

# 클라우드 기반 정보시스템 구축 전문가 양성 국비 과정

3조 - Shell Work) 쿠버네티스 프로젝트

김진호  
김현욱  
임원택  
정혁준



## 문제1) ETCD 백업

https://127.0.0.1:2379에서 실행 중인 etcd의 snapshot을 생성하고 snapshot을 etcd-snapshot.db에 저장합니다.

그런 다음 다시 스냅샷을 복원합니다.

etcdctl을 사용하여 서버에 연결하기 위해 다음 TLS 인증서/키가 제공됩니다.

CA certificate: /etc/kubernetes/pki/etcd/ca.crt

Client certificate: /etc/kubernetes/pki/etcd/server.crt

Client key: /etc/kubernetes/pki/etcd/server.key

### 1. [kubernetes.io/docs/home](https://kubernetes.io/docs/home) 사이트에서 ETCD Backup 검색 후 필요한 명령어 찾기

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> \  
snapshot save <backup-file-location>
```

### 2. root 계정으로 전환 후 etcd-client 설치

```
ubuntu@master:~$ sudo -i  
root@master:~# apt install etcd-client
```

### 3. /data 디렉터리 생성

```
root@master:~# mkdir /data
```

### 4. 찾은 명령어 내용 중 <> 안에 있는 내용들을 문제에 맞게 수정 후, 스냅샷 생성 및 저장

```
root@master:~# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt  
--key=/etc/kubernetes/pki/etcd/server.key \  
snapshot save /data/etcd-snapshot.db
```

### 5. 스냅샷 복원

```
root@master:~# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt  
--key=/etc/kubernetes/pki/etcd/server.key \  
snapshot restore /data/etcd-snapshot.db
```

## 6. 스냅샷 복원 여부 확인

```
root@master:~# tree default.etcd/member/
```

```
default.etcd/member/
```

```
├── snap
│   ├── 000000000000000001-0000000000000001.snap
│   └── db
└── wal
    └── 0000000000000000-0000000000000000.wal
```

```
3 directories, 3 files
```

## 문제2) Cluster Upgrade

마스터 노드의 모든 Kubernetes control plane 및 node 구성 요소를 버전 1.28.8-1.1 버전으로 업그레이드합니다.

master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon해야 합니다.

"주의사항" 반드시 Master Node에서 root 권한을 가지고 작업을 실행해야 한다.

### 1. root 계정으로 전환 후 업그레이드 할 버전 선택

```
ubuntu@master:~$ sudo -i
root@master:~# apt update
root@master:~# apt-cache madison kubeadm
```

### 2. kubeadm upgrade 호출

```
root@master:~# apt-mark unhold kubeadm
Canceled hold on kubeadm.
root@master:~# sudo apt-get update && sudo apt-get install -y kubeadm='1.30.14-*'
root@master:~# apt-mark hold kubeadm
kubeadm set on hold.
```

### 3. 업그레이드 할 버전 선택 후, 적용

```
root@master:~# kubeadm upgrade apply v1.30.14-1.1
```

### 4. root 계정에서 exit 후 master 노드 drain

```
ubuntu@master:~$ kubectl drain master --ignore-daemonsets
node/master cordoned
```

### 5. 다시 root 계정 전환 후 kubelet과 kubectl 업그레이드

```
root@master:~# apt-mark unhold kubelet kubectl
Canceled hold on kubelet.

root@master:~# apt-get update && apt-get install -y kubelet=1.30.14-1.1 kubectl=1.30.14-1.1

root@master:~# apt-mark hold kubelet kubectl
kubelet set on hold.
```

## 6. kubelet 다시 시작

```
root@master:~# systemctl daemon-reload  
root@master:~# systemctl restart kubelet
```

## 7. root 계정에서 exit 후 master 노드 uncordon

```
ubuntu@master:~$ kubectl uncordon master  
node/master uncordoned
```

## 문제3) Service Account, Role, RoleBinding 생성

애플리케이션 운영중 특정 namespace의 Pod들을 모니터할수 있는 서비스가 요청되었습니다. api-access 네임스페이스의 모든 pod를 view할 수 있도록 다음의 작업을 진행하시오.

1. api-access라는 새로운 namespace에 pod-viewer라는 이름의 Service Account를 만듭니다.
2. podreader-role이라는 이름의 Role과 podreader-rolebinding이라는 이름의 RoleBinding을 만듭니다.
3. 앞서 생성한 ServiceAccount를 API resource Pod에 대하여 watch, list, get을 허용하도록 매핑하시오.

### 1. api-access라는 이름의 Namespace와 pod-viewer라는 이름의 Service Account 생성

```
ubuntu@master:~$ kubectl create ns api-access
namespace/api-access created

ubuntu@master:~$ kubectl create serviceaccount pod-viewer -n api-access
serviceaccount/pod-viewer created
```

### 2. 생성된 Namespace와 Service Account 확인

```
ubuntu@master:~$ kubectl get ns
```

NAME	STATUS	AGE
api-access	Active	97s
default	Active	6d5h
kube-node-lease	Active	6d5h
kube-public	Active	6d5h
kube-system	Active	6d5h

```
ubuntu@master:~$ kubectl get sa -n api-access
```

NAME	SECRETS	AGE
default	0	2m28s
pod-viewer	0	2m6

### 3. watch, list, get 을 허용하도록 하는 podreader-role 이라는 이름의 Role 생성

```
ubuntu@master:~$ kubectl create role podreader-role -n api-access --resource=pod --verb=watch,list,get
role.rbac.authorization.k8s.io/podreader-role created
```

#### 4. Role 생성 확인

```
ubuntu@master:~$ kubectl describe role -n api-access
```

Name: podreader-role

Labels: <none>

Annotations: <none>

PolicyRule:

Resources	Non-Resource URLs	Resource Names	Verbs
-----	-----	-----	-----
Pods	[]	[]	[watch list get]

#### 5. podreader-rolebinding 라는 이름의 RoleBinding 생성

```
ubuntu@master:~$ kubectl create rolebinding podreader-rolebinding
```

```
--serviceaccount=api-account:pod-viewer --role=podreader-role -n api-access
```

```
rolebinding.rbac.authorization.k8s.io/podreader-rolebinding created
```

#### 6. RoleBinding 생성 확인

```
ubuntu@master:~$ kubectl describe rolebinding -n api-access
```

Name: podreader-rolebinding

Labels: <none>

Annotations: <none>

Role:

Kind: Role

Name: podreader-role

Subjects:

Kind	Name	Namespace
-----	-----	-----
ServiceAccount	pod-viewer	api-account

## 문제4) Service Account, ClusterRole, ClusterRoleBinding 생성

애플리케이션 배포를 위해 새로운 ClusterRole을 생성하고 특정 namespace의 ServiceAccount를 바인드하시오.

다음의 resource type에서만 Create가 허용된 ClusterRole deployment-clusterrole을 생성합니다.

Resource Type: Deployment StatefulSet DaemonSet

미리 생성된 namespace api-access 에 cicc-token이라는 새로운 ServiceAccount를 만듭니다.

ClusterRole deployment-clusterrole을 namespace api-access 로 제한된 새 ServiceAccount cicc-token에 바인딩하세요.

### 1. cicc-token이라는 새로운 Service Account 생성

```
ubuntu@master:~$ kubectl create sa cicc-token -n api-access
serviceaccount/cicc-token created
```

### 2. 생성한 Service Account 확인

```
ubuntu@master:~$ kubectl describe sa -n api-access cicc-token
Name:          cicc-token
Namespace:     api-access
```

### 3. resource type에서만 Create가 허용된 deployment-clusterrole 라는 이름의 ClusterRole 생성

```
ubuntu@master:~$ kubectl create clusterrole deployment-clusterrole
--resource=deployment,statefulset,daemonset --verb=create
clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
```

### 4. 생성된 ClusterRole 확인

```
ubuntu@master:~$ kubectl describe clusterrole deployment-clusterrole
Name:      deployment-clusterrole
Labels:    <none>
Annotations: <none>
PolicyRule:
```

Resources	Non-Resource URLs	Resource Names	Verbs
-----	-----	-----	----
daemonsets.apps	[ ]	[ ]	[create]
deployments.apps	[ ]	[ ]	[create]
statefulsets.apps	[ ]	[ ]	[create]



## 5. 생성한 ClusterRole을 cicc-token Service Account에 바인딩 하도록 ClusterRoleBinding 생성

```
ubuntu@master:~$ kubectl create clusterrolebinding deployment-clusterrolebinding
--serviceaccount=api-access:cicc-token --clusterrole=deployment-clusterrole -n api-access
clusterrolebinding.rbac.authorization.k8s.io/deployment-clusterrolebinding created
```

## 6. 생성된 ClusterRoleBinding 확인

```
ubuntu@master:~$ kubectl describe -n api-access clusterrolebinding deployment-clusterrolebinding
```

Name: deployment-clusterrolebinding

Labels: <none>

Annotations: <none>

Role:

Kind: ClusterRole

Name: deployment-clusterrole

Subjects:

Kind	Name	Namespace
----	----	-----
ServiceAccount	cicc-token	api-access

## 문제5) 노드 비우기

k8s-worker2 노드를 스케줄링 불가능하게 설정하고, 해당 노드에서 실행 중인 모든 Pod을 다른 node로 reschedule 하세요.

### 1. 설정하기 전 worker2 노드 확인

```
ubuntu@master:~$ kubectl get no
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane	6d5h	v1.30.14
worker1	Ready	<none>	6d5h	v1.30.14
worker2	Ready	<none>	6d5h	v1.30.14

### 2. worker2 노드 drain

```
ubuntu@master:~$ kubectl drain worker2
node/worker2 cordoned
```

### 3. worker2 노드 확인

```
ubuntu@master:~$ kubectl get no
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane	6d5h	v1.30.14
worker1	Ready	<none>	6d5h	v1.30.14
worker2	Ready,SchedulingDisabled	<none>	6d5h	v1.30.14

### 4. worker2 uncordon 후 노드 확인

```
ubuntu@master:~$ kubectl uncordon worker2
node/worker2 uncordoned
```

```
ubuntu@master:~$ kubectl get no
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane	6d5h	v1.30.14
worker1	Ready	<none>	6d5h	v1.30.14
worker2	Ready	<none>	6d5h	v1.30.14

## 문제6) Pod Scheduling

다음의 조건으로 pod를 생성하세요.

Name: eshop-store

Image: nginx

NodeSelector: disktype=ssd

### 1. worker1, 2 노드에 라벨링 추가

```
ubuntu@master:~$kubectl label nodes worker1 disktype=ssd
node/worker1 labeled
```

```
ubuntu@master:~$kubectl label nodes worker2 disktype=hdd
node/worker2 labeled
```

### 2. 노드 라벨 확인

```
ubuntu@master:~$kubectl get nodes -L disktype
```

NAME	STATUS	ROLES	AGE	VERSION	DISKTYPE
master	Ready	control-plane	6d3h	v1.30.14	
worker1	Ready	<none>	6d3h	v1.30.14	ssd
worker2	Ready	<none>	6d3h	v1.30.14	hdd

### 3. eshop-store.yaml 파일 생성

```
ubuntu@master:~$kubectl run eshop-store --image=nginx --dry-run=client -o yaml > eshop-store.yaml
```

### 4. eshop-store.yaml 파일 수정

```
ubuntu@master:~$ vi eshop-store.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: eshop-store
    name: eshop-store
spec:
  containers:
  - image: nginx
    name: eshop-store
  nodeSelector:
    disktype: ssd
```

## 5. eshop-store.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f eshop-store.yaml  
pod/eshop-store created
```

## 6. pod 생성 및 적용 확인

```
ubuntu@master:~$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
eshop-store	1/1	Running	0	41s	172.16.235.149	worker1	<none>	<none>

## 문제7) 환경변수, command, args 적용

'cka-exam'이라는 namespace를 만들고, 'cka-exam' namespace에 아래와 같은 Pod를 생성하시오.

pod Name: pod-01

image: busybox

환경변수 : CERT = "CKA-cert"

command: /bin/sh

args: "-c", "while true; do echo \$(CERT); sleep 10;done"

### 1. 네임스페이스 생성

```
ubuntu@master:~$ kubectl create namespace cka-exam
namespace/cka-exam created
```

### 2. 네임스페이스 생성 확인

```
ubuntu@master:~$ kubectl get namespaces | grep cka-exam
cka-exam      Active      56s
```

### 3. pod-01.yaml 파일 생성

```
ubuntu@master:~$ kubectl run pod-01 --image=busybox -n cka-exam --env=CERT="CKA-cert"
--dry-run=client -o yaml > pod-01.yaml
```

### 4. pod-01.yaml 파일 수정

```
ubuntu@master:~$ vi pod-01.yaml

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod-01
  name: pod-01
  namespace: cka-exam
spec:
  containers:
  - env:
    - name: CERT
      value: CKA-cert
    image: busybox
    name: pod-01
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(CERT); sleep 10;done"]
```

## 5. pod-01.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f pod-01.yaml
pod/pod-01 created
```

## 6. pod 생성 확인

```
ubuntu@master:~$ kubectl get po pod-01 -n cka-exam -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
pod-01	1/1	Running	0	10s	172.16.189.89	worker2	<none>	<none>

## 문제8) Static Pod 생성

worker1 노드에 nginx-static-pod.yaml 라는 이름의 Static Pod를 생성하세요.

pod name: nginx-static-pod

image: nginx

port : 80

### 1. worker1에 접속

```
ubuntu@master:~$ ssh worker1
```

```
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-71-generic x86_64).
```

```
...
```

```
ubuntu@worker1:~$
```

### 2. worker1의 static pod 위치 확인 및 이동

```
ubuntu@worker1:~$ sudo -i
```

```
root@worker1:~# cd /var/lib/kubelet/
```

```
root@worker1:/var/lib/kubelet# ls
```

```
checkpoints  cpu_manager_state  kubeadm-flags.env  pki  plugins_registry  pods  
config.yaml  device-plugins     memory_manager_state  plugins  pod-resources
```

```
root@worker1:/var/lib/kubelet# cat config.yaml | grep -i static
```

```
staticPodPath: /etc/kubernetes/manifests
```

```
root@worker1:/var/lib/kubelet# cd /etc/kubernetes/manifests/
```

```
root@worker1:/etc/kubernetes/manifests#
```

### 3. nginx-static-pod.yaml 라는 이름의 Static Pod 생성

```
root@worker1:/etc/kubernetes/manifests# vi nginx-static-pod.yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
  name: nginx-static-pod
```

```
spec:
```

```
  containers:
```

```
    - name: nginx-static-pod
```

```
      image: nginx
```

```
      ports:
```

```
        - containerPort: 80
```

#### 4. static pod 생성 확인

```
ubuntu@master:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-static-pod-worker1	1/1	Running	0	20s



## 문제9) 로그 확인

Pod "nginx-static-pod-worker1"의 log를 모니터링하고, 메시지를 포함하는 로그라인을 추출하세요.  
추출된 결과는 /home/ubuntu/pod-log에 기록하세요.

### 1. nginx-static-pod-worker1의 동작 확인

```
ubuntu@master:~$ kubectl get pod nginx-static-pod-worker1
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-static-pod-worker1	1/1	Running	0	6m16s

### 2. 로그 확인

```
ubuntu@master:~$ kubectl logs nginx-static-pod-worker1
```

```
/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
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/08/12 07:16:20 [notice] 1#1: using the "epoll" event method
2025/08/12 07:16:20 [notice] 1#1: nginx/1.29.0
2025/08/12 07:16:20 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14+deb12u1)
2025/08/12 07:16:20 [notice] 1#1: OS: Linux 6.8.0-71-generic
2025/08/12 07:16:20 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/08/12 07:16:20 [notice] 1#1: start worker processes
2025/08/12 07:16:20 [notice] 1#1: start worker process 29
2025/08/12 07:16:20 [notice] 1#1: start worker process 30
2025/08/12 07:16:20 [notice] 1#1: start worker process 31
2025/08/12 07:16:20 [notice] 1#1: start worker process 32
```

### 3. 로그 저장

```
ubuntu@master:~$ kubectl logs nginx-static-pod-worker1 > /home/ubuntu/pod-log
```

## 문제10) Multi Container Pod 생성

4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성하시오.  
pod image: nginx, redis, memcached, consul

### 1. eshop-frontend-pod.yaml 파일 생성

```
ubuntu@master:~$ kubectl run eshop-frontend --image=nginx --dry-run=client -o yaml >eshop-frontend.yaml
```

### 2. eshop-frontend.yaml 파일 수정

```
ubuntu@master:~$ vi eshop-frontend.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: eshop-frontend
    name: eshop-frontend
spec:
  containers:
    - image: nginx
      name: nginx
    - image: redis
      name: redis
    - image: memcached
      name: memcached
    - image: consul
      name: consul
```

### 3. eshop-frontend.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f eshop-frontend.yaml
pod/eshop-frontend created
```

### 4. eshop-frontend Pod 확인

```
ubuntu@master:~$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
eshop-frontend	3/4	ErrImagePull	0	39s

## 문제11) Rolling Update & Roll Back

Deployment를 이용해 nginx 파드를 3개 배포한 다음 컨테이너 이미지 버전을 rolling update하고 update record를 기록합니다.

마지막으로 컨테이너 이미지를 previous version으로 roll back 합니다.

name: eshop-payment

Image : nginx

Image version: 1.16

update image version: 1.17

label: app=payment, environment=production

### 1. eshop-payment.yaml Deployment 파일 생성

```
ubuntu@master:~$ kubectl create deploy eshop-payment --image=nginx:1.16 --replicas=3
--dry-run=client -o yaml > eshop-payment.yaml
```

### 2. eshop-payment-deployment.yaml 파일 생성

```
ubuntu@master:~$ vi eshop-payment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: payment
    environment: production
  name: eshop-payment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: payment
      environment: production
  template:
    metadata:
      labels:
        app: payment
        environment: production
    spec:
      containers:
        - image: nginx:1.16
          name: nginx
```

### 3. eshop-payment.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f eshop-payment.yaml --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/eshop-payment created
```

### 4. image 버전 업데이트

```
ubuntu@master:~$ kubectl set image deploy eshop-payment nginx=nginx:1.17 --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/eshop-payment image updated
```

### 5. rolling update 확인

```
ubuntu@master:~$ kubectl rollout history deploy eshop-payment
deployment.apps/eshop-payment
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=eshop-payment.yaml --record=true
2          kubectl set image deploy eshop-payment nginx=nginx:1.17 --record=true
```

### 6. roll back 실행

```
ubuntu@master:~$ kubectl rollout undo deploy eshop-payment
deployment.apps/eshop-payment rolled back
```

### 7. image 버전 확인

```
ubuntu@master:~$ kubectl rollout history deploy eshop-payment
deployment.apps/eshop-payment
REVISION  CHANGE-CAUSE
2          kubectl set image deploy eshop-payment nginx=nginx:1.17 --record=true
3          kubectl apply --filename=eshop-payment.yaml --record=true

ubuntu@master:~$ kubectl describe pod eshop-payment-6855fb78d6- | grep nginx
nginx:
Image:      nginx:1.16
Image ID:
docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf30
```

## 문제 12) ClusterIP

'devops' namespace에서 deployment eshop-order를 다음 조건으로 생성하시오.

- image: nginx, replicas: 2, label: name=order

'eshop-order' deployment의 Service를 만드세요.

Service Name: eshop-order-svc

Type: ClusterIP

Port: 80

### 1. devops 네임스페이스 생성

```
ubuntu@master:~$ kubectl create namespace devops
namespace/devops created
```

### 2. eshop-order.yaml Deployment 파일 생성

```
ubuntu@master:~$ kubectl create deploy eshop-order -n devops --replicas=2 --image=nginx
--dry-run=client -o yaml > eshop-order.yaml
```

### 3. eshop-order.yaml 파일 수정

```
ubuntu@master:~$ vi eshop-order.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: order
    name: eshop-order
    namespace: devops
spec:
  replicas: 2
  selector:
    matchLabels:
      name: order
  template:
    metadata:
      labels:
        name: order
    spec:
      containers:
        - image: nginx
          name: nginx
```

#### 4. eshop-order.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f eshop-order.yaml
deployment.apps/eshop-order created
```

#### 5. eshop-order Deployment 서비스 생성

```
ubuntu@master:~$ kubectl expose deploy eshop-order -n devops --name=eshop-order-svc --port=80
--target-port=80
service/eshop-order-svc exposed
```

#### 6. eshop-order Deployment 및 서비스 확인

```
ubuntu@master:~$ kubectl get deploy,svc -n devops
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/eshop-order	2/2	2	2	2m19s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/eshop-order-svc	ClusterIP	10.105.111.221	<none>	80/TCP	113s

## 문제 13) NodePort

'front-end' deployment를 다음 조건으로 생성하시오.

image: nginx, replicas: 2, label: run=nginx

'front-end' deployment의 nginx 컨테이너를 expose하는 'front-end-nodesvc'라는 새 service를 만듭니다.  
Front-end로 동작중인 Pod에는 node의 30200 포트로 접속되어야 합니다.

### 1. front-end.yaml Deployment 파일 생성

```
ubuntu@master:~$ kubectl create deploy front-end --image=nginx --replicas=2 --dry-run=client -o yaml > front-end.yaml
```

### 2. front-end.yaml 파일 수정

```
ubuntu@master:~$ vi front-end.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: nginx
  name: front-end
spec:
  replicas: 2
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
        - image: nginx
          name: nginx
```

### 3. front-end.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f front-end.yaml
deployment.apps/front-end created
```

#### 4. front-end-nodesvc 서비스 생성

```
ubuntu@master:~$ kubectl expose deploy front-end --name=front-end-nodesvc --port=80 --target-port=80
--type=NodePort --dry-run=client -o yaml > front-end-nodesvc.yaml
```

#### 5. front-end-nodesvc.yaml 파일 수정

```
ubuntu@master:~$ vi front-end-nodesvc.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nginx
  name: front-end-nodesvc
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
      nodePort: 30200
  selector:
    run: nginx
  type: NodePort
```

#### 6. front-end-nodesvc.yaml 파일 적용

```
ubuntu@master:~$ kubectl apply -f front-end-nodesvc.yaml
service/front-end-nodesvc created
```

#### 7. front-end 디플로이먼트 및 서비스 확인

```
ubuntu@master:~$ kubectl get deploy,svc
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/front-end	2/2	2	2	3m57s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/front-end-nodesvc	NodePort	10.104.209.193	<none>	80:30200/TCP	76s
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	7m38s



## 문제 14) Network Policy

customera, customerb를 생성한 후, 각각 PARTITION=customera, PARTITION=customerb를 라벨링하시오.

default namespace에 다음과 같은 pod를 생성하세요.

name: poc

image: nginx

port: 80

label: app=poc

"partition=customera"를 사용하는 namespace에서만 poc의

80포트로 연결할 수 있도록 default namespace에 'allow-web-from-customera'라는 network Policy를 설정하세요.

보안 정책상 다른 namespace의 접근은 제한합니다.

### 1. Namespace 생성 및 확인

```
ubuntu@master:~$ kubectl create namespace customera
```

```
namespace/customera created
```

```
ubuntu@master:~$ kubectl create namespace customerb
```

```
namespace/customerb created
```

```
ubuntu@master:~$ kubectl get ns customera customerb
```

NAME	STATUS	AGE
customera	Active	2h
customerb	Active	2h

### 2. Namespace labeling 및 확인

```
ubuntu@master:~$ kubectl label namespaces customera PARTITION=customera
```

```
namespace/customera labeled
```

```
ubuntu@master:~$ kubectl label namespaces customerb PARTITION=customerb
```

```
namespace/customerb labeled
```

```
ubuntu@master:~$ kubectl get ns -L PARTITION
```

NAME	STATUS	AGE	PARTITION
customera	Active	27h	customera
customerb	Active	27h	customerb

### 3. Pod 생성

```
ubuntu@master:~$ kubectl run poc --image=nginx --port=80 --labels=app=poc
pod/poc created
```

### 4. networkpolicy.yaml 파일 생성

```
ubuntu@master:~$ vi networkpolicy.yaml

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-web-from-customer
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: poc
  policyTypes:
    - Ingress
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            partition: customer
      ports:
        - protocol: TCP
          port: 80
```

### 5. networkpolicy.yaml 파일 적용 및 확인

```
ubuntu@master:~$ kubectl apply -f netpol.yaml
networkpolicy.networking.k8s.io/allow-web-from-customer created

ubuntu@master:~$ kubectl get networkpolicy
NAME                                POD-SELECTOR          AGE
allow-web-from-customer             app=poc                2h
```

## 문제 15) Ingress

Create a new nginx Ingress resource as follows:

- Name: ping
- Namespace: ing-internal
- Exposing service hi on path /hi using service port 5678

### 1. Namespace 생성 및 확인

```
ubuntu@master:~$ kubectl create namespace ing-internal
namespace/ing-internal created
ubuntu@master:~$ kubectl get ns ing-internal
```

NAME	STATUS	AGE
ing-internal	Active	2h

### 2. ingress.yaml 파일 생성 및 수정

```
ubuntu@master:~$ vi ingress.yaml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /hi
        pathType: Prefix
      backend:
        service:
          name: hi
          port:
            number: 5678
```

### 3. ingress.yaml 적용 및 확인

```
ubuntu@master:~$ kubectl apply -f ingress.yaml
ingress.networking.k8s.io/ping created
ubuntu@master:~$ kubectl get ing -n ing-internal
```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
ping	nginx-example	*		80	15s

## 문제 16) Service and DNS Lookup

image nginx를 사용하는 resolver pod를 생성하고 resolver-service라는 service를 구성합니다.

클러스터 내에서 service와 pod 이름을 조회할 수 있는지 테스트합니다.

- dns 조회에 사용하는 pod 이미지는 busybox:1.28이고, service와 pod 이름 조회는 nslookup을 사용합니다.

- service 조회 결과는 /home/ubuntu/nginx.svc에 pod name 조회 결과는 /home/ubuntu/nginx.pod 파일에 기록합니다.

### 1. Pod 생성 및 확인

```
ubuntu@master:~$ kubectl run resolver --image=nginx --port=80
pod/resolver created
```

```
ubuntu@master:~$ kubectl get po resolver
```

NAME	READY	STATUS	RESTARTS	AGE
resolver	1/1	Running	0	21s

### 2. 서비스 생성 및 확인

```
ubuntu@master:~$ kubectl expose pod resolver --name=resolver-service --port=80
service/resolver-service exposed
```

```
ubuntu@master:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2m16s
resolver-service	ClusterIP	10.103.218.60	<none>	80/TCP	11s

### 3. 서비스 DNS 조회 및 저장

```
ubuntu@master:~$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup
10.103.218.60
```

```
Server: 10.96.0.10
```

```
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local
```

```
Name: 10.103.218.60
```

```
Address 1: 10.103.218.60 resolver-service.default.svc.cluster.local
```

```
pod "testpod" deleted
```

```
ubuntu@master:~$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup
10.103.218.60 > /home/ubuntu/nginx.svc
```

#### 4. Pod DNS 조회 및 저장

```
ubuntu@master:~$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup
10-103-218-60.default.pod.cluster.local
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: 10-103-218-60.default.pod.cluster.local
Address 1: 10.103.218.60 resolver-service.default.svc.cluster.local
pod "testpod" deleted

ubuntu@master:~$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup
10-103-218-60.default.pod.cluster.local > nginx.pod
```

#### 5. 확인

```
ubuntu@master:~$ cat nginx.pod
Server: 10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: 10-103-218-60.default.pod.cluster.local
Address 1: 10.103.218.60 resolver-service.default.svc.cluster.local
pod "testpod" deleted
```

## 문제 17) emptyDir Volume

다음 조건에 맞춰서 nginx 웹서버 pod가 생성한 로그파일을 받아서 STDOUT으로 출력하는 busybox 컨테이너를 운영하시오.

Pod Name: weblog

Web container:

- Image: nginx:1.17
- Volume mount : /var/log/nginx
- Readwrite

Log container:

- Image: busybox
- args: /bin/sh, -c, "tail -n+1 -f /data/access.log"
- Volume mount : /data
- readonly

emptyDir 볼륨을 통한 데이터 공유

### 1. Pod 생성용 weblog.yaml파일 생성

```
ubuntu@master:~$ kubectl run weblog --image=nginx:1.17 --dry-run=client -o yaml > weblog.yaml
```

### 2. weblog.yaml 수정

```
ubuntu@master:~$ vi weblog.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: weblog
spec:
  containers:
    - image: nginx:1.17
      name: weblog
      volumeMounts:
        - mountPath: /var/log/nginx
          name: weblog
    - image: busybox
      args: [/bin/sh, -c, "tail -n+1 -f /data/access.log"]
      name: log
      volumeMounts:
        - mountPath: /data
          name: weblog
          readOnly: true
  volumes:
    - name: weblog

emptyDir: {}
```

### 3. weblog.yaml 적용 및 확인

```
ubuntu@master:~$ kubectl apply -f weblog.yaml
```

```
pod/weblog created
```

```
ubuntu@master:~$ kubectl describe pod weblog
```

```
Name:          weblog
```

```
...
```

```
Containers:
```

```
weblog:
```

```
Container ID:
```

```
containerd://cbda42a0416299d2cee4afde6b21adcdf84d046339bfada7becdeb69d8d261e
```

```
Image:         nginx:1.17
```

```
Image ID:
```

```
docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
```

```
Port:          <none>
```

```
Host Port:     <none>
```

```
State:         Running
```

```
Started:       Thu, 14 Aug 2025 04:45:34 +0000
```

```
Ready:         True
```

```
Restart Count: 0
```

```
Environment:   <none>
```

```
Mounts:
```

```
/var/log/nginx from weblog (rw)
```

```
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-zhsvd (ro)
```

```
log:
```

```
Container ID:
```

```
containerd://89c94bcdee111fdca9207a999345bef497468a5c38936ed23334f242393ab32f
```

```
Image:         busybox
```

```
Image ID:
```

```
docker.io/library/busybox@sha256:f9a104fddb33220ec80fc45a4e606c74aadf1ef7a3832eb0b05be9e90cd61f5f
```

```
Port:          <none>
```

```
Host Port:     <none>
```

```
Args:
```

```
/bin/sh
```

```
-c
```

```
tail -n+1 -f /data/access.log
```

```
State:         Running
```

```
Started:       Thu, 14 Aug 2025 04:45:35 +0000
```

```
Ready:         True
```

```
Restart Count: 0
```

```
Environment:   <none>
```

```
Mounts:
```

```
/data from weblog (ro)
```

```
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-zhsvd (ro)
```

```
...
```

## 문제18) HostPath Volume

1. /data/cka/fluentsd.yaml 파일을 만들어 새로운 Pod 생성하세요

신규생성 Pod Name: fluentd, image: fluentd, namespace: default)

2. 위 조건을 참고하여 다음 조건에 맞게 볼륨마운트를 설정하시오.

1. Worker node의 도커 컨테이너 디렉토리 : /var/lib/docker/containers 동일 디렉토리로 pod에 마운트하시오.

2. Worker node의 /var/log 디렉토리를 fluentd Pod에 동일이름의 디렉토리 마운트하시오.

### 1. cka 폴더 생성 및 이동

```
ubuntu@master:~$ mkdir cka
ubuntu@master:~$ cd cka
```

### 2. Pod 생성

```
ubuntu@master:~/cka$ kubectl run fluentd --image=fluentd --port=80
pod/fluentd created
```

### 3. 생성된 Pod의 fluentsd.yaml 파일 생성 및 수정

```
ubuntu@master:~/cka$ kubectl get po fluentd -o yaml > fluentd.yaml
ubuntu@master:~/cka$ vi fluentd.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: fluentd
spec:
  containers:
  - image: fluentd
    name: fluentd
    ports:
    - containerPort: 80
      protocol: TCP
    volumeMounts:
    - mountPath: /var/lib/docker/containers
      name: containersdir
    - mountPath: /var/log
      name: logdir
  volumes:
  - name: containersdir
    hostPath:
      path: /var/lib/docker/containers
  - name: logdir
    hostPath:
      path: /var/log
```



#### 4. Pod 삭제

```
ubuntu@master:~/cka$ kubectl delete po fluentd  
pod "fluentd" deleted
```

#### 5. fluentd.yaml 적용 및 확인

```
ubuntu@master:~/cka$ kubectl apply -f fluentd.yaml  
pod/fluentd created  
  
ubuntu@master:~/cka$ kubectl describe pod fluentd  
Name:          fluentd  
...  
Containers:  
  fluentd:  
    Container ID:    
    Image:         fluentd  
    Image ID:        
    Port:          80/TCP  
    Host Port:     0/TCP  
    State:         Waiting  
      Reason:      ContainerCreating  
    Ready:         False  
    Restart Count: 0  
    Environment:   <none>  
    Mounts:  
      /var/lib/docker/containers from containersdir (rw)  
      /var/log from logdir (rw)  
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-p448z (ro)  
    ...
```

## 문제 19) Persistent Volume

pv001라는 이름으로 size 1Gi, access mode ReadWriteMany를 사용하여 persistent volume을 생성합니다.

volume type은 hostPath이고 위치는 /tmp/app-config입니다.

### 1. pv.001.yaml 생성 및 수정

```
ubuntu@master:~$ vi pv001.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /tmp/app-config
```

### 2. 적용 및 확인

```
ubuntu@master:~$ kubectl apply -f pv001.yaml
```

```
persistentvolume/pv001 created
```

```
ubuntu@master:~$ kubectl get persistentvolume
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
pv001	1Gi	RWX	Retain	Available				38s