

**[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**
* **Client certificate: /etc/kubernetes/pki/etcd/server.crt**
* **Client key: /etc/kubernetes/pki/etcd/server.key**

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>

1. root 계정으로 전환

# sudo -i

1. etcd-client 설치

# apt install etcd-client

1. /data 디렉터리 생성

# mkdir /data

1. 스냅샷 생성 후 저장

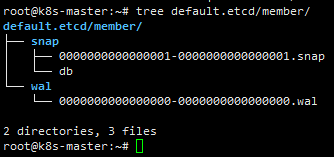
#ETCDCTL\_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save /data/etcd-snapshot.db

mkdir /data

1. 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

1. 스냅샷 복원 여부 확인



**[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

1. 업그레이드 버전 결정

# apt update

# apt-cache madison kubeadm

1. kubeadm 업그레이드 호출

# apt-mark unhold kubeadm

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

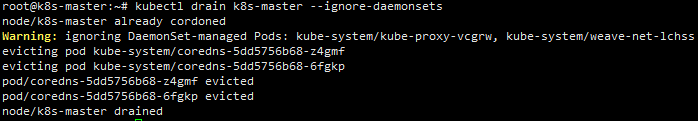
# apt-mark hold kubeadm

1. 업그레이드 버전 선택 후 실행

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

1. 노드를 예약 불가능으로 표시하고 유지 관리 준비

# kubectl drain k8s-master --ignore-daemonsets



1. 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

1. kubelet 다시 시작

# sudo systemctl daemon-reload

# sudo systemctl restart kubelet

1. 노드 차단 해제

# kubectl uncordon k8s-master

**[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

1. pod-viewer라는 이름의 Service Account 생성

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

1. Service Account 생성 확인

# kubectl get sa -n api-access

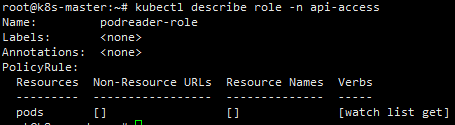


1. watch,list,get 허용하도록 Podreader-role라는 이름의 Role생성

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

1. Role 생성 확인

# kubectl describe role -n api-access



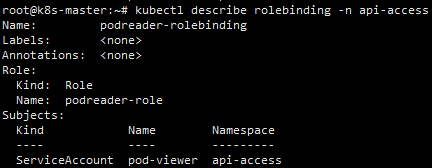
1. podreader-rolebinding라는 이름의 RoleBinding 생성

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

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

1. RoleBinding 생성 확인

# kubectl describe rolebinding -n 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**

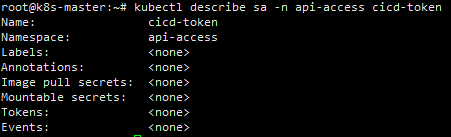
**3. ClusterRole deployment-clusterrole을 namespace api-access로 제한된 cicd-token에 바인딩**

1. cicd-tocken이라는 새로운 SerivceAccount 생성

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

1. ServiceAccount 생성 확인

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



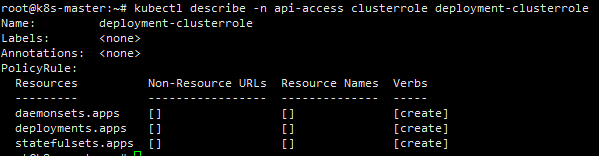
1. create가 허용되고 리소스 타입이 들어간 deployment-clusterrole라는 이름의 clusterRole생성

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

--verb=create

1. ClusterRole 생성 확인

# kubectl describe -n api-acces clusterrole deployment-clusterrole

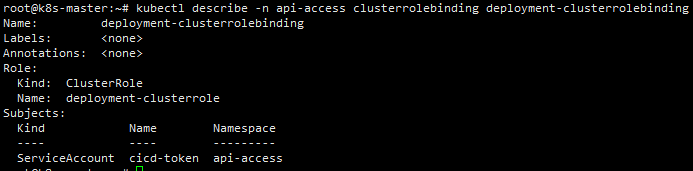


1. 생성한 clusterrole을 새 serviceaccount에 바인딩하도록 ClusterRoleBinding 생성

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

1. ClusterRoleBinding 확인

# kubectl describe -n api-access clusterrolebinding deployment-clusterrolebinding



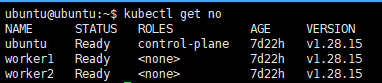
**[문제 5] 노드 관리 - 노드 비우기**

**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

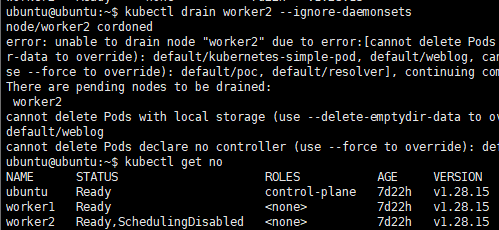


1. Worker2에서 실행 중인 모든 pod를 다른 node로 reschedule 설정

# kubectl drain worker2

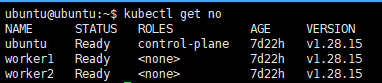
1. Worker2 노드 확인

# kubectl drain worker --ignore-daemonsets



1. Worker2 drain 설정 해제 및 노드 확인

# kubectl uncordon worker2



**[문제 6] 노드 관리 - Pod Scheduling**

**1. 다음의 조건으로 pod를 생성**

* **Name: eshop-store**
* **Image: nginx**
* **Nodeselector: disktype=ssd**

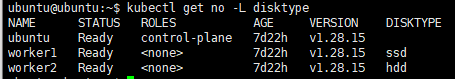
1. Worker1,2 노드에 각각 disktpye=ssd & disktype=hdd 라벨링 추가

# kubectl label node worker1 disktype=ssd

# kubectl label node worker2 disktype=hdd

1. 라벨링 추가 여부 확인

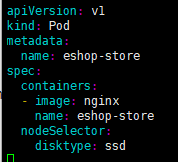
# kubectl get no -L disktype



1. eshop-store 파일 생성

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

1. eshop-store.yaml파일 수정

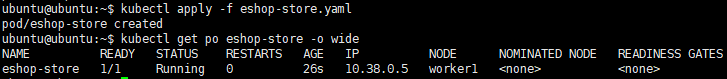


1. eshop-store.yaml파일 적용

# kubectl apply -f eshop-store.yaml

1. pod생성 확인

# kubectl get po eshop-store -o wide



**[문제 7] 파드 생성**

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

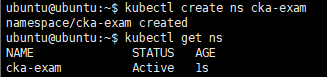
* **pod Name: pod-01**
* **image: busybox**
* **환경변수 : CERT = "CKA-cert"**
* **command: /bin/sh**
* **args: "-c", "while true; do echo $(CERT); sleep 10;done“**

1. ‘cka-exam’ namespace 생성

# kubectl create ns cka-exam

1. namespace 생성 여부 확인

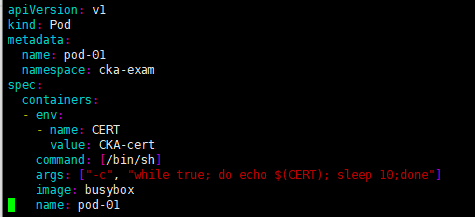
# kubectl create ns cka-exam



1. 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

1. pod-01.yaml파일 수정

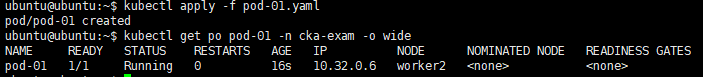


1. Pod-01.yaml파일 적용

# kubectl apply -f pod-01.yaml

1. pod생성 확인

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



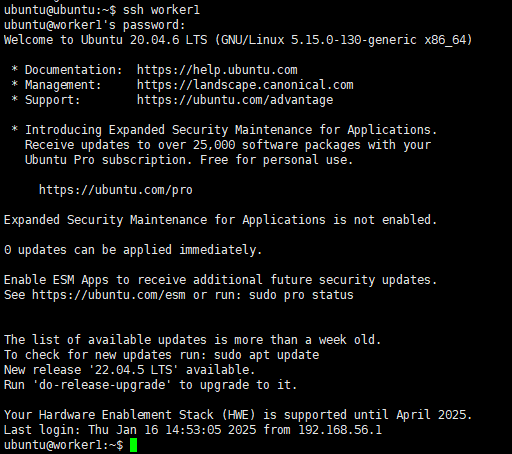
**[문제 8] 파드 생성 - Static Pod 생성**

**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

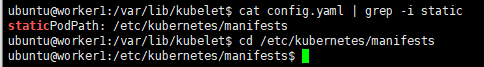


1. worker1의 static위치 확인 및 해당 위치로 이동

# cd /var/lib/kubelet

# cat config.yaml | grep -i static

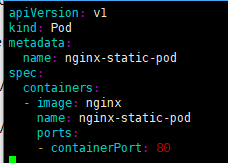
# cd /etc/kubernetes/manifests/



1. 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

1. vi nginx-static-pod.yaml파일을 조건에 맞게 수정



1. nginx-static-pod.yaml파일 적용

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

1. pod생성 확인

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



**[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



1. root 계정으로 전환

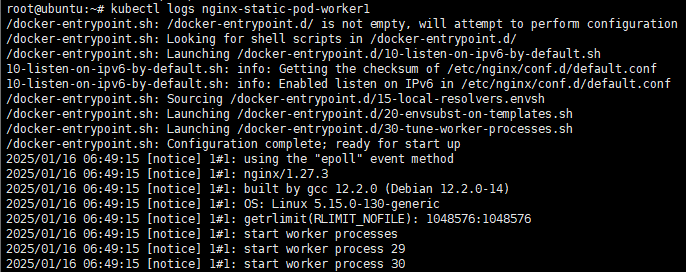
# sudo -i

1. /opt/REPORT/2023 디렉터리 생성

# mkdir -p /opt/REPORT/2023/

1. nginx-static-pod-worker1의 로그를 모니터링

# kubectl logs nginx-static-pod-worker1

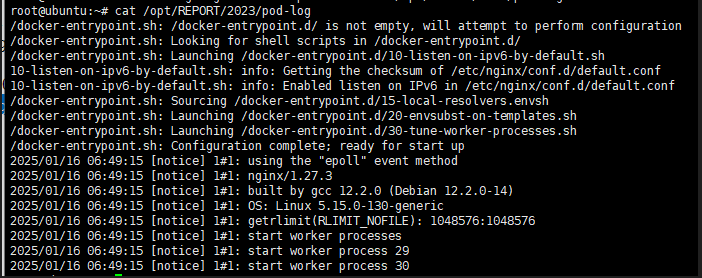


1. 추출된 결과 /opt/REPORT/2023/pod-log에 기록

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

1. 기록 여부 확인

# cat /opt/ REPORT/2023/pod-log



**[문제10] Multi Container Pod 생성**

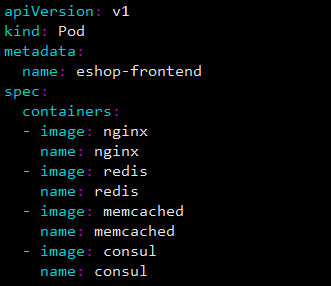
**1. 4개의 컨테이너를 동작시키는 eshop-frontend Pod를 생성**

**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

1. eshop-frontend.yaml 수정 (vi 사용)



1. eshop-frontend.yaml 적용

# kubectl apply -f eshop-fronted.yaml

1. eshop-frontend Pod 확인

# kubectl get po eshop-frontend



**[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**

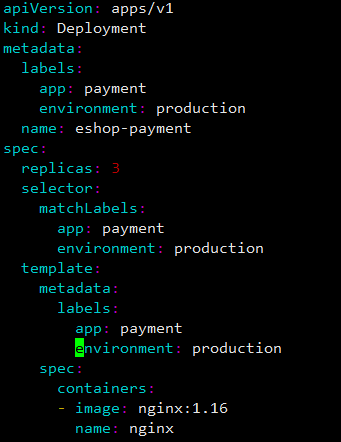
1. eshop-frontend Deployment 생성

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

1. eshop-payment.yaml 수정 (vi 사용)

2-1) label값 수정

2-2) replicas값, image 버전 확인



1. eshop-frontend.yaml 적용

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

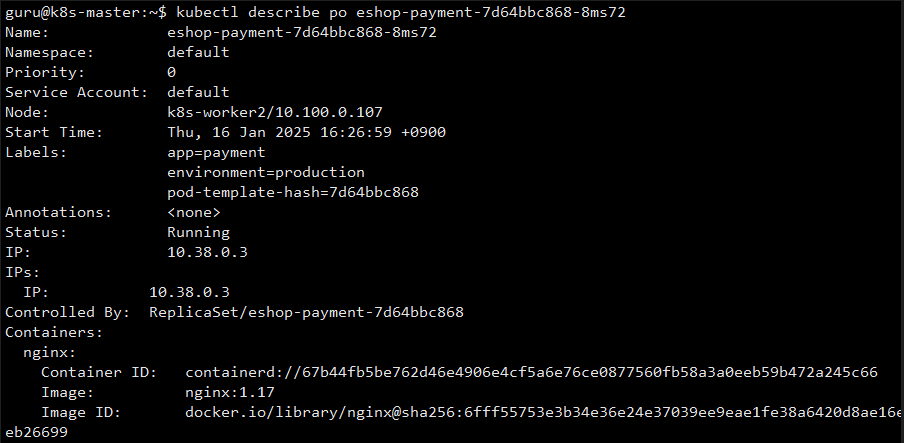
1. image 버전 업데이트

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

1. pod 및 rolling update 확인

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

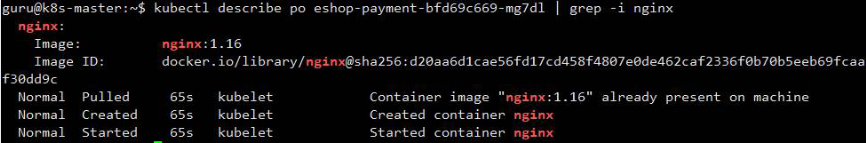
# kubectl describe po eshop-payment-7d64bbc868-8ms72



1. roll back 실행 및 확인

# kubectl rollout undo deploy eshop-payment

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

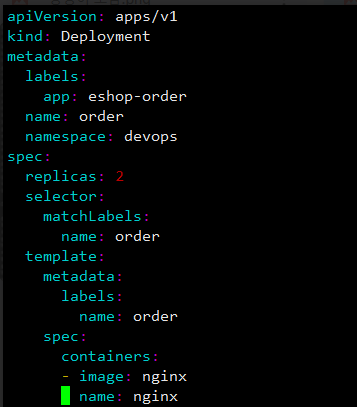
1. 'devops' namespace 생성

# kubectl create ns devops

1. ‘eshop-order’ 디플로이먼트 생성

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

1. eshop-order.yaml 파일 수정



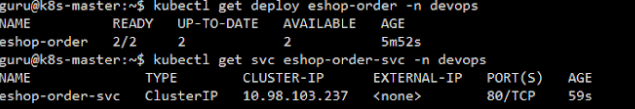
1. ‘eshop-order’ 디플로이먼트 서비스 생성

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

1. ‘eshop-order’ 디플로이먼트 및 서비스 확인

# kubectl get deploy eshop-order -n devops

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



**[문제13] NodePort**

**1. ‘front-end’ deployment를 다음 조건으로 생성**

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

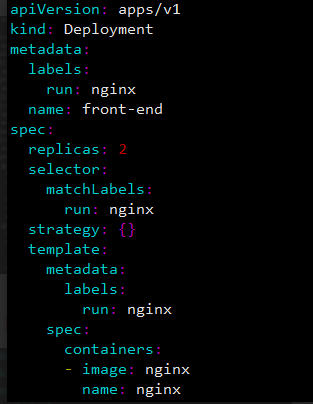
**2. 'front-end' deployment의 nginx 컨테이너를 expose하는 'front-end-nodesvc'라는 새 service를 생성**

**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

1. front-end.yaml 수정



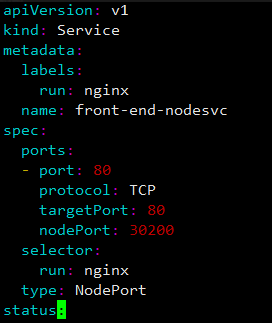
1. front-end.yaml 적용

# kubectl apply -f front-end.yaml

1. '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

1. front-end-nodesvc.yaml 파일 수정

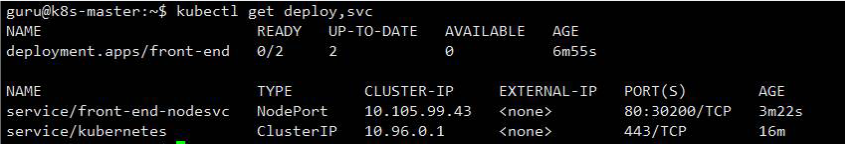


1. front-end-nodesvc.yaml 적용

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

1. ‘front-end’ 디플로이먼트 및 서비스 확인

# kubectl get deploy,svc



**[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**

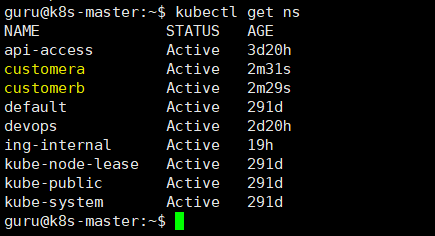
1. Kubernetes 사이트에서 [net pol] 검색
2. customera, customerb 네임스페이스 생성

# kubectl create ns customera

# kubectl create ns customerb

1. 네임스페이스 확인

# kubectl get ns



1. customera, customerb 파티션 및 라벨링

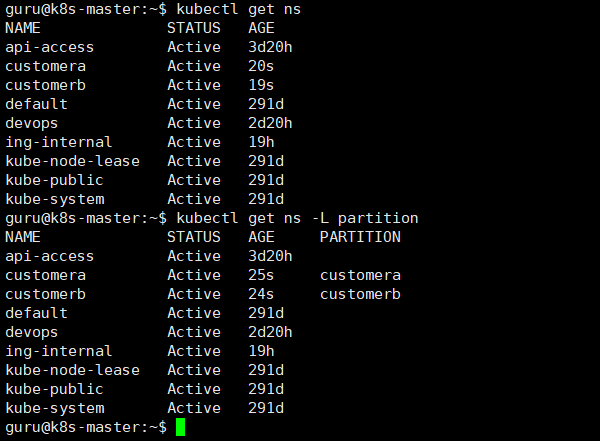
# kubectl label ns customera partition=customera

# kubectl label ns custpmerb partition=customerb

1. 네임스페이스 Partition 및 라벨 확인

# Kubectl get ns

# Kubectl get ns -L partition



1. poc 파드 생성

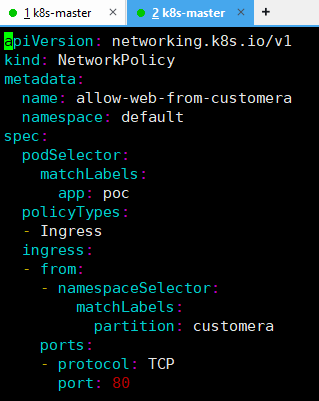
# kubectl run poc --image=nginx --port=80 --labels=app=poc

1. pod 파드 확인

# kubectl get po poc



1. netpol.yaml 생성 및 수정



1. netpol.yaml 적용  
   # kubectl apply -f netpol.yaml
2. Netpol 파드 확인

# kubectl get netpol



**[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

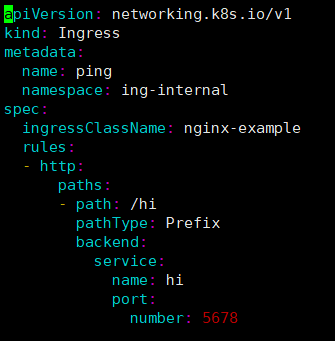
# kubectl create ns ing-internal

1. 네임스페이스 확인

# kubectl get ns ing-internal



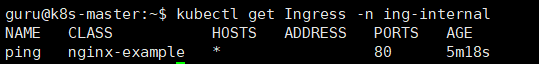
1. ingress.yaml 수정



1. ingress.yaml 파일 적용

# kubectl apply -f ingress.yaml

1. Ingress pod 확인  
   # kubectl get Ingress -n ing-internal



**[****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



1. 서비스 생성  
   # kubectl expose pod resolver --name=resolver-service --port=80
2. 서비스 확인  
   # kubectl get svc resolver-service



1. Root 전환 및 디렉터리 생성

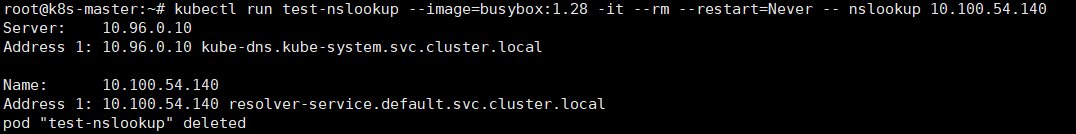
# sudo -i

# mkdir -p /var/CKA2023

1. service 조회 결과 /var/CKA2023/nginx.svc파일에 기록

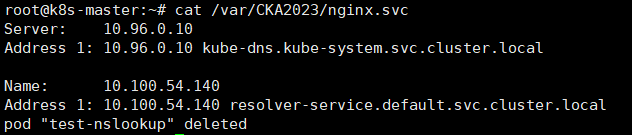
# kubectl run test-nslookup --image=busybox:1.28 -it --rm --restart=Never -- nslookup 10.100.54.140

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



1. 주소 확인

# cat /var/cka2023/nginx.svc



1. resolver 파드 ip 확인

# kubectl get po resolver -o wide



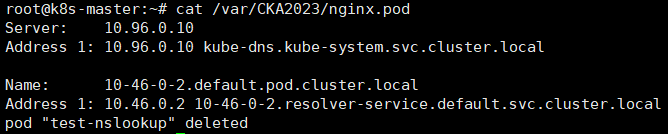
1. pod name 조회 결과 /var/CKA2023/nginx.pod 파일에 기록

# 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

1. 확인

# cat /var/CKA2023/nginx.pod



**[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**

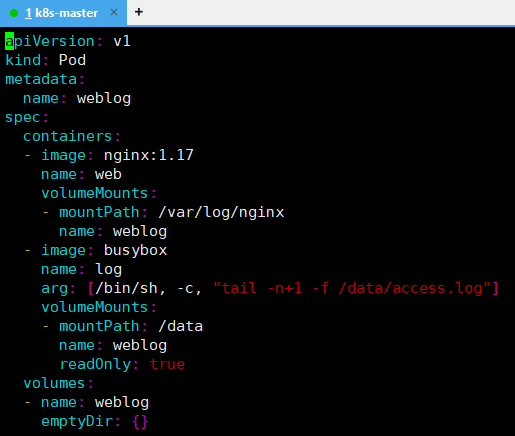
**Log container:**

* **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

1. weblog.yaml 수정



1. weblog yaml 파일 적용

# kubectl apply -f weblog.yaml

1. weblog 파드 확인

# kubectl get po weblog



**[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**

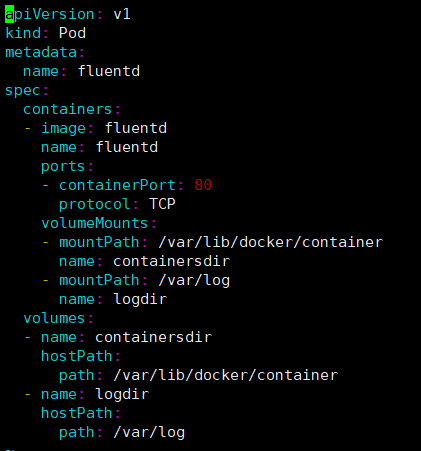
**4. Worker node의 /var/log 디렉토리를 fluentd Pod에 동일이름의 디렉토리 마운트**

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

# cd /data/cak

# vi fluentd.yaml

1. fluentd.yaml 코드 붙여넣기 및 수정



1. fluentd yaml 파일 적용

# kubectl apply -f fluentd.yaml

1. fluentd pod 확인

# kubectl get po fluentd

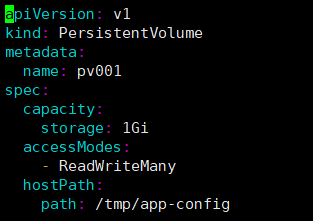


**[문제19] Persistent Volume**

**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



1. pv001.yaml 적용

# kubectl apply -f pv001.yaml

1. pv001 pv 확인

# kubectl get pv pv001

