대우능력개발원
DAEWOO DEVELOPMENT OF ABILITY

Kubernetes Project

kubernetes

FRONT_1 조

김선혁

정세환

임영철

박채령

목차

[문제 1] ETCD Backup https://127.0.0.1:2379 에서 실행 중인 etcd 의 snapshot 을 생성하고 snapshot 을 /data/etcd snapshot.db 에 저장합니다. 그런 다음 /data/etcd-snapshot-previous.db 에 있는 기존의 이전 스냅샷을 복원합니다. etcdctl 을 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.Etcd 의 snapshot 생성, /data/etcd-snapshot.db 에 저장

```
 252  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
```

2.Find 명령어를 통해 파일이 정확히 들어갔는지 확인

```
root@master:~# find /data/etcd-snapshot.db
/data/etcd-snapshot.db
```

3.Snapshot 복원

```
 255  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
```

[문제 2] Cluster upgrade 마스터 노드의 모든 Kubernetes control plane 및 node 구성 요소를 버전 1.29.6-1.1 버전으로 업 그레이드합니다. master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon 해야 합니다. "주의사항" 반드시 Master Node 에서 root 권한을 가지고 작업을 실행해야 한다.

1.버전 확인

```
root@master:~# kubectl get no
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   7d22h   v1.28.15
worker1   Ready    <none>          7d21h   v1.28.15
worker2   Ready    <none>          7d21h   v1.28.15
```

2.현재 지원되는 패키지 버전 확인

```
root@master:~# sudo apt-cache madison kubeadm
   kubeadm | 1.28.15-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.14-2.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.13-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.12-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.11-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.10-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.9-2.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.8-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.7-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.6-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.5-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.4-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.3-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.2-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.1-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
   kubeadm | 1.28.0-1.1 | https://pkgs.k8s.io/core:/stable:/v1.28/deb  Packages
```

3.다운 그레이드를 위해 고정 취소

```
root@master:~# sudo apt-mark unhold kubeadm
kubeadm 패키지 고정 취소함.
```

4.현재 1.28.15 로 지원되는 버전 중 최상위 버전이기 때문에 1.28.14-2.1 로 다운그레이드

```
root@master:~# sudo apt-get update && sudo apt-get install -y kubeadm='1.28.14-2.1'
기존:1 https://download.docker.com/linux/ubuntu focal InRelease
기존:3 http://security.ubuntu.com/ubuntu focal-security InRelease
기존:4 http://kr.archive.ubuntu.com/ubuntu focal InRelease
기존:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.28/deb  InRelease
기존:5 http://kr.archive.ubuntu.com/ubuntu focal-updates InRelease
기존:6 http://kr.archive.ubuntu.com/ubuntu focal-backports InRelease
패키지 목록을 읽는 중입니다... 완료
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
다음 패키지를 다운그레이드할 것입니다:
  kubeadm
0개 업그레이드, 0개 새로 설치, 1개 업그레이드, 0개 제거 및 76개 업그레이드 안 함.
E: 패키지는 다운그레이드 되었으며 --allow-downgrades 옵션 없이 -y옵션이 사용되었습니다.
```

5.자동 업데이트를 방지하기 위해 다시 고정

```
root@master:~# sudo apt-mark hold kubeadm
kubeadm 패키지 고정으로 설정.
```

6.Control Plane 업그레이드 절차 수행

```
root@master:~# kubeadm upgrade apply v1.28.14 --force
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
```

7.Drain 을 통하여 클러스터 다운 그레이드 중, 노드를 클러스터에서 안전하게 비우는 작업

```
root@master:~# kubectl drain master --ignore-daemonsets
node/master cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/fluentd-elasticsearch-q2nx5, kube-system/kube-proxy-kfw2g, kube-system/weave-net-mrsz9
evicting pod kube-system/coredns-5dd5756b68-lrn5v
pod/coredns-5dd5756b68-lrn5v evicted
node/master drained
```

8.Kubelet 다운 그레이드 위해서 고정 취소

```
root@master:~# apt-mark unhold kubelet kubectl
kubectl은(는) 고정하지 않았습니다.
kubelet 패키지 고정 취소함.
```

9.이후, 같은 과정으로 패키지 다운 그레이드 후 서비스 재시작

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

10.적용 결과

```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.28.14". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading
```

[문제 3] ServiceAccount, Role, RoleBinding 애플리케이션 운영중 특정 namespace 의 Pod 들을 모니터할수 있는 서비스가 요청되었습니다. api-access 네임스페이스의 모든 pod 를 view 할 수 있도록 다음의 작업을 진행하시오. 1. api-access 라는 새로운 namespace 에 pod-viewer 라는 이름의 Service Account 를 만듭니다. 2. podreader-role 이라는 이름의 Role 과 podreader-rolebinding 이라는 이름의 RoleBinding 을 만 듭니다. 3. 앞서 생성한 ServiceAccount 를 API resource Pod 에 대하여 watch, list, get 을 허용하도록 매핑 하시오.

1.Kubectl ns api-access 를 통하여 네임 스페이스 생성 후, 생성 여부 확인

```
root@master:~# kubectl get ns | grep api-access
api-access        Active   127m
```

2.Kubectl create sa pod-viewer -n api-access

> api-access 네임 스페이스 안에 pod-viewer 서비스 어카운트 생성 후 확인

```
root@master:~# kubectl get sa pod-viewer -n api-access
NAME         SECRETS   AGE
pod-viewer   0         128m
```

```
root@master:~# kubectl describe sa pod-viewer -n api-access
Name:                pod-viewer
Namespace:           api-access
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   <none>
Tokens:              <none>
Events:              <none>
```

3.Kubectl create role podreader-role -n api-access –resource=pod –verb=watch,list,get

api-access 네임스페이스 내부에 리소스는 pod 에 대해 watch,list,get 을 허용하는 롤 생성 후, 확인

```
ubuntu@master:~$ kubectl describe role -n api-access
Name:         podreader-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  pods       []                 []              [watch list get]
```

4.Kubectl create rolebinding podreader-rolebinding –serviceaccount=api-access:pod-viewer --role=podreader-role -n api-access

api-access 네임스페이스 안에 있는 podreader-role 과 service account pod-viewer 묶어주는 롤 바인딩 생성 후 확인

```
root@master:~# kubectl get rolebinding podreader-rolebinding -o yaml -n api-access
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2025-01-16T05:11:12Z"
  name: podreader-rolebinding
  namespace: api-access
  resourceVersion: "171478"
  uid: 2f148ac1-c4c9-4041-a3c4-9ff698fd1cf2
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: podreader-role
subjects:
- kind: ServiceAccount
  name: pod-viewer
  namespace: api-access
```

[문제 4] ServiceAccount, ClusterRole, ClusterRoleBinding 작업 Context 에서 애플리케이션 배포를 위해 새로운 ClusterRole 을 생성하고 특정 namespace 의 ServiceAccount 를 바인드하시오. 다음의 resource type 에서만 Create 가 허용된 ClusterRole deployment-clusterrole 을 생성합니다. Resource Type: Deployment StatefulSet DaemonSet 미리 생성된 namespace api-access 에 cicd-token 이라는 새로운 ServiceAccount 를 만듭니다. ClusterRole deployment-clusterrole 을 namespace api-access 로 제한된 새 ServiceAccount cicd-token 에 바인딩 하세요

1.Kubectl create sa cicd-token -n api-access

api-access 네임스페이스 안에 cicd-token 이라는 service account 생성 후 확인

```
root@master:~# kubectl get sa cicd-token -n api-access
NAME         SECRETS   AGE
cicd-token   0         132m
```

```
root@master:~# kubectl get sa cicd-token -o yaml -n api-access
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2025-01-16T05:14:46Z"
  name: cicd-token
  namespace: api-access
  resourceVersion: "171801"
  uid: aa822c08-da71-4d7e-a662-b25a5cffa101
```

2.Kubectl create clusterrole deployment-clusterrole –resource=deployment,statefulset, daemonset –verb=create > deploy, statefulset ,daemonset 타입의 리소스에 대해 create 가 허용된 cluster role 생성 후 확인

```
root@master:~# kubectl get clusterrole deployment-clusterrole
NAME                     CREATED AT
deployment-clusterrole   2025-01-16T05:13:54Z
```

```
root@master:~# kubectl get clusterrole deployment-clusterrole -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: "2025-01-16T05:13:54Z"
  name: deployment-clusterrole
  resourceVersion: "171722"
  uid: 0c3fe5f6-728e-467b-92fe-c85d62a36b60
rules:
- apiGroups:
  - apps
  resources:
  - deployments
  - statefulsets
  - daemonsets
  verbs:
  - create
```

3.Kubectl create clusterrolebinding deployment-clusterrolebinding –clusterrole= deployment-clusterrole –serviceaccount=api-access:cicd-token -n api-access

api-access 네임스페이스 안에 있는 service account 와 clusterrole 을 묶어주는 cluster role binding 생성 후, 확인

```
root@master:~# kubectl get clusterrolebinding deployment-clusterrolebinding
NAME                            ROLE                                AGE
deployment-clusterrolebinding   ClusterRole/deployment-clusterrole   138m
```

```
root@master:~# kubectl get clusterrolebinding deployment-clusterrolebinding -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: "2025-01-16T05:16:34Z"
  name: deployment-clusterrolebinding
  resourceVersion: "171965"
  uid: c814a4ff-fa6f-4994-abca-26578fc49175
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deployment-clusterrole
subjects:
- kind: ServiceAccount
  name: cicd-token
  namespace: api-access
```

[문제 5] 노드 비우기 작업 클러스터 : k8s k8s-worker2 노드를 스케줄링 불가능하게 설정하고, 해당 노드에서 실행 중인 모든 Pod을 다른 node로 reschedule 하세요

1.Ready 상태 확인

```
ubuntu@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   7d22h   v1.28.15
worker1   Ready    <none>          7d22h   v1.28.15
worker2   Ready    <none>          7d22h   v1.28.15
```

2.Drain을 통하여 worker2를 스케줄링 불가능 상태로 전환한 뒤

해당 노드에 존재하는 모든 파드를 다른 노드로 재배치

```
ubuntu@master:~$ kubectl drain worker2 --ignore-daemonsets
node/worker2 already cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/fluentd-elasticsearch-qdrjq, kube-system/kube-proxy-gs5rz, kube-system/weave-net-bw2lw
evicting pod kube-system/coredns-5dd5756b68-pjpsx
pod/coredns-5dd5756b68-pjpsx evicted
node/worker2 drained
```

3.스케줄링 불가 결과

```
ubuntu@master:~$ kubectl get nodes
NAME      STATUS                     ROLES           AGE     VERSION
master    Ready                      control-plane   7d22h   v1.28.15
worker1   Ready                      <none>          7d22h   v1.28.15
worker2   Ready,SchedulingDisabled   <none>          7d22h   v1.28.15
```

4.Worker2 노드 다시 스케줄링 가능 상태로 전환

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

[문제 6] Pod Scheduling 작업 클러스터 : k8s 다음의 조건으로 pod 를 생성하세요. Name: eshop-store Image: nginx Nodeselector: disktype=ssd

1.Kubectl label node worker1 disktype=ssd

kubectl label node worker2 disktype=hdd

worker1,2 에 알맞은 disktype label 생성 후 확인

```
root@master:~# kubectl get nodes -L disktype
NAME      STATUS   ROLES           AGE    VERSION    DISKTYPE
master    Ready    control-plane   8d     v1.28.15
worker1   Ready    <none>          104m   v1.28.15   ssd
worker2   Ready    <none>          104m   v1.28.15   hdd
```

2.Kubectl run eshop-store –image=nginx –dry-run=client -o yaml > eshop-store.yaml

vi eshop-store 을 통하여 yaml 파일에 nodeSelector 추가 후,

kubectl apply -f eshop-store 을 통하여 적용, pod 를 생성

그 후 내용 확인

```
root@master:~# kubectl describe pod eshop-store | grep Image
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:0a399eb16751829e1af26fea27b20c3ec28d7ab1fb72182879dcae1cca21206a
root@master:~# kubectl describe pod eshop-store | grep Node-Sele
Node-Selectors:              disktype=ssd
```

[문제 7] Pod 생성하기 작업 클러스터 : k8s 'cka-exam'이라는 namespace 를만들고, 'cka-exam' namespace 에 아래와 같은 Pod 를 생성하시오. pod Name: pod-01 image: busybox 환경변수 : CERT = "CKA-cert" command: /bin/sh args: "-c", "while true; do echo $(CERT); sleep 10;done"

1.Kubectl get ns 를 통하여 cka-exam 네임 스페이스 존재 여부 확인 후,
없으면 kubectl create ns cka-exam 을 통하여 생성

```
root@master:~# kubectl get ns | grep cka-exam
cka-exam          Active   103m
root@master:~#
```

2.Kubectl run pod-01 –image=busybox –env=CERT="CKA-cert" -n cka-exam
--dry-run=client -o yaml > pod-01.yaml 을 통하여 yaml 파일 생성
vi pod-01.yaml 을 통하여 내용 수정 후, kubectl apply -f pod-01.yaml 로 적용

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

```
Containers:
  pod-01:
    Container ID:  containerd://fc7cf629fb3618dc5860bc5483c1559c01d9d293d21435b4563fe98d2827a3da
    Image:         busybox
    Image ID:      docker.io/library/busybox@sha256:2919d0172f7524b2d8df9e50066a682669e6d170ac0f6a49676d54358fe970
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do echo $(CERT); sleep 10;done
    State:          Running
      Started:      Thu, 16 Jan 2025 15:03:09 +0900
    Ready:          True
    Restart Count:  0
    Environment:
      CERT:  CKA-CERT
```

[문제 8] Pod "nginx-static-pod-k8s-worker1"의 log 를 모니터링하고, 메세지를 포함하는 로그라인을 추출하세요. 추출된 결과는 /opt/REPORT/2023/pod-log 에 기록하세요.
문제 9 를 통하여 nginx-static-pod-worker1 을 생성 후 진행 하였습니다.

1.Nginx-static-pod-worker1 존재 여부 확인

```
ubuntu@master:~$ kubectl get po | grep -i nginx-static
nginx-static-pod-worker1        1/1     Running
```

2.Sudo mkdir -p /opt/REPORT/2023/을 통하여 디렉터리 생성,
kubectl logs nginx-static-pod-worker1 > /opt/REPORT/2023/pod-log
위 디렉터리에 pod-log 라는 이름의 파일로 로그라인 추출 후 확인

```
root@master:~# find /opt/REPORT/2023/pod-log
/opt/REPORT/2023/pod-log
root@master:~# 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 06:09:48 [notice] 1#1: using the "epoll" event method
2025/01/16 06:09:48 [notice] 1#1: nginx/1.27.3
2025/01/16 06:09:48 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/01/16 06:09:48 [notice] 1#1: OS: Linux 5.15.0-130-generic
2025/01/16 06:09:48 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/01/16 06:09:48 [notice] 1#1: start worker processes
2025/01/16 06:09:48 [notice] 1#1: start worker process 29
2025/01/16 06:09:48 [notice] 1#1: start worker process 30
```

[문제 9] staticpod 생성하기 worker1 노드에 nginx-static-pod.yaml 라는 이름의 Static Pod 를 생성하세요. pod name: nginx-static-pod image: nginx port : 80
master 세션 복제 후, 하나의 세션은 ssh worker1 을 통해 접속하여 진행 했습니다.

1.K8s 노드에서 Static Pod 정의 파일이 기본적으로 위치하는 경로를 찾음
그 후, 이 경로대로 이동 cd /etc/kubernetes/manifests (ssh1 세션)

```
root@master:~# cat /var/lib/kubelet/config.yaml | grep static
staticPodPath: /etc/kubernetes/manifests
```

2.Image = nginx , port = 80, 이름은 nginx-static-pod 인 yaml 파일 생성 (master 세션)
그 후, vi nginx-static-pod.yaml 을 통하여 복사 후, ssh1 세션의 ~~/manifests 에서
똑같이 vi nginx-static-pod.yaml 위치에 붙여넣기 > :wq 로 저장 후 나가기
history 는 master 에서 yaml 파일 제작 후, apply 를 하지 않았다는 증거입니다.

```
root@master:~# history | grep nginx-static | grep dry
  217  kubectl run nginx-static-pod --image=nginx --port=80 --dry-run=client -o yaml > nginx-static-pod.yaml
  228  kubectl run nginx-static-pod --image=nginx --port=80 --dry-run=client -o yaml > nginx-static-pod.yaml
```

```
root@master:~# history | grep nginx-static | grep apply
  384  history | grep nginx-static | grep apply
```

3.Static Pod 확인

```
root@master:~# kubectl get pod -o wide | grep static
nginx-static-pod-worker1       1/1     Running           0          107m   10.44.0.2   worker1   <none>          <none>
```

[문제 10]multicontainer Pod 생성하기 4 개의 컨테이너를 동작시키는 eshop-frontend Pod 를 생성하시오. pod image: nginx, redis, memcached, consul

1.Kubectl run eshop-frontend –image=nginx –dry-run=client -o yaml > eshop-frontend.yaml

yaml 파일 생성 후, vi eshop-frontend.yaml

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

2.Kubectl apply -f eshop-frontend.yaml 을 통하여 적용, Pod 를 생성 후 확인

```
root@master:~# kubectl get pod eshop-frontend
NAME             READY   STATUS             RESTARTS   AGE
eshop-frontend   3/4     ImagePullBackOff   0          105m
```

```
Containers:
  nginx:
    Container ID:   containerd://e89585cd1452550d2d91751f5b6b3b1db0443585eafb50db6519acc599f8dc98
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:0a399eb16751829e1af26fea27b20c3ec28d7ab1fb72182
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jan 2025 15:15:37 +0900
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5rclv (ro)
  redis:
    Container ID:   containerd://56626004f2363902f6b03f70e6ee8bbaaa9a21abf5625b3b633bd73af2201db4
    Image:          redis
    Image ID:       docker.io/library/redis@sha256:26bfdb90be9e92338a6e20d18db7152fb58e886773b67cd
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jan 2025 15:15:43 +0900
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5rclv (ro)
  memcached:
    Container ID:   containerd://74f4a7caf153039168938c50bf7f88657680e2eca956cca82d00ac651013a592
    Image:          memcached
    Image ID:       docker.io/library/memcached@sha256:0e49fcdaf62a232b012ae931555a3618540936440026
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 16 Jan 2025 15:15:47 +0900
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5rclv (ro)
  consul:
    Container ID:
    Image:          consul
```

[문제 11]Rolling update & Roll back 작업 클러스터 : k8s Deployment 를 이용해 nginx 파드를 3 개 배포한 다음 컨테이너 이미지 버전을 rolling update 하 고 update record 를 기록합니다. 마지막으로 컨테이너 이미지를 previous version 으로 roll back 합니다. name: eshop-payment Image : nginx Image version: 1.16 update image version: 1.17 label: app=payment, environment=production

1.Deployment 생성하기 전, Yaml 파일 생성

```
ubuntu@master:~$ kubectl create deploy eshop-payment --image=nginx:1.16 --replicas=3 --dry-run=client -o yaml > eshop-payment.yaml
ubuntu@master:~$ ls
dns-deployment.yaml  eshop-frontend.yaml  eshop-store.yaml       pod-01.yaml  다운로드  바탕화면  사진  템플릿
dnsconfig-pod.yaml   eshop-payment.yaml   nginx-static-pod.yaml  공개         문서      비디오    음악
```

2.yaml 파일 내용 > environment 값, app 값 추가 (label 내용 수정)

vi eshop-payment.yaml

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

3.yaml 파일 적용, Deployment 생성, 레코드 번호 생성

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

4.nginx 1.16 > nginx 1.17로 업데이트, 레코드 번호 생성

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

5.이미지 업데이트 확인

Kubectl describe pod | grep -I image

```
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
  Normal  Pulling    58s   kubelet            Pulling image "nginx:1.17"
  Normal  Pulled     24s   kubelet            Successfully pulled image "nginx:1.17" in 33.616s (33.616s including waiting)
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
  Normal  Pulling    97s   kubelet            Pulling image "nginx:1.17"
  Normal  Pulled     62s   kubelet            Successfully pulled image "nginx:1.17" in 35.257s (35.257s including waiting)
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e36e24e37039ee9eae1fe38a6420d8ae16ef37c92d1eb26699
  Normal  Pulled     21s   kubelet            Container image "nginx:1.17" already present on machine
```

6.이미지, 레코드 번호 확인

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

7.이미지 롤백 후 레코드 번호 확인

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

8.롤백 이미지 버전 확인

Kubectl describe pod | grep -I image

```
    Image:          nginx:1.16
    Image ID:       docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
  Normal  Pulled     62s   kubelet            Container image "nginx:1.16" already present on machine
    Image:          nginx:1.16
    Image ID:       docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
  Normal  Pulled     58s   kubelet            Container image "nginx:1.16" already present on machine
    Image:          nginx:1.16
    Image ID:       docker.io/library/nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
  Normal  Pulled     54s   kubelet            Container image "nginx:1.16" already present on machine
```

[문제 12] ClusterIP 'devops' namespace 에서 deployment eshop-order 를 다음 조건으로 생성하시오. - image: nginx, replicas: 2, label: name=ord 'eshop-order' deployment 의 Service 를 만드세요. Service Name: eshop-order-svc Type: ClusterIP Port: 80

1.Kubectl get ns devops 를 통하여 존재여부 확인, 없으면 kubectl create ns devops

```
root@master:~# kubectl get ns | grep devops
devops            Active   88m
root@master:~#
```

2.Kubectl create deploy eshop-order -n devops –image=nginx –replicas=2 –dry-run=client -o yaml > eshop-order.yaml 을 통하여 yaml 파일 생성, vi eshop-order.yaml 을 통해 yaml 파일 내용 수정
그 후, kubectl apply -f eshop-order.yaml

```
root@master:~# kubectl describe deploy eshop-order -n devops
Name:                   eshop-order
Namespace:              devops
CreationTimestamp:      Thu, 16 Jan 2025 15:46:38 +0900
Labels:                 name=order
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               name=order
Replicas:               2 desired | 2 updated | 2 total | 2 availab
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  name=order
  Containers:
   nginx:
    Image:        nginx
```

3.Kubectl expose deploy eshop-order -n devops –name=eshop-order-svc –port=80 –target-port=80
devops 네임스페이스 안에 있는 eshop-order 라는 이름의 deploy 안에,
eshop-order-svc 서비스 생성, port = 80

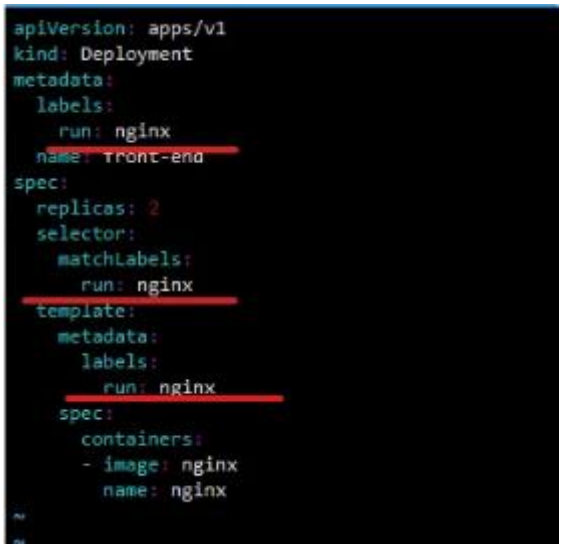
```
root@master:~# kubectl describe svc eshop-order-svc -n devops
Name:              eshop-order-svc
Namespace:         devops
Labels:            name=order
Annotations:       <none>
Selector:          name=order
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.104.68.177
IPs:               10.104.68.177
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.42.0.4:80,10.44.0.3:80
Session Affinity:  None
Events:            <none>
```

[문제 13 NodePort]

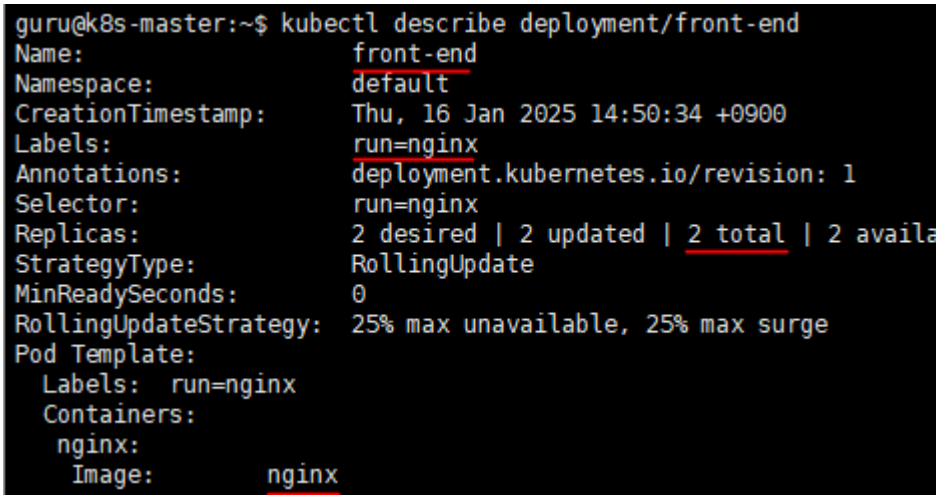
'front-end' deployment 를 다음 조건으로 생성하시오. image: nginx, replicas: 2, label: run=nginx 'front-end' deployment 의 nginx 컨테이너를 expose 하는 'front-end-nodesvc'라는 새 service 를 만듭니다. Front-end 로 동작중인 Pod 에는 node 의 **30200** 포트로 접속되어야 합니다

1.Kubectl create deploy front-end –image=nginx –replicas=2 –dry-run=client -o yaml >

front-end.yaml -> vi front-end.yaml 을 통한 yaml 파일 생성, 수정



2.Kubectl apply -f front-end.yaml 을 통하여 yaml 파일 적용, 확인



3.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

yaml 파일 생성, 수정

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

4.kubectl apply -f front-end-nodesvc.yaml 을 통하여 적용 후, 확인

```
guru@k8s-master:~$ kubectl describe svc front-end-nodesvc
Name:                     front-end-nodesvc
Namespace:                default
Labels:                   run=nginx
Annotations:              <none>
Selector:                 run=nginx
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.111.71.1
IPs:                      10.111.71.1
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30200/TCP
Endpoints:                10.32.0.4:80,10.38.0.2:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

```
guru@k8s-master:~$ kubectl get svc front-end-nodesvc
NAME               TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
front-end-nodesvc  NodePort   10.111.71.1   <none>        80:30200/TCP   24h
```

[문제 14 Network Policy]

customera, customerb 를 생성한 후, 각각 PARTITION=customera, PARTITION=customerb 를 라벨링하시오. default namespace 에 다음과 같은 pod 를 생성하세요. name: poc image: nginx port: 80 label: app=poc "partition=customera"를 사용하는 namespace 에서만 poc 의 80 포트로 연결할 수 있도록 default namespace 에 'allow-web-from-customera'라는 network Policy 를 설정하세요. 보안 정책상 다른 namespace 의 접근은 제한합니다.

1. kubectl create ns customera , kubectl create ns customerb > customer a,b 네임스페이스 생성

```
guru@k8s-master:~$ kubectl get ns | grep customer
customera         Active   23h
customerb         Active   23h
```

2. kubectl label ns customera partition=customera , kubectl label ns customerb partition=customerb > 네임스페이스 라벨링 부여

```
guru@k8s-master:~$ kubectl get ns --show-labels | grep custo
customera         Active   23h    kubernetes.io/metadata.name=customera,partition=customera
customerb         Active   23h    kubernetes.io/metadata.name=customerb,partition=customerb
```

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

```
guru@k8s-master:~$ kubectl describe pod/poc
Name:             poc
Namespace:        default
Priority:         0
Service Account:  default
Node:             k8s-worker1/10.100.0.106
Start Time:       Thu, 16 Jan 2025 15:00:37 +0900
Labels:           app=poc
```

4. vi netpol.yaml > Netpol.yaml 파일 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:
          name: customera
          partition: customera
    ports:
    - protocol: TCP
      port: 80
```

5. kubectl apply -f netpol.yaml 을 통하여 적용

```
guru@k8s-master:~$ kubectl get netpol allow-web-from-customera -o yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"networking.k8s.io/v1","kind":"NetworkPolicy","meta
espace":"default"},"spec":{"ingress":[{"from":[{"namespaceSelector":{"m
orts":[{"port":80,"protocol":"TCP"}]}],"podSelector":{"matchLabels":{"a
  creationTimestamp: "2025-01-15T05:07:32Z"
  generation: 2
  name: allow-web-from-customera
  namespace: default
  resourceVersion: "121898"
  uid: 1c8989bf-d51a-48ad-a62a-7344bd834636
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: customera
          partition: customera
    ports:
    - port: 80
      protocol: TCP
  podSelector:
    matchLabels:
      app: poc
  policyTypes:
  - Ingress
```

[문제 15 Ingress]

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

1. kubectl create ns ing-internal> ing-internal 네임스페이스 생성

```
guru@k8s-master:~$ kubectl get ns ing-internal
NAME           STATUS   AGE
ing-internal   Active   23h
```

2. ingress.yaml 파일 생성 및 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
```

3. kubectl apply -f ingress.yaml > ingress.yaml 을 적용 후, 확인

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

[문제 16 Service and DNS Lookup]

image nginx 를 사용하는 resolver pod 를 생성하고 resolver-service 라는 service 를 구성합니다. 클러스터 내에서 service 와 pod 이름을 조회할 수 있는지 테스트합니다.- dns 조회에 사용하는 pod 이미지는 busybox:1.28 이고, service 와 pod 이름 조회는 nlsookup 을 사용합니다.- service 조회 결과는 /var/CKA2023/nginx.svc 에 pod name 조회 결과는 /var/CKA2023/nginx.pod 파일에 기록합니다.

1. kubectl run resolver --image=nginx --port=80 > resolver pod 생성
2. kubectl expose pod resolver --name=resolver-service --port=80 > resolver-service 구성

```
guru@k8s-master:~$ kubectl get all | grep resolver
pod/resolver                      1/1     Running           6 (24m ago)      22h
service/resolver-service    ClusterIP   10.103.96.149   <none>        80/TCP           22h
```

3. kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10.103.96.149

```
service/resolver-service    ClusterIP   10.103.96.149   <none>        80/TCP           22h   run=resolver
```

4. sudo mkdir -p /var/CKA2023 > 디렉터리 생성
5. kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10.103.96.149 > /var/CKA2023/nginx.svc = 서비스 조회결과 > /var/CKA2023/nginx.svc 에 기록
6. .kubectl get po resolver -o wide (파드 주소 확인)

```
guru@k8s-master:~$ kubectl get po resolver -o wide
NAME       READY   STATUS    RESTARTS      AGE   IP          NODE          NOMINATED NODE   READINESS GATES
resolver   1/1     Running   7 (73m ago)   4d    10.32.0.8   k8s-worker1   <none>           <none>
```

7. .kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10-32-0-6.default.pod.cluster.local > /var/CKA2023/nginx.pod > pod name 조회결과 > /var/CKA2023/nginx.pod 에 기록
8. /var/CKA2023/nginx.pod , nginx.svc 기록 확인

```
guru@k8s-master:~$ find /var/CKA2023/nginx.pod
/var/CKA2023/nginx.pod
guru@k8s-master:~$ find /var/CKA2023/nginx.svc
/var/CKA2023/nginx.svc
guru@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:      10-32-0-8.default.pod.cluster.local
Address 1: 10.32.0.8 10-32-0-8.resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
guru@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:      10.103.96.149
Address 1: 10.103.96.149 resolver-service.default.svc.cluster.local
pod "test-nslookup" deleted
```

[문제 17 emptyDir Volume]

다음 조건에 맞춰서 nginx 웹서버 pod 가 생성한 로그파일을 받아서 STDOUT 으로 출력하는 busybox 컨테이너를 운영하시오. Pod Name: **weblog** **Web container:**- Image: **nginx:1.17**- Volume mount : **/var/log/nginx**- Readwrite Log container:- Image: busybox- args: /bin/sh, -c, "tail -n+1 -f /data/access.log"- Volume mount : /data- readonly emptyDir 볼륨을 통한 데이터 공유

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

2. vi weblog.yaml > weblog.yaml 파일 수정 (args, readonly, 컨테이너 추가)

```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
    name: log
    args : ["/bin/sh, -c", "tail -n+1 -f /data/access.log"]
    volumeMounts:
    - mountPath: /data
      name: weblog
      readOnly: true
  volumes:
  - name: weblog
    emptyDir: {}
```

3. kubectl apply -f weblog.yaml > weblog.yaml 적용

```
guru@k8s-master:~$ kubectl get pod weblog
NAME     READY   STATUS             RESTARTS       AGE
weblog   1/2     CrashLoopBackOff   63 (41s ago)   21h
guru@k8s-master:~$
```

4. Weblog 파드 확인

```
guru@k8s-master:~$ kubectl describe pod weblog
Name:             weblog
Namespace:        default
```

```
Containers:
  web:
    Container ID:   containerd://cfba3d8822e5d4684873cc0286cca9d89a
    Image:          nginx:1.17
    Image ID:       docker.io/library/nginx@sha256:6fff55753e3b34e3
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 17 Jan 2025 13:47:34 +0900
    Last State:     Terminated
      Reason:       Unknown
      Exit Code:    255
      Started:      Fri, 17 Jan 2025 12:24:35 +0900
      Finished:     Fri, 17 Jan 2025 13:44:39 +0900
    Ready:          True
    Restart Count:  5
    Environment:    <none>
    Mounts:
      /var/log/nginx from weblog (rw)
```

```
log:
  Container ID:  containerd://717bdf89c6e328562944e50d72435fc6a594edbe798d25873daf6
  Image:         busybox
  Image ID:      docker.io/library/busybox@sha256:a5d0ce49aa801d475da48f8cb163c354a
  Port:          <none>
  Host Port:     <none>
  Args:
    /bin/sh, -c
    tail -n+1 -f /data/access.log
  State:         Waiting
    Reason:      CrashLoopBackOff
  Last State:    Terminated
    Reason:      StartError
    Message:     failed to create containerd task: failed to create shim task: OCI
e to start container process: exec: "/bin/sh, -c": stat /bin/sh, -c: no such file or
    Exit Code:   128
    Started:     Thu, 01 Jan 1970 09:00:00 +0900
    Finished:    Fri, 17 Jan 2025 13:59:23 +0900
  Ready:         False
  Restart Count: 58
  Environment:   <none>
  Mounts:
    /data from weblog (ro)
    /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-55hsd (ro)
```

[문제 18 HostPath Volume]

/data/cka/fluentd.yaml 파일을 만들어 새로은 Pod 생성하세요 신규생성 Pod Name: fluentd, image: fluentd, namespace: default)

위 조건을 참고하여 다음 조건에 맞게 볼륨마운트를 설정하시오.

Worker node 의 도커 컨테이너 디렉토리 : /var/lib/docker/containers 동일 디렉토리로 pod 에 마운트 하시오.

Worker node 의 /var/log 디렉토리를 fluentd Pod 에 동일이름의 디렉토리 마운트하시오.

1. .mkdir -p /data/cka > /data/cka 디렉터리 생성
2. cd /data/cka > /data/cka 디렉터리로 이동
3. kubectl run fluentd --image=fluentd --port=80 --dry-run=client -o yaml > fluentd.yaml > pod 생성

4. vi fluentd.yam; > fluentd.yaml 파일 수정

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: fluentd
spec:
  containers:
  - image: fluentd
    name: fluentd
    ports:
    - containerPort: 80
    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. kubectl apply -f fluentd.yaml > fluentd.yaml 적용

```
guru@k8s-master:~$ kubectl get po fluentd
NAME      READY   STATUS    RESTARTS      AGE
fluentd   1/1     Running   6 (54m ago)   3d22h
guru@k8s-master:~$ kubectl describe po fluentd
Name:             fluentd
Namespace:        default
Priority:         0
Service Account:  default
Node:             k8s-worker1/10.100.0.106
Start Time:       Thu, 16 Jan 2025 16:14:11 +0900
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.32.0.5
IPs:
  IP:  10.32.0.5
```

[문제 19 Persistent Volume]

pv001 라는 이름으로 size 1Gi, access mode ReadWriteMany 를 사용하여 persistent volume 을 생성합니다. volume type 은 hostPath 이고 위치는 /tmp/app-config 입니다.

1. vi pv.yaml 생성 및 수정

name:pv001

size :1Gi , access mode : ReadWriteMany

volume type : hostPath , path: /tmp/app-config

2. kubectl apply -f pv.yaml > pv.yaml 적용 및 확인

```
guru@k8s-master:~$ kubectl get pv pv001
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv001   1Gi        RWX            Retain           Available                                   44h
```

```
guru@k8s-master:~$ kubectl describe pv pv001
Name:            pv001
Labels:          <none>
Annotations:     <none>
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:
Status:          Available
Claim:
Reclaim Policy:  Retain
Access Modes:    RWX
VolumeMode:      Filesystem
Capacity:        1Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /tmp/app-config
    HostPathType:
Events:          <none>
```

학습 효과

1. **클러스터 생성, 백업, 업그레이드**

   1. ETCD 백업 및 복원 : TLS 인증서 / 키를 사용한 보안 연결 설정의 중요성 이해

   2. Cluster Upgrade : 버전 업그레이드 주요 단계 및 드레인, 언코든 등 이해

   컨트롤 플레인, 노드 구성 요소를 개별적으로 관리하는 방법 이해

2. **역할 기반 접근 제어**

   1. Role / ClusterRole 설정 : 특정 리소스 타입에 대한 세분화된 권한 설정 이해

   2. Service Account 활용 : SA 와 (Cluster)role/rolebinding 간의 관계 이해

3. **노드 관리 및 파드 스케줄링**

   노드 드레인 및 스케줄링 제어를 통한 특정 노드에서 파드를 재배치하고

   노드 스케줄링 불가능 상태로 설정하는 기술을 익힙니다.

   NodeSelector 와 같은 스케줄링 전략을 활용한 자원 배치를 이해 합니다.

4. **파드 생성 및 관리**

   1. 환경 변수, 커맨드, args 적용

   파드의 동작 방식을 커스터마이징 하는 방법을 익힙니다.

   동적 환경에서의 로그 관리 및 모니터링 기술을 이해 합니다.

   2. Static Pod 및 Multi-Container Pod

   특정 노드에서 강제적으로 실행 관리, 컨테이너 간 데이터 공유 및 작업 분리 방법 이해

5. **컨트롤러와 네트워킹**

   1. Rolling Update, Rollback :기록 가능한 업데이트를 통해 배포 이력 추적 기술 이해

   2. Service 생성 및 네트워크 노출 : ClusterIP, NodePort 차이점 이해, 적절 서비스 설계

6. **Network Policy 및 Ingress**

   1. Network Policy : ns 기반의 트래픽 제어를 구현, 보안 강화하는 방법 이해

   특정 라벨을 사용하여 제한된 접근만 허용하는 정책을 설정합니다.

   2. Ingress : 여러 서비스의 통합된 접근점을 관리하는 기술을 이해합니다.

7. **리소스 관리**

   1. EmptyDir 및 HostPath Volume : 파드 내 컨테이너 간 데이터 공유 및 호스트

   디렉터리 사용 방법을 이해 합니다.

   또한, 로그 데이터를 관리하고 외부 시스템과 연동하는 기술을 이해 합니다.

   2. Persistent Volume : 지속적인 데이터 저장소를 관리하는 방법을 이해 합니다.

   스토리지 리소스, 쿠버네티스 클러스터를 통합하는 방법을 이해 합니다.

대우 능력 개발원 FRONT_1 조

감사합니다.