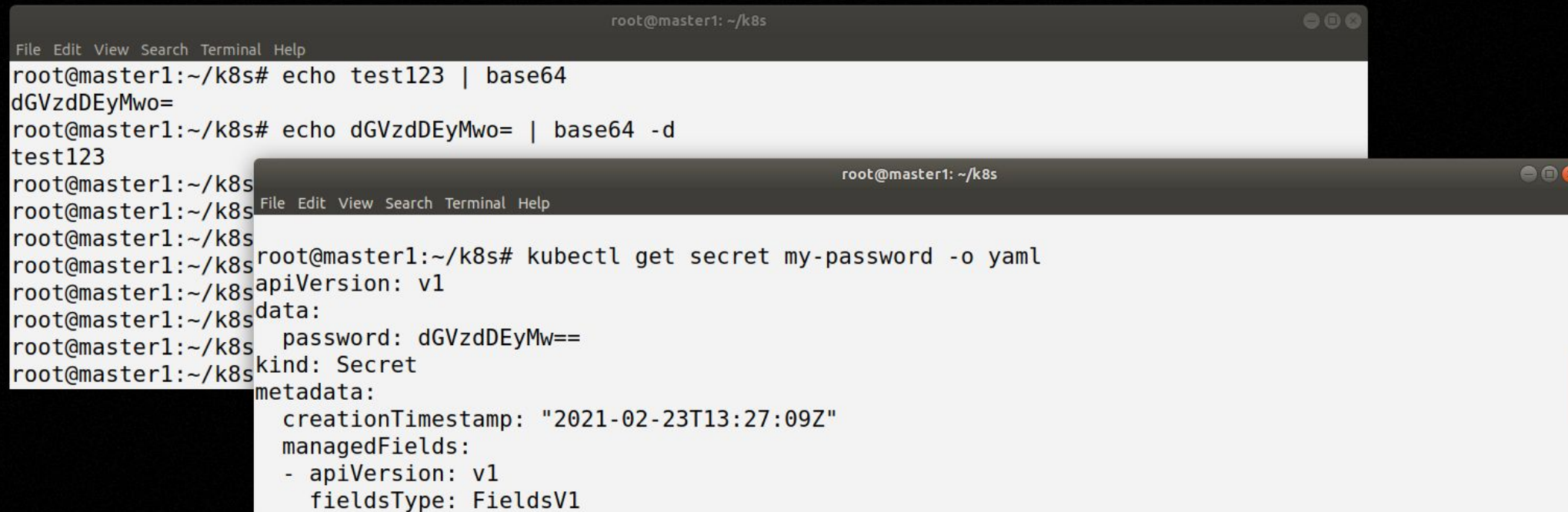
```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat mount-configmap.yml
apiVersion: v1
kind: Pod
metadata:
  name: container-volume-test
spec:
  containers:
    - name: my-container
      image: busybox
      args: ['tail', '-f', '/dev/null']
      volumeMounts:
        - name: configmap-volume
          mountPath: /etc/config

  volumes:
    - name: configmap-volume
      configMap:
        name: testmap
root@master1:~/k8s#
root@master1:~/k8s# kubectl apply -f mount-configmap.
pod/container-volume-test created
root@master1:~/k8s#
```

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# kubectl exec container-volume-test -- ls /etc/config
container
root@master1:~/k8s# kubectl exec container-volume-test -- cat /etc/config/container
dockerroot@master1:~/k8s#
root@master1:~/k8s#
root@master1:~/k8s# kubectl exec container-volume-test -- cat /etc/config/k8s
kubernetesroot@master1:~/k8s#
root@master1:~/k8s#
```


시크릿(Secret) : 사용방법의 이해

- kubectl describe 와 kubectl get secrets -o yaml 로 my-password 내용확인하기



```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# echo test123 | base64
dGVzdDEyMwo=
root@master1:~/k8s# echo dGVzdDEyMwo= | base64 -d
test123
root@master1:~/k8s# kubectl get secret my-password -o yaml
apiVersion: v1
data:
  password: dGVzdDEyMwo=
kind: Secret
metadata:
  creationTimestamp: "2021-02-23T13:27:09Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
```

- yaml 파일로부터 시크릿을 생성할 때에도 데이터의 값에 base64로 인코딩된 문자열을 사용해야 한다

시크릿(Secret) : 사용방법의 이해

- 시크릿에 저장된 모든 key-value 쌍을 포드의 환경 변수로 가져오기

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat env-from-secret.yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-example
spec:
  containers:
  - name: my-container
    image: busybox
    args: ['tail', '-f', '/dev/null']
    envFrom:
    - secretRef:
        name: my-password
root@master1:~/k8s#
```

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# kubectl exec secret-env-example env
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future
version. Use kubectl exec [POD] -- [COMMAND] instead.
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=secret-env-example
password=test123
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
```

- 시크릿을 사용할 때 yaml 파일에 base64로 인코딩한 값을 입력했더라도 시크릿을 포드의 환경 변수나 볼륨으로 가져오면 base64 로 디코딩된 원래의 값을 사용하게 된다

시크릿(Secret) : 사용방법의 이해

- 이미지 레지스트리 접근을 위한 시크릿의 사용
 - "kubectl get secrets" 에서의 TYPE Opaque 는 시크릿 생성시 "generic" 사용으로 만들어진 일반적인 타입
 - 비공개 레지스트리 접근시 인증설정 시크릿
사설 레지스트리, 클라우드 레지스트리를 사용하고 있다면 로그인 인증절차를 거친 뒤, 이미지를 다운로드 할 수 있어야 한다. 단일 서버에서 도커데몬을 사용할 경우 "docker login" 을 사용하지만, 쿠버네티스에서는 시크릿을 생성하는 명령어에서 직접 도커 로그인 정보를 입력하여 사설 저장소로 접근할 수 있다

```
$kubectl create secret docker-registry registry-auth-by-cmd --docker-username=beomtaek  
--docker-password=test123
```

```
$kubectl create secret docker-registry registry-auth-by-registry \  
--docker-username=beomtaek --docker-password=test123 \  
--docker-server=beomtaek.registry.com
```

--docker-server 옵션이 없으면 도커 허브(docker.io)를 사용, 사용하면 다른 사설 레지스트리를 사용할 수 있다

시크릿(Secret) : 사용방법의 이해

- 이미지 레지스트리 접근을 위한 시크릿의 사용

```
root@master1: ~/k8s
File Edit View Search Terminal Help

root@master1:~/k8s# kubectl get secrets
NAME                                TYPE                                DATA  AGE
default-token-m9ft2                kubernetes.io/service-account-token 3      129m
my-password                        Opaque                             1      55m
our-password                       Opaque                             2      50m
root@master1:~/k8s# kubectl create secret docker-registry registry-auth-registry --docker-username=
beomtaek --docker-password=test123 --docker-server=beomtaek.registry.com
secret/registry-auth-registry created
root@master1:~/k8s#
root@master1:~/k8s# kubectl get secrets
NAME                                TYPE                                DATA  AGE
default-token-m9ft2                kubernetes.io/service-account-token 3      129m
my-password                        Opaque                             1      55m
our-password                       Opaque                             2      51m
registry-auth-registry             kubernetes.io/dockerconfigjson      1      7s
root@master1:~/k8s#
```

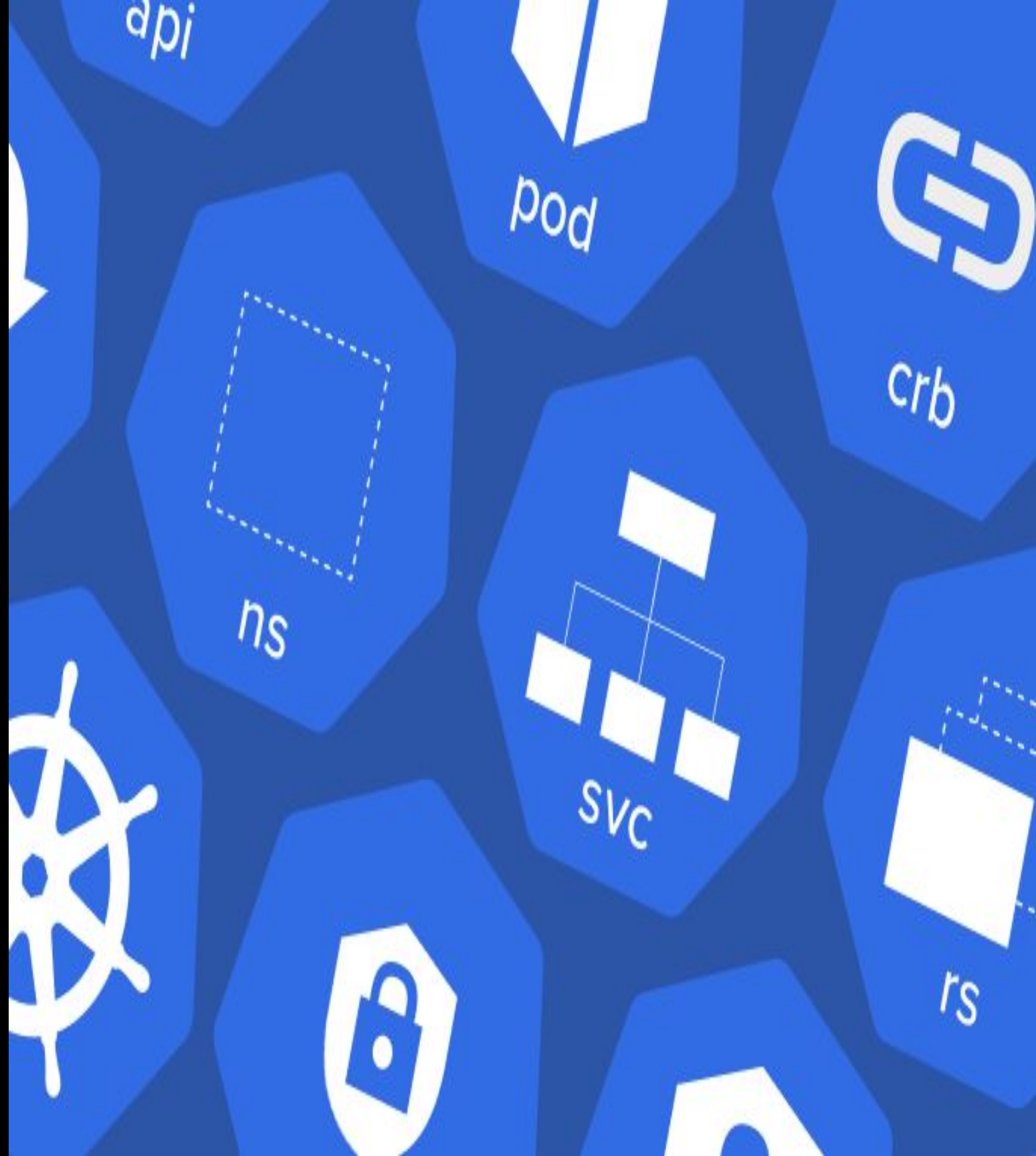
시크릿(Secret) : 사용방법의 이해

- 이미지 레지스트리 접근을 위한 시크릿의 사용
 - 앞서 생성한 시크릿은 디플로이먼트, 포드 등에서 사설 레지스트리로부터 이미지를 불러올 때 사용할 수 있다. 예를 들어 도커 허브의 사설 저장소(**Private Repository**)에 저장된 이미지를 통해 포드를 생성하려면 다음과 같이 **YAML** 파일에서 **imagePullSecret** 항목을 정의한다

```
apiVersion: apps/v1
kind: Deployment
...
spec:
  containers:
    - name: test-container
      image: [직접 만든 이미지명]
  imagePullSecrets:
    - name: registry-auth-registry
```

Kubernetes 의 고급 기능들

- 포드의 자원 사용량 제한
- ingress (작성중)
- 퍼시스턴트 볼륨(PV)과 퍼시스턴트 볼륨 클레임(PVC)



포드(Pod)의 자원사용량 제한

- 컨테이너 오케스트레이션 툴은 여러 대의 서버를 묶어 리소스 풀로 사용가능
- 클러스터의 CPU, 메모리 등의 자원이 부족할 때, 필요한 용량을 동적으로 추가하여 수평확장
- 자원 사용률의 확장

: 쿠버네티스에서 실행중인 컨테이너의 CPU, 메모리의 사용률이 현저히 낮거나, 유휴 상태의 컨테이너에 불필요하게 많은 자원을 할당했다면 이는 컴퓨팅 자원의 활용률이 낮다고 할 수 있다. 이러한 상황을 방지하려면 각 컨테이너의 자원 사용량을 적절히 제한해야 하며, 남은 자원을 어떻게 사용할 수 있을지에 대한 전략을 세워야 한다

포드(Pod)의 자원사용량 제한

컨테이너와 포드의 자원 사용량 제한 : Limit

- 컨테이너 오케스트레이션 툴은 여러 대의 서버를 묶어 리소스 풀로 사용가능
- 컨테이너의 자원 사용량 제한

--memory : 컨테이너의 메모리 제한, m(megabyte)/g(gigabyte) 지정 가능하며 최소 메모리는 4MB

```
# docker container run -d --memory="1g" nginx
```

```
# docker container run -d --memory=200m --memory-swap=500m nginx
```

--cpu-shares : 컨테이너에 CPU를 한 개씩 할당하는 방식이 아닌, 시스템에 존재하는 CPU를 어느 비중만큼 나눠 쓸 것인지를 명시. 1024는 CPU 할당에서 1의 비중을 의미

```
# docker container run -d --cpu-shares 1024 nginx
```

두 컨테이너가 각각 1024, 512 지정하면 66%, 33% 를 사용한다

포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Limit

- 컨테이너의 자원 사용량 제한

--cpuset-cpu : 컨테이너가 특정 CPU만 사용하도록 설정

```
# docker container run -d --cpuset-cpus=2 nginx
```

apt-get install htop(yum -y install epel-release htop) 후 htop 을 통해 확인

--cpu-period, --cpu-quota : 컨테이너는 [--cpu-quota 값] / [--cpu-period 값] 만큼 CPU 시간을 할당받는다. period 기본값은 1000000(100ms)

```
# docker container run -d --cpu-period=100000 \
--cpu-quota=25000
```

1/4 만큼 CPU를 적게 사용하게 된다

포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Limit

- 컨테이너의 자원 사용량 제한

--cpus : 직관적으로 CPU의 개수를 직접 지정
docker container run -d --cpus=0.5 nginx

CPU의 50% 를 점유하여 사용

- 자원 할당량을 설정하지 않으면 포드의 컨테이너가 노드의 물리 자원을 모두 사용할 수 있기 때문에 노드의 자원이 모두 고갈되는 상황이 발생할 수 있다. 이를 예방하기 위해서는 포드 자체에 자원 사용량을 명시적으로 설정하는 것이다

포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Limit

- CPU, 메모리 사용량 제한을 위한 YAML 파일

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-limit-pod
  labels:
    name: resource-limit-pod
spec:
  containers:
  - name : nginx
    image: nginx:latest
  resources:
    limits:
      memory: "256Mi"
      cpu: "1000m"          # --cpus 1 과 동일
```

```
# kubectl apply -f resources-limit-pod.yaml
# kubectl get pods -o wide
# kubectl describe node [node명]
```

포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Requests

- **"Requests"** : 적어도 이 만큼의 자원은 컨테이너에게 보장해 주어야 한다는 의미이며 **Requests** 는 쿠버네티스에서 자원의 오버커밋(**overcommit**)을 가능하게 만드는 기능을 제공한다.
- **"Overcommit"** : 한정된 컴퓨팅 자원을 효율적으로 사용하기 위한 방법으로, 사용할 수 있는 자원보다 더 많은 양을 가상머신이나 컨테이너에게 할당함으로써 전체 자원의 사용률 (**Utilization**)을 높이는 방법
- 두 컨테이너에 **500MB** 메모리를 할당했을 경우, **A**는 메모리 사용률이 낮고 **B**는 높다면 자원 사용의 불균형이 발생할 것이다. 이 경우 오버커밋을 통해 실제 물리 자원보다 더 많은 양의 자원을 할당하는 기능을 제공한다.

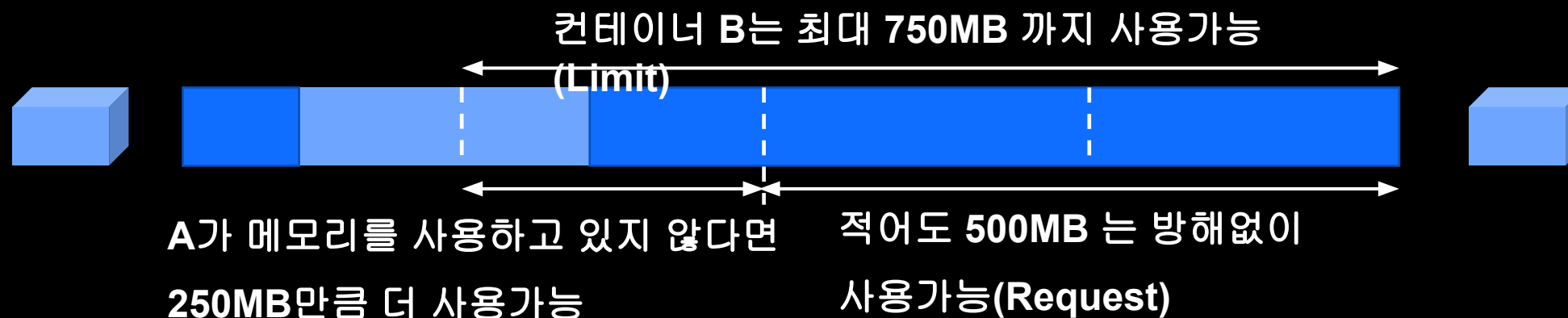
포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Requests

- 전체 자원보다 더 많은 양을 할당하는 오버커밋



- 컨테이너 A가 500MB를 사용하고 있을 때 컨테이너 B가 750MB 을 사용하려 한다면? 각 컨테이너가 '사용을 보장받을 수 있는 경계선' 을 정한다



포드(Pod)의 자원사용량 제한(Cont.)

컨테이너와 포드의 자원 사용량 제한 : Requests

- 최소한 128Mi의 메모리 사용은 보장되지만, 유휴 메모리 자원이 있다면 최대 256Mi까지 사용할 수 있다. (1개의 CPU=1000m, 2 CPU가 있다면 최대 2000m 만큼의 CPU Requests 를 포드에 할당가능)

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-limit-with-request-pod
  labels:
    name: resource-limit-with-request-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    resources:
      limits: "256Mi"
      cpu: "1000m"
      requests:
        memory: "128Mi"
        cpu: "500m"
```

최대 1개의 CPU 만큼을 사용할 수 있음

최소한 0.5 개의 CPU만큼의 사용을 보장받을 수 있음

인그레스(Ingress)

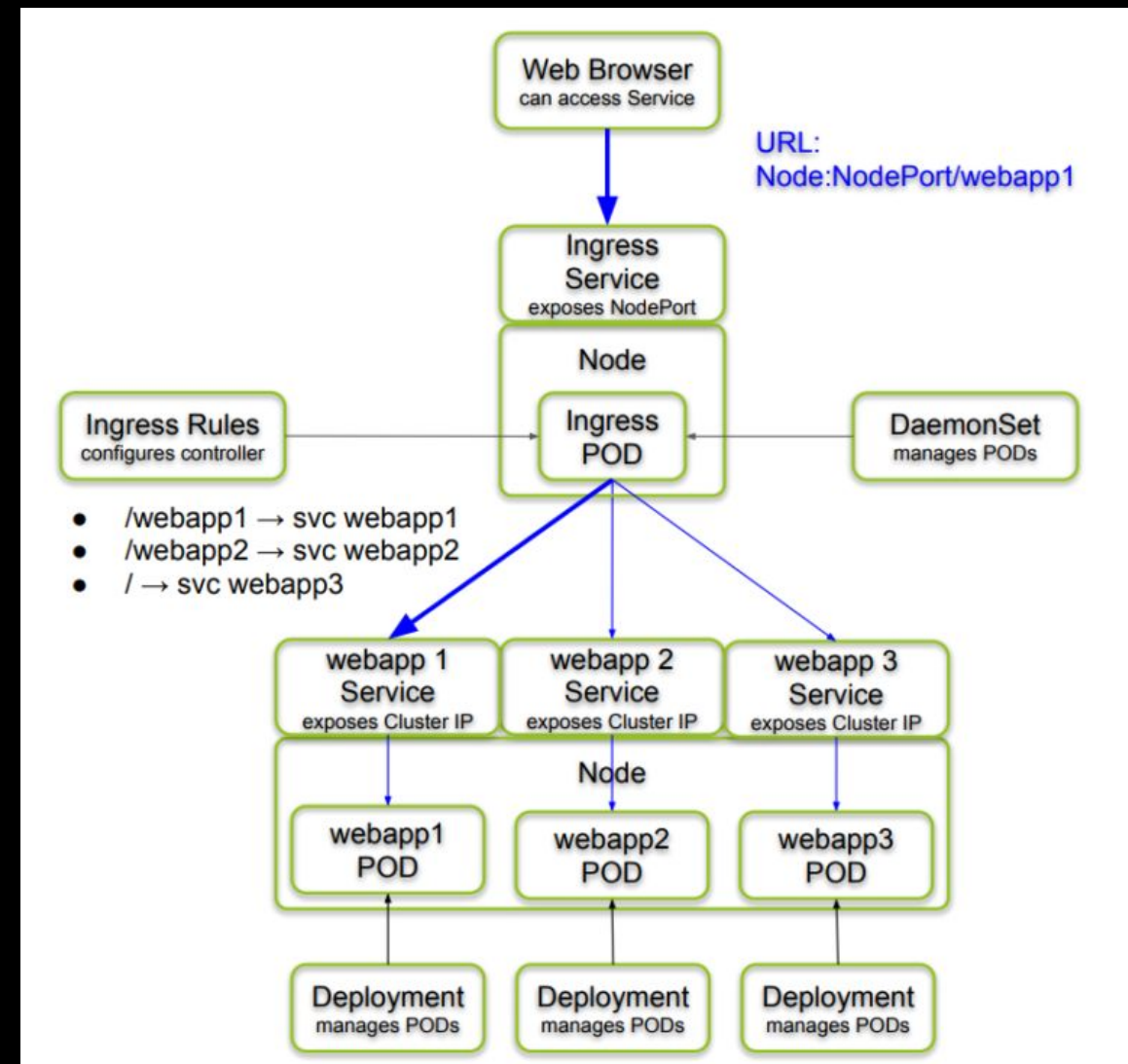
인그레스의 이해

- 서비스 오브젝트는 외부 요청을 받아들이기 위한 것
- "인그레스" 는 외부 요청을 어떻게 처리할 것인지를 정의하는 오브젝트
 - 외부 요청의 라우팅 : `/apple`, `/apple/red` 등과 같이 특정 경로로 들어온 요청을 어떠한 서비스로 전달할 것인지 정의하는 라우팅 규칙을 설정
 - 가상 호스트 기반의 요청 처리 : 같은 IP에 대해 다른 도메인 이름으로 요청이 도착했을 때, 어떻게 처리할 것인지 정의
 - SSL/TLS 보안 처리 : 여러 개의 서비스로 요청을 라우팅할 때, 보안 연결을 위한 인증서를 쉽게 적용

인그레스(Ingress)

인그레스의 사용이유

- 라우팅 정의나 보안 연결 등과 같은 세부 설정은 서비스와 디플로이먼트가 아닌 인그레스에 의해 수행
- 각 디플로이먼트에 대해 일일이 설정을 적용할 필요 없이 하나의 설정 지점에서 처리 규칙을 정의하기만 하면 됨. 즉, 외부 요청에 대한 처리 규칙을 쿠버네티스 자체의 기능으로 편리하게 관리할 수 있다



인그레스(Ingress)

인그레스의 사용이유

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

PV와 PVC의 이해

- MySQL 디플로이먼트로 포드를 생성하였다고 하더라도 포드 내부의 정보는 영구적이지 않다. 즉, 포드 삭제시 데이터도 삭제된다. 이를 해결하기 위해 포드의 데이터를 영속적으로 저장하기 위한 옵션이 필요한데 이를 **Persistent Volume(PV)** 라고 한다
 - 도커에서의 아래의 상황과 비슷

```
$ docker volume create myvolume
$ docker container run -it --name test -v myvolume:/mnt ubuntu
```
- 각 노드의 디렉토리를 각 포드와 공유하는 것이 아니라 스토리지를 마운트하여 데이터를 영속적으로 저장할 수 있는 볼륨을 퍼시스턴트 볼륨이라고 함. 포드가 다른 노드로 이동하더라도 데이터를 계속 사용할 수 있음
- 네트워크 퍼시스턴트 볼륨의 대표적인 예
 - NFS, aws 의 EBS(Elastic Block Store), Ceph, GlusterFS 등

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

NFS를 네트워크 볼륨으로 사용하기

■ NFS 서버 생성

211.183.3.99 를 사용하는 CentOS7 에서 아래의 내용을 실행

```
# yum -y install nfs-utils
```

```
# mkdir /k8s ; chmod 777 /k8s
```

```
# vi /etc/exports
```

```
/k8s    211.183.3.*(rw,sync)
```

```
# systemctl restart nfs-server ; systemctl stop firewalld
```

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

NFS를 네트워크 볼륨으로 사용하기

■ nfs-pod 생성하기

NFS 볼륨의 마운트는 컨테이너 자체에서의 접근이 아닌 워커 노드에서 발생하므로 각 워커 노드에서

먼저 "apt-get install -y nfs-common" 으로 nfs 클라이언트를 설치한 뒤 pod를 생성해 보자

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat nfs-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nfs-pod
spec:
  containers:
    - name: nfs-mount-container
      image: busybox
      args: [ "tail", "-f", "/dev/null" ]
      volumeMounts:
        - name: nfs-volume
          mountPath: /mnt
  volumes:
    - name: nfs-volume
      nfs:
        path: /k8s
        server: 211.183.3.99
root@master1:~/k8s# kubectl apply -f nfs-pod.yaml
```

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

NFS를 네트워크 볼륨으로 사용하기

- 마운트 확인
 - pod 에서 쉘로 파일 생성해 보기

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s#
root@master1:~/k8s# kubectl exec -it nfs-pod -- sh
/ # touch /mnt/test_from_k8s_worker.txt
/ #
```

- nfs 서버에서 생성된 파일 확인하기

```
[root@lbaas ~]# ls /
bin boot dev etc home k8s lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
[root@lbaas ~]# cat /etc/exports
/k8s 211.183.3.*(rw,sync)
[root@lbaas ~]#
[root@lbaas ~]# systemctl restart nfs-server
[root@lbaas ~]# systemctl stop firewalld
[root@lbaas ~]# ls /k8s
test_from_k8s_worker.txt
[root@lbaas ~]#
```

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

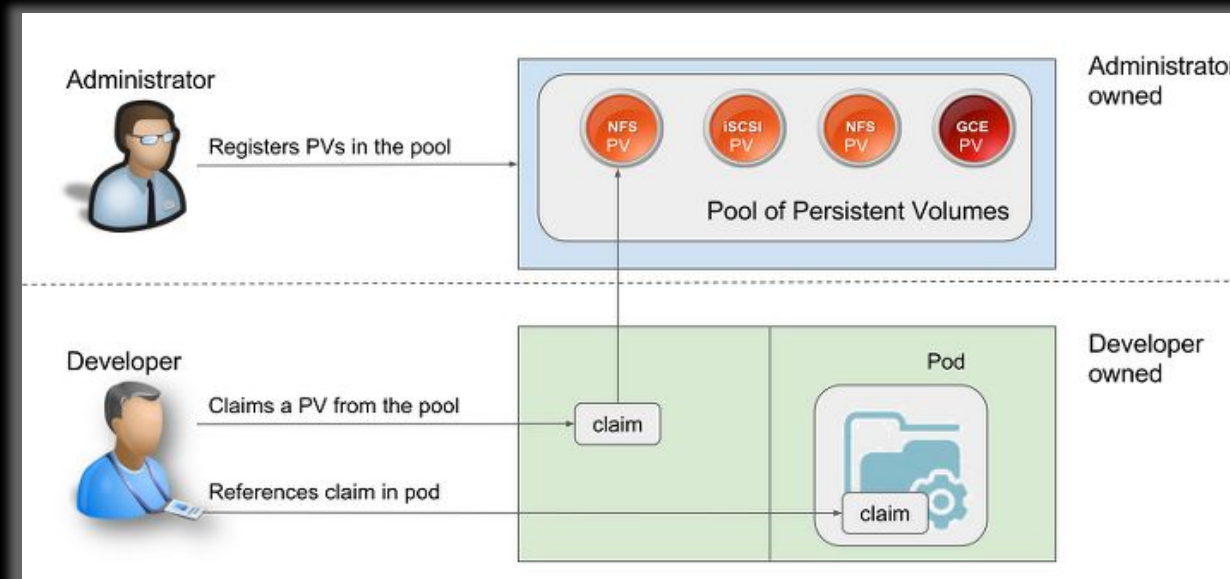
PV/PVC 사용이유

■ 일반적인 볼륨 사용 방법

- ❶ iscsi, nfs, ceph, aws ebs 별 yaml 파일을 만들어 미리 배포해 두어야 함
- ❷ 개발자가 마운트 해야할 스토리지(서버)의 주소, 디렉토리를 명확히 알고 있어야 함

■ PV, PVC 사용시 장점

- ❶ 인프라 관리자는 네트워크 볼륨의 정보를 이용하여 퍼시스턴트 볼륨 리소스를 미리 생성해 둔다
- ❷ 개발자는 YAML 파일에 '포드에 볼륨이 필요하다'를 정의하기만 하면 된다. 상세정보 필요하지 않음
- ❸ 쿠버네티스는 인프라 관리자가 생성해 둔 PV의 속성과 개발자가 요청한 PVC의 요구 사항이 일치한다면 둘을 바인딩한다



퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

NFS를 이용한 PV, PVC

- 인프라 담당자가 생성한 PV

- nfs 서버에 공유 디렉토리 생성

`mkdir /pvpvc ; chmod 777 /pvpvc` 이후 `/etc/exports` 에 허용 디렉토리 등록

- 개발자에게 제공할 1GB 크기의 PV 생성

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat nfs-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /pvpvc
    server: 211.183.3.99
    readOnly: false
root@master1:~/k8s# kubectl apply -f nfs-pv.yaml
persistentvolume/nfs-pv created
root@master1:~/k8s#
root@master1:~/k8s# kubectl get pv
NAME      CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM   STORAGECLASS  REASON   AGE
nfs-pv    1Gi       RWX           Retain          Available             Available    7s
root@master1:~/k8s#
```


퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

NFS를 이용한 PV, PVC

개발자 생성한 PVC, Pod

```

root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat nfs-pod-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-nfs-pvc
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: nfs-mount-container
spec:
  containers:
    - name: nfs-mount-container
      image: busybox
      args: [ "tail", "-f", "/dev/null" ]
      volumeMounts:
        - name: nfs-volume
          mountPath: /mnt
  volumes:
    - name: nfs-volume
      persistentVolumeClaim:
        claimName: my-nfs-pvc
root@master1:~/k8s# kubectl apply -f nfs-pod-pvc.yaml
persistentvolumeclaim/my-nfs-pvc created
pod/nfs-mount-container created
root@master1:~/k8s#

```

```

root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# kubectl get pv,pvc
NAME                                CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM                STORAGECLASS  REASON  AGE
persistentvolume/nfs-pv            1Gi       RWX           Retain          Bound   default/my-nfs-pvc                81s

NAME                                STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
persistentvolumeclaim/my-nfs-pvc    Bound   nfs-pv   1Gi       RWX           default/my-nfs-pvc   54s
root@master1:~/k8s#
root@master1:~/k8s# kubectl get pod | grep nfs-mount
nfs-mount-container                1/1      Running    0          59s
root@master1:~/k8s#

```

```

root@master1: ~/k8s
File Edit View Search Terminal Help
NAME                                STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
persistentvolumeclaim/my-nfs-pvc    Bound   nfs-pv   1Gi       RWX           default/my-nfs-pvc   54s
root@master1:~/k8s#
root@master1:~/k8s# kubectl get pod | grep nfs-mount
nfs-mount-container                1/1      Running    0          59s
root@master1:~/k8s#
root@master1:~/k8s#
root@master1:~/k8s#
root@master1:~/k8s# kubectl exec nfs-mount-container -- sh
root@master1:~/k8s#
root@master1:~/k8s# kubectl exec -it nfs-mount-container -- sh
/ # df -h | grep /mnt
211.183.3.99:/pvpcv                17.0G    1.3G    15.7G    8% /mnt
/ #

```

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

퍼시스턴트 볼륨을 선택하기 위한 조건 명시

- **accessModes** 와 스토리지 클래스, 라벨 셀렉터

- **accessModes**

accessModes 이름	kubectl get 에서 출력되는 이름	속성 설명
ReadWriteOnce	RWO	1:1 마운트만 가능, 읽기 쓰기 가능
ReadOnlyMany	ROX	1:N 마운트 가능, 읽기전용
ReadWriteMany	RWX	1:N 마운트 가능, 읽기 쓰기 가능

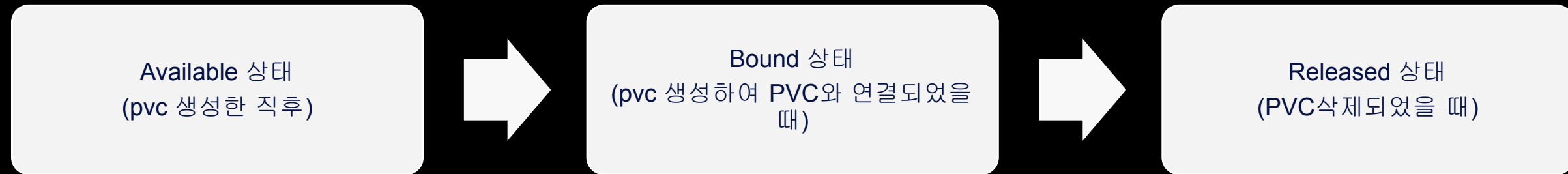
- 볼륨 생성시 스토리지 클래스를 설정하면 해당 클래스를 요청하는 PVC 과 연결하여 바인딩. "" 으로 설정되면 동일하게 스토리지 클래스가 설정되지 않은 PV 또는 PVC 와 연결된다.
 - **svc, deplyment** 를 서로 연결할 때 처럼 라벨 셀렉터를 사용했던 것처럼 PVC 에 라벨 셀렉터인 **matchLabels** 항목을 정의하여 특정 PV와 바인딩할 수 있다

퍼시스턴트 볼륨(PV)과 볼륨 클레임(PVC)

Reclaim Policy

- 퍼시스턴트 볼륨의 라이프 사이클

- 퍼시스턴트 볼륨의 라이프 사이클 설정이 없는 경우(Reclaim Policy 가 기본값인 경우)



RECLAIM POLICY 가 **Retain** 으로 설정된 경우 PV 은 연결된 PVC를 삭제한 뒤에 **Released** 상태로 전환되며, 스토리지에 저장된 데이터를 그대로 보존된다

- Delete** : PV 사용이 끝난 뒤에 자동으로 PV 가 삭제되며, 가능하다면 연결된 외부 스토리지도 함께 삭제된다
- Recycle** : **Delete** 와 같이 실제 저장된 데이터를 모두 삭제한다는 점은 같지만, **PV/외부 스토리지 자체를 삭제하지는 않는다**

실습예제



kubernetes

① LAN 환경에 Kubernetes 설치하기

- 설치환경 개요

- Network : vmnet8 (NAT, 211.183.3.0/24)
- Node :

Node 구분	CPU/RAM/DISK	IP주소	기타
master1	4/4/20	211.183.3.100	
worker1	2/2/20	211.183.3.101	
worker2	2/2/20	211.183.3.102	
worker3	2/2/20	211.183.3.103	

- 설치 이미지 : ubuntu 18.04 desktop
- master1 에 필요한 사항들을 모두 설치한 뒤에 이를 clone 하여 사용

① LAN 환경에 Kubernetes 설치하기

▪ master1 구성하기

```
apt-get update
apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
apt-cache madison docker-ce
apt-cache madison docker-ce-cli // 설치가능 버전 찾기
apt-get install docker-ce=5:18.09.9~3-0~ubuntu-bionic docker-ce-cli=5:18.09.9~3-0~ubuntu-bionic containerd.io
docker --version
apt-get update
```

① LAN 환경에 Kubernetes 설치하기

- master1 구성하기

```
127.0.0.1    localhost
127.0.1.1    master1

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters

211.183.3.100 master1
211.183.3.101 worker1
211.183.3.102 worker2
211.183.3.103 worker3
```

- /etc/hosts 파일에 master1, worker1~worker3 추가하기
- 두번째 줄의 127.0.1.1 master1 은 각 노드에서 해당 노드의 호스트명으로 수정해 주어야 함

① LAN 환경에 Kubernetes 설치하기

- master1 구성하기

```
root@master1:~# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -  
OK
```

```
root@master1:~# cat <<EOF > /etc/apt/sources.list.d/kubernetes.list  
> deb http://apt.kubernetes.io/ kubernetes-xenial main  
> EOF
```

```
root@master1:~# apt-get update
```

```
root@master1:~# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

- 위 까지 설정된 master1 을 clone 하여 worker1~worker3 을 생성한다

① LAN 환경에 Kubernetes 설치하기

- 쿠버네티스 초기화하기

```
root@master1:~# swapoff -a
```

```
root@master1:~# kubeadm init --apiserver-advertise-address 211.183.3.100 --pod-network-cidr=192.168.0.0/16
```

```
root@master1:~# mkdir -p $HOME/.kube
```

```
root@master1:~# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
root@master1:~# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
root@master1:~# export KUBECONFIG=/etc/kubernetes/admin.conf
```

```
root@master1:~#
```

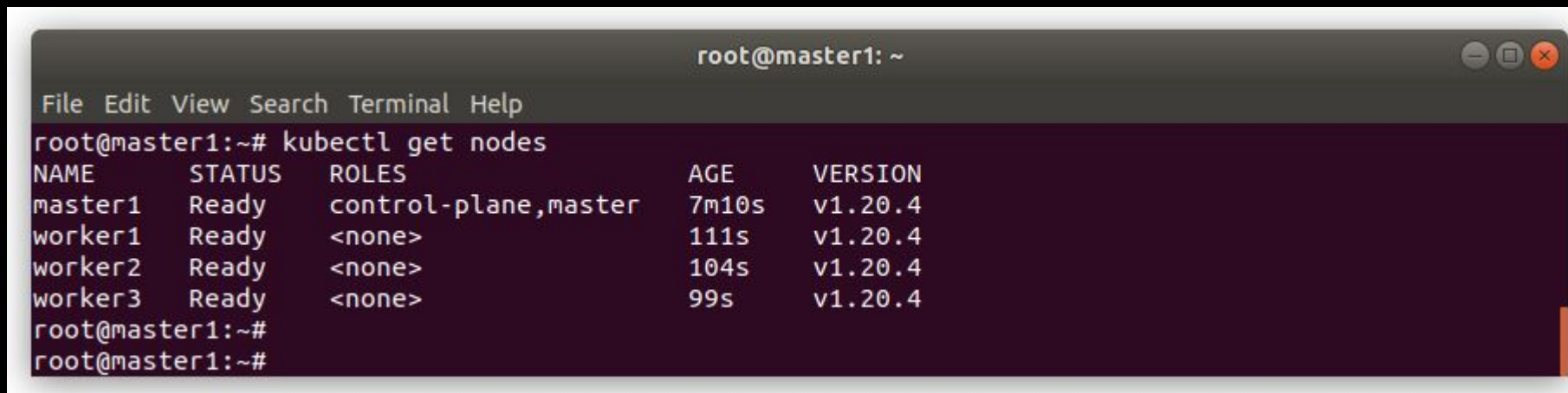
```
kubeadm join 211.183.3.100:6443 --token b4cu29.4nt1ct5xazuqqoz9 \
--discovery-token-ca-cert-hash sha256:56246eb54ec26ffd597bd9837fb8b65b4a5751cc5ab084be874948b
3374d32f8
```

- 위와 같이 발행된 토큰 정보를 복사하여 각 worker 노드에 붙여넣기를 한다

① LAN 환경에 Kubernetes 설치하기

- 쿠버네티스 초기화하기

```
root@master1:~# kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml
```



```
root@master1: ~
File Edit View Search Terminal Help
root@master1:~# kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
master1       Ready     control-plane,master  7m10s  v1.20.4
worker1       Ready     <none>          111s   v1.20.4
worker2       Ready     <none>          104s   v1.20.4
worker3       Ready     <none>          99s    v1.20.4
root@master1:~#
root@master1:~#
```

- 위와 같이 클러스터 환경에 포함된 각 노드를 확인할 수 있다

② LAN 환경에 설치된 k8s에서 서비스 배포하기

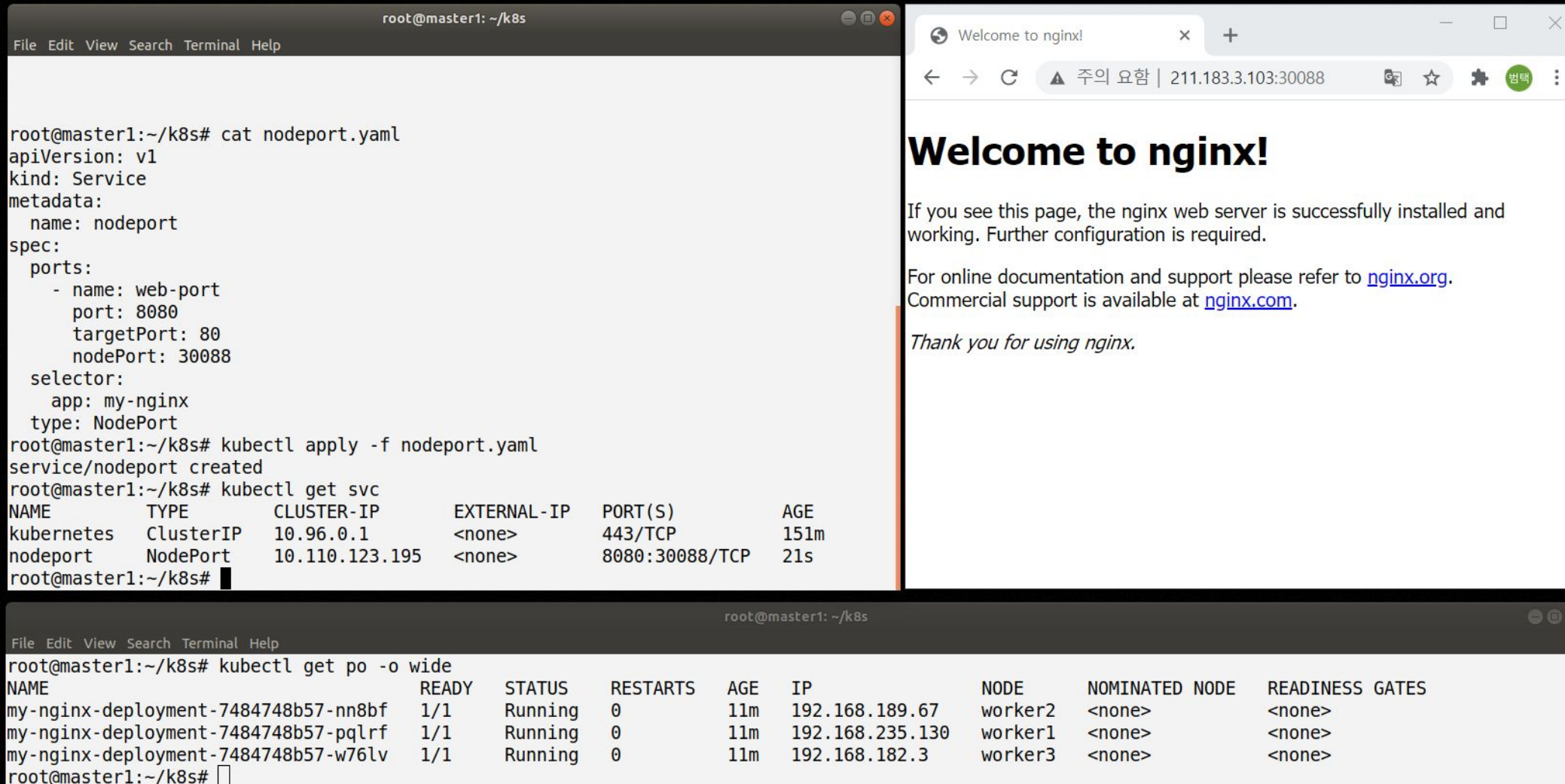
■ nginx 배포하기

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# cat deploy-nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-nginx
  template:
    metadata:
      name: my-nginx-pod
      labels:
        app: my-nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.10
        ports:
        - containerPort: 80
root@master1:~/k8s#
```

```
root@master1: ~/k8s
File Edit View Search Terminal Help
root@master1:~/k8s# kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx-deployment 3/3      3             3           61s
root@master1:~/k8s# kubectl get rs
NAME                DESIRED   CURRENT   READY   AGE
my-nginx-deployment-7484748b57 3         3         3       83s
root@master1:~/k8s# kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
my-nginx-deployment-7484748b57-nn8bf 1/1     Running   0          91s
my-nginx-deployment-7484748b57-pqlrf 1/1     Running   0          91s
my-nginx-deployment-7484748b57-w76lv 1/1     Running   0          91s
root@master1:~/k8s#
```

② LAN 환경에 설치된 k8s에서 서비스 배포하기

- Service(NodePort) 이용하여 외부연결



The screenshot shows a terminal window on a master node and a web browser displaying the nginx welcome page.

Terminal Output:

```
root@master1: ~/k8s
File Edit View Search Terminal Help

root@master1:~/k8s# cat nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  name: nodeport
spec:
  ports:
    - name: web-port
      port: 8080
      targetPort: 80
      nodePort: 30088
  selector:
    app: my-nginx
  type: NodePort
root@master1:~/k8s# kubectl apply -f nodeport.yaml
service/nodeport created
root@master1:~/k8s# kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1     <none>         443/TCP          151m
nodeport     NodePort    10.110.123.195 <none>         8080:30088/TCP   21s
root@master1:~/k8s#
```

Web Browser Output:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Terminal Output (kubectl get po -o wide):

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
my-nginx-deployment-7484748b57-nn8bf	1/1	Running	0	11m	192.168.189.67	worker2	<none>	<none>
my-nginx-deployment-7484748b57-pqlrf	1/1	Running	0	11m	192.168.235.130	worker1	<none>	<none>
my-nginx-deployment-7484748b57-w76lv	1/1	Running	0	11m	192.168.182.3	worker3	<none>	<none>

root@master1:~/k8s#

② LAN 환경에 설치된 k8s에서 서비스 배포하기

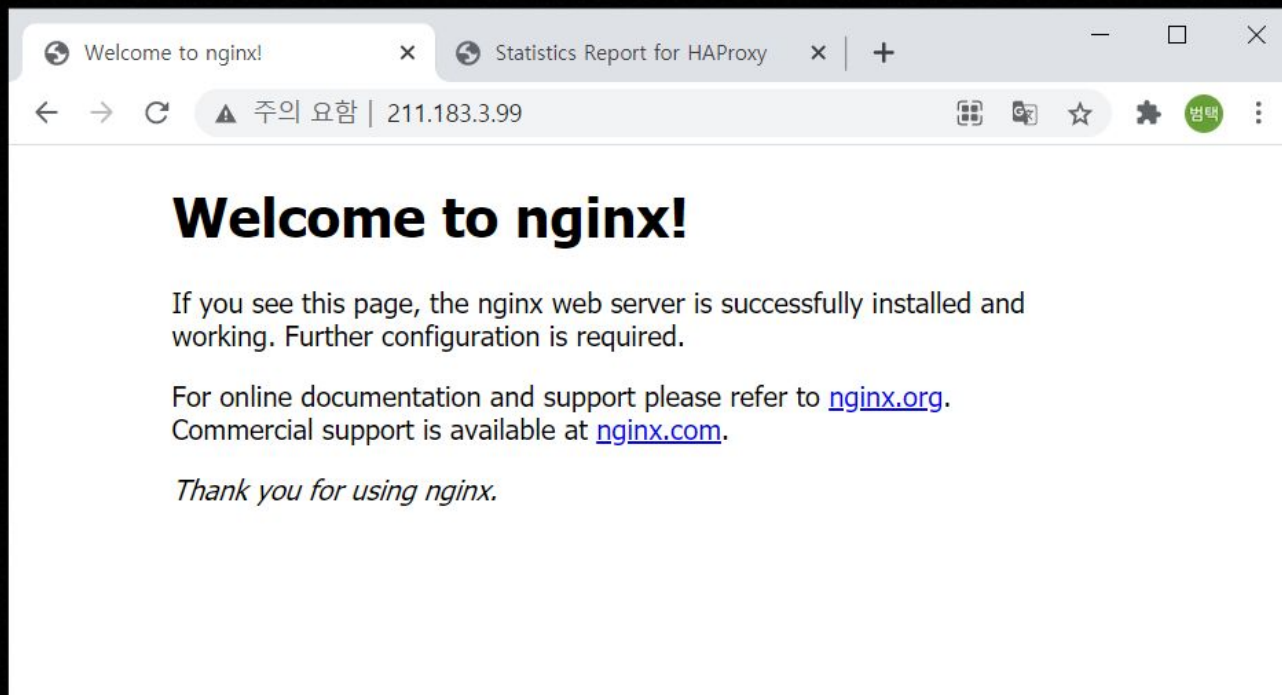
- HAProxy 와 연결을 통한 외부 노출

vi /etc/haproxy/haproxy.cfg 중 주요 부분

```
root@lbaas:~  
    timeout server          1m  
    timeout http-keep-alive 10s  
    timeout check           10s  
    maxconn                 3000  
  
frontend http-in  
    bind 211.183.3.99:80  
    stats enable  
    stats uri /stats  
    default_backend app  
    option forwardfor  
  
backend app  
    balance roundrobin  
    server app1 211.183.3.101:30088 check  
    server app2 211.183.3.102:30088 check  
    server app3 211.183.3.103:30088 check
```


② LAN 환경에 설치된 k8s에서 서비스 배포하기

- HAProxy 와 연결을 통한 외부 노출
haproxy 를 통합 웹 접속 확인



The screenshot shows a web browser window with two tabs: "Welcome to nginx!" and "Statistics Report for HAProxy". The address bar shows the URL "211.183.3.99/stats". The main content of the page is a statistics report for HAProxy, showing various metrics for the "http-in" and "app" sections.

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

http-in	
Queue	Session rate
Cur Max Limit	Cur Max Limit
Frontend	0 2 - 2 2 3 000 4

app	
Queue	Session rate
Cur Max Limit	Cur Max Limit
app1	0 0 - 0 2 0 1 - 3 3 2s 1 794 525 0 0 0 0 0 0 38s UP L4OK in 1ms 1
app2	0 0 - 0 1 0 1 - 2 2 3s 1 196 350 0 0 0 0 0 0 38s UP L4OK in 1ms 1
app3	0 0 - 0 1 0 1 - 2 2 3s 1 196 350 0 0 0 0 0 0 38s UP L4OK in 1ms 1
Backend	0 0 0 4 0 1 300 7 7 2s 4 186 1 225 0 0 0 0 0 0 38s UP 3

③ 이미지 생성과 저장

- cloudbuild 를 활용하여 작성된 이미지를 gcp 사설 저장소에 업로드하기

```
root@lab-k8s:~/btstore/kube# cat nginxtest/Dockerfile
FROM centos:7
RUN yum -y install httpd
EXPOSE 80
COPY index.html /var/www/html/index.html
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
root@lab-k8s:~/btstore/kube#
root@lab-k8s:~/btstore/kube# cat nginxtest/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>k8s test</title>
  </head>
  <body>
    <center><h2>HELLO FROM NGINX</h2></center>
  </body>
</html>
root@lab-k8s:~/btstore/kube#
```

③ 이미지 생성과 저장

- cloudbuild 를 활용하여 작성된 이미지를 gcp 사설 저장소에 업로드하기

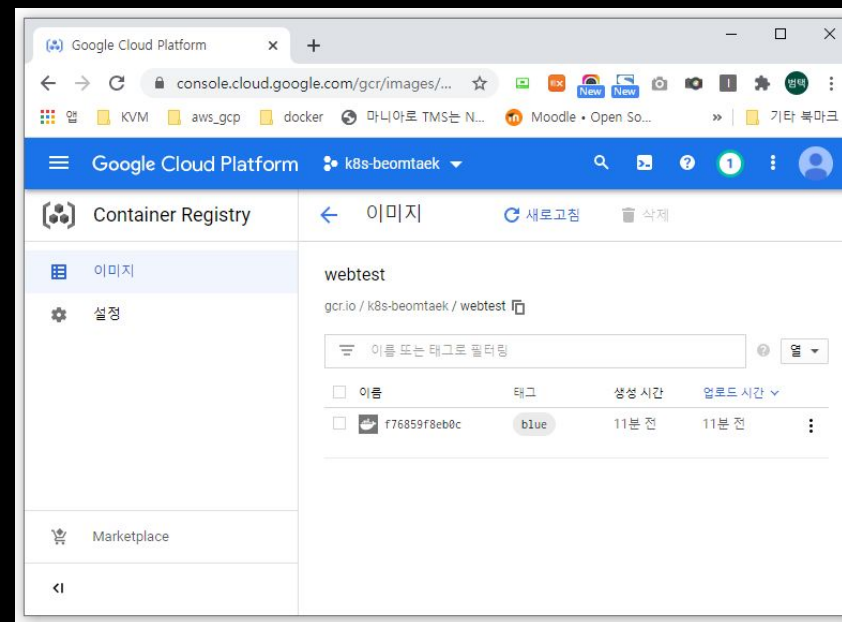
```
root@lab-k8s:~/btstore/kube# cat config/cloudbuild2.yaml
steps:
```

```
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/webtest:blue', './nginxtest']
```

```
images: ['gcr.io/$PROJECT_ID/webtest:blue']
```

```
root@lab-k8s:~/btstore/kube# PROJECT_ID=$(gcloud config list project --format "value(core.project)")
```

```
root@lab-k8s:~/btstore/kube# gcloud builds submit --config config/cloudbuild2.yaml
```



③ 생성된 이미지를 활용하여 서비스배포(Deployment)

- gcp 사설저장소(Container Registry)의 이미지를 활용한 디플로이 생성(매니페스트파일)

```
root@lab-k8s:~/btstore/kube# cat config/deployment-webtest-blue.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: my-webtest-deployment
```

```
spec:
```

```
  replicas: 3
```

```
  selector:
```

```
    matchLabels:
```

```
      app: my-webtest
```

```
template:
```

```
  metadata:
```

```
    name: my-webtest-pod
```

```
    labels:
```

```
      app: my-webtest
```

```
  spec:
```

```
    containers:
```

```
      - name: webtest
```

```
        image: gcr.io/k8s-beomtaek/webtest:blue
```

```
        ports:
```

```
          - containerPort: 80
```

```
root@lab-k8s:~/btstore/kube#
```

③ 생성된 이미지를 활용하여 서비스배포(Deployment)

- gcp 사설저장소(Container Registry) 의 이미지를 활용한 디플로이 생성(서비스상태확인)

```
root@lab-k8s:~/btstore/kube# kubectl apply -f config/deployment-webtest-blue.yaml
```

```
deployment.apps/my-webtest-deployment created
```

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube# kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
my-webtest-deployment	3/3	3	3	19s

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube# kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
my-webtest-deployment-68546bcb5	3	3	3	32s

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-webtest-deployment-68546bcb5-4t9qb	1/1	Running	0	39s
my-webtest-deployment-68546bcb5-g7d29	1/1	Running	0	39s
my-webtest-deployment-68546bcb5-s44bj	1/1	Running	0	39s

```
root@lab-k8s:~/btstore/kube#
```

③ 생성된 이미지를 활용하여 서비스배포(Deployment)

- gcp 사설저장소(Container Registry) 의 이미지를 활용한 디플로이 생성(동작확인 및 삭제)

```
root@lab-k8s:~/btstore/kube# kubectl get pods
NAME                                READY  STATUS   RESTARTS  AGE
my-webtest-deployment-68546bcb5-mxt9v 1/1    Running  0         3s
my-webtest-deployment-68546bcb5-s8zn6 1/1    Running  0         3s
my-webtest-deployment-68546bcb5-w4h65 1/1    Running  0         3s
root@lab-k8s:~/btstore/kube#
root@lab-k8s:~/btstore/kube# kubectl delete pods my-webtest-deployment-68546bcb5-mxt9v
pod "my-webtest-deployment-68546bcb5-mxt9v" deleted
```

```
root@lab-k8s:~/btstore/kube#
root@lab-k8s:~/btstore/kube# kubectl get pods
NAME                                READY  STATUS   RESTARTS  AGE
my-webtest-deployment-68546bcb5-97t4d 1/1    Running  0         15s
my-webtest-deployment-68546bcb5-s8zn6 1/1    Running  0         56s
my-webtest-deployment-68546bcb5-w4h65 1/1    Running  0         56s
root@lab-k8s:~/btstore/kube#
root@lab-k8s:~/btstore/kube# kubectl delete -f config/deployment-webtest-blue.yaml
deployment.apps "my-webtest-deployment" deleted
root@lab-k8s:~/btstore/kube#
```

③ 생성된 이미지를 활용하여 서비스배포(Service)

- LB 를 이용한 서비스 배포

```
root@lab-k8s:~/btstore/kube# cat config/service2.yaml
```

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  name: webtest-svc-lb
```

```
spec:
```

```
  ports:
```

```
    - name: web-port
```

```
      port: 80
```

```
      targetPort: 80
```

```
  selector:
```

```
    app: my-webtest
```

```
  type: LoadBalancer
```

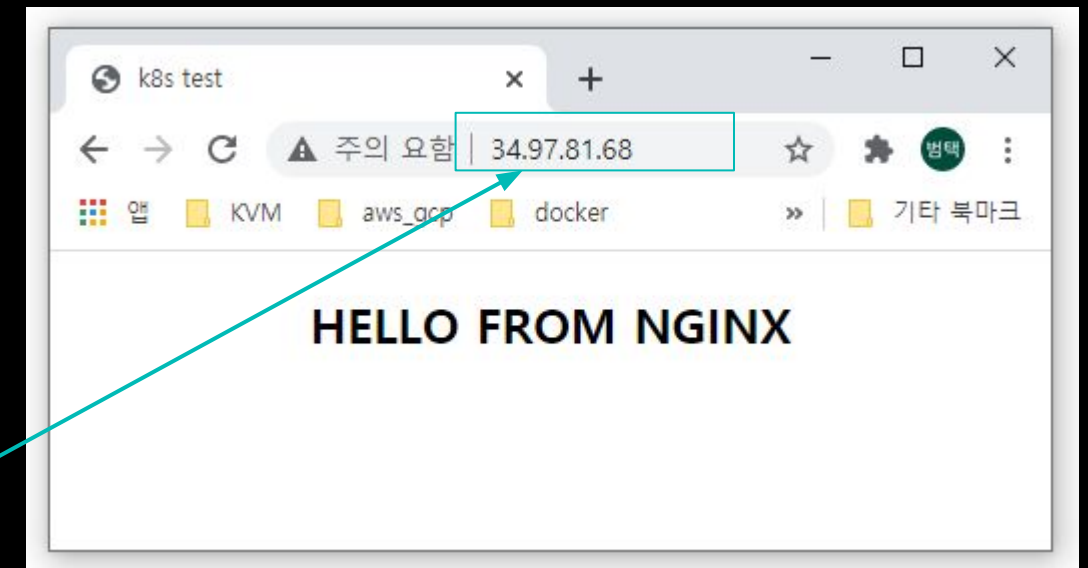
```
root@lab-k8s:~/btstore/kube# kubectl apply -f config/service2.yaml
```

```
service/webtest-svc-lb created
```

```
root@lab-k8s:~/btstore/kube# kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	18h
webtest-svc-lb	LoadBalancer	10.96.15.120	34.97.81.68	80:31262/TCP	4m18s

```
root@lab-k8s:~/btstore/kube#
```



③ 생성된 이미지를 활용하여 서비스배포(Service)

- LB 를 이용한 서비스 상태확인

webtest-svc-lb - 서비스 세부정보

console.cloud.google.com/kubernetes/service/asia-northeast2-a/myk8s/default...

Google Cloud Platform k8s-beomtaek

Kubernetes Engine

서비스 세부정보

my-webtest-deployment OK 3/3

제공 pod

이름	상태	엔드포인트	재시작	생성일
my-webtest-deployment-68546bcb5-mdl24	Running	10.92.0.6	0	2021. 2. 17. PM 12:03:35
my-webtest-deployment-68546bcb5-lskfm	Running	10.92.1.7	0	2021. 2. 17. PM 12:03:35
my-webtest-deployment-68546bcb5-7dhf6	Running	10.92.2.9	0	2021. 2. 17. PM 12:03:35

포트

이름	포트	노드 포트	대상 포트	프로토콜
web-port	80	31262	80	TCP

포트 전달

webtest-svc-lb - 서비스 세부정보

console.cloud.google.com/kubernetes/service/asia-northeast2-a/myk8s/default...

Google Cloud Platform k8s-beomtaek

Kubernetes Engine

서비스 세부정보

myk8s

클러스터

네임스페이스

default

라벨

설정된 라벨이 없습니다.

로그

my-webtest-deployment

유형

LoadBalancer

외부 엔드포인트

34.97.81.68:80

부하 분산기

클러스터 IP

10.96.15.120

부하 분산기 IP

34.97.81.68

부하 분산기

ad16b8bc24f11460181acaebb1f0eb1f

③ 서비스 롤링 업데이트

- LB 를 이용한 서비스 상태확인

```
root@lab-k8s:~/btstore/kube# cat config/cloudbuild3.yaml
steps:
```

```
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/webtest:green', './httpdtest']
```

```
images: ['gcr.io/$PROJECT_ID/webtest:green']
```

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube# cat httpdtest/index.html
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>k8s test</title>
```

```
  </head>
```

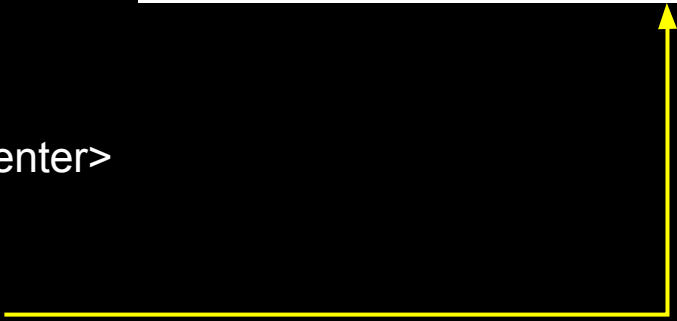
```
  <body>
```



```
    <center><h2><font color="green">HELLO FROM HTTPD TEST</font></h2></center>
```

```
  </body>
```

```
</html>
```

```
root@lab-k8s:~/btstore/kube# gcloud builds submit --config config/cloudbuild3.yaml
```



webtest				
gcr.io / k8s-beomtaek / webtest				
이름 또는 태그로 필터링				
<input type="checkbox"/>	이름	태그	생성 시간	업로드 시간
<input type="checkbox"/>	 bd2b9504696e	green	방금 전	방금 전
<input type="checkbox"/>	 f76859f8eb0c	blue	1시간 전	1시간 전

③ 서비스 롤링 업데이트

- LB 를 이용한 서비스 상태확인

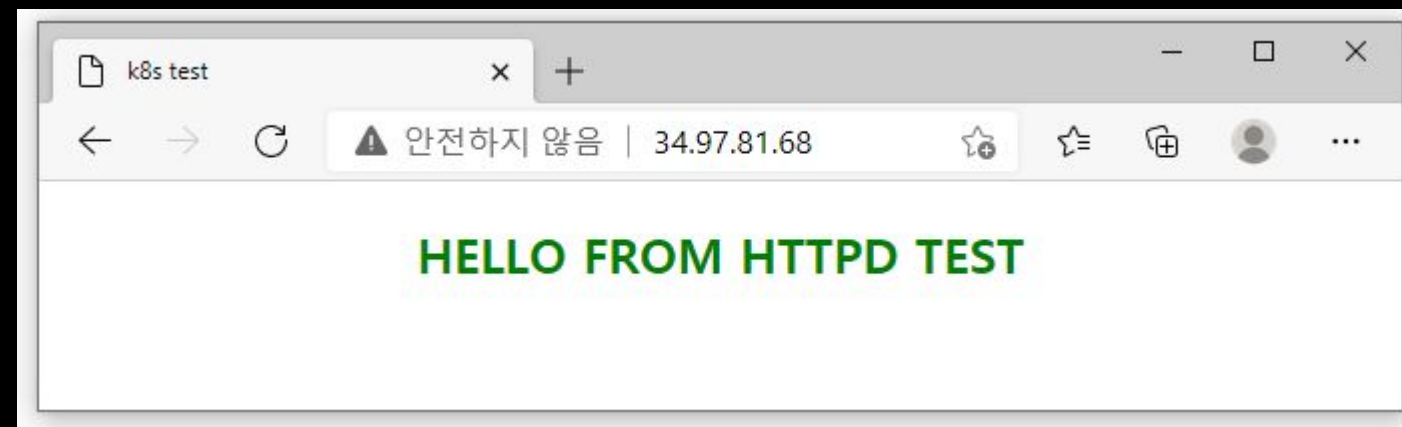
```
root@lab-k8s:~/btstore/kube# kubectl get deployment
NAME                READY  UP-TO-DATE  AVAILABLE  AGE
my-webtest-deployment 3/3    3           3          23m
```

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube#
```

```
root@lab-k8s:~/btstore/kube# kubectl set image deployment my-webtest-deployment webtest=gcr.io/k8s-beomtaek/webtest:green --record
deployment.apps/my-webtest-deployment image updated
```

```
root@lab-k8s:~/btstore/kube#
```



```
root@lab-k8s:~/btstore/kube# kubectl delete deployment,pod,rs,svc --all
```