Team : Rest

Kubernetes

프로젝트

REPORT

강승환

고동우

유세종

최성민

한시완

[문제 1] – ETCD 백업

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

그런 다음 다시 스냅샷을 복원

etcdctl을 사용하여 서버에 연결하기 위해 다음 TLS 인증서/키가 제공

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/에서 etcd backup 검색 후 해당 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 계정 전환

sudo -i

(3) etcd-client 설치

apt install etcd-client

(4) 스냅샷 생성 및 저장

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 etcd-snapshot.db

(5) 스냅샷 복원

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 etcd-snapshot.db

(6) tree 설치

apt install tree

(7) 스냅샷 복원 확인

```
root@master:~# tree default.etcd/member
default.etcd/member
├── snap
│   ├── 0000000000000001-0000000000000001.snap
│   └── db
└── wal
    └── 0000000000000000-0000000000000000.wal

3 directories, 3 files
```

[문제 2] – Cluster Upgrade

마스터 노드의 모든 Kubernetes control plane및 node 구성 요소를 버전 1.28.8-1.1 버전으로 업그레이드

master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon

"주의사항" 반드시 Master Node에서 root권한을 가지고 작업을 실행

(1) 현재 버전 확인

kubectl get nodes

```
ubuntu@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   6d3h   v1.30.14
worker1   Ready    <none>          6d3h   v1.30.14
worker2   Ready    <none>          6d3h   v1.30.14
```

(2) 업그레이드 버전 결정

apt update

apt-cache madison kubeadm

(3) kubeadm 업그레이드 호출

apt-mark unhold kubeadm

apt-get update && sudo apt-get install -y kubeadm='1.28.8-1.1'

apt-mark hold kubeadm

(4) 업그레이드 버전 선택 후 실행

sudo kubeadm upgrade apply --force v1.28.8-1.1

(5) 노드를 예약 불가능으로 표시하고 유지 관리 준비

kubectl drain master --ignore-daemonsets

(6) kubelet 및 kubectl 업그레이드

sudo apt-mark unhold kubelet kubectl

sudo apt-get update && sudo apt-get install -y kubelet='1.28.8-1.1' kubectl='1.28.8-1.1'

sudo apt-mark hold kubelet kubectl

(7) kubelet 다시 시작

sudo systemctl daemon-reload

sudo systemctl restart kubelet

(8) 노드 차단 해제

kubectl uncordon master

[문제 3] – 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 생성

2. podreader-role이라는 이름의 Role과 podreader-rolebinding이라는 이름의 RoleBinding을 생성

3. 앞서 생성한 ServiceAccount를 API resource Pod에 대하여 watch, list, get을 허용하도록 매핑

(1) NameSpace 생성 및 확인

kubectl create ns api-access

```
ubuntu@master:~$ kubectl get ns
NAME              STATUS   AGE
api-access        Active   7s
```

(2) ServiceAccount 생성 및 확인

kubectl create serviceaccount pod-viewer -n api-access

```
ubuntu@master:~$ kubectl get serviceaccount -n api-access
NAME         SECRETS   AGE
default      0         24m
pod-viewer   0         111s
```

(3) Role 생성 및 확인

kubectl create role podreader-role -n api-access --resource=pod --verb=watch,list,get

```
ubuntu@master:~$ kubectl get role -n api-access
NAME             CREATED AT
podreader-role   2025-08-12T05:27:19Z
```

(4) RoleBinding 생성 및 확인

kubectl create rolebinding podreader-rolebinding --role=podreader-role --serviceaccount=api-access:pod-viewer -n api-access

```
ubuntu@master:~$ kubectl get rolebinding -n api-access
NAME                    ROLE                   AGE
podreader-rolebinding   Role/podreader-role    21m
```

[문제 4] - Service Account, ClusterRole, ClusterRoleBinding 생성

애플리케이션 배포를 위해 새로운 ClusterRole을 생성하고 특정 namespace의 ServiceAccount를 바인드한다.

다음의 resource type에서만 Create가 허용된 ClusterRole deployment-clusterrole을 생성

Resource Type: Deployment StatefulSet DaemonSet

미리 생성된 namespace api-access에 cicd-token이라는 새로운 ServiceAccount를 생성

ClusterRole deployment-clusterrole을 namespace api-access로 제한된 ServiceAccount cicd-token에 바인딩

(1) ServiceAccount 생성 및 확인

kubectl create sa cicd-token -n api-access

```
ubuntu@master:~$ kubectl get sa -n api-access
NAME         SECRETS   AGE
cicd-token   0         51s
```

(2) ClusterRole 생성 및 확인

kubectl create clusterrole deployment-clusterrole --resource=deployment,statefulset,daemonset --verb=create

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

(3) ClusterRoleBinding 생성 및 확인

kubectl create clusterrolebinding deployment-clusterrolebinding --clusterrole=deployment-clusterrole --serviceaccount=api-access:cicd-token -n api-access

```
ubuntu@master:~$ kubectl describe clusterrolebinding deployment-clusterrolebinding
Name:         deployment-clusterrolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  deployment-clusterrole
Subjects:
  Kind            Name        Namespace
  ----            ----        ---------
  ServiceAccount  cicd-token  api-access
```

[문제 5] – 노드 비우기

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

(1) 노드 확인

kubectl get no

```
ubuntu@master:~$ kubectl get no
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   6d4h   v1.30.14
worker1   Ready    <none>          6d4h   v1.30.14
worker2   Ready    <none>          6d4h   v1.30.14
```

(2) 노드 드레인

kubectl drain worker2 –ignore-daemonsets --force

```
ubuntu@master:~/k8s$ kubectl drain worker2 --ignore-daemonsets --force
node/worker2 already cordoned
Warning: deleting Pods that declare no controller: default/nginx-static-pod; ignoring DaemonSet-managed Pods: kube-sy
stem/calico-node-f2s59, kube-system/kube-proxy-ddpxz
evicting pod default/nginx-static-pod
pod/nginx-static-pod evicted
node/worker2 drained
```

(3) 노드 확인

```
ubuntu@master:~/k8s$ 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) 노드 리스케쥴링 및 확인

kubectl uncordon worker2

```
ubuntu@master:~/k8s$ kubectl uncordon worker2
node/worker2 uncordoned
ubuntu@master:~/k8s$
ubuntu@master:~/k8s$ 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
```

[문제 6] – Pod Sheduling

다음의 조건으로 pod를 생성하세요.

Name: eshop-store

Image: nginx

Nodeselector: disktype=ssd

(1) 노드 라벨 부여 및 확인

kubectl label node worker1 disktype=ssh

kubectl label node worker2 disktype=hdd

kubectl get no -L disktype

```
ubuntu@master:~/k8s$ kubectl get no -L disktype
NAME      STATUS   ROLES           AGE    VERSION    DISKTYPE
master    Ready    control-plane   6d6h   v1.30.14
worker1   Ready    <none>          6d5h   v1.30.14   ssh
worker2   Ready    <none>          6d5h   v1.30.14   hdd
```

(2) YAML 파일 생성

kubectl run eshop-store --image=nginx --dry-run=client -o yaml > eshop-store.yaml

(3) YAML 파일 수정

vi eshop-store.yaml

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

(4) YAML 파일 적용 및 확인

kubectl apply -f eshop-store.yaml

```
ubuntu@master:~/k8s$ kubectl get po -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
eshop-store   1/1     Running   0          26m   172.17.235.129   worker1   <none>           <none>
```

[문제 7] – 환경변수, command, args 적용

'cka-exam'이라는 namespace를만들고, '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) NameSpace 생성 및 확인

kubectl create ns cka-exam

```
ubuntu@master:~$ kubectl get ns
NAME              STATUS   AGE
cka-exam          Active   12s
```

(2) YAML 파일 생성

kubectl run pod-01 --image=busybox -n cka-exam --env=CERT="CKA-cert" --dry-run=client -o yaml > pod-01.yaml

(3) YAML 파일 수정

vi pod-01.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  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"]
```

(4) YAML 파일 적용

kubectl apply -f pod-01.yaml

(5) pod 생성 확인

kubectl get po pod-01 -n cka-exam -o wide

```
ubuntu@master:~/k8s$ 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          3m42s   172.16.235.129   worker1   <none>           <none>
```

kubectl describe pod pod-01 -n cka-exam

```
Command:
  /bin/sh
Args:
  -c
  while true; do echo $(CERT); sleep 10;done
State:          Running
  Started:      Tue, 12 Aug 2025 07:02:51 +0000
Ready:          True
Restart Count:  0
Environment:
  CERT:  CKA-cert
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-zqmbb (ro)
```

[문제 8] – Static Pod 생성

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

pod name: nginx-static-pod

image: nginx

port : 80

(1) ssh로 worker1 접속

ssh worker1

(2) worker1의 static 위치 확인 및 이동

cd /var/lib/kubelet

cat config.yaml | grep -I static

cd /etc/kubernetes/manifests

(3) YAML 파일 생성

kubectl run nginx-static-pod --image=nginx --port=80 --dry-run=client -o yaml > nginx-static-pod.yaml

(4) YAML 파일 수정

vi nginx-static-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx-static-pod
  name: nginx-static-pod
spec:
  containers:
  - image: nginx
    name: nginx-static-pod
    ports:
    - containerPort: 80
```

(5) YAML 파일 적용 및 확인

kubectl apply -f nginx-static-pod.yaml

kubectl get po nginx-static-pod -o wide

```
ubuntu@master:~/k8s$ kubectl get po nginx-static-pod-worker1 -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
nginx-static-pod-worker1  1/1     Running   0          81s   172.17.235.130  worker1   <none>           <none>
```

[문제 9] – 로그 확인

Pod "nginx-static-pod-k8s-worker1"의 log를 모니터링하고, 메세지를 포함하는 로그라인을 추출하세요.

추출된 결과는 /home/ubuntu/pod-log에 기록하세요.

(1) 파드 확인

kubectl get po nginx-static-pod-worker1

```
ubuntu@master:~/k8s$ kubectl get po nginx-static-pod-worker1
NAME                       READY   STATUS    RESTARTS   AGE
nginx-static-pod-worker1   1/1     Running   0          7m8s
```

(2) root 계정 전환

sudo -i

(3) 디렉터리 생성

mkdir -p /opt/REPORT/2023/pod-log

(4) 로그 모니터링

kubectl logs nginx-static-pod-worker1

```
root@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
2025/08/13 04:51:24 [notice] 1#1: using the "epoll" event method
2025/08/13 04:51:24 [notice] 1#1: nginx/1.29.0
2025/08/13 04:51:24 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14+deb12u1)
2025/08/13 04:51:24 [notice] 1#1: OS: Linux 6.8.0-71-generic
2025/08/13 04:51:24 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/08/13 04:51:24 [notice] 1#1: start worker processes
2025/08/13 04:51:24 [notice] 1#1: start worker process 29
2025/08/13 04:51:24 [notice] 1#1: start worker process 30
```

(5) 로그 저장 및 확인

sudo kubectl logs nginx-static-pod-worker1 > /opt/REPORT/2023/pod-log/log.txt

cat /opt/REPORT/2023/pod-log/log.txt

```
root@master:~# cat /opt/REPORT/2023/pod-log/log.txt
/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
2025/08/13 04:51:24 [notice] 1#1: using the "epoll" event method
2025/08/13 04:51:24 [notice] 1#1: nginx/1.29.0
2025/08/13 04:51:24 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14+deb12u1)
2025/08/13 04:51:24 [notice] 1#1: OS: Linux 6.8.0-71-generic
2025/08/13 04:51:24 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/08/13 04:51:24 [notice] 1#1: start worker processes
2025/08/13 04:51:24 [notice] 1#1: start worker process 29
2025/08/13 04:51:24 [notice] 1#1: start worker process 30
```

[문제 10] – Multi Container Pod 생성

4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성하시오.

pod image: nginx, redis, memcached, consul

(1) YAML 파일 생성

kubectl run eshop-frontend --image=nginx --dry-run=client -o yaml > eshop-frontend.yaml

(2) YAML 파일 수정

vi eshop-frontend.yaml

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

(3) YAML 파일 적용

kubectl apply -f eshop-frontend.yaml

(4) 파드 확인

kubectl get po eshop-frontend

```
ubuntu@master:~/k8s$ k get po eshop-frontend
NAME             READY   STATUS             RESTARTS   AGE
eshop-frontend   3/4     ImagePullBackOff   0          115s
```

[문제 11] – 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) YAML 파일 생성

kubectl create deploy eshop-payment --image=nginx:1.16 --replicas=3 --dry-run=client -o yaml > eshop-payment.yaml

(2) YAML 파일 수정

vi eshop-payment.yaml

```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) YAML 파일 적용 및 확인

kubectl apply -f eshop-payment.yaml --record

kubectl get deploy,rs,po

```
ubuntu@master:~/k8s$ kubectl get deploy,rs,po
NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-payment    3/3     3            3           2m15s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/eshop-payment-6855fb78d6    3         3         3       2m15s

NAME                                    READY   STATUS    RESTARTS   AGE
pod/eshop-payment-6855fb78d6-9b8p2      1/1     Running   0          2m15s
pod/eshop-payment-6855fb78d6-bx9kl      1/1     Running   0          2m15s
pod/eshop-payment-6855fb78d6-kc7xj      1/1     Running   0          2m15s
```

(4) Rolling Update

kubectl set image deploy eshop-payment nginx=nginx:1.17 --record

kubectl rollout history deploy eshop-payment

kubectl describe po eshop-payment | grep -I nginx

```
ubuntu@master:~/k8s$ kubectl describe po eshop-payment | grep -i nginx
  nginx:
    Image:          nginx:1.17
```

(5) RollBack

kubectl rollout undo deploy eshop-payment

kubectl rollout history deploy eshop-payment

kubectl describe po eshop-payment | grep -I nginx

```
ubuntu@master:~/k8s$ kubectl describe po eshop-payment | grep -i nginx
  nginx:
    Image:          nginx:1.16
```

[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) NameSpace 생성 및 확인

kubectl create ns devops

```
ubuntu@master:~/k8s$ kubectl get ns devops
NAME      STATUS   AGE
devops    Active   26m
```

(2) YAML 파일 생성

kubectl create deploy eshop-order -n devops --image=nginx --replicas=2 --dry-run=client -o yaml > eshop-order.yaml

(3) YAML 파일 수정

vi eshop-order.yaml

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

(4) YAML 파일 적용 및 확인

kubectl apply -f eshop-order.yaml

(5) 서비스 생성 및 확인

kubectl expose deploy eshop-order -n devops --name=eshop-order-svc --port=80 --target-port=80

kubectl get deploy,svc -n devops

```
ubuntu@master:~/k8s$ kubectl get deploy,svc -n devops
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-order    2/2     2            2           44s

NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/eshop-order-svc     ClusterIP   10.104.95.135   <none>        80/TCP    7s
```

[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) YAML 파일 생성

kubectl create deploy eshop-order -n devops --image=nginx --replicas=2 --dry-run=client -o yaml > eshop-order.yaml

(2) YAML 파일 수정

vi front-end.yaml

```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) YAML 파일 적용 및 확인

kubectl get apply -f front-end.yaml

kubectl get deploy front-end --show-labels

```
ubuntu@master:~/k8s$ kubectl get deploy front-end --show-labels
NAME        READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
front-end   2/2     2            2           78s   run=nginx
```

(4) Service YAML 파일 생성

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) Service YAML 파일 수정

vi front-end-nodesvc.yaml

```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) Service YAML 파일 적용 및 확인

kubectl apply -f front-end-nodesvc.yaml

kubectl get svc front-end-nodesvc

```
ubuntu@master:~/k8s$ kubectl get svc front-end-nodesvc
NAME                TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
front-end-nodesvc   NodePort   10.97.193.151   <none>        80:30200/TCP   36s
```

[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 설정

보안 정책상 다른 namespace의 접근은 제한합니다.

(1) NameSpace 생성 및 확인

kubectl create ns customera

kubectl create ns customerb

```
ubuntu@master:~/k8s$ kubectl get ns customera customerb
NAME        STATUS   AGE
customera   Active   43s
customerb   Active   43s
```

(2) NameSpace 라벨링 및 확인

kubectl label ns customera PARTITION=customera

kubectl label ns customerb PARTITION=customerb

kubectl get ns -L PARTITION

```
ubuntu@master:~/k8s$ kubectl get ns -L PARTITION
NAME              STATUS   AGE     PARTITION
customera         Active   2m58s   customera
customerb         Active   2m58s   customerb
```

(3) Pod 생성 및 확인

kubectl run poc --image=nginx --port=80 --labels=app=poc

kubectl get po poc

```
ubuntu@master:~/k8s$ k get po poc --show-labels
NAME   READY   STATUS    RESTARTS   AGE   LABELS
poc    1/1     Running   0          8s    app=poc
```

(4) Network Policy YAML 생성 및 수정

Kubernetes 사이트에서 network policy 검색 후 사용할 yaml 파일 복사

vi netpol.yaml

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

(5) Network Policy YAML 적용 및 확인

kubectl apply -f netpol.yaml

kubectl get netpol

```
ubuntu@master:~/k8s$ kubectl get netpol
NAME                        POD-SELECTOR   AGE
allow-web-from-customera    app=poc        2m49s
```

[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

kubectl create ns ing-internal

```
ubuntu@master:~/k8s$ kubectl get ns ing-internal
NAME           STATUS   AGE
ing-internal   Active   7s
```

(2) Ingress YAML 파일 생성 및 수정

Kubernetes 사이트에서 ingress 검색 후 사용할 yaml 파일 복사

vi ingress.yaml

```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

kubectl apply -f ingress.yaml

kubectl get ing -n ing-internal

```
ubuntu@master:~/k8s$ kubectl get ing -n ing-internal
NAME   CLASS           HOSTS   ADDRESS   PORTS   AGE
ping   nginx-example   *                 80      23s
```

[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 nlsookup을 사용

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

/home/ubuntu/nginx.pod 파일에 기록합니다.

(1) Pod 생성 및 확인

kubectl run resolver --image=nginx --port=80

kubectl get po resolver

```
ubuntu@master:~/k8s$ k get po resolver
NAME       READY   STATUS    RESTARTS   AGE
resolver   1/1     Running   0          27s
```

(2) Service 생성 및 확인

kubectl expose pod resolver --name=resolver-service --port=80

kubectl get svc resolver-service

```
ubuntu@master:~/k8s$ kubectl get svc resolver-service
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
resolver-service   ClusterIP   10.105.184.16   <none>        80/TCP    42s
```

(3) Service DNS 조회 및 저장

kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10.105.184.16

kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10.105.184.16 > /home/ubuntu/nginx.svc

cat nginx.svc

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

Name:      10.105.184.16
Address 1: 10.105.184.16 resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
```

(4) Pod DNS 조회 및 저장

kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10-105-184-16.default.pod.cluster.local > /home/ubuntu/nginx.pod

cat nginx.svc

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

Name:      10-105-184-16.default.pod.cluster.local
Address 1: 10.105.184.16 resolver-service.default.svc.cluster.local
pod "test-nslookup" 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

(1) Pod YAML 파일 생성

kubectl run weblog --image=nginx:1.17 --dry-run=client -o yaml > weblog.yaml

(2) Pod YAML 파일 수정

vi weblog.yaml

```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) Pod YAML 파일 적용 및 확인

kubectl apply -f weblog.yaml

kubectl get po weblog

kubectl describe po weblog

```
ubuntu@master:~/k8s$ kubectl get po weblog
NAME     READY   STATUS    RESTARTS   AGE
weblog   2/2     Running   0          16s
```

```
Containers:
  weblog:
    Container ID:   containerd://d403f78d671bb8b1bc1f4d45a14c9664318aab10e541e469e843e9c14ddc8275
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 14 Aug 2025 06:23:17 +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-fbwr2 (ro)
  log:
    Container ID:  containerd://5f01c77472013fd5248602b8c9a3e5ae08c9e9053c843e603dd455a23a4a04dd
    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 06:23:19 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /data from weblog (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fbwr2 (ro)
```

[문제 18] – HostPath Volume

1. /data/cka/fluentd.yaml 파일을 만들어 새로은 Pod 생성하세요

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

2. 위 조건을 참고하여 다음 조건에 맞게 볼륨마운트를 설정하시오.

3. Worker node의 도커 컨테이너 디렉토리 : /var/lib/docker/containers 동일 디렉토리로 pod에 마운트

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

(1) cka 폴더 생성 및 이동

mkdir cka

cd cka

(2) Pod YAML 파일 생성

kubectl run fluentd --image=fluentd --port=80 --dry-run=client -o yaml > fluentd.yaml

(3) Pod YAML 파일 수정

vi fluentd.yaml

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

(4) Pod YAML 파일 적용 및 확인

kubectl apply -f fluentd.yaml

kubectl describe po fluentd

```
Volumes:
  containersdir:
    Type:          HostPath (bare host directory volume)
    Path:          /var/lib/docker/containers
    HostPathType:
  logdir:
    Type:          HostPath (bare host directory volume)
    Path:          /var/log
```

[문제 19] – Persistent Volume

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

volume type은 hostPath이고 위치는 /tmp/app-config입니다.

(1) Pod YAML 파일 생성 및 수정

Kubernetes 사이트에서 pv 검색 후 사용할 yaml 파일 복사

vi pv001.yaml

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

(2) YAML 파일 적용 및 확인

kubectl apply -f pv001.yaml

kubectl get pv pv001


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