올인원 패키지 Online.

CKA 쿠버네티스 자격증 과정

PART1 | CKA 자격증 소개
자격증 등록과 Hands-On 실습 환경 만들기

PART2 | **Kubernetes 아키텍쳐**
Etcd 백업/복구, kubernets 업그레이드, 쿠버네티스 인증 실습

PART3 | Workloads & Scheduling
Application deploy 후 scale/rollback, pod 스테줄링 실습

PART4 | Services & Networking
Service 운영과 접근제한(NetworkPolicy),Ingress 운영 실습

PART5 | Storage
StorageClass 적용한 PV, PVC 생성하여 pod에 적용하기 실습

PART6 | Troubleshooting
controller component 설정 정보 수정및 node/pod 문제해결

PART7 | 실전문제풀이
실전 문제를 직접풀어보고, 답안 리뷰 확인

Part 2 Cluster Architecture, Installation & Configuration

01 ETCD Backup & Restore
02 kubernets Upgrade
03 RBAC 인증

Part2 Cluster Architecture Installation & Configuration

01 ETCD Backup & Restore

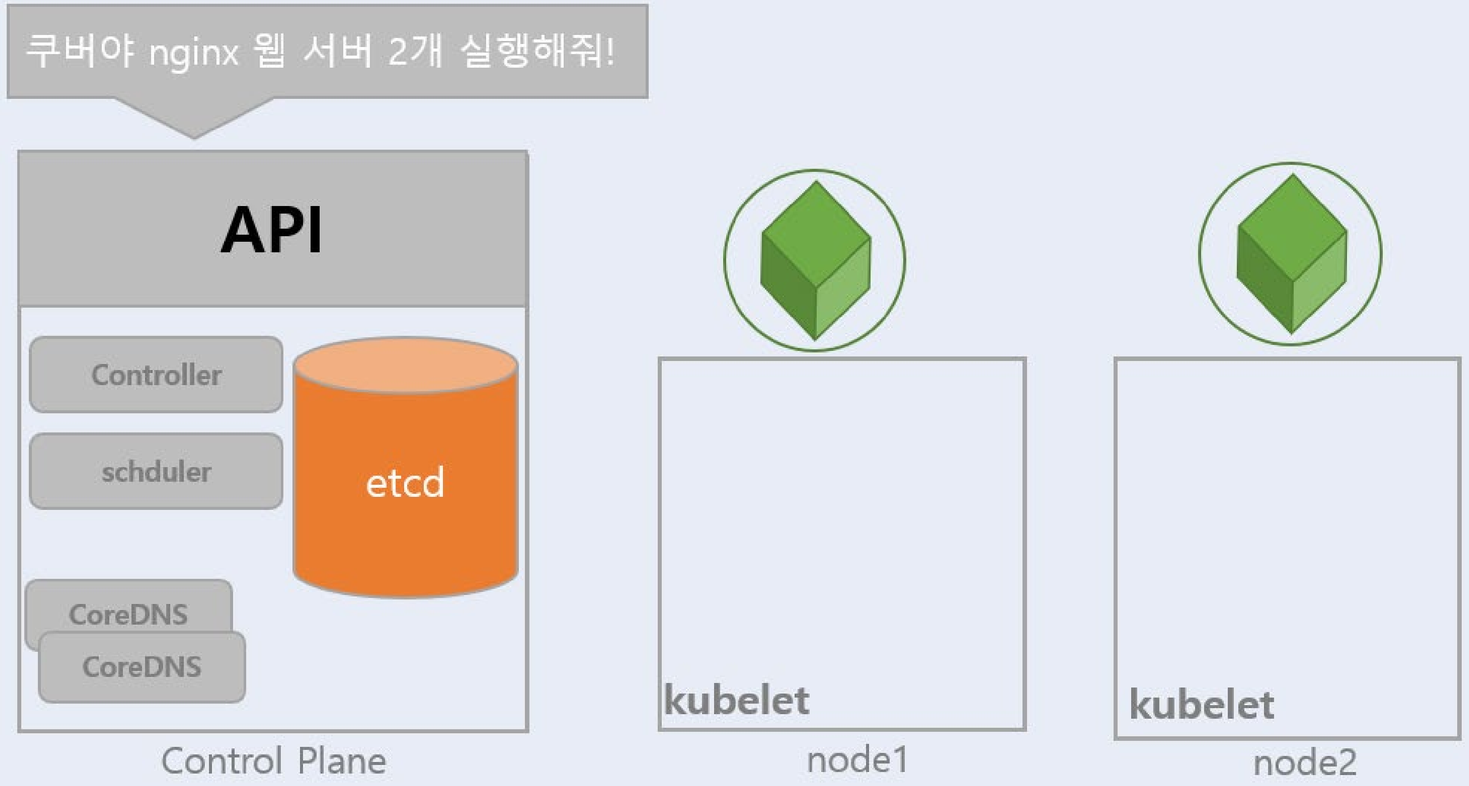
ETCD

- Coreos가 만든 분산 key:value 형태의 데이터 스토리지
- 쿠버네티스 클러스터의 정보를 저장(memory)해서 사용
- 모든 etcd 데이터는 etcd 데이터베이스 파일에 보관 : /var/lib/etcd

- etcd 관리 명령 : etcdctl
- etcdctl 설치확인

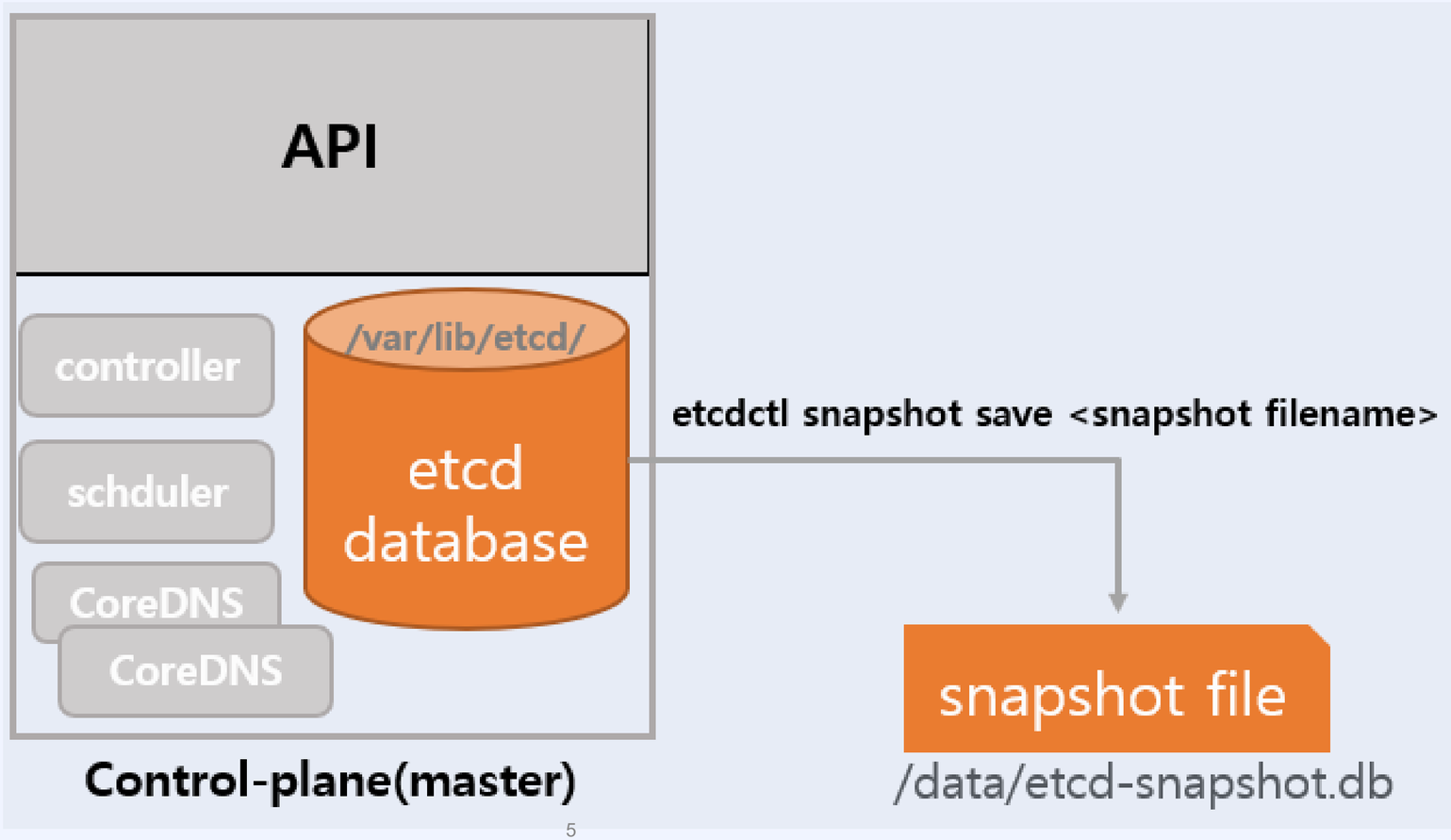
  ETCD를 호스팅 할 시스템에 ssh 로그인
  $ ssh k8s-master

  동작중인 etcd 버전과
  etcdctl 툴이 설치여부 확인
  $ etcd --version
  $ etcdctl version



ETCD backup

- mater의 장애와 같은 예기치 못한 사고로 인해 ETCD 데이터베이스가 유실될 경우를 대비해서 Backup API를 제공
- ETCD snapshot



ETCD Restore

- Snapshot으로 저장한 database파일을 동작중인 etcd에 적용하여 snapshot 생성시점으로 되돌리기
- 단계
  (1) snapshot 파일을 데이터베이스 파일로 복원
  (2) 동작중인 etcd Pod의 구성정보를 복원된 데이터베이스 위치로 수정 적용

etcd backup & restore

- ETCD를 호스팅 할 시스템에 ssh 로그인
  $ ssh k8s-master

- etcdctl 툴이 설치 여부 확인
  $ etcdctl version

- ETCD Backup
  https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#backing-up-an-etcd-cluster

```
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>
```

trusted-ca-file 확인
$ ps -ef | grep kube | grep trusted-ca-file

cert-file 확인
$ ps -ef | grep kube | grep cert-file

key-file 확인
$ ps –ef | grep kube | grep key-file

**$ sudo 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 /tmp/etcd-backup

- 현재 etcd 상태를 수정 하고 snapshot 파일을 이용해 복원했을 때 원래대로 복원되었는지 확인
  $ kubectl get pods
  $ kubectl delete deployment XXXXX
  $ kubectl get pods

- ETCD Restore
  https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#restoring-an-etcd-cluster

```
ETCDCTL_API=3 etcdctl --data-dir <data-dir-location> snapshot restore snapshotdb
```

$ sudo ETCDCTL_API=3 etcdctl --data-dir=/var/lib/etcd-new snapshot restore /tmp/etcd-backup
$ sudo tree /var/lib/etcd-new/

- etcd Pod에 복원된 etcd-data 위치를 적용하고 Pod 다시 시작
  $ **sudo vi /etc/kubernetes/manifests/etcd.yaml**
  …
  - hostPath:
    path: /var/lib/etcd-new
    type: DirectoryOrCreate
  name: etcd-data

#docker 명령으로 etcd가 restart 되었는지 확인
$ **sudo docker ps -a | grep etcd**

복원되었는지 확인
$ **kubectl get pods**

문제1:
ETCD Backup

■ 작업 클러스터 : k8s

https://127.0.0.1:2379에서 실행 중인 etcd의 snapshot을 생성하고 snapshot을 /data/etcd-snapshot.db에 저장합니다.
그런 다음 /data/etcd-snapshot-previous.db에 있는 기존의 이전 스냅샷을 복원합니다.
etcdctl을 사용하여 서버에 연결하기 위해 다음 TLS 인증서/키가 제공됩니다.

- CA certificate: /etc/kubernetes/pki/etcd/ca.crt
- Client certificate: /etc/kubernetes/pki/etcd/server.crt
- Client key: /etc/Kubernetes/pki/etcd/server.key

```
$ sudo 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
```

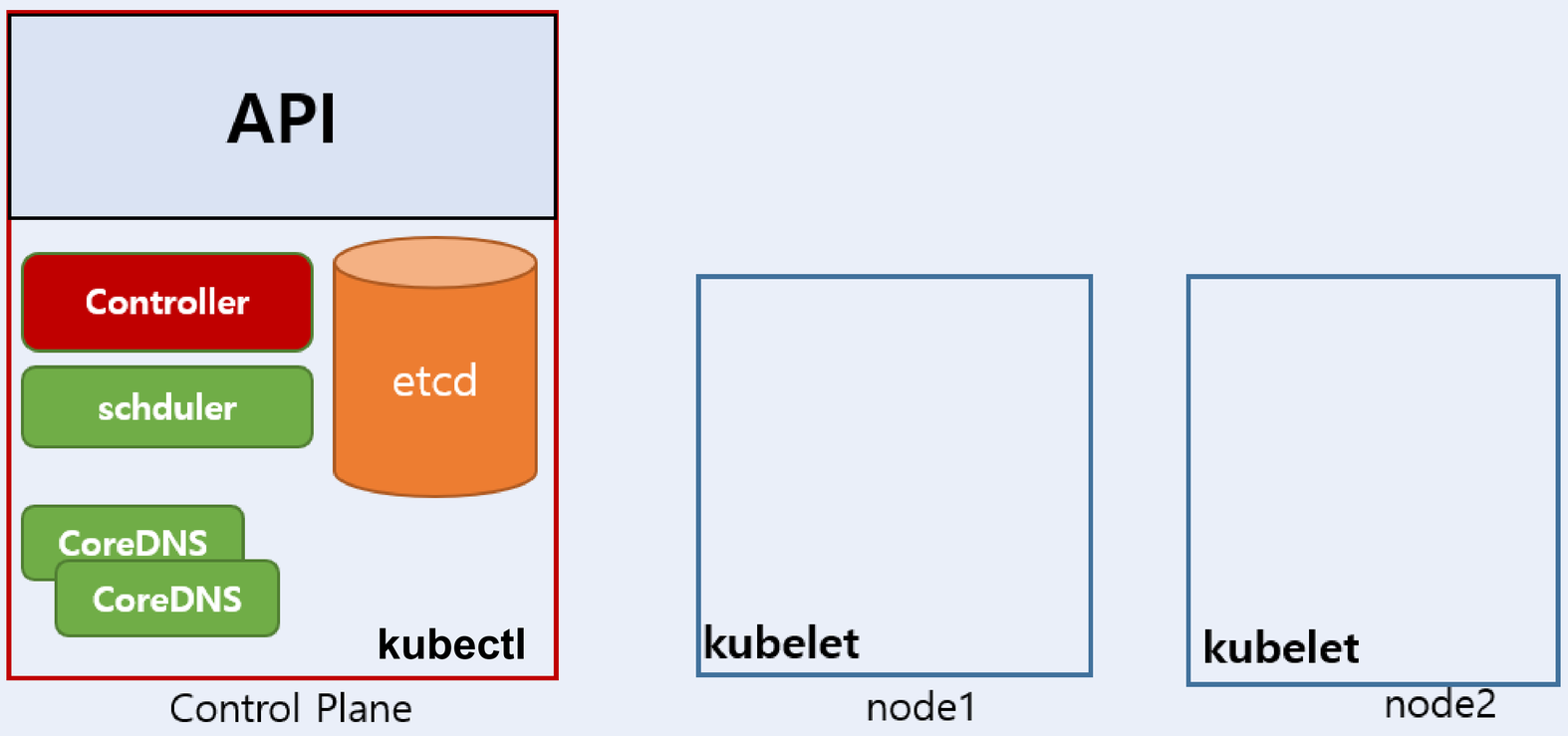
```
$ sudo ETCDCTL_API=3  etcdctl --data-dir=/var/lib/etcd-previous
snapshot restore /data/etcd-snapshot-previous.유
$ sudo tree /var/lib/etcd-previous/

$ sudo vi /etc/kubernetes/manifests/etcd.yaml
...
 - hostPath:
     path: /var/lib/etcd-previous
     type: DirectoryOrCreate
   name: etcd-data

$ sudo docker ps -a | grep etcd
```

Part2 Cluster Architecture Installation & Configuration

02 Kubernets Upgrade

Kubernetes Cluster upgrade

- Kubernetes Packages
  - kubeadm : 클러스터를 부트스트랩하는 명령
  - kubelet : Pod와 Container시작과 같은 작업을 수행하는 컴포넌트
  - kubectl : 클러스터와 통신하기 위한 커맨드 라인 유틸리티



Control plane upgrade

- Kubernetes cluster upgrade
  - kubeadm, kubelet, kubectl을 1.22.4에서 1.23.3 버전으로 업그레이드

- Control-plane Upgrade
  1. Upgrade할 master에 접속 : ssh <master>
  2. 업그레이드 할 버전 확인

     sudo yum list --showduplicates kubeadm --disableexcludes=kubernetes | tail -5

  3. kubeadm 업그레이드

     sudo yum install -y kubeadm-1.23.3-0 --disableexcludes=kubernetes
     kubeadm version
     sudo kubeadm upgrade plan v1.23.3
     sudo kubeadm upgrade apply v1.23.3

  4. 노드 드레인 : console이나 master에서 실행

     kubectl drain hk8s-m --ignore-daemonsets

  5. kubelet과 kubectl 업그레이드

     sudo yum install -y kubelet-1.23.3-0 kubectl-1.23.3-0 --disableexcludes=kubernetes
     sudo systemctl daemon-reload
     sudo systemctl restart kubelet

  6. 노드 uncordon

     kubectl uncordon hk8s-m

Node upgrade

- Kubernetes cluster upgrade
  - kubeadm, kubelet, kubectl을 1.22.4에서 1.23.3 버전으로 업그레이드

- Worker node Upgrade
  1. Upgrade할 node에 접속 : ssh <node>
  2. kubeadm 업그레이드
     sudo yum install -y kubeadm-1.23.3-0 --disableexcludes=kubernetes
  3. "kubeadm upgrade" 호출
     sudo kubeadm upgrade node
  4. 노드 드레인
     kubectl drain <node> --ignore-daemonsets
  5. kubelet, kubeadm 업그레이드
     sudo yum install -y kubelet-1.23.3-0 kubectl-1.23.3-0 --disableexcludes=kubernetes
  6. 노드 uncordon
     kubectl uncordon <node>

문제2:
Cluster upgrade

■ 작업 클러스터 : k8s

마스터 노드의 모든 Kubernetes control plane및 node 구성 요소를 버전 1.23.3으로 업그레이드합니다.
master 노드를 업그레이드하기 전에 drain 하고 업그레이드 후에 uncordon해야 합니다.

1. Control-plane: cluster upgrade를 1.23.3 버전으로 업그레이드

```
# kubeadm 업그레이드
sudo yum install -y kubeadm-1.23.3-0 --disableexcludes=kubernetes
kubeadm version
sudo kubeadm upgrade plan v1.23.3
sudo kubeadm upgrade apply v1.23.3

# 노드 드레인
kubectl drain hk8s-m --ignore-daemonsets

# kubelet과 kubectl 업그레이드
sudo yum install -y kubelet-1.23.3-0 kubectl-1.23.3-0 --
disableexcludes=kubernetes
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# 노드 uncordon
kubectl uncordon hk8s-m
```

2. Worker Node upgrade

```
# kubeadm 업그레이드
sudo yum install -y kubeadm-1.23.3-0 --disableexcludes=kubernetes
sudo kubeadm upgrade node

# 노드 드레인
kubectl drain k8s-worker2 --ignore-daemonsets

# kubelet과 kubectl 업그레이드
sudo yum install -y kubelet-1.23.3-0 kubectl-1.23.3-0 --
disableexcludes=kubernetes
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# 노드 uncordon
kubectl uncordon k8s-worker2
```

Part2 Cluster Architecture Installation & Configuration

03 RBAC 인증

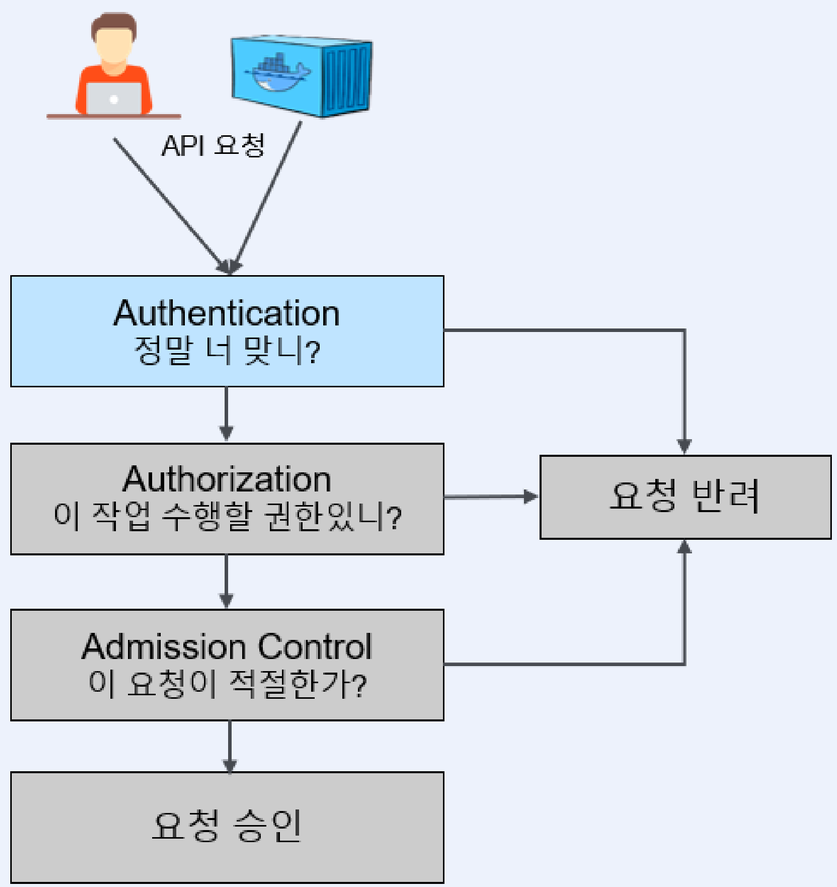
API 인증: RBAC

- API 서버에 접근하기 위해서는 인증작업 필요
- Role-based access control(RBAC. 역할기반 액세스 제어)
  사용자의 역할에 따라 리소스에 대한 접근 권한를 가짐
- User : 클러스터 외부에서 쿠버네티스를 조작하는 사용자 인증

```
$ cat .kube/config
..
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
..
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRU--인증서
```

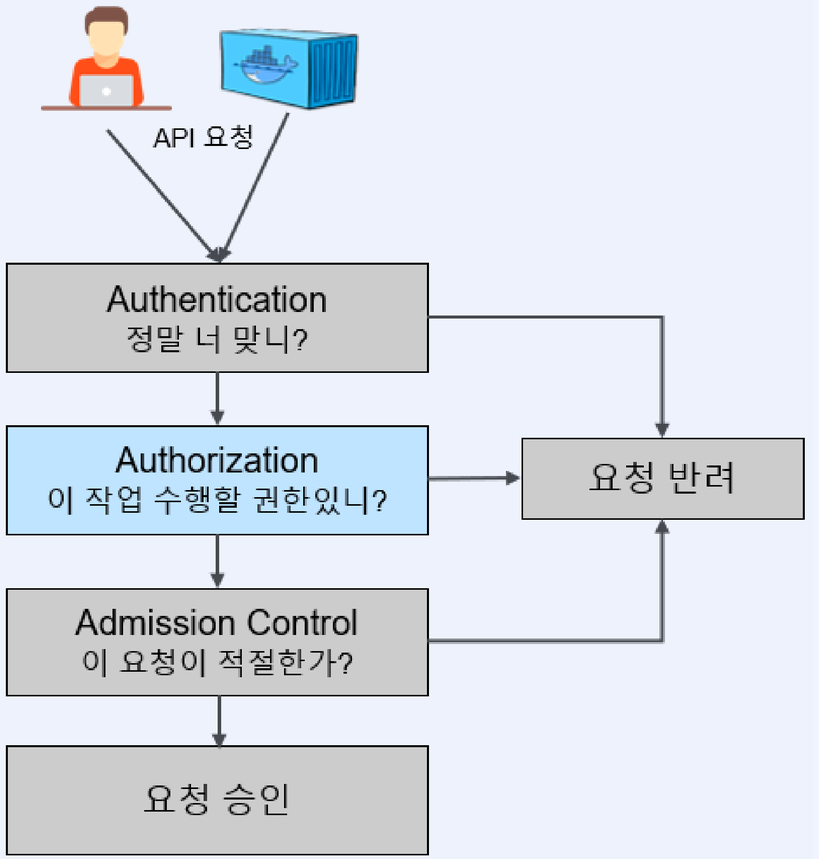
- Service Account : Pod가 쿠버네티스 API를 다룰 때 사용하는 계정

```
$ kubectl get pods eshop-cart-app -o yaml | grep -i serviceaccount
$ kubectl get secrets default-token-gvdn7
```

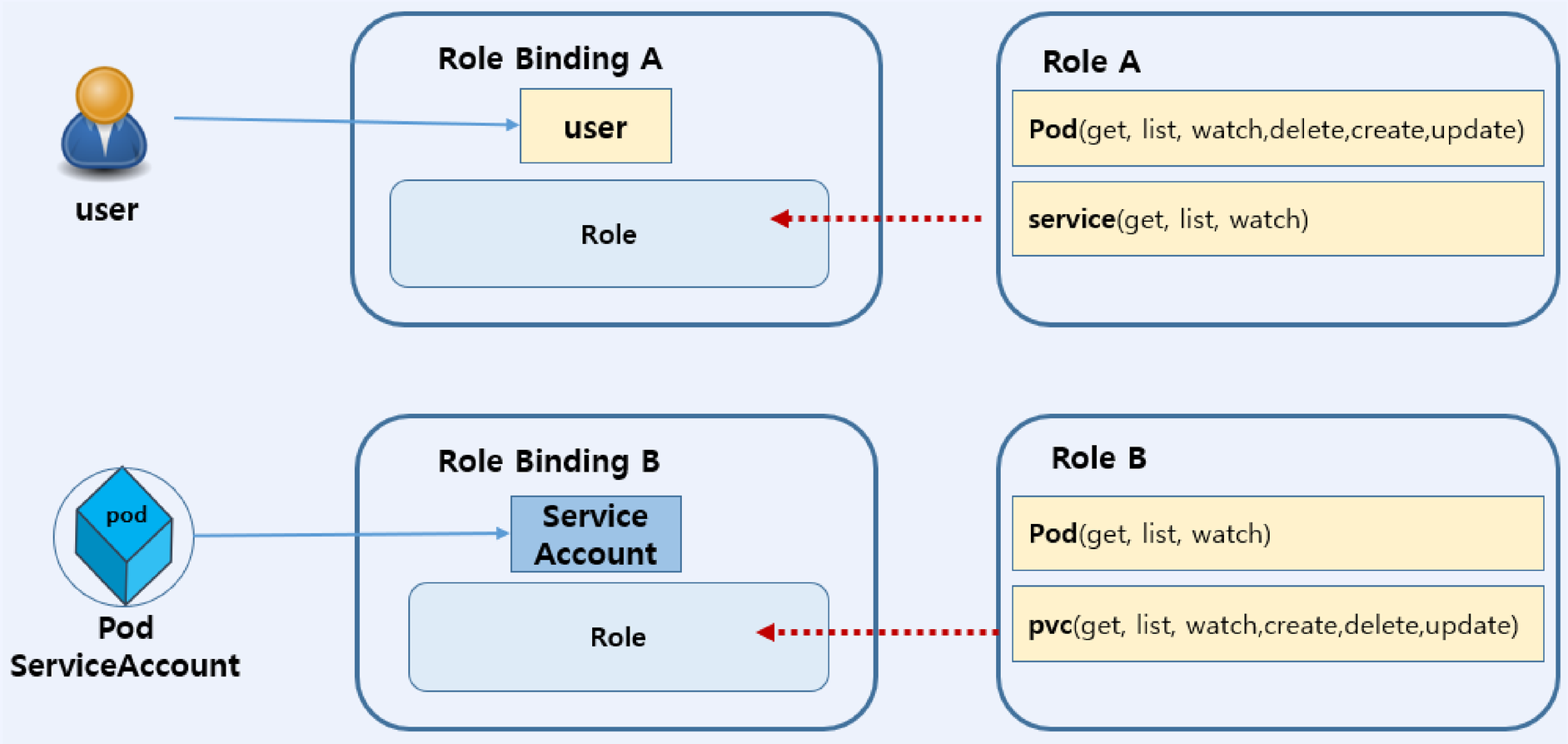


Role & RoleBinding 를 이용한 권한 제어

- 특정 유저나 ServiceAccount가 접근하려는 API에 접근 권한을 설정.
- 권한 있는 User만 접근하도록 허용
- 권한제어
- Role
  - 어떤 API를 이용할 수 있는지의 정의
  - 쿠버네티스의 사용권한을 정의
  - 지정된 네임스페이스에서만 유효
- RoleBinding
  - 사용자/그룹 또는 Service Account와 role을 연결



Role & RoleBinding 를 이용한 권한 제어



Role

- Role
  - 어떤 API를 사용할 수 있는지 권한정의. 지정된 네임스페이스에서만 유효
- Role 예제 : default 네임스페이스의 Pod에 대한 get, watch, list 할 수 있는 권한

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  verbs: ["get", "watch", "list"]
```

verb	의미
create	새로운 리소스 생성
get	개별 리소스 조회
list	여러 건의 리소스 조회
update	기존 리소스 내용 전체 업데이트
patch	기존 리소스 중 일부 내용 변경
delete	개별 리소스 삭제
deletecollection	여러 리소스 삭제

  - pod는 코어 API이기 때문에 apiGroups를 따로 지정하지 않는다. 만약 리소스가 job이라면 apiGroups에 "batch"를 지정
  - resources에는 pods, deployments, services 등 사용할 API resource들을 명시한다.
  - verbs에는 단어 그대로 나열된 API 리소스에 허용할 기능을 나열한다.

RoleBinding

- RoleBinding
  - 사용자/그룹 또는 Service Account와 role을 연결
- RoleBinding 예제 : default 네임스페이스에서 유저 jane 에게 pod-reader의 role을 할당

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

유저 jane 에게 앞서 정의한 pod-reader의 role을 할당

serviceAccount 적용 예:

```
subjects:
- kind: ServiceAccount
  name: podman
  apiGroup: rbac.authorization.k8s.io
```

앞서 정의한 Role을 명세
apiGroup에는 rbac api를 사용하기 때문에 rbac.authorization.k8s.io로 설정

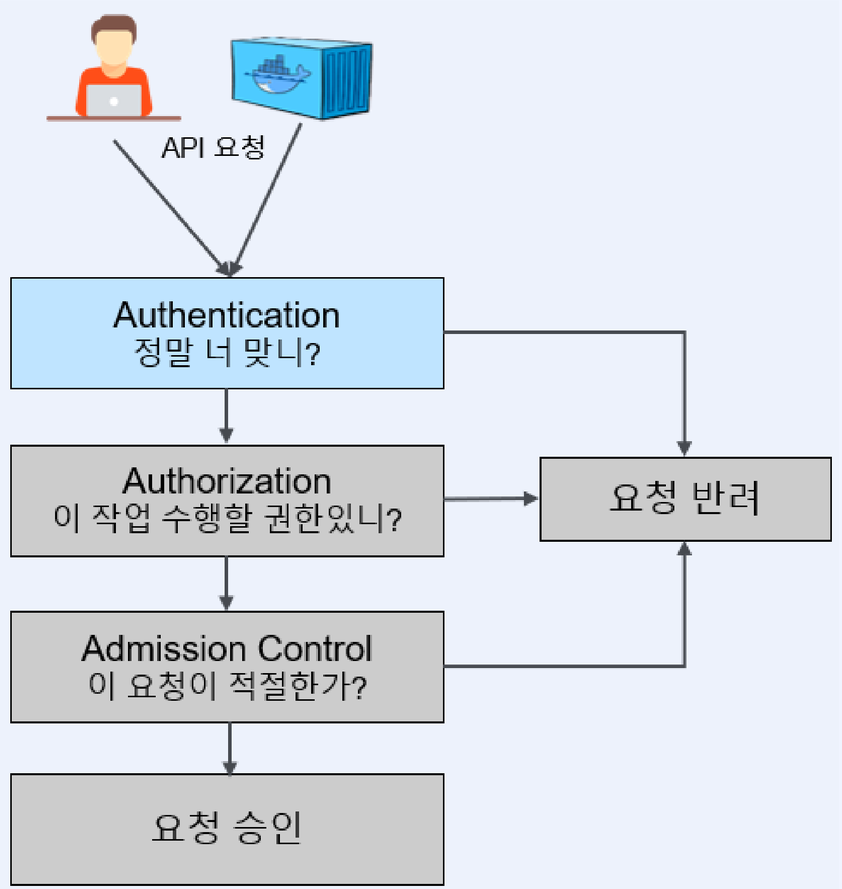
API 인증

- API 서버에 접근하기 위해서는 인증작업 필요
- User : 클러스터 외부에서 쿠버네티스를 조작하는 사용자 인증

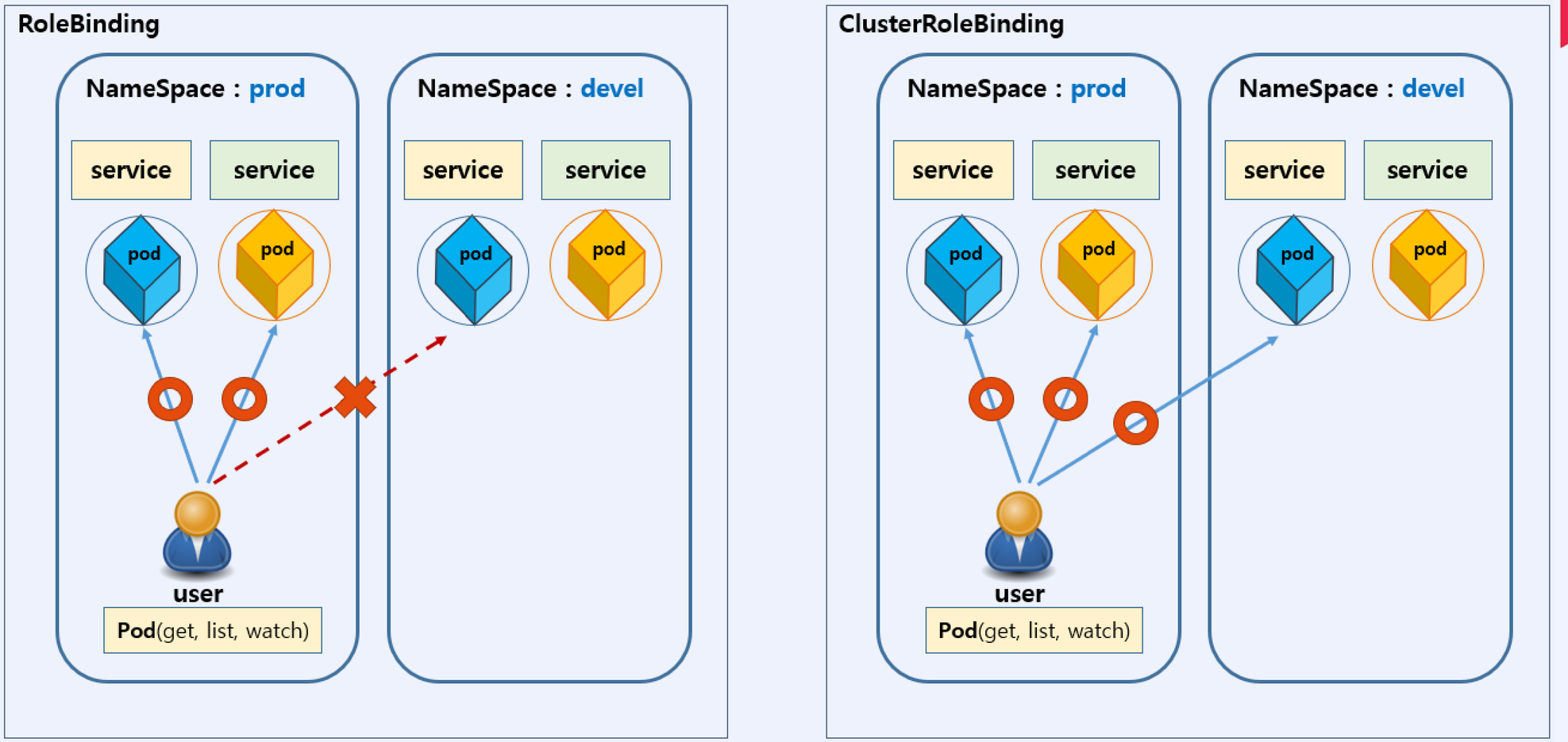
```
$ cat .kube/config
..
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
..
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRU--인증서
```

- Service Account : Pod가 쿠버네티스 API를 다룰 때 사용하는 계정

```
$ kubectl get pods eshop-cart-app -o yaml | grep -i serviceaccount
 serviceAccount: default
  serviceAccountName: default
      - serviceAccountToken:
$ kubectl get secrets default-token-gvdn7
NAME                  TYPE                                  DATA   AGE
default-token-gvdn7   kubernetes.io/service-account-token   3      62d
```



Cluster Role과
Cluster Role Binding



Cluster Role과 Cluster Role Binding

- ClusterRole
  - 어떤 API를 사용할 수 있는지 권한 정의. 클러스터 전체(전체 네임스페이스)에서 유효
  - ClusterRole 예제 : 전체 네임스페이스의 Secret에 대한 get, watch, list 할 수 있는 권한

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Cluster Role과 Cluster Role Binding

- ClusterRoleBinding
  - 사용자/그룹 또는 Service Account와 role을 연결
  - ClusterRoleBinding 예제 : manager 그룹의 모든 사용자가 모든 네임스페이스의 secret을 읽을 수 있도록 구성

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

→ manager 그룹

→ ClusterRole

문제3:
ServiceAccount, Role, RoleBinding

■ 작업 클러스터 : k8s

- 애플리케이션 운영중 특정 namespace의 Pod들을 모니터할수 있는 서비스가 요청되었습니다. api-access 네임스페이스의 모든 pod를 view할 수 있도록 다음의 작업을 진행하시오.

  - api-access라는 새로운 namespace에 pod-viewer라는 이름의 Service Account를 만듭니다.
  - podreader-role이라는 이름의 Role과 podreader-rolebinding이라는 이름의 RoleBinding을 만듭니다.
  - 앞서 생성한 ServiceAccount를 API resource Pod에 대하여 watch, list, get을 허용하도록 매핑하시오.

```
$ kubectl create namespace api-access
$ kubectl create serviceaccount  pod-viewer -n api-access
$ kubectl get serviceaccounts  --namespace api-access

$ kubectl create role podreader-role  --verb=watch --verb=list --verb=get --resource=pods --namespace api-access

$ kubectl create rolebinding podreader-rolebinding --role=podreader-role --serviceaccount=api-access:pod-viewer -n api-access
$ kubectl describe -n api-access rolebindings.rbac.authorization.k8s.io podreader-rolebinding
```

문제4:
ServiceAccount, ClusterRole,
ClusterRoleBinding

■ 작업 클러스터 : k8s

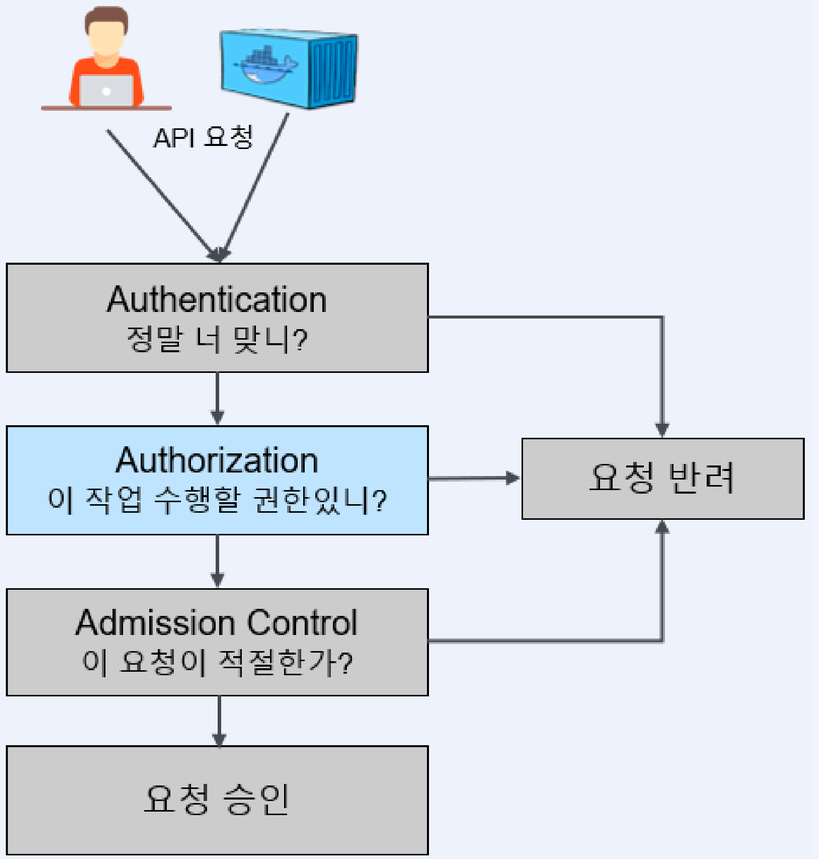
- 작업 Context에서 애플리케이션 배포를 위해 새로운 ClusterRole을 생성하고 특정 namespace의 ServiceAccount를 바인드하시오.

  - 다음의 resource type에서만 Create가 허용된 ClusterRole deployment-clusterrole을 생성합니다.
    Resource Type: Deployment StatefulSet DaemonSet
  - 미리 생성된 namespace api-access 에 cicd-token이라는 새로운 ServiceAccount를 만듭니다.
  - ClusterRole deployment-clusterrole을 namespace api-access 로 제한된 새 ServiceAccount cicd-token에 바인딩하는 합니다.

```
$ kubectl create clusterrole deployment-clusterrole --verb=create --resource=deployment,statefulset,daemonset
$ kubectl get clusterrole deployment-clusterrole

$ kubectl create serviceaccount cicd-token --namespace=api-access
$ kubectl get serviceaccounts --namespace api-access

$ kubectl create clusterrolebinding deployment-clusterrolebinding --clusterrole=deployment-clusterrole --serviceaccount=api-access:cide-token --namespace=api-access
$  kubectl describe clusterrolebindings deployment-clusterrolebinding
```

Role & RoleBinding 를 이용한 권한 제어

- 특정 유저나 ServiceAccount가 접근하려는 API에 접근 권한을 설정.
- 권한 있는 User만 접근하도록 허용
- 권한제어
- Role
  - 어떤 API를 이용할 수 있는지의 정의
  - 쿠버네티스의 사용권한을 정의
  - 지정된 네임스페이스에서만 유효
- RoleBinding
  - 사용자/그룹 또는 Service Account와 role을 연결



문제5:
User, ClusterRole, ClusterRoleBinding

■ 작업 클러스터 : k8s

CSR(Certificate Signing Request)를 통해 app-manager 인증서를 발급받은 user app-manager 에게 cluster내 모든 namespace의 deployment, pod, service 리소스를 create, list, get, update, delete 할 수 있는 권한을 할당하시오.

- user name : app-manager
- certificate name: app-manager
- clusterRole name : app-access
- clusterRoleBinding name: app-access-binding

```
$ openssl genrsa -out app-manager.key 2048
$ openssl req -new -key app-manager.key -out app-manager.csr -subj "/CN=app-manager"

$ cat app-manager.csr | base64 | tr -d "\n"
$ vi app-manager.yml
$ kubectl apply -f app-manager.yaml

$ kubectl get csr app-manager -o jsonpath='{.status.certificate}'| base64 -d > app-manager.crt
$ kubectl certificate approve app-manager
$ kubectl get csr
$ kubectl get csr/app-manager -o yaml
```

```
$ kubectl create clusterrole app-access --verb=create,list,get,update,delete --resource=deployment,pod,service
$ kubectl create clusterrolebinding app-access-binding --clusterrole=app-access --user=app-manager

# Add to kubeconfig
$ kubectl config set-credentials app-manager --client-key=app-manager.key --client-certificate=app-manager.crt --embed-certs=true
$ kubectl config set-context app-manager --cluster=kubernetes --user=app-manager
$ kubectl config use-context app-manager
```