대우직업능력개발원
Daewoo Development Institute for Vocational ability

쿠버네티스 프로젝트

이정일
서재권
김민호
임재근

APEX

01 [ETCD 백업]

1. https://127.0.0.1:2379에서 실행 중인 etcd의 snapshot을 생성하고 snapshot을 etcd-snapshot.db에 저장
 후 스냅샷 복원
2. 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/docs에서 etcd backup 검색 후 명령어 찾기

2. 루트 계정으로 전환

sudo -i

3. ETCD 클라이언트 설치

apt install etcd-client

4.명령어 수정
#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. 위 명령어 적용(백업)

```
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 etcd-snapshot.db
{"level":"info","ts":1754896487.9092126,"caller":"snapshot/v3_snapshot.go:119","msg":"created
 temporary db file","path":"etcd-snapshot.db.part"}
{"level":"info","ts":"2025-08-11T07:14:47.917208Z","caller":"clientv3/maintenance.go:212","ms
g":"opened snapshot stream; downloading"}
{"level":"info","ts":1754896487.917287,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching
 snapshot","endpoint":"https://127.0.0.1:2379"}
{"level":"info","ts":"2025-08-11T07:14:48.014038Z","caller":"clientv3/maintenance.go:220","ms
g":"completed snapshot read; closing"}
{"level":"info","ts":1754896488.0295432,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched
 snapshot","endpoint":"https://127.0.0.1:2379","size":"7.9 MB","took":0.119854689}
{"level":"info","ts":1754896488.029675,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","
path":"etcd-snapshot.db"}
Snapshot saved at etcd-snapshot.db
```

6. 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 etcd-snapshot.db
{"level":"info","ts":1754896567.1978636,"caller":"snapshot/v3_snapshot.go:306","msg":"restori
ng snapshot","path":"etcd-snapshot.db","wal-dir":"default.etcd/member/wal","data-dir":"defaul
t.etcd","snap-dir":"default.etcd/member/snap"}
{"level":"info","ts":1754896567.244799,"caller":"mvcc/kvstore.go:388","msg":"restored last co
mpact revision","meta-bucket-name":"meta","meta-bucket-name-key":"finishedCompactRev","restor
ed-compact-revision":128694}
{"level":"info","ts":1754896567.2539272,"caller":"membership/cluster.go:392","msg":"added mem
ber","cluster-id":"cdf818194e3a8c32","local-member-id":"0","added-peer-id":"8e9e05c52164694d"
,"added-peer-peer-urls":["http://localhost:2380"]}
{"level":"info","ts":1754896567.2591524,"caller":"snapshot/v3_snapshot.go:326","msg":"restore
d snapshot","path":"etcd-snapshot.db","wal-dir":"default.etcd/member/wal","data-dir":"default
.etcd","snap-dir":"default.etcd/member/snap"}
```

7. tree 설치

apt install tree

8. 복원 여부 확인

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

3 directories, 3 files
root@master:~#
```

02 [Cluster Upgrade]

1. 마스터 노드의 모든 Kubernetes control plane및 node 구성 요소를 버전 1.28.8-1.1 버전으로 업그레이드 합니다.
2. master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon해야 합니다.
3. "주의사항" 반드시 Master Node에서 root권한을 가지고 작업을 실행해야 한다.

1. kubectl get node 버전 확인

```
ubuntu@master:~$ kubectl get node
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   5d5h   v1.30.14
worker1   Ready    <none>          5d5h   v1.30.14
worker2   Ready    <none>          5d5h   v1.30.14
```

2. kubeadm upgrade검색

3. 업그레이드 버전 결정
sudo apt update
sudo apt-cache madison kubeadm
4. kubeadm 업그레이드 호출
sudo apt-mark unhold kubeadm

```
ubuntu@master:~$ sudo apt-mark unhold kubeadm
Canceled hold on kubeadm.
```

sudo apt-get update && sudo apt-get install -y kubeadm='1.30.14-*'
sudo apt-mark hold kubeadm

```
ubuntu@master:~$ sudo apt-mark hold kubeadm
kubeadm set on hold.
```

kubectl get node

```
ubuntu@master:~$ kubectl get node
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   5d6h   v1.30.14
worker1   Ready    <none>          5d6h   v1.30.14
worker2   Ready    <none>          5d6h   v1.30.14
```

5. 업그레이드 버전 선택 후 실행
sudo kubeadm upgrade plan
sudo kubeadm upgrade apply v1.30.14 - -v5

6. 노드를 예약 불가능으로 표시하고 유지 관리 준비
kubectl drain master - -ignore-daemonsets

```
ubuntu@master:~$ kubectl drain master --ignore-daemonsets
node/master cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/calico-node-r5qwx, kube-system/kube-pro
xy-nb66m
evicting pod kube-system/coredns-55cb58b774-xws4m
evicting pod kube-system/coredns-55cb58b774-2mbl6
pod/coredns-55cb58b774-2mbl6 evicted
pod/coredns-55cb58b774-xws4m evicted
node/master drained
```

7. kubelet 및 kubectl 업그레이드
sudo apt-mark unhold kubelet kubectl
sudo apt-get update && sudo apt-get install -y kubelet='1.30.14-*' kubectl='1.30.14-*'
sudo apt-mark hold kubelet kubectl

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

8. kubelet 다시 시작
sudo systemctl daemon-reload
sudo systemctl restart kubelet
9. 노드 차단 해제
3-15. kubectl uncordon master

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

10. kubectl get node 버전 재확인
kubectl get no

```
ubuntu@master:~$ kubectl get no
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   5d6h   v1.30.14
worker1   Ready    <none>          5d6h   v1.30.14
worker2   Ready    <none>          5d6h   v1.30.14
ubuntu@master:~$
```

03 [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 새로운 namespace 생성
kubectl create ns api-access

```
ubuntu@master:~/k8s$ kubectl create namespace api-access
namespace/api-access created
ubuntu@master:~/k8s$
ubuntu@master:~/k8s$ kubectl get namespace
NAME              STATUS   AGE
api-access        Active   20s
default           Active   2d3h
kube-node-lease   Active   2d3h
kube-public       Active   2d3h
kube-system       Active   2d3h
ubuntu@master:~/k8s$
```

2. pod-viewer라는 이름의 Service Account 생성 및 확인
kubectl create serviceaccount pod-viewer -n api-access
kubectl get serviceaccount -n api-aacess

```
ubuntu@master:~/k8s$ k create serviceaccount pod-viewer -n api-access
serviceaccount/pod-viewer created
ubuntu@master:~/k8s$ k get serviceaccount -n api-access
NAME         SECRETS   AGE
default      0         10m
pod-viewer   0         8m
```

3. watch,list,get 허용하도록 Podreader-role라는 이름의 Role생성
kubectl create role podreader-role -n api-access --resource=pod --verb=watch,list,get
kubectl describe role -namespace api-access

```
ubuntu@master:~/k8s$ kubectl create role podreader-role -n api-access --resource=pod --verb=
watch,list,get
role.rbac.authorization.k8s.io/podreader-role created
ubuntu@master:~/k8s$
ubuntu@master:~/k8s$ kubectl get role -n api-access
NAME              CREATED AT
podreader-role    2025-08-08T04:56:00Z
ubuntu@master:~/k8s$
```

4. podreader-rolebinding라는 이름의 RoleBinding 생성
kubectl create rolebinding podreader-rolebinding --serviceaccount=api-account:pod-viewer --role=podreader-role -n api-access
kubectl get rolebinding -namespace api-access

```
ubuntu@master:~/k8s$ kubectl create rolebinding podreader-rolebinding --serviceaccount=api-a
ccount:pod-viewer --role=podreader-role -n api-access
rolebinding.rbac.authorization.k8s.io/podreader-rolebinding created
ubuntu@master:~/k8s$ kubectl get rolebinding -n api-access
NAME                    ROLE                   AGE
podreader-rolebinding   Role/podreader-role    53s
ubuntu@master:~/k8s$
```

04 [Service Account, ClusterRole, ClusterRoleBinding 생성 생성]

1. 애플리케이션 배포를 위해 새로운 ClusterRole을 생성하고 특정 namespace의 ServiceAccount를 바인드 하시오.
2. 다음의 resource type에서만 Create가 허용된 ClusterRole deployment-clusterrole을 생성합니다.
3. Resource Type: Deployment StatefulSet DaemonSet
4. 미리 생성된 namespace api-access 에 cicd-token이라는 새로운 ServiceAccount를 만듭니다.
5. ClusterRole deployment-clusterrole을 namespace api-access 로 제한된 새 ServiceAccount cicd-token에 바인딩하세요

1. service account 생성

kubectl create serviceaccount cicd-token -n api-access

```
ubuntu@master:~/k8s$ kubectl create serviceaccount cicd-token -n api-access
serviceaccount/cicd-token created
ubuntu@master:~/k8s$ kubectl get serviceaccount -n api-access
NAME          SECRETS   AGE
cicd-token    0         13s
default       0         24m
pod-viewer    0         22m
ubuntu@master:~/k8s$
```

2. cluster role 생성

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

```
ubuntu@master:~/k8s$ kubectl create clusterrole deployment-clusterrole --resource=deployment
,statefulset,daemonset --verb=create
clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
ubuntu@master:~/k8s$ kubectl get clusterrole | grep deployment-clusterrole
deployment-clusterrole                                                 2025-08-08T05:13:58Z
ubuntu@master:~/k8s$
```

3. cluster role binding 생성

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

```
ubuntu@master:~/k8s$ kubectl create clusterrolebinding deployment-clusterrolebinding --clusterrole --serviceac
count=api-access:cicd-token -n api-access
clusterrolebinding.rbac.authorization.k8s.io/deployment-clusterrolebinding created
ubuntu@master:~/k8s$ k get clusterrolebinding | grep deployment
deployment-clusterrolebinding                          ClusterRole/--serviceaccount=api-access:cicd-t
oken                                 24s
system:controller:deployment-controller                ClusterRole/system:controller:deployment-contr
oller                                2d3h
ubuntu@master:~/k8s$
```

05 [노드 비우기]

1. k8s-worker2 노드를 스케줄링 불가능하게 설정
2. 해당 노드에서 실행 중인 모든 Pod을 다른 node로 reschedule

1. worker2 노드 동작 확인

kubectl get nodes

```
ubuntu@master:~/k8s$ kubectl get nodes
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   2d3h   v1.30.14
worker1   Ready    <none>          2d3h   v1.30.14
worker2   Ready    <none>          2d3h   v1.30.14
ubuntu@master:~/k8s$
```

2. Worker2 노드 드레인

kubectl drain worker2 --ignore-daemonsets

```
ubuntu@master:~/k8s$ kubectl drain worker2 --ignore-daemonsets
node/worker2 cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/calico-node-rjfkh, kube-system/kube-pro
xy-g6lcp
node/worker2 drained
ubuntu@master:~/k8s$
ubuntu@master:~/k8s$ kubectl get nodes
NAME      STATUS                     ROLES           AGE    VERSION
master    Ready                      control-plane   2d3h   v1.30.14
worker1   Ready                      <none>          2d3h   v1.30.14
worker2   Ready,SchedulingDisabled   <none>          2d3h   v1.30.14
ubuntu@master:~/k8s$
```

3. Worker2 노드 리스케줄링

kubectl uncordon worker2

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

4. Worker2 노드 상태 확인

kubectl get nodes

```
ubuntu@master:~/k8s$ kubectl get nodes
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   2d3h   v1.30.14
worker1   Ready    <none>          2d3h   v1.30.14
worker2   Ready    <none>          2d3h   v1.30.14
ubuntu@master:~/k8s$
```

06 [PodScheduling]

1. 다음의 조건으로 pod를 생성하세요.
- Name: eshop-store
- Image: nginx
- Nodeselector: disktype=ssd

1. 검색 kubernetes.io/docs에서 nodeselector 검색 후 명령어 찾기

2. Worker1,2 노드에 disktpye=ssd & disktype=hdd 라벨 부여

- kubectl label node worker1 disktype=ssd
- kubectl label node worker2 disktype=hdd

라벨 부여 전

```
ubuntu@master:~/k8s$ k get no --show-labels
NAME      STATUS   ROLES           AGE    VERSION    LABELS
master    Ready    control-plane   2d4h   v1.30.14   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,
kubernetes.io/arch=amd64,kubernetes.io/hostname=master,kubernetes.io/os=linux,node-role.kubernetes.io/control-p
lane=,node.kubernetes.io/exclude-from-external-load-balancers=
worker1   Ready    <none>          2d3h   v1.30.14   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,
kubernetes.io/arch=amd64,kubernetes.io/hostname=worker1,kubernetes.io/os=linux
worker2   Ready    <none>          2d3h   v1.30.14   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,
kubernetes.io/arch=amd64,kubernetes.io/hostname=worker2,kubernetes.io/os=linux
ubuntu@master:~/k8s$
```

3. Worker1,2 노드 라벨 확인

#kubectl get no -L disktype

```
ubuntu@master:~/k8s$ k get no -L disktype
NAME      STATUS   ROLES           AGE    VERSION    DISKTYPE
master    Ready    control-plane   2d4h   v1.30.14
worker1   Ready    <none>          2d3h   v1.30.14   ssd
worker2   Ready    <none>          2d3h   v1.30.14   hdd
ubuntu@master:~/k8s$
```

4. 1. eshop-store.yaml 파일 생성
kubectl run eshop-store - -image=nginx - -dry-run=client -o yaml >eshop-store.yaml

```
ubuntu@master:~/k8s$ kubectl run eshop-store --image=nginx --dry-run=client -o yaml > eshop-store.yaml
ubuntu@master:~/k8s$ vi eshop-store.yaml
```

5. vi 로 eshop-store.yaml 파일 내용 수정(nodeselector 추가)

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: eshop-store
  name: eshop-store
spec:
  containers:
  - image: nginx
    name: eshop-store
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

→

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

6. eshop-store.yaml 파일 적용

```
ubuntu@master:~/k8s$ k apply -f eshop-store.yaml
pod/eshop-store created
ubuntu@master:~/k8s$ k get po -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
eshop-store   1/1     Running   0          11s   172.17.235.177   worker1   <none>           <none>
```

07 [환경변수, command, args 적용]

1. 'cka-exam'이라는 namespace를만들고, 아래와 같은 Pod를 생성
- 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
kubectl create ns cka-exam

```
ubuntu@master:~/k8s$ kubectl create ns cka-exam
namespace/cka-exam created
ubuntu@master:~/k8s$ k get ns
NAME              STATUS   AGE
api-access        Active   79m
cka-exam          Active   5s
default           Active   2d4h
kube-node-lease   Active   2d4h
kube-public       Active   2d4h
kube-system       Active   2d4h
ubuntu@master:~/k8s$
```

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

```
ubuntu@master:~/k8s$ kubectl run pod-01 --image=busybox -n cka-exam --env=CERT="CKA-cert" --dry-run=client -o yaml > pod-01.yaml
ubuntu@master:~/k8s$ vi pod-01.yaml
```

4. pod-01.yaml 파일 수정

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod-01
  name: pod-01
  namespace: cka-exam
spec:
  containers:
  - env:
    - name: CERT
      value: CKA-cert
    image: busybox
    name: pod-01
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

→

```
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"]
~
~
```

5. pod-01.yaml 파일 적용
kubectl apply -f pod-01.yaml pod/pod-01 created

```
ubuntu@master:~/k8s$ kubectl apply -f pod-01.yaml
pod/pod-01 created
ubuntu@master:~/k8s$ kubectl get po -n cka-exam
NAME     READY   STATUS    RESTARTS   AGE
pod-01   1/1     Running   0          99s
ubuntu@master:~/k8s$ kubectl logs pod-01 -n cka-exam
CKA-cert
CKA-cert
CKA-cert
```

08 [Static Pod 생성]

1. worker1 노드에 nginx-static-pod.yaml 라는 이름의 Static Pod를 생성
- pod name: nginx-static-pod
- image: nginx
- port : 80

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

2. ssh로 worker1 접속

3. worker1의 static위치 확인 및 해당 위치로 이동
cd/var/lib/kubelet
cat config.yaml | grep -i static
cd /etc/kubernetes/manifests/

```
ubuntu@worker1:~$ sudo -i
[sudo] password for ubuntu:
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#
```

4. nginx-static-pod.yaml 파일 수정 (worker1에 적용)

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

4-1 vi로 worker1에서 내용 수정

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx-static-pod
  name: nginx-static-pod
spec:
  containers:
  - image: nginx
    name: nginx-static-pod
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

→

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

5. 결과창

```
ubuntu@master:~/k8s$ vi nginx-static-pod.yaml
ubuntu@master:~/k8s$ k get po
NAME                       READY   STATUS    RESTARTS   AGE
eshop-store                1/1     Running   0          49m
nginx-static-pod-worker1   1/1     Running   0          14s
ubuntu@master:~/k8s$
```

09 [로그 확인]

1. Pod "nginx-static-pod-k8s-worker1"의 log를 모니터링하고, 메세지를 포함하는 로그라인을 추출
2. 추출된 결과는 /opt/REPORT/ubuntu/pod-log에 기록

1. Pod 확인

```
ubuntu@master:~/k8s$ k get po
NAME                     READY   STATUS    RESTARTS   AGE
eshop-store              1/1     Running   0          49m
nginx-static-pod-worker1   1/1     Running   0          14s
```

2. 로그 저장

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

```
ubuntu@master:~$ kubectl logs nginx-static-pod-worker1 > /home/ubuntu/pod-log
ubuntu@master:~$ cat pod-log
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configurat
ion
/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.co
nf
/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 06:09:34 [notice] 1#1: using the "epoll" event method
2025/08/12 06:09:34 [notice] 1#1: nginx/1.29.0
2025/08/12 06:09:34 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14+deb12u1)
2025/08/12 06:09:34 [notice] 1#1: OS: Linux 6.8.0-71-generic
2025/08/12 06:09:34 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/08/12 06:09:34 [notice] 1#1: start worker processes
2025/08/12 06:09:34 [notice] 1#1: start worker process 29
2025/08/12 06:09:34 [notice] 1#1: start worker process 30
2025/08/12 06:09:34 [notice] 1#1: start worker process 31
2025/08/12 06:09:34 [notice] 1#1: start worker process 32
```

10 [Multi Container Pod 생성]

1. 1.4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성하시오.
2. pod image: nginx, redis, memcached, consul

1. 야물 생성(eshop-frontend.yaml파일 생성)
- kubectl run eshop-frontend - -image=nginx - -dry-run=client -o yaml > multi-pod.yaml

```
ubuntu@master:~/k8s$ kubectl run eshop-frontend --image=nginx --dry-run=client -o yaml > multi-pod.yaml
ubuntu@master:~/k8s$
```

2. eshop-frontend.yaml 파일 수정
- vi multi-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: eshop-frontend
  name: eshop-frontend
spec:
  containers:
  - image: nginx
    name: eshop-frontend
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

→

```yaml
apiVersion: v1
kind: Pod
metadata:
    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:~/k8s$ k apply -f multi-pod.yaml
pod/eshop-frontend created
ubuntu@master:~/k8s$ k get po
NAME                     READY   STATUS              RESTARTS   AGE
eshop-frontend           0/4     ContainerCreating   0          4s
eshop-store              1/1     Running             0          80m
nginx-static-pod-worker1 0/1     Completed           0          30m
ubuntu@master:~/k8s$
```

11 [Rolling Update & Roll Back]

1. Deployment를 이용해 nginx 파드를 3개 배포한 다음 컨테이너 이미지 버전을 rolling update하고 update record를 기록
2. 마지막으로 컨테이너 이미지를 previous version으로 roll back
- name: eshop-payment
- Image : nginx
- Image version: 1.16
- update image version: 1.17
- label: app=payment, environment=production

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

```
ubuntu@master:~/k8s$ kubectl create deploy eshop-payment --image=nginx:1.16 --replicas=3 --dry-run=client -o ya
ml > eshop-payment.yaml~
ubuntu@master:~/k8s$
```

2. 1. eshop-payment.yaml 파일 수정
- vi eshop-payment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: eshop-payment
  name: eshop-payment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eshop-payment
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: eshop-payment
    spec:
      containers:
      - image: nginx:1.16
        name: nginx
        resources: {}
status: {}
```

→

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: payment
    environment: production
  name: eshop-paymebt
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. 1. eshop-payment.yaml 파일 적용(- -record)
kubectl apply -f eshop-payment.yaml - -record

```
ubuntu@master:~/k8s$ kubectl apply -f eshop-payment.yaml --record
Flag --record has been deprecated, --record will be removed in the future
error: the path "eshop-payment.yaml" does not exist
ubuntu@master:~/k8s$ kubectl get deploy,rs,po
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-payment     3/3     3            3           102s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/eshop-payment-6855fb78d6    3         3         3       102s

NAME                                  READY   STATUS    RESTARTS   AGE
pod/eshop-payment-6855fb78d6-8ch4s    1/1     Running   0          102s
pod/eshop-payment-6855fb78d6-hlpgd    1/1     Running   0          102s
pod/eshop-payment-6855fb78d6-nlbqw    1/1     Running   0          102s
pod/nginx-static-pod-worker1          1/1     Running   0          4s
```

```
ubuntu@master:~/k8s$ kubectl rollout history deploy eshop-payment
deployment.apps/eshop-payment
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=eshop-payment.yaml --record=true
```

4. 롤링 업데이트

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

```
ubuntu@master:~/k8s$ 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
ubuntu@master:~/k8s$ kubectl get deploy,rs,po
NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-payment    3/3     3            3           4m32s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/eshop-payment-6855fb78d6    0         0         0       4m32s
replicaset.apps/eshop-payment-696dd55885    3         3         3       15s

NAME                                  READY   STATUS    RESTARTS   AGE
pod/eshop-payment-696dd55885-7jhbc    1/1     Running   0          14s
pod/eshop-payment-696dd55885-fpc8t    1/1     Running   0          15s
pod/eshop-payment-696dd55885-qh2b8    1/1     Running   0          13s
pod/nginx-static-pod-worker1          1/1     Running   0          2m54s
ubuntu@master:~/k8s$ 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
```

5. 롤백

kubectl rollout undo deploy eshop-payment

```
ubuntu@master:~/k8s$ kubectl rollout undo deploy eshop-payment
deployment.apps/eshop-payment rolled back
ubuntu@master:~/k8s$ 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:~/k8s$ kubectl describe pod eshop-payment-6855fb78d6-bjjjw | grep -i image
    Image:          nginx:1.16
    Image ID:       docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336
f0b70b5eeb69fcaaf30dd9c
  Normal  Pulled     2m5s  kubelet            Container image "nginx:1.16" already present o
n machine
```

12 [ClusterIP]

1. 'devops' namespace에서 deployment eshop-order를 다음 조건으로 생성하시오.
- image: nginx, replicas: 2, label: name=order
2. 'eshop-order' deployment의 Service를 만드세요.
- Service Name: eshop-order-svc
- Type: ClusterIP
- Port: 80

1. devops 네임스페이스 생성
kubectl create ns devops
kubectl get namespace

```
ubuntu@master:~/k8s$ kubectl create namespace devops
namespace/devops created
ubuntu@master:~/k8s$ kubectl get namespace
NAME              STATUS   AGE
api-access        Active   3h13m
cka-exam          Active   114m
default           Active   2d6h
devops            Active   8s
kube-node-lease   Active   2d6h
kube-public       Active   2d6h
kube-system       Active   2d6h
```

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

```
ubuntu@master:~/k8s$ kubectl create deploy eshop-order -n devops --image=nginx --replicas=2
--dry-run=client -o yaml > eshop-order.yaml
```

3. YAML 수정

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: eshop-order
  name: eshop-order
  namespace: devops
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eshop-order
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: eshop-order
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

→

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

4. eshop-order.yaml 파일 적용
kubectl apply -f eshop-order.yaml

```
ubuntu@master:~/k8s$ kubectl apply -f eshop-order.yaml
deployment.apps/eshop-order created
ubuntu@master:~/k8s$ kubectl get deploy -n devops
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
eshop-order   2/2     2            2           13s
```

4. 서비스 생성
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,rs,po -n devops
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-order   2/2     2            2           41s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/eshop-order-d6d584684    2         2         2       41s

NAME                                 READY   STATUS    RESTARTS   AGE
pod/eshop-order-d6d584684-h8dsj      1/1     Running   0          41s
pod/eshop-order-d6d584684-vh5dd      1/1     Running   0          41s
ubuntu@master:~/k8s$ kubectl expose deploy eshop-order -n devops --name=eshop-order-svc --po
rt=80 --target-port=80
service/eshop-order-svc exposed
ubuntu@master:~/k8s$ kubectl get deploy,svc -n devops
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eshop-order   2/2     2            2           3m56s

NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/eshop-order-svc   ClusterIP   10.110.199.210   <none>        80/TCP    20s
```

13 [NodePort]

1. 'front-end' deployment를 다음 조건으로 생성하시오.
 - image: nginx, replicas: 2, label: run=nginx
2. 'front-end' deployment의 nginx 컨테이너를 expose하는 'front-end-nodesvc'라는 새 service를 만듭니다.
3. Front-end로 동작중인 Pod에는 node의 30200 포트로 접속되어야 합니다.

1. front-end.yaml 파일 생성
kubectl create deploy front-end - -image=nginx - -replicas=2 - -dry-run=client -o yaml > front-end.yaml

```
ubuntu@master:~/k8s$ kubectl create deploy front-end --image=nginx --replicas=2 --dry-run=client -o yaml >
front-end.yaml
ubuntu@master:~/k8s$
```

2. front-end.yaml 파일 수정
- vi front-end.yaml
- run=nginx → run: nginx

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: front-end
  name: front-end
spec:
  replicas: 2
  selector:
    matchLabels:
      app: front-end
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: front-end
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

→

```
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
kubectl apply -f front-end.yaml
kubectl get deploy,rs,po

```
ubuntu@master:~/k8s$ vi front-end.yaml
ubuntu@master:~/k8s$ kubectl apply -f front-end.yaml
deployment.apps/front-end created
ubuntu@master:~/k8s$ kubectl get deploy,rs,po
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/front-end     2/2     2            2           10s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/front-end-7858f98fb4     2         2         2       10s

NAME                                 READY   STATUS    RESTARTS   AGE
pod/front-end-7858f98fb4-k8s95       1/1     Running   0          10s
pod/front-end-7858f98fb4-vthxc       1/1     Running   0          10s
ubuntu@master:~/k8s$
```

4. front-end-nodesvc.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

```
ubuntu@master:~/k8s$ 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
ubuntu@master:~/k8s$
```

5.front-end-nodesvc.yaml 파일수정
vi front-end-nodesvc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: front-end-nodesvc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: nginx
  type: NodePort
status:
  loadBalancer: {}
```

→

```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
kubectl apply -f front-end-nodesvc.yaml

```
ubuntu@master:~/k8s$ kubectl apply -f front-end-nodesvc.yaml
service/front-end-nodesvc created
ubuntu@master:~/k8s$ kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
front-end-nodesvc   NodePort    10.110.167.65   <none>        80:30200/TCP   10s
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP        16m
ubuntu@master:~/k8s$
```

14 [Network Policy]

1. customera, customerb를 생성한 후, 각각 PARTITION=customera, PARTITION=customerb를 라벨링 하시오.
2. default namespace에 다음과 같은 pod를 생성하세요.
 - name: poc
 - image: nginx
 - port: 80
 - label: app=poc
 - "partition=customera"를 사용하는 namespace에서만 poc의
3. 80포트로 연결할 수 있도록 default namespace에 'allow-web-from-customera'라는 network Policy를 설정하세요.
보안 정책상 다른 namespace의 접근은 제한합니다.

1. customera, customerb 네임스페이스 생성 및 확인
kubectl create ns customera
kubectl create ns customerb
kubectl get ns customera customerb

```
ubuntu@master:~/k8s$ kubectl create ns customera
namespace/customera created
ubuntu@master:~/k8s$ kubectl create ns customerb
namespace/customerb created
ubuntu@master:~/k8s$ kubectl get ns customera customerb
NAME         STATUS   AGE
customera    Active   33s
customerb    Active   28s
ubuntu@master:~/k8s$
```

2. customera, customerb 네임스페이스 라벨링
kubectl label ns customera PARTITION=customera
kubectl label ns customerb PARTITION=customerb
kubectl get ns —show-labels

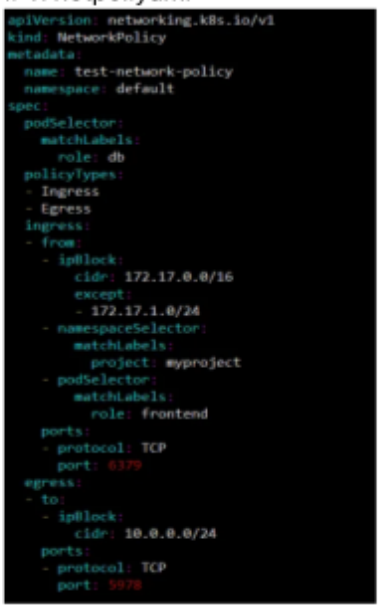
```
ubuntu@master:~/k8s$ kubectl label ns customera PARTITION=customera
namespace/customera labeled
ubuntu@master:~/k8s$ kubectl label ns customerb PARTITION=customerb
namespace/customerb labeled
ubuntu@master:~/k8s$ kubectl get ns --show-labels
NAME              STATUS   AGE     LABELS
customera         Active   2m52s   PARTITION=customera,kubernetes.io/metadata.name=customera
customerb         Active   2m47s   PARTITION=customerb,kubernetes.io/metadata.name=customerb
default           Active   5d1h    kubernetes.io/metadata.name=default
kube-node-lease   Active   5d1h    kubernetes.io/metadata.name=kube-node-lease
kube-public       Active   5d1h    kubernetes.io/metadata.name=kube-public
kube-system       Active   5d1h    kubernetes.io/metadata.name=kube-system
ubuntu@master:~/k8s$
```

3. 파드 생성
kubectl run poc - -image=nginx - -port=80 - -labels=app=poc
kubectl get po poc - -show-labels

```
ubuntu@master:~/k8s$ kubectl run poc --image=nginx --port=80 --labels=app=poc
pod/poc created
ubuntu@master:~/k8s$ kubectl get po poc --show-labels
NAME   READY   STATUS    RESTARTS   AGE   LABELS
poc    1/1     Running   0          11s   app=poc
ubuntu@master:~/k8s$
```

4. netpol.yaml 파일 생성
검색 : networkpolicy(쿠버네티스 사이트)

5. netpol.yaml 파일 수정
vi netpol.yaml



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

→

```
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
```

6. netpol.yaml 파일 적용
kubectl apply -f netpol.yaml
kubectl get netpo

```
ubuntu@master:~/k8s$ kubectl apply -f netpol.yaml
networkpolicy.networking.k8s.io/allow-web-from-customera created
ubuntu@master:~/k8s$ kubectl get netpol
NAME                       POD-SELECTOR   AGE
allow-web-from-customera   app=poc        8s
ubuntu@master:~/k8s$
```

15 [Ingress]

1. Create a new nginx Ingress resource as follows:
 - Name: ping
 - Namespace: ing-internal
 - Exposing service hi on path /hi using service port 5678

1. Ing-internal 네임스페이스 생성 및 확인
kubectl create ns ing-internal
kubectl get ns

```
ubuntu@master:~/k8s$ kubectl create ns ing-internal
namespace/ing-internal created
ubuntu@master:~/k8s$ kubectl get ns
NAME              STATUS   AGE
customera         Active   83m
customerb         Active   83m
default           Active   5d3h
ing-internal      Active   10s
kube-node-lease   Active   5d3h
kube-public       Active   5d3h
kube-system       Active   5d3h
ubuntu@master:~/k8s$
```

2. ingress.yaml 파일 생성
검색 : ingress(쿠버네티스 사이트)
3. ingress.yaml 파일 수정
vi ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

→

```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
```

4. ingress.yaml 파일 적용
kubectl apply-f ingress.yaml
kubectl get ing -n ing-internal

```
ubuntu@master:~/k8s$ kubectl apply -f ingress.yaml
ingress.networking.k8s.io/ping created
ubuntu@master:~/k8s$ kubectl get ing -n ing-internal
NAME   CLASS           HOSTS   ADDRESS   PORTS   AGE
ping   nginx-example   *                 80      37s
ubuntu@master:~/k8s$
```

16 [Service and DNS Lookup]

1. image nginx를 사용하는 resolver pod를 생성하고 resolver-service라는 service를 구성

2. 클러스터 내에서 service와 pod 이름을 조회할 수 있는지 테스트

- dns 조회에 사용하는 pod 이미지는 busybox:1.28이고, service와 pod 이름 조회는 nlsookup을 사용

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

1. resolver pod 생성 및 확인

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

```
ubuntu@master:~/k8s$ kubectl run resolver --image=nginx --port=80
pod/resolver created
ubuntu@master:~/k8s$ kubectl get po resolver
NAME       READY   STATUS    RESTARTS   AGE
resolver   1/1     Running   0          9s
ubuntu@master:~/k8s$
```

2. 서비스 생성 및 확인

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

```
ubuntu@master:~/k8s$ kubectl expose pod resolver --name=resolver-service --port=80
service/resolver-service exposed
ubuntu@master:~/k8s$ kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
front-end-nodesvc   NodePort    10.110.167.65   <none>        80:30200/TCP   108m
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP        125m
resolver-service    ClusterIP   10.102.45.25    <none>        80/TCP         9s
ubuntu@master:~/k8s$
```

3. 서비스 DNS 조회 및 저장
kubectl run testpod - -image=busybox:1.28 -it - -restart=Never - -rm - -nslookup 10.102.45.25

```
ubuntu@master:~/k8s$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10.102.4
5.25
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10.102.45.25
Address 1: 10.102.45.25 resolver-service.default.svc.cluster.local
pod "testpod" deleted
```

mkdir cka
kubectl run testpod - -image=busybox:1.28 -it - -restart=Never - -rm - -nslookup 10.102.45.25 > nginx.svc
ls

```
ubuntu@master:~/k8s$ mkdir cka
ubuntu@master:~/k8s$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10.102.4
5.25 > /home/ubuntu/cka/nginx.svc
-bash: /home/ubuntu/cka/nginx.svc: No such file or directory
ubuntu@master:~/k8s$ ls
annotation.yaml        development-label01.yaml  front-end-nodesvc.yaml  loadbalancer.yaml  service.yaml
cka                    ex.yaml                   front-end.yaml          netpol.yaml
deployment-nginx.yaml  front                     ingress.yaml            nodeport.yaml
ubuntu@master:~/k8s$
```

cat nginx.svc

```
ubuntu@master:~/k8s$ cat nginx.svc
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10.102.45.25
Address 1: 10.102.45.25 resolver-service.default.svc.cluster.local
pod "testpod" deleted
ubuntu@master:~/k8s$
```

4. 파드 DNS 조회 및 저장
kubectl run testpod - -image=busybox:1.28 -it - -restart=Never - -rm - -nslookup 10-101-78-250.default.pod.cluster.local
kubectl run testpod - -image=busybox:1.28 -it - -restart=Never - -rm - -nslookup 10-101-78-250.default.pod.cluster.local > nginx.pod
cat nginx.pod

```
ubuntu@master:~/k8s$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10-101-78-250.default.pod.cluster.local
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10-101-78-250.default.pod.cluster.local
Address 1: 10.101.78.250
pod "testpod" deleted
ubuntu@master:~/k8s$ kubectl run testpod --image=busybox:1.28 -it --restart=Never --rm -- nslookup 10-101-78-250.default.pod.cluster.local > nginx.pod
ubuntu@master:~/k8s$ cat nginx.pod
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10-101-78-250.default.pod.cluster.local
Address 1: 10.101.78.250
pod "testpod" deleted
```

17 [emptyDir Volume]

1. 다음 조건에 맞춰서 nginx 웹서버 pod가 생성한 로그파일을 받아서 STDOUT으로 출력하는 busybox 컨테이너를 운영
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. weblog.yaml 생성
kubectl run weblog --image=nginx:1.17 --dry-run=client -o yaml > weblog.yaml

2. weblog.yaml 수정
vi weblog.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: weblog
  name: weblog
spec:
  containers:
  - image: nginx:1.17
    name: weblog
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

→

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: weblog
spec:
  containers:
  - image: nginx:1.17
    name: web
    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 파일 적용
kubectl apply -f weblog.yaml

```
ubuntu@master:~/k8s$ kubectl apply -f weblog.yaml
pod/weblog created
ubuntu@master:~/k8s$ kubectl get pod weblog
NAME     READY   STATUS    RESTARTS   AGE
weblog   2/2     Running   0          7s
Containers:
  weblog:
    Container ID:   containerd://7f9c0ab4bea18d1762592 5f576a82c58d320f0f78de49b5c8e4cb1d2c177df6f
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 11 Aug 2025 06:07:55 +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-5rlc5 (ro)
  log:
    Container ID:  containerd://1c0e459c4ad3bbaf1b49c22b54a5a20b08585b78d9c4240ddf0a4efce4ae908e
    Image:         nginx:1.17
    Image ID:      docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      tail -n+1 -f /data/access.log
    State:          Running
      Started:      Mon, 11 Aug 2025 06:07:55 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /data from weblog (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5rlc5 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
```

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

1. 검색 : hostpath 검색((쿠버네티스 사이트)
2. cka 폴더 생성 및 이동
mkdir cka
cd cka

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

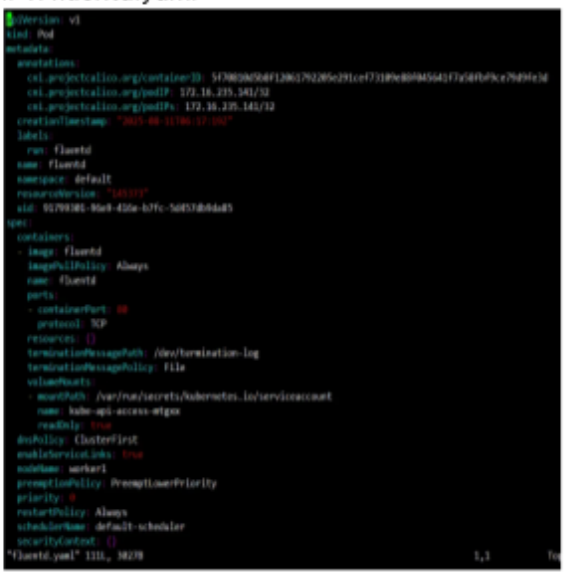
3. fluentd pod 파드 생성
kubectl run fluentd - -image=fluentd - -port=80

```
ubuntu@master:~/k8s$ kubectl run fluentd --image=fluentd --port=80
pod/fluentd created
ubuntu@master:~/k8s$
```

4. fluentd yaml파일 생성
kubectl get po fluentd -o yaml > fluentd.yaml

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

5. fluentd yaml 파일 수정
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
~
~
```

5. fluentd yaml 파일 적용, 파드 삭제
kubectl delete po fluentd --force

```
ubuntu@master:~/cka$ kubectl delete po fluentd --force
Warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. T
he resource may continue to run on the cluster indefinitely.
pod "fluentd" force deleted
```

6.fluentd yaml 파일 적용
kubectl apply -f fluentd.yaml
kubectl describe pod fluentd

```
ubuntu@master:~/cka$ kubectl apply -f fluentd.yaml
pod/fluentd created
ubuntu@master:~/cka$ kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
fluentd   1/1     Running   0          6s
weblog    2/2     Running   0          37m
```

19 [Persistent Volume]

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

생성합니다.

2. volume type은 hostPath이고 위치는 /tmp/app-config입니다.

1. hostpath 검색
kind : Persistent

2. pv001.yaml 파일 복사 붙여넣기
vi pv001.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
~
```

3. hostpath 검색 후 복사
hostPath:
  pasth: /any/path/it/will/be/replaced

4.pv001.yaml 파일 수정

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

5. pv001.yaml 파일 적용
kubectl apply -f pv001.yaml
kubectl get pv

```
ubuntu@master:~$ kubectl apply -f pv001.yaml
persistentvolume/pv001 created
ubuntu@master:~$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   VOLUMEATTRIBUTESCLASS
   REASON   AGE
pv001   1Gi        RWX            Retain           Available                          <unset>
          9s
ubuntu@master:~$
```

대우직업능력개발원
Deawoo Development Institute for Vocational ability

감사합니다

APEX