대우능력개발원
DAEWOO DEVELOPMENT OF ABILITY

쿠버네티스 프로젝트

BSFAN | 김태경
박종승
윤재영
김효은

[문제 1] ETCD 백업

1. https://127.0.0.1:2379에서 실행 중인 etcd의 snapshot을 생성하고snapshot을 /data/etcd-snapshot.db에 저장 후 복원
2. etcdctl을 사용하여 서버에 연결하기 위해 다음 TLS 인증서/키가 제공
   - ✓ CA certificate: /etc/kubernetes/pki/etcd/ca.crt

1) Docs.io에서 etcd 백업 검색 후 사용할 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

```
# sudo -i
```

3) etcd-client 설치

```
# apt install etcd-client
```

4) /data 디렉터리 생성

```
# mkdir /data
```

5) 스냅샷 생성 후 저장

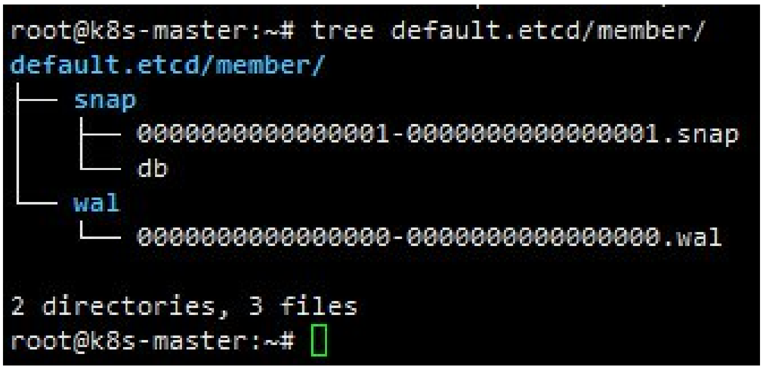
```
#ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /data/etcd-snapshot.db
```

```
mkdir /data
```

6) 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 /data/etcd-snapshot.db
```

7) 스냅샷 복원 여부 확인



```
root@k8s-master:~# tree default.etcd/member/
default.etcd/member/
├── snap
│   ├── 0000000000000001-0000000000000001.snap
│   └── db
└── wal
    └── 0000000000000000-0000000000000000.wal

2 directories, 3 files
root@k8s-master:~# 
```

[문제 2] Cluster Upgrade

1. 마스터 노드의 모든 구성 요소를 버전 1.29.6-1.1 버전으로 업그레이드

2. master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon

3. "주의사항" 반드시 Master Node에서 root권한을 가지고 작업을 실행

1) root 계정으로 전환

```
# sudo -i
```

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 k8s-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

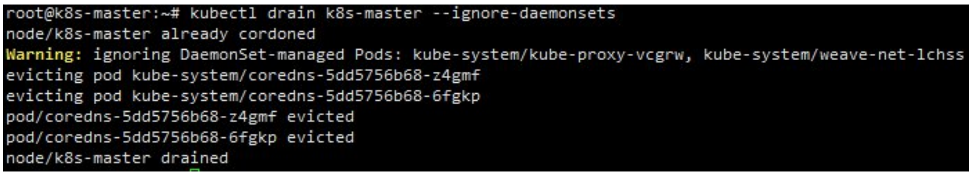
7) kubelet 다시 시작

# sudo systemctl daemon-reload

# sudo systemctl restart kubelet

8) 노드 차단 해제

# kubectl uncordon k8s-master



```
root@k8s-master:~# kubectl drain k8s-master --ignore-daemonsets
node/k8s-master already cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kube-proxy-vcgrw, kube-system/weave-net-lchss
evicting pod kube-system/coredns-5dd5756b68-z4gmf
evicting pod kube-system/coredns-5dd5756b68-6fgkp
pod/coredns-5dd5756b68-z4gmf evicted
pod/coredns-5dd5756b68-6fgkp evicted
node/k8s-master drained
```

[문제 3] Service Account & Role & RoleBinding 생성

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) api-access라는 새로운 namespace 생성

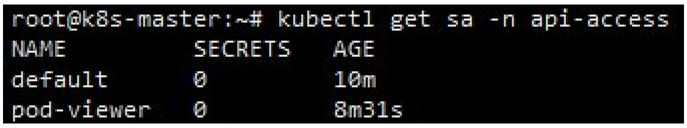
# kubectl create ns api-access

2) pod-viewer라는 이름의 Service Account 생성

# kubectl create sa pod-viewer -n api-access

3) Service Account 생성 확인

# kubectl get sa -n api-access



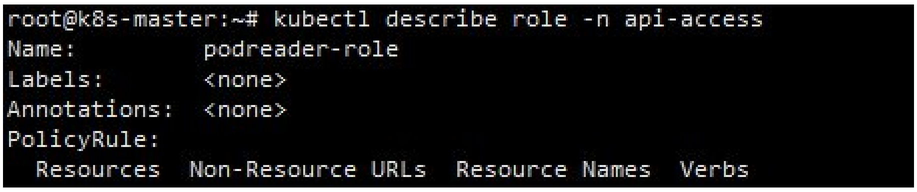
```
root@k8s-master:~# kubectl get sa -n api-access
NAME          SECRETS   AGE
default       0         10m
pod-viewer    0         8m31s
```

4) watch,list,get 허용하도록 Podreader-role라는 이름의 Role생성

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

5) Role 생성 확인

kubectl describe role -n api-access



```
root@k8s-master:~# kubectl describe role -n api-access
Name:         podreader-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
```

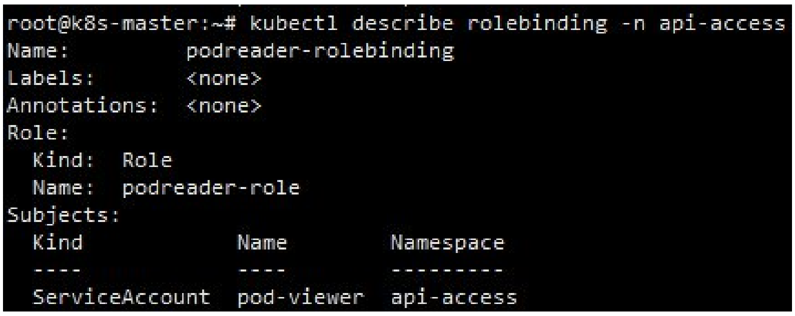
6) podreader-rolebinding라는 이름의 RoleBinding 생성

# kubectl create rolebinding podreader-rolebinding --role=podreader-role

--serviceaccount=api-access:pod-viewer -n api-access

7) RoleBinding 생성 확인

kubectl describe rolebinding -n api-access



```
root@k8s-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-access
```

[문제 4] Service Account & ClusterRole & ClusterRoleBinding 생성

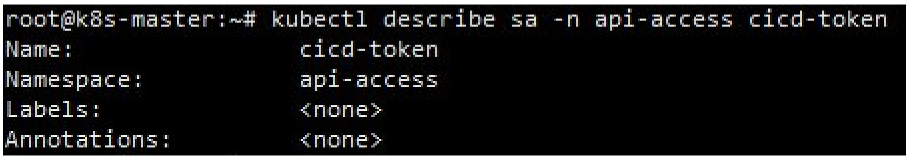
1. resource type에서만 Create가 허용된 ClusterRole deployment-clusterrole을 생성
    - ✓ Resource Type: Deployment StatefulSet DaemonSet
2. 미리 생성된 namespace api-access 에 cicd-token이라는 새로운 ServiceAccount를 생성
3. ClusterRole deployment-clusterrole을 namespace api-access로 제한된 cicd-token에 바인딩

1) cicd-tocken이라는 새로운 ServiceAccount 생성

# kubectl create sa cicd-token -n api-access

2) ServiceAccount 생성 확인

kubectl describe sa -n api-access cicd-token



```
root@k8s-master:~# kubectl describe sa -n api-access cicd-token
Name:                cicd-token
Namespace:           api-access
Labels:              <none>
Annotations:         <none>
```

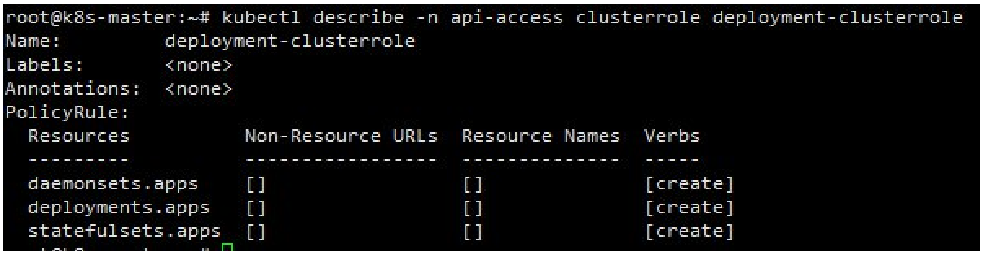
3) create가 허용되고 리소스 타입이 들어간 deployment-clusterrole라는 이름의 clusterRole생성

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

--verb=create

4) ClusterRole 생성 확인

kubectl describe -n api-acces clusterrole deployment-clusterrole



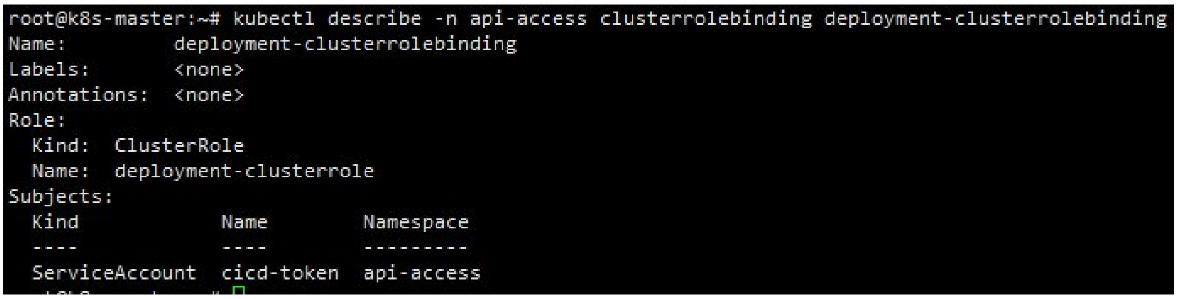
```
root@k8s-master:~# kubectl describe -n api-access clusterrole deployment-clusterrole
Name:         deployment-clusterrole
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources          Non-Resource URLs  Resource Names  Verbs
  ---------          -----------------  --------------  -----
  daemonsets.apps    []                 []              [create]
  deployments.apps   []                 []              [create]
  statefulsets.apps  []                 []              [create]
```

5) 생성한 clusterrole을 새 serviceaccount에 바인딩하도록 ClusterRoleBinding 생성

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

6) ClusterRoleBinding 확인

kubectl describe -n api-access clusterrolebinding deployment-clusterrolebinding



```
root@k8s-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  cicd-token  api-access
```

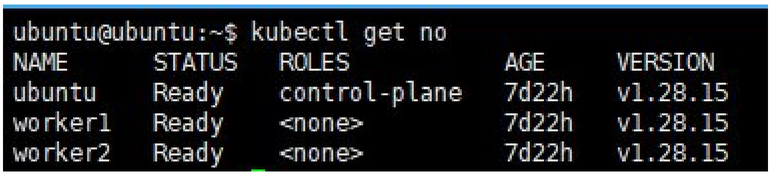
[문제 5] 노드 관리 - 노드 비우기

1. k8s-worker2 노드를 스케줄링 불가능하게 설정

2. 해당 노드에서 실행 중인 모든 Pod을 다른 node로 reschedule

1) 설정 전 worker2 노드 확인

# kubectl get no



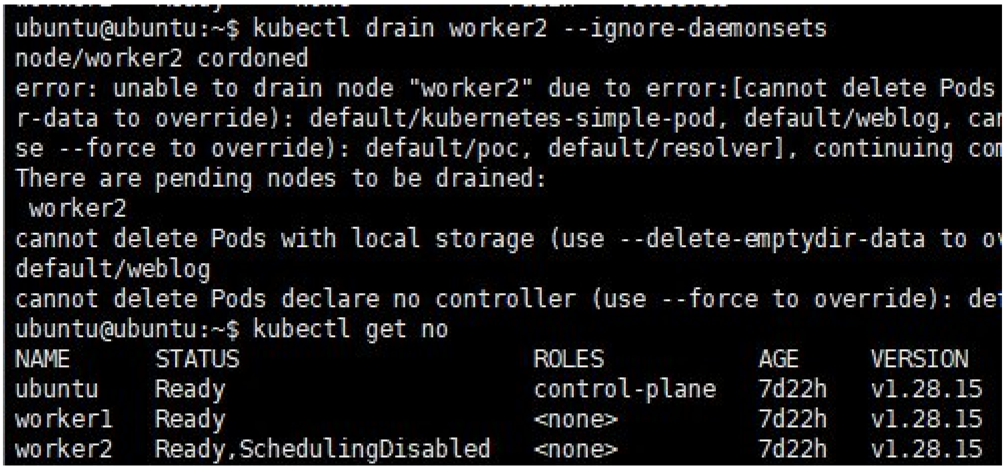
```
ubuntu@ubuntu:~$ kubectl get no
NAME      STATUS   ROLES           AGE     VERSION
ubuntu    Ready    control-plane   7d22h   v1.28.15
worker1   Ready    <none>          7d22h   v1.28.15
worker2   Ready    <none>          7d22h   v1.28.15
```

2) Worker2에서 실행 중인 모든 pod를 다른 node로 reschedule 설정

# kubectl drain worker2

3) Worker2 노드 확인

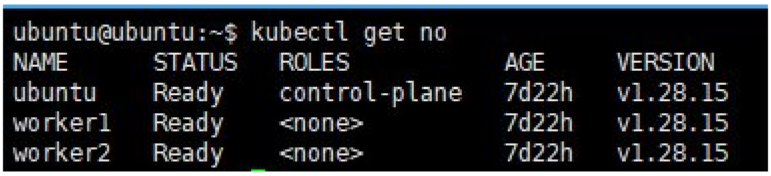
# kubectl drain worker --ignore-daemonsets



```
ubuntu@ubuntu:~$ kubectl drain worker2 --ignore-daemonsets
node/worker2 cordoned
error: unable to drain node "worker2" due to error:[cannot delete Pods
r-data to override): default/kubernetes-simple-pod, default/weblog, car
se --force to override): default/poc, default/resolver], continuing com
There are pending nodes to be drained:
 worker2
cannot delete Pods with local storage (use --delete-emptydir-data to ov
default/weblog
cannot delete Pods declare no controller (use --force to override): def
ubuntu@ubuntu:~$ kubectl get no
NAME      STATUS                     ROLES           AGE     VERSION
ubuntu    Ready                      control-plane   7d22h   v1.28.15
worker1   Ready                      <none>          7d22h   v1.28.15
worker2   Ready,SchedulingDisabled   <none>          7d22h   v1.28.15
```

4) Worker2 drain 설정 해제 및 노드 확인

# kubectl uncordon worker2



```
ubuntu@ubuntu:~$ kubectl get no
NAME      STATUS   ROLES           AGE     VERSION
ubuntu    Ready    control-plane   7d22h   v1.28.15
worker1   Ready    <none>          7d22h   v1.28.15
worker2   Ready    <none>          7d22h   v1.28.15
```

[문제 6] 노드 관리 - Pod Scheduling

1. 다음의 조건으로 pod를 생성

✓ Name: eshop-store

✓ Image: nginx

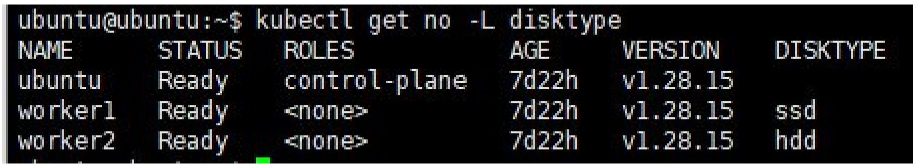
1) Worker1,2 노드에 각각 disktpye=ssd & disktype=hdd 라벨링 추가

   # kubectl label node worker1 disktype=ssd

   # kubectl label node worker2 disktype=hdd

2) 라벨링 추가 여부 확인

   # kubectl get no -L disktype

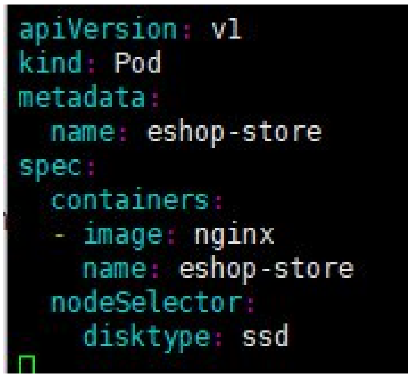


```
ubuntu@ubuntu:~$ kubectl get no -L disktype
NAME      STATUS   ROLES           AGE     VERSION    DISKTYPE
ubuntu    Ready    control-plane   7d22h   v1.28.15
worker1   Ready    <none>          7d22h   v1.28.15   ssd
worker2   Ready    <none>          7d22h   v1.28.15   hdd
```

3) eshop-store 파일 생성

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

4) eshop-store.yaml파일 수정



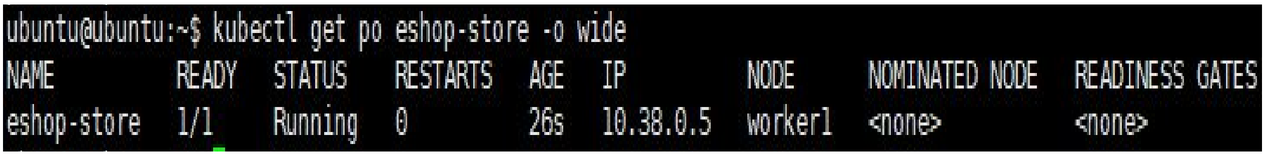
```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-store
spec:
  containers:
  - image: nginx
    name: eshop-store
  nodeSelector:
    disktype: ssd
```

5) eshop-store.yaml파일 적용

   # kubectl apply -f eshop-store.yaml

6) pod생성 확인

   # kubectl get po eshop-store -o wide



```
ubuntu@ubuntu:~$ kubectl get po eshop-store -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP          NODE      NOMINATED NODE   READINESS GATES
eshop-store   1/1     Running   0          26s   10.38.0.5   worker1   <none>           <none>
```

[문제 7] 파드 생성

1. 'cka-exam'이라는 namespace를 만들고 아래와 같은 Pod를 생성

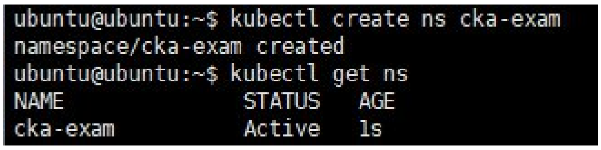
- ✓ pod Name: pod-01
- ✓ image: busybox
- ✓ 환경변수 : CERT = "CKA-cert"
- ✓ command: /bin/sh
- ✓ args: "-c", "while true; do echo $(CERT); sleep 10;done“

1) ‘cka-exam’ namespace 생성

# kubectl create ns cka-exam

2) namespace 생성 여부 확인

# kubectl create ns cka-exam

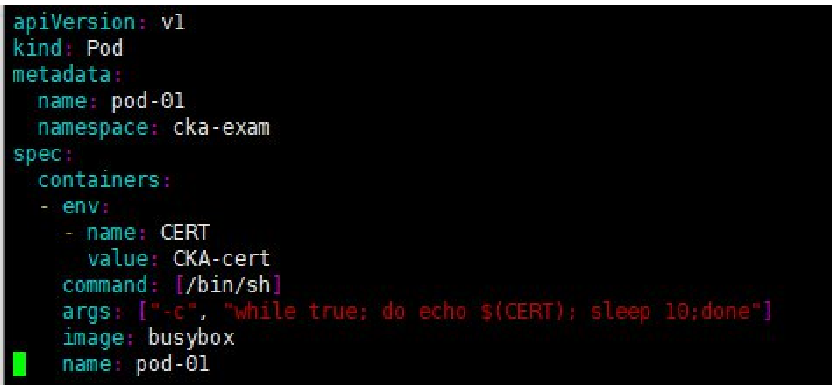


```
ubuntu@ubuntu:~$ kubectl create ns cka-exam
namespace/cka-exam created
ubuntu@ubuntu:~$ kubectl get ns
NAME                 STATUS   AGE
cka-exam             Active   1s
```

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

4) pod-01.yaml파일 수정



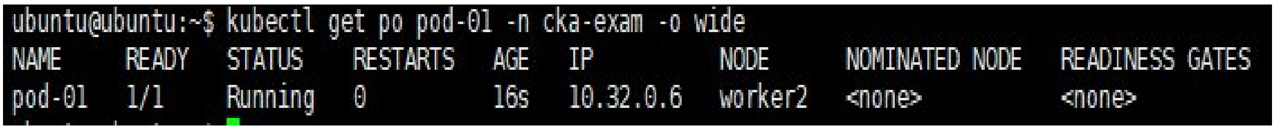
```
apiVersion: v1
kind: Pod
metadata:
  name: pod-01
  namespace: cka-exam
spec:
  containers:
  - env:
    - name: CERT
      value: CKA-cert
    command: [/bin/sh]
    args: ["-c", "while true; do echo $(CERT); sleep 10;done"]
    image: busybox
    name: pod-01
```

5) Pod-01.yaml파일 적용

# kubectl apply -f pod-01.yaml

6) pod생성 확인

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



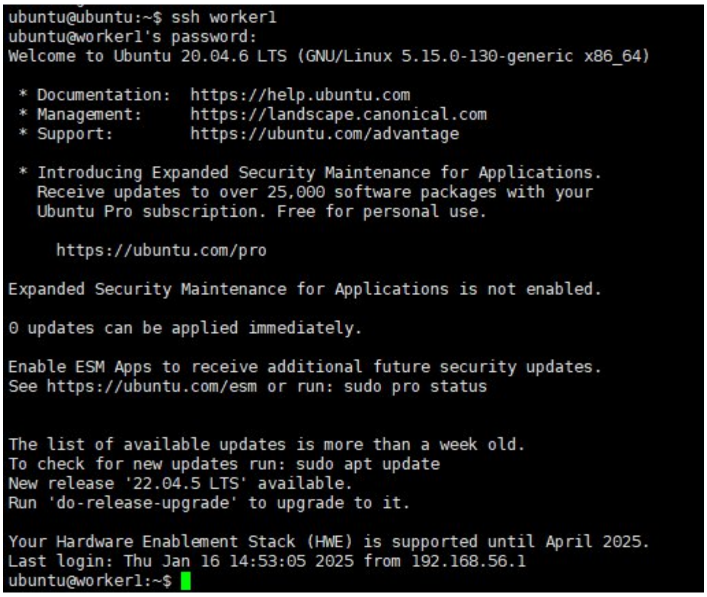
```
ubuntu@ubuntu:~$ kubectl get po pod-01 -n cka-exam -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP          NODE      NOMINATED NODE   READINESS GATES
pod-01   1/1     Running   0          16s   10.32.0.6   worker2   <none>           <none>
```

[문제 8] 파드 생성 - Static Pod 생성

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

1) ssh로 worker1 접속 및 확인

# ssh worker1



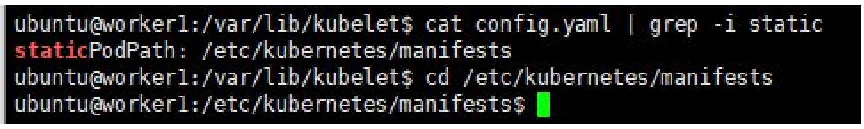
```
ubuntu@ubuntu:~$ ssh worker1
ubuntu@worker1's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-130-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Thu Jan 16 14:53:05 2025 from 192.168.56.1
ubuntu@worker1:~$
```

2) worker1의 static위치 확인 및 해당 위치로 이동

# cd /var/lib/kubelet

# cat config.yaml | grep -i static

# cd /etc/kubernetes/manifests/

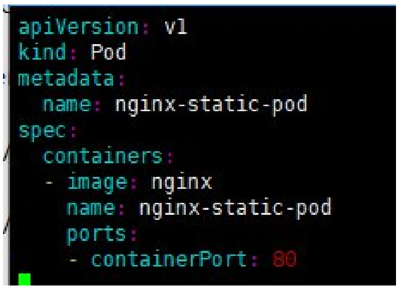


```
ubuntu@worker1:/var/lib/kubelet$ cat config.yaml | grep -i static
staticPodPath: /etc/kubernetes/manifests
ubuntu@worker1:/var/lib/kubelet$ cd /etc/kubernetes/manifests
ubuntu@worker1:/etc/kubernetes/manifests$
```

3) Static Pod 생성을 위한 nginx-static-pod.yaml파일 생성

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

4) vi nginx-static-pod.yaml파일을 조건에 맞게 수정



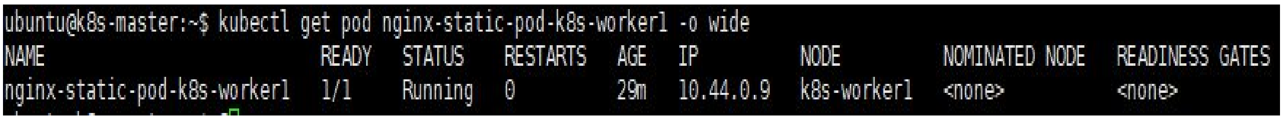
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-pod
spec:
  containers:
  - image: nginx
    name: nginx-static-pod
    ports:
    - containerPort: 80
```

5) nginx-static-pod.yaml파일 적용

# kubectl apply -f nginx-static-pod.yaml

6) pod생성 확인

# kubectl get pod nginx-static-pod-k8s-worker1 -o wide



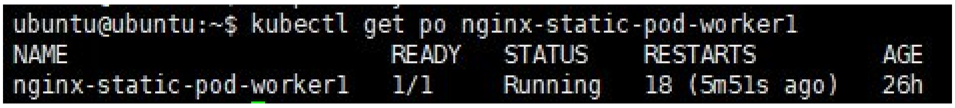
```
ubuntu@k8s-master:~$ kubectl get pod nginx-static-pod-k8s-worker1 -o wide
NAME                           READY   STATUS    RESTARTS   AGE   IP          NODE          NOMINATED NODE   READINESS GATES
nginx-static-pod-k8s-worker1   1/1     Running   0          29m   10.44.0.9   k8s-worker1   <none>           <none>
```

[문제 9]파드 생성 - 로그 확인

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

2. 추출된 결과는 /opt/REPORT/2023/pod-log에 기록

1) nginx-static-pod-worker1의 작동 여부 확인



```
ubuntu@ubuntu:~$ kubectl get po nginx-static-pod-worker1
NAME                       READY   STATUS    RESTARTS        AGE
nginx-static-pod-worker1   1/1     Running   18 (5m51s ago)   26h
```

2) root 계정으로 전환

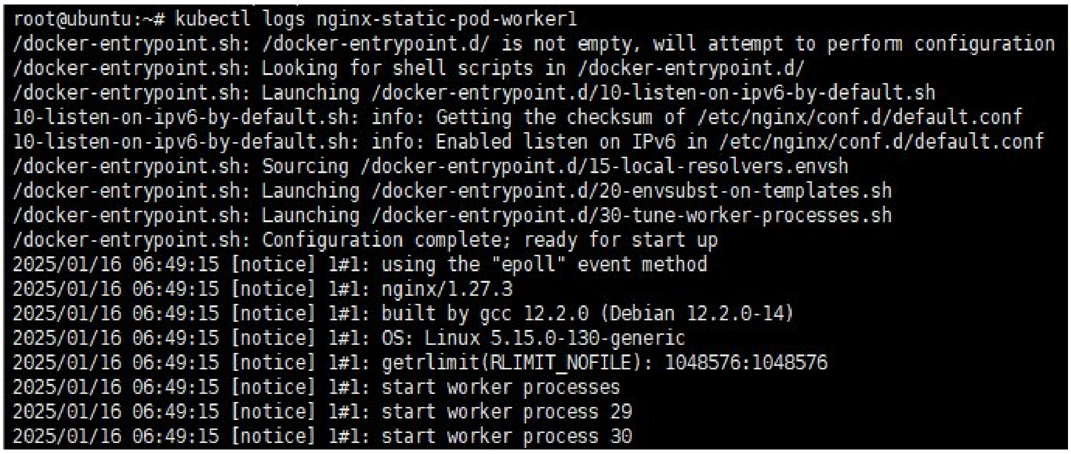
# sudo -i

3) /opt/REPORT/2023 디렉터리 생성

# mkdir -p /opt/REPORT/2023/

4) nginx-static-pod-worker1의 로그를 모니터링

# kubectl logs nginx-static-pod-worker1



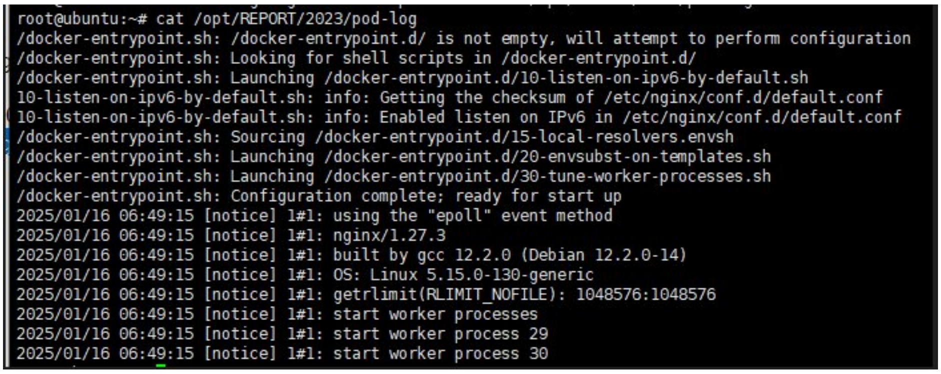
```
root@ubuntu:~# kubectl logs nginx-static-pod-worker1
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
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
2025/01/16 06:49:15 [notice] 1#1: using the "epoll" event method
2025/01/16 06:49:15 [notice] 1#1: nginx/1.27.3
2025/01/16 06:49:15 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/01/16 06:49:15 [notice] 1#1: OS: Linux 5.15.0-130-generic
2025/01/16 06:49:15 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/01/16 06:49:15 [notice] 1#1: start worker processes
2025/01/16 06:49:15 [notice] 1#1: start worker process 29
2025/01/16 06:49:15 [notice] 1#1: start worker process 30
```

5) 추출된 결과 /opt/REPORT/2023/pod-log에 기록

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

6) 기록 여부 확인

cat /opt/ REPORT/2023/pod-log



```
root@ubuntu:~# cat /opt/REPORT/2023/pod-log
/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
2025/01/16 06:49:15 [notice] 1#1: using the "epoll" event method
2025/01/16 06:49:15 [notice] 1#1: nginx/1.27.3
2025/01/16 06:49:15 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/01/16 06:49:15 [notice] 1#1: OS: Linux 5.15.0-130-generic
2025/01/16 06:49:15 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/01/16 06:49:15 [notice] 1#1: start worker processes
2025/01/16 06:49:15 [notice] 1#1: start worker process 29
2025/01/16 06:49:15 [notice] 1#1: start worker process 30
```

[문제10] Multi Container Pod 생성

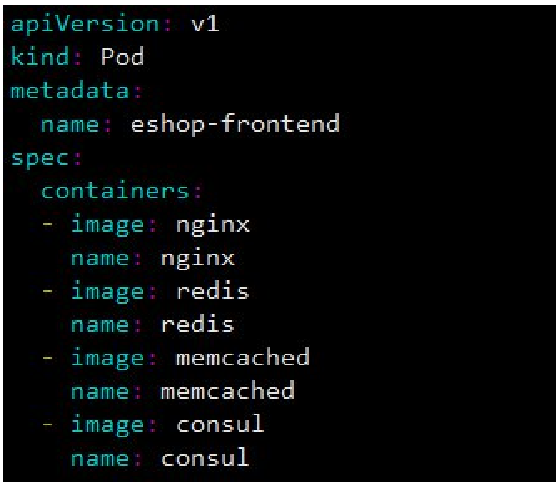
1. 4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성

2. pod image: nginx, redis, memcached, consul

1) eshop-frontend Pod를 생성

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

2) eshop-frontend.yaml 수정 (vi 사용)



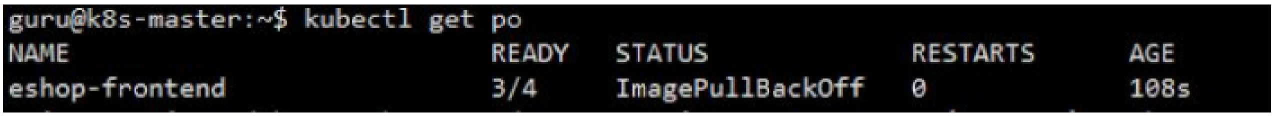
```
apiVersion: v1
kind: Pod
metadata:
  name: eshop-frontend
spec:
  containers:
  - image: nginx
    name: nginx
  - image: redis
    name: redis
  - image: memcached
    name: memcached
  - image: consul
    name: consul
```

3) eshop-frontend.yaml 적용

```
# kubectl apply -f eshop-fronted.yaml
```

4) eshop-frontend Pod 확인

```
# kubectl get po eshop-frontend
```



```
guru@k8s-master:~$ kubectl get po
NAME                                   READY   STATUS             RESTARTS     AGE
eshop-frontend                         3/4     ImagePullBackOff   0            108s
```

[문제11] Rolling Updatae & Rolling 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

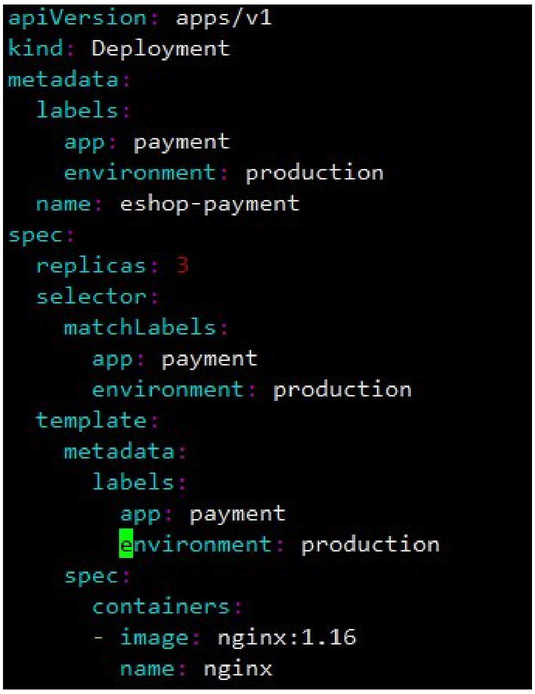
1) eshop-frontend Deployment 생성

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

2) eshop-payment.yaml 수정 (vi 사용)

2-1) label값 수정

2-2) replicas값, image 버전 확인



```
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-frontend.yaml 적용

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

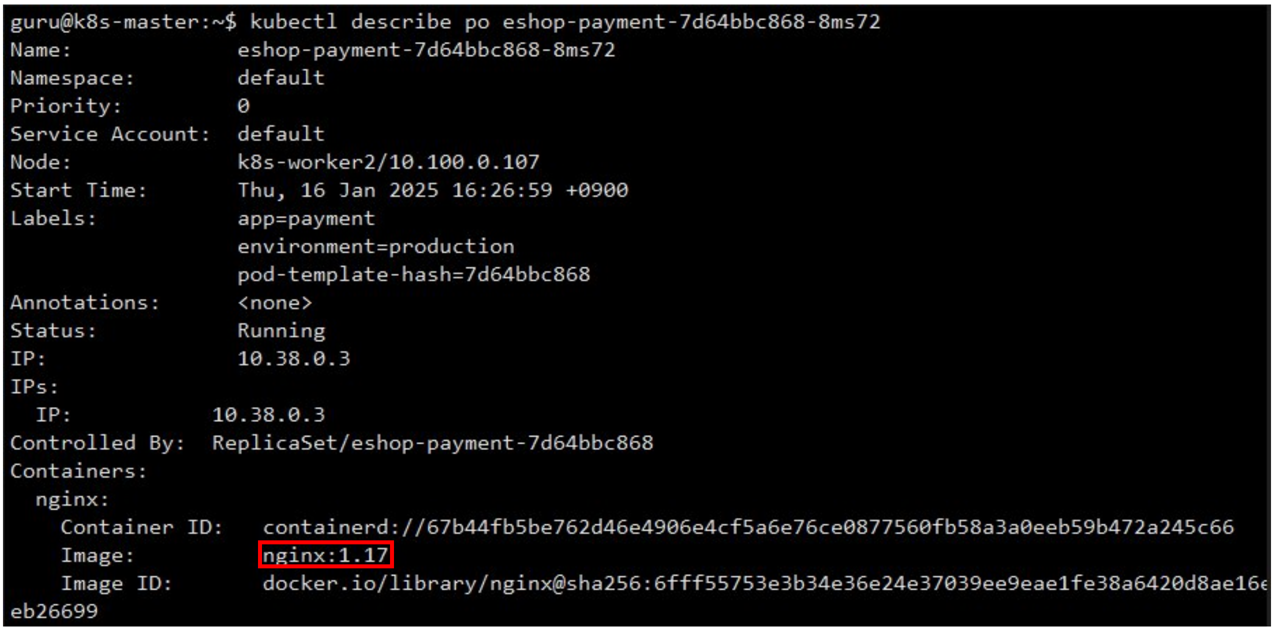
4) image 버전 업데이트

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

5) pod 및 rolling update 확인

```
# kubectl get deploy,po | grep -i eshop-payment
```

# kubectl describe po eshop-payment-7d64bbc868-8ms72

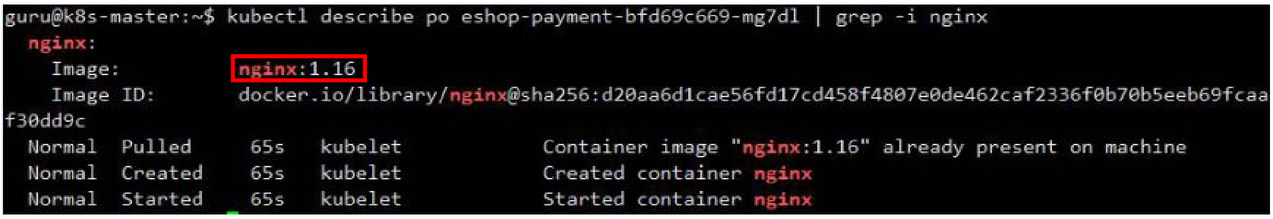


```
guru@k8s-master:~$ kubectl describe po eshop-payment-7d64bbc868-8ms72
Name:             eshop-payment-7d64bbc868-8ms72
Namespace:        default
Priority:         0
Service Account:  default
Node:             k8s-worker2/10.100.0.107
Start Time:       Thu, 16 Jan 2025 16:26:59 +0900
Labels:           app=payment
                  environment=production
                  pod-template-hash=7d64bbc868
Annotations:      <none>
Status:           Running
IP:               10.38.0.3
IPs:
  IP:           10.38.0.3
Controlled By:  ReplicaSet/eshop-payment-7d64bbc868
Containers:
  nginx:
    Container ID:   containerd://67b44fb5be762d46e4906e4cf5a6e76ce0877560fb58a3a0eeb59b472a245c66
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16e
eb26699
```

6) roll back 실행 및 확인

# kubectl rollout undo deploy eshop-payment

# kubectl describe po eshop-payment-7d64bbc868-mg7d1 | grep -i nginx



```
guru@k8s-master:~$ kubectl describe po eshop-payment-bfd69c669-mg7dl | grep -i nginx
  nginx:
    Image:          nginx:1.16
    Image ID:       docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
f30dd9c
  Normal  Pulled     65s   kubelet            Container image "nginx:1.16" already present on machine
  Normal  Created    65s   kubelet            Created container nginx
  Normal  Started    65s   kubelet            Started container nginx
```

[문제12] Multi Container Pod 생성

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

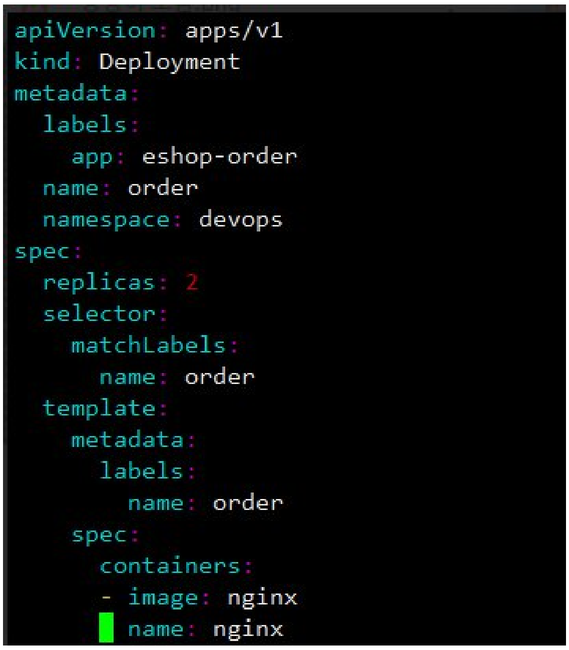
1) 'devops' namespace 생성

# kubectl create ns devops

2) ‘eshop-order’ 디플로이먼트 생성

```
# 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



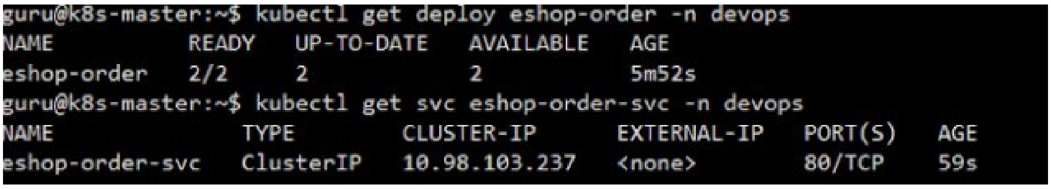
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: eshop-order
  name: order
  namespace: devops
spec:
  replicas: 2
  selector:
    matchLabels:
      name: order
  template:
    metadata:
      labels:
        name: order
    spec:
      containers:
      - image: nginx
        name: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

4) ‘eshop-order’ 디플로이먼트 서비스 생성

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

5) ‘eshop-order’ 디플로이먼트 및 서비스 확인

```
# kubectl get deploy eshop-order -n devops

# kubectl get svc eshop-order-svc -n devops
```



```
guru@k8s-master:~$ kubectl get deploy eshop-order -n devops
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
eshop-order   2/2     2            2           5m52s
guru@k8s-master:~$ kubectl get svc eshop-order-svc -n devops
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
eshop-order-svc   ClusterIP   10.98.103.237   <none>        80/TCP    59s
```

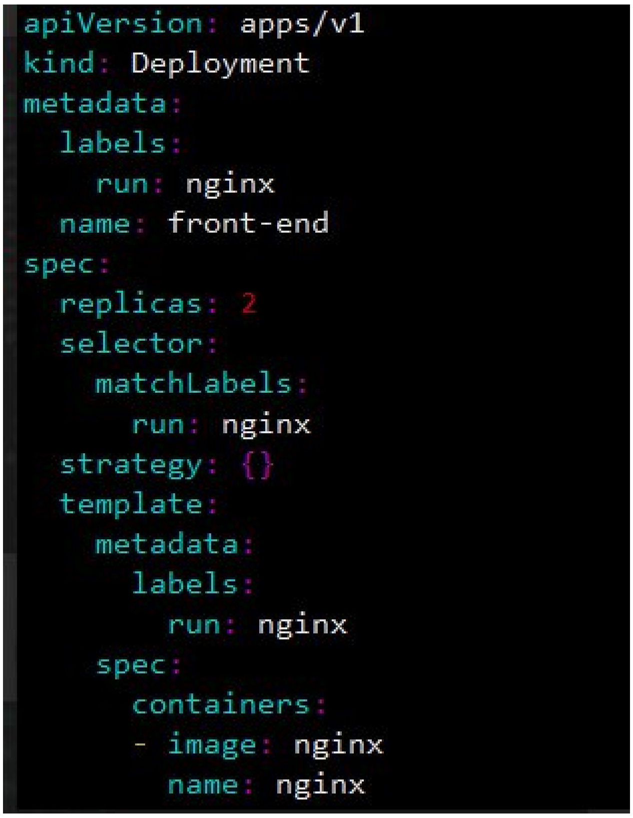
[문제13] NodePort

1. ‘front-end’ deployment를 다음 조건으로 생성
   - ✓ image: nginx, replicas: 2, label: run=nginx
2. 'front-end' deployment의 nginx 컨테이너를 expose하는 'front-end-nodesvc'라는 새 service를 생성
3. Front-end로 동작중인 Pod에는 node의 **30200** 포트로 접속

1) ‘front-end’ deployment 생성

```
# kubectl create deploy front-end --image=nginx --replicas=2 --dry-run=client -o yaml > front-end.yaml
```

2) front-end.yaml 수정



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

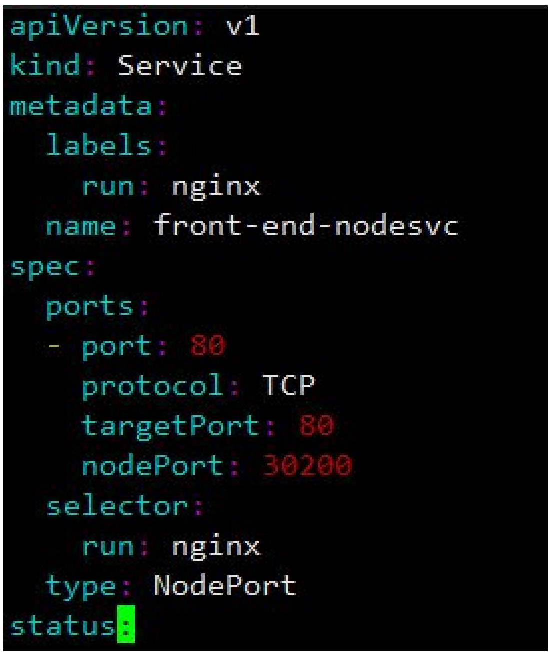
3) front-end.yaml 적용

# kubectl apply -f front-end.yaml

4) 'front-end-nodesvc' service 생성

# kubectl expose deploy front-end --name=front-end-nodesvc --port=80 --type=NodePort --target-port=80 --dry-run=client -o yaml > front-end-nodesvc.yaml

5) front-end-nodesvc.yaml 파일 수정



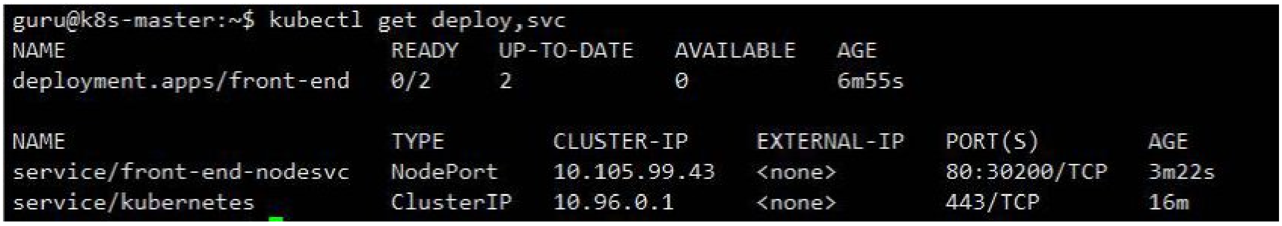
```
apiVersion: v1
kind: Service
metadata:
  labels:
    run: nginx
  name: front-end-nodesvc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30200
  selector:
    run: nginx
  type: NodePort
status:
```

6) front-end-nodesvc.yaml 적용

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

7) 'front-end' 디플로이먼트 및 서비스 확인

# kubectl get deploy,svc



```
guru@k8s-master:~$ kubectl get deploy,svc
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/front-end    0/2     2            0           6m55s

NAME                         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
service/front-end-nodesvc    NodePort    10.105.99.43   <none>        80:30200/TCP   3m22s
service/kubernetes           ClusterIP   10.96.0.1      <none>        443/TCP        16m
```

[문제14] Network Policy

1. customera, customerb를 생성한 후, 각각 PARTITION=customera, PARTITION=customer 를 라벨링

2. default namespace에 다음과 같은 pod를 생성

   ✓ name: pocimage: nginxport: 80

   ✓ label: app=poc

   ✓ "partition=customera"를 사용하는 namespace에서만 poc의

3. 80포트로 연결할 수 있도록 default namespace에 'allow-web-from-customera'라는 network Policy를 설정, 보안 정책상 다른 namespace의 접근은 제한

1) Kubernetes 사이트에서 [net pol] 검색

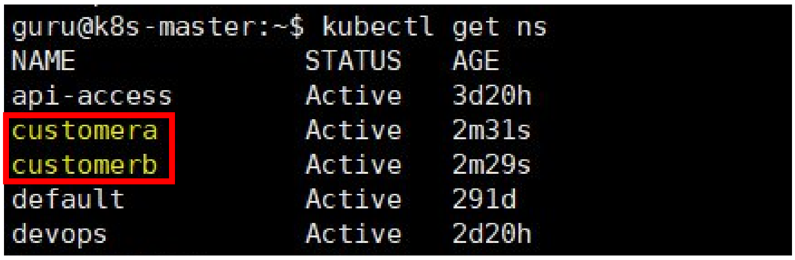
2) customera, customerb 네임스페이스 생성

# kubectl create ns customera

# kubectl create ns customerb

3) 네임스페이스 확인

# kubectl get ns



```
guru@k8s-master:~$ kubectl get ns
NAME                  STATUS   AGE
api-access            Active   3d20h
customera             Active   2m31s
customerb             Active   2m29s
default               Active   291d
devops                Active   2d20h
```

4) customera, customerb 파티션 및 라벨링

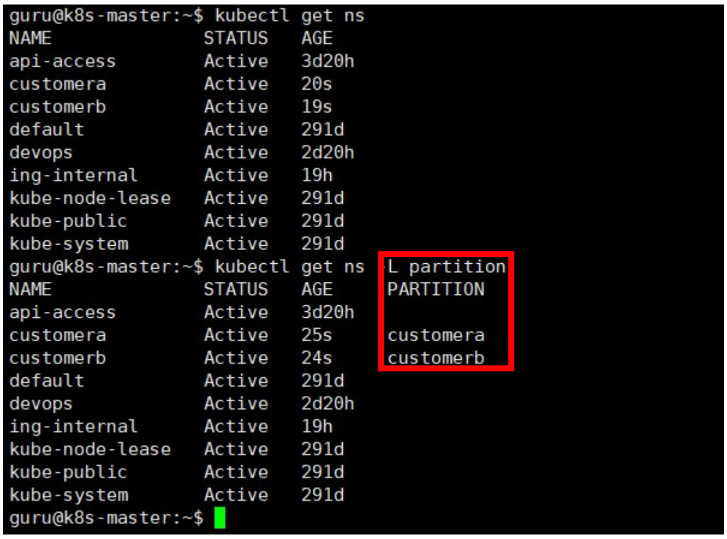
# kubectl label ns customera partition=customera

# kubectl label ns custpmerb partition=customerb

5) 네임스페이스 Partition 및 라벨 확인

# Kubectl get ns

# Kubectl get ns -L partition



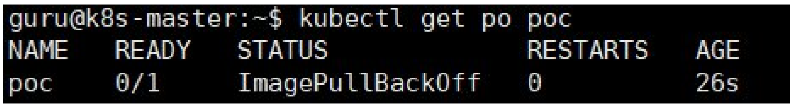
```
guru@k8s-master:~$ kubectl get ns
NAME                  STATUS   AGE
api-access            Active   3d20h
customera             Active   20s
customerb             Active   19s
default               Active   291d
devops                Active   2d20h
ing-internal          Active   19h
kube-node-lease       Active   291d
kube-public           Active   291d
kube-system           Active   291d
guru@k8s-master:~$ kubectl get ns -L partition
NAME                  STATUS   AGE     PARTITION
api-access            Active   3d20h
customera             Active   25s     customera
customerb             Active   24s     customerb
default               Active   291d
devops                Active   2d20h
ing-internal          Active   19h
kube-node-lease       Active   291d
kube-public           Active   291d
kube-system           Active   291d
guru@k8s-master:~$
```

6) poc 파드 생성

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

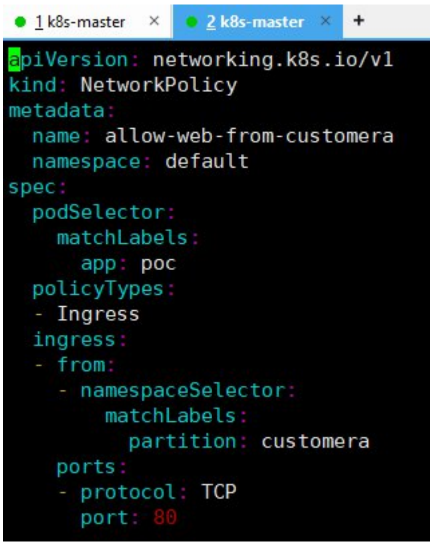
7) pod 파드 확인

# kubectl get po poc



```
guru@k8s-master:~$ kubectl get po poc
NAME   READY   STATUS             RESTARTS   AGE
poc    0/1     ImagePullBackOff   0          26s
```

8) netpol.yaml 생성 및 수정



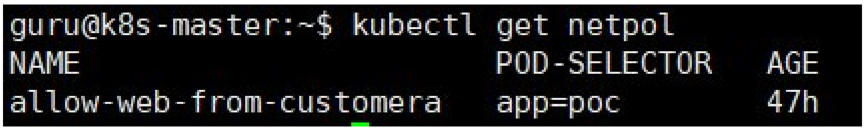
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-web-from-customera
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: poc
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          partition: customera
 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

9) netpol.yaml 적용

# kubectl apply -f netpol.yaml

10) Netpol 파드 확인

# kubectl get netpol



```
guru@k8s-master:~$ kubectl get netpol
NAME                       POD-SELECTOR   AGE
allow-web-from-customera   app=poc        47h
```

[문제15] Ingress

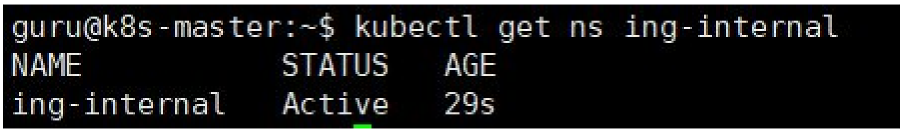
1. Create a new nginx Ingress resource as follows
   - ✓ Name : ping
   - ✓ Namespace : ing-internal
   - ✓ Exposing service hi on path /hi using service port 5678

1) Ing-internal 네임스페이스 생성

# kubectl create ns ing-internal

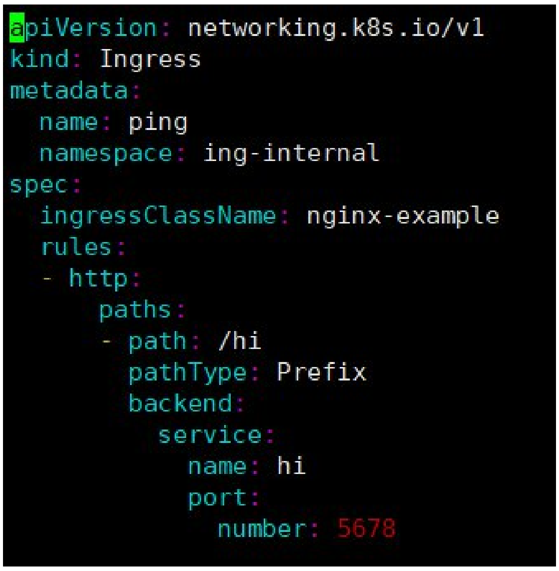
2) 네임스페이스 확인

# kubectl get ns ing-internal



```
guru@k8s-master:~$ kubectl get ns ing-internal
NAME           STATUS   AGE
ing-internal   Active   29s
```

3) ingress.yaml 수정



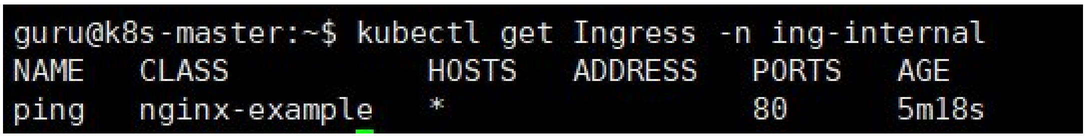
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /hi
        pathType: Prefix
        backend:
          service:
            name: hi
            port:
          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

4) ingress.yaml 파일 적용

# kubectl apply -f ingress.yaml

5) Ingress pod 확인

# kubectl get Ingress -n ing-internal



```
guru@k8s-master:~$ kubectl get Ingress -n ing-internal
NAME    CLASS           HOSTS    ADDRESS    PORTS    AGE
ping    nginx-example   *                   80       5m18s
```

[문제16] Service and DNS Lookup

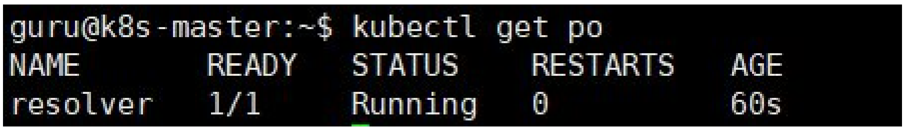
1. image nginx를 사용하는 resolver pod를 생성하고 resolver-service라는 service를 구성
2. 클러스터 내에서 service와 pod 이름을 조회할 수 있는지 테스트
   - ✓ dns 조회에 사용하는 pod 이미지는 busybox:1.28이고, service와 pod 이름 조회는 nlsookup을 사용
   - ✓ service 조회 결과는 /var/CKA2023/nginx.svc에 pod name 조회 결과는 /var/CKA2023/nginx.pod 파일에 기록

1) resolver pod 생성

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

2) resolver pod 확인

# kubectl get po



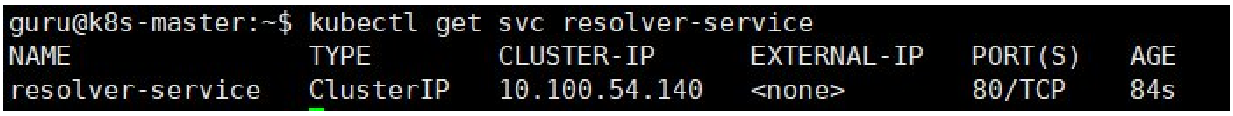
```
guru@k8s-master:~$ kubectl get po
NAME        READY   STATUS    RESTARTS   AGE
resolver    1/1     Running   0          60s
```

3) 서비스 생성

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

4) 서비스 확인

# kubectl get svc resolver-service



```
guru@k8s-master:~$ kubectl get svc resolver-service
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
resolver-service   ClusterIP   10.100.54.140   <none>        80/TCP    84s
```

5) Root 전환 및 디렉터리 생성

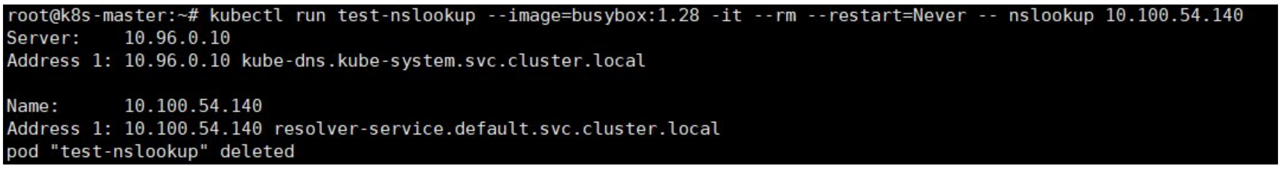
# sudo -i

# mkdir -p /var/CKA2023

6) service 조회 결과 /var/CKA2023/nginx.svc파일에 기록

# kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10.100.54.140

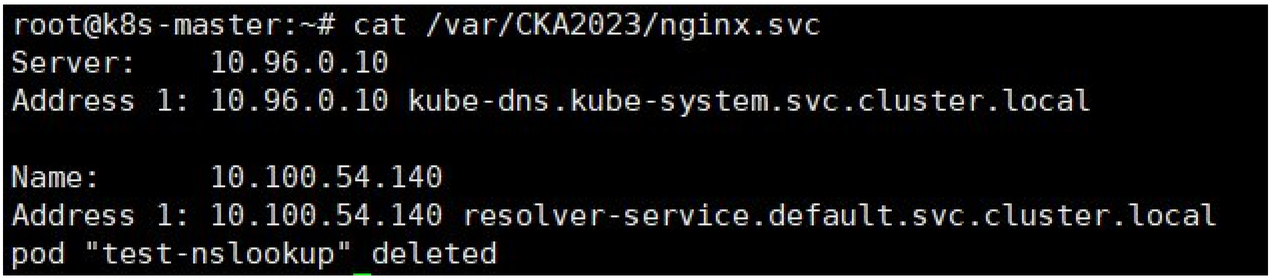
# kubectl run test-nslookup 10.100.54.140 -image=busybox:1.28 -it --rm --restart=Never --resolver 10.100.54.140 > nginx.svc



```
root@k8s-master:~# kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10.100.54.140
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10.100.54.140
Address 1: 10.100.54.140 resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
```

7) 주소 확인

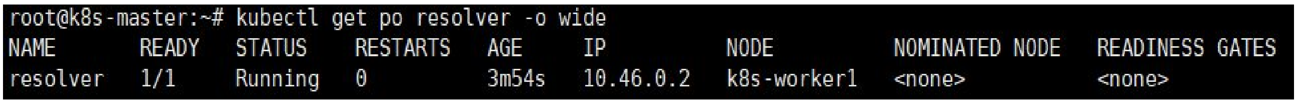
# cat /var/cka2023/nginx.svc



```
root@k8s-master:~# cat /var/CKA2023/nginx.svc
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10.100.54.140
Address 1: 10.100.54.140 resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
```

8) resolver 파드 ip 확인

kubectl get po resolver -o wide



```
root@k8s-master:~# kubectl get po resolver -o wide
NAME       READY   STATUS    RESTARTS   AGE     IP          NODE          NOMINATED NODE   READINESS GATES
resolver   1/1     Running   0          3m54s   10.46.0.2   k8s-worker1   <none>           <none>
```

9) pod name 조회 결과 /var/CKA2023/nginx.pod 파일에 기록

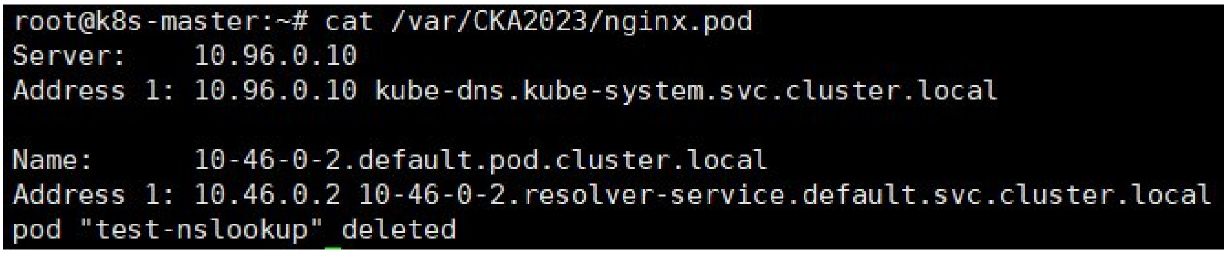
# kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookp 10-46-0-2 default.pod.cluster.local

# kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookp 10-46-0-2

default.pod.cluster.local > /var/CKA2023/nginx.pod

10) 확인

```
# cat /var/CKA2023/nginx.pod
```



```
root@k8s-master:~# cat /var/CKA2023/nginx.pod
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      10-46-0-2.default.pod.cluster.local
Address 1: 10.46.0.2 10-46-0-2.resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
```

[문제17] 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

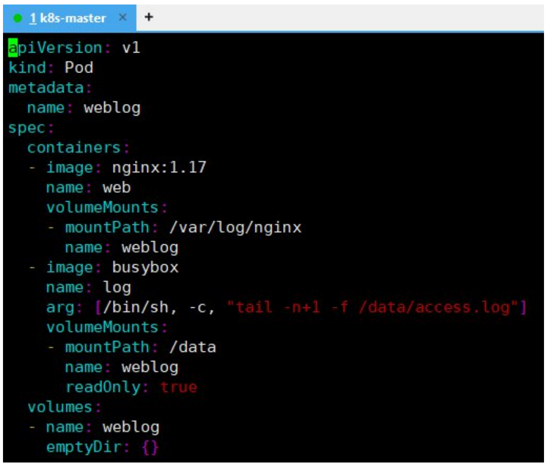
   Log container:

   - ✓ 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



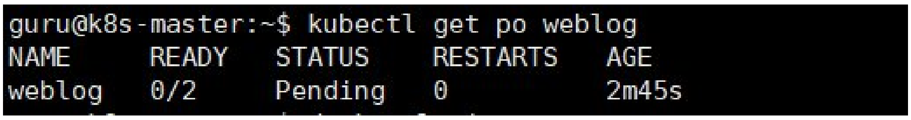
```
apiVersion: v1
kind: Pod
metadata:
  name: weblog
spec:
  containers:
  - image: nginx:1.17
    name: web
    volumeMounts:
    - mountPath: /var/log/nginx
      name: weblog
  - image: busybox
    name: log
    arg: [/bin/sh, -c, "tail -n+1 -f /data/access.log"]
    volumeMounts:
    - mountPath: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

3) weblog yaml 파일 적용

# kubectl apply -f weblog.yaml

4) weblog 파드 확인

# kubectl get po weblog



```
guru@k8s-master:~$ kubectl get po weblog
NAME      READY    STATUS     RESTARTS    AGE
weblog    0/2      Pending    0           2m45s
```

[문제18] HostPath Volume

1. /data/cka/fluentd.yaml 파일을 만들어 새로운 Pod 생성 및 볼륨마운트 설정

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

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

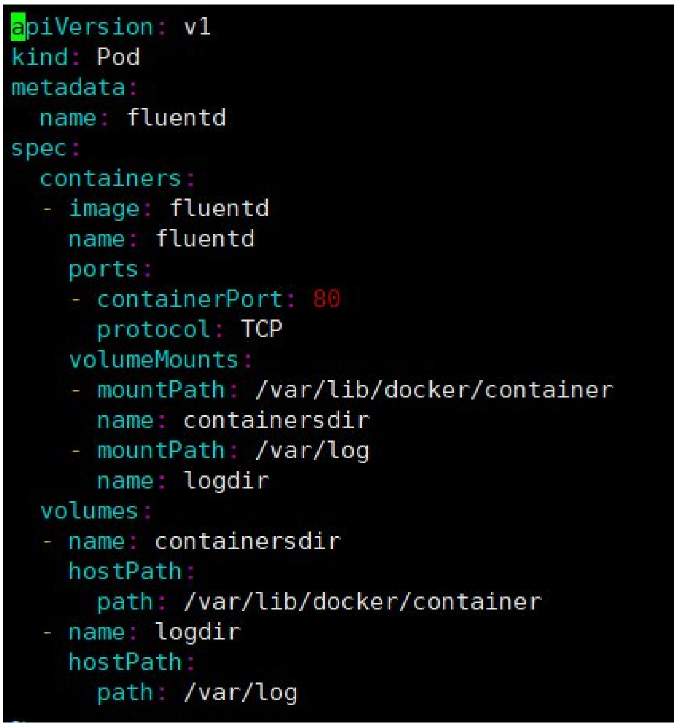
4. Worker node의 /var/log 디렉토리를 fluentd Pod에 동일이름의 디렉토리 마운트

1) /data/cka/fluentd.yaml 이동 및 fluentd.yaml 생성

# cd /data/cak

# vi fluentd.yaml

2) fluentd.yaml 코드 붙여넣기 및 수정



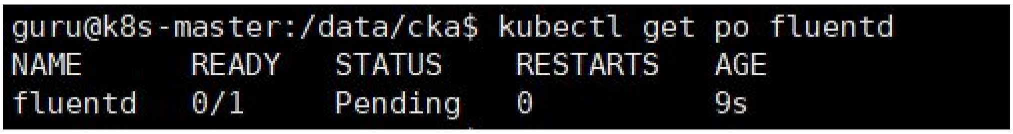
```
apiVersion: v1
kind: Pod
metadata:
  name: fluentd
spec:
  containers:
  - image: fluentd
    name: fluentd
    ports:
    - containerPort: 80
      protocol: TCP
    volumeMounts:
    - mountPath: /var/lib/docker/container
      name: containersdir
    - mountPath: /var/log
      name: logdir
 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

3) fluentd yaml 파일 적용

# kubectl apply -f fluentd.yaml

4) fluentd pod 확인

# kubectl get po fluentd



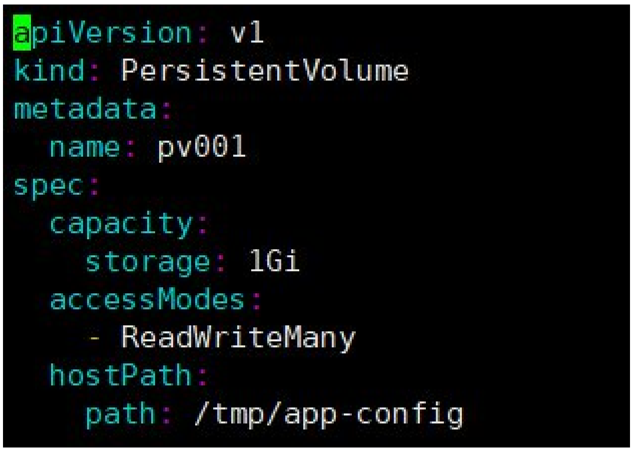
```
guru@k8s-master:/data/cka$ kubectl get po fluentd
NAME      READY   STATUS    RESTARTS   AGE
fluentd   0/1     Pending   0          9s
```

[문제19] Persistent Volume

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

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

1) pv001.yaml 생성 후 코드 붙여넣기 및 수정
# vi pv001.yaml



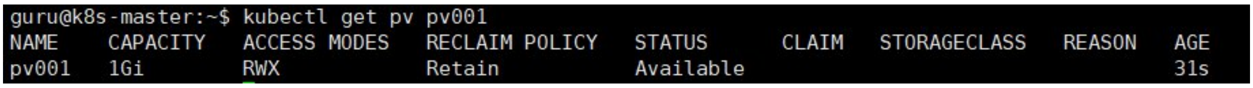
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv001
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /tmp/app-config
```

2) pv001.yaml 적용

   # kubectl apply -f pv001.yaml

3) pv001 pv 확인

   # kubectl get pv pv001



```
guru@k8s-master:~$ kubectl get pv pv001
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv001   1Gi        RWX            Retain           Available                                   31s
```