# k8s 기반 CI/CD 자동화
## ( 내 PC로 실습하는 )

## 11. CD를 위한 k8s - Part4

# 1. k8s의 인증, 인가 개요

## ❖인증(Authentication)과 인가(Authorization)

- 인증 : 보안주체(Principal: 사용자, 그룹 등)가 자신의 신원을 확인하는 프로세스
- 인가 : 인증된 보안주체에 대해 액세스 권한을 부여하는 프로세스

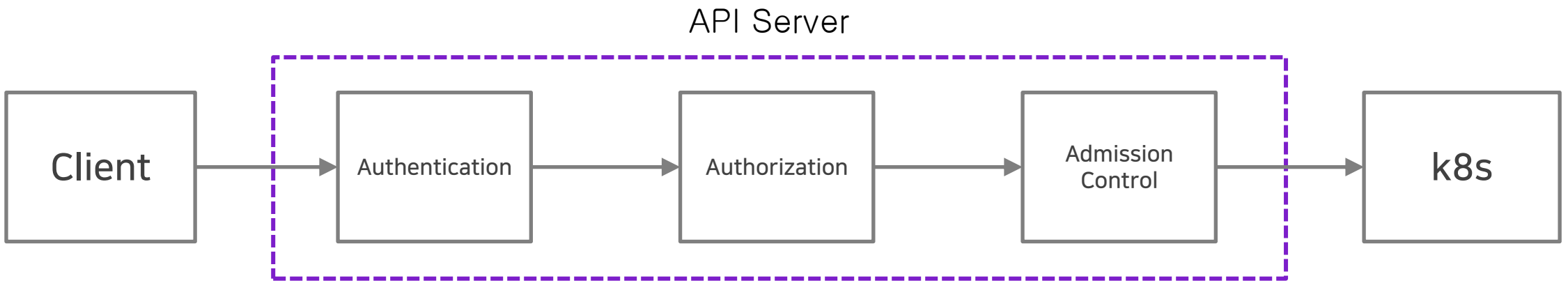
## ❖k8s에서의 인증, 인가

- k8s에는 보안주체라는 개념은 존재하지만 사용자 정보를 저장하고 관리하는 기능은 없음
  - 즉 인증 기능이 없다는 뜻
  - 사용자 관리, 인증 기능은 k8s 클러스터를 운영하는 환경의 IDP(Identity Provider)를 이용하라는 것
- 인증 방법
  - 권장하지 않는 방법 : 자격증명 갱신이 어렵고, 외부시스템 연동이 불가능
    - X.509 Client Certificate : 사용자 클라이언트 인증서를 이용한 인증
    - Static Token : 사용자 토큰을 파일로 만들어 API 서버에 등록해 두고 이를 이용한 인증
  - 권장하는 방식
    - Webhook 방식
      - » Keycloak, LDAP, AWS IAM과 같은 외부 인증 서버에 HTTP Post로 토큰 검증을 요청하는 방식
    - OIDC(OpenID Connect) 방식
      - » Google, Keycloak, Azure AD 등과 연동 하는 방식
    - ServiceAccount JWT 이용 : ServiceAccount에 대한 JWT를 생성하여 사용하는 방식, 주로 Pod간 통신에 사용함

# 1. k8s의 인증, 인가 개요

## ❖K8s의 접근 제어 방식

- Authentication: 인증. 사용자, 클라이언트의 신원 확인
- Authorization: 인가. 권한 확인, 권한 부여
- Admission Control
  - API 서버로 들어온 요청이 etcd에 저장되거나 클러스터 리소스에 반영되기 전에 이것을 가로채어 검사하거나 변경하는 단계.
  - 인증과 권한 부여가 끝난 후 리소스에 대한 접근을 최종적으로 허용할지 거부할지를 결정함.



# 1. k8s의 인증, 인가 개요

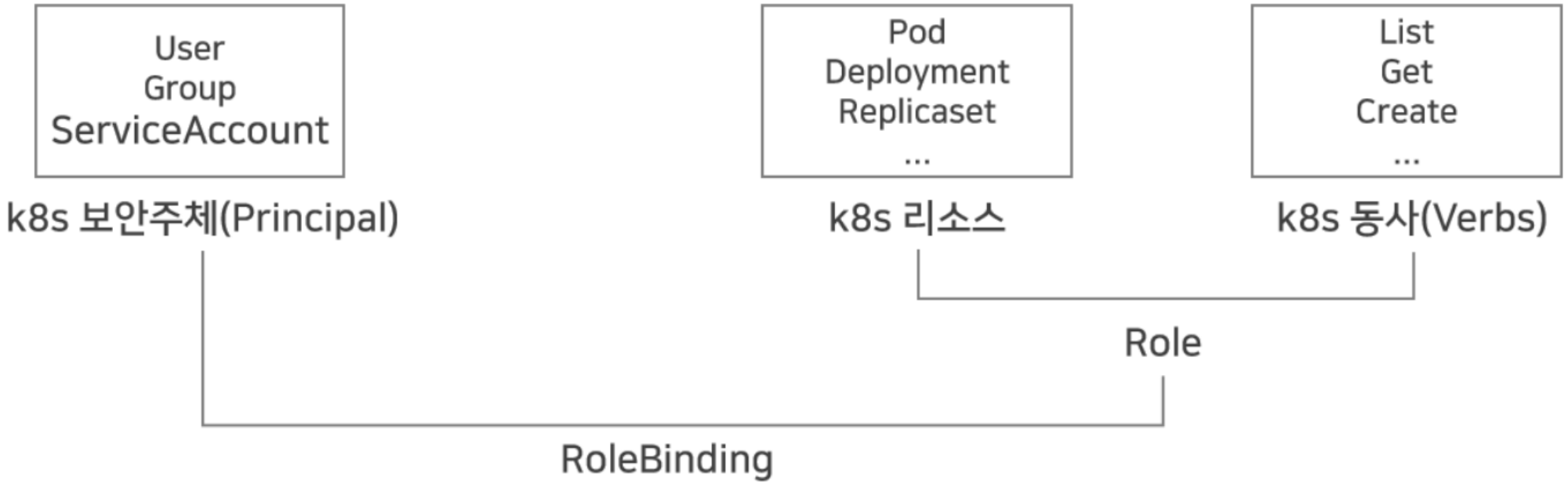
## ❖k8s의 보안 주체(Principal)

- User
  - k8s에서는 자체적인 user 관리 기능이 없음.
  - API 서버는 사용자의 실제 이름, 이메일 주소, 속성 등의 자세한 정보를 알 필요가 없음
    - 단지 user를 단순한 식별자로 사용하여 RBAC 규칙과 비교하여 권한부여 여부를 결정함
  - API 서버는 사용자가 어떤 방식으로 생성, 관리되는지 알 필요가 없음.
    - LDAP인지 OIDC 토큰인지 알 필요 없음
- Group
  - 여러 User, ServiceAccount를 묶어서 관리하기 위한 논리적인 그룹
  - RBAC Role을 group 과 바인딩 하여 권한을 부여할 수 있음
  - k8s API 서버 또는 연동된 외부 인증 시스템에 의해 정의되어짐
- ServiceAccount
  - k8s 클러스터 내부에서 실행되는 프로세스를 위한 자격증명
  - 특정 네임스페이스에 종속적임
  - ServiceAccount는 하나의 Secret과 연결될 수 있으며, Secret에는 JWT 토큰이 포함됨.
    - Secret을 생성하지 않고 ServiceAccount를 이용해 동적으로 JWT 토큰을 생성할 수 있음

# 1. k8s의 인증, 인가 개요

## ❖k8s의 인가 방식

- RBAC Role
  - 어느 리소스에 어떠한 권한을 부여할 것인가를 정의한 객체
- RBAC RoleBinding
  - RBAC Role과 k8s 보안주체를 연결하는 객체



# 2. kube config

## ❖kube config란?

- k8s 클러스터에 접근하기 위한 클라이언트의 설정 파일
- 기본 경로 : ~/.kube/config

## ❖ kube config 구조

- clusters
  - 연결하려는 클러스터의 API 서버 주소와 인증서 정보들을 지정함
- users
  - 클러스터에 연결하려는 사용자 정보들을 지정함
  - 인증서, 토큰 등 다양한 방법이 존재함
- contexts
  - cluster + user 조합의 정보.
  - 클라이언트에서 어떤 컨텍스트를 사용할 것인지를 지정하여 클러스터에 접근함
    - kubectl config use-context  컨텍스트명

# 2. kube config

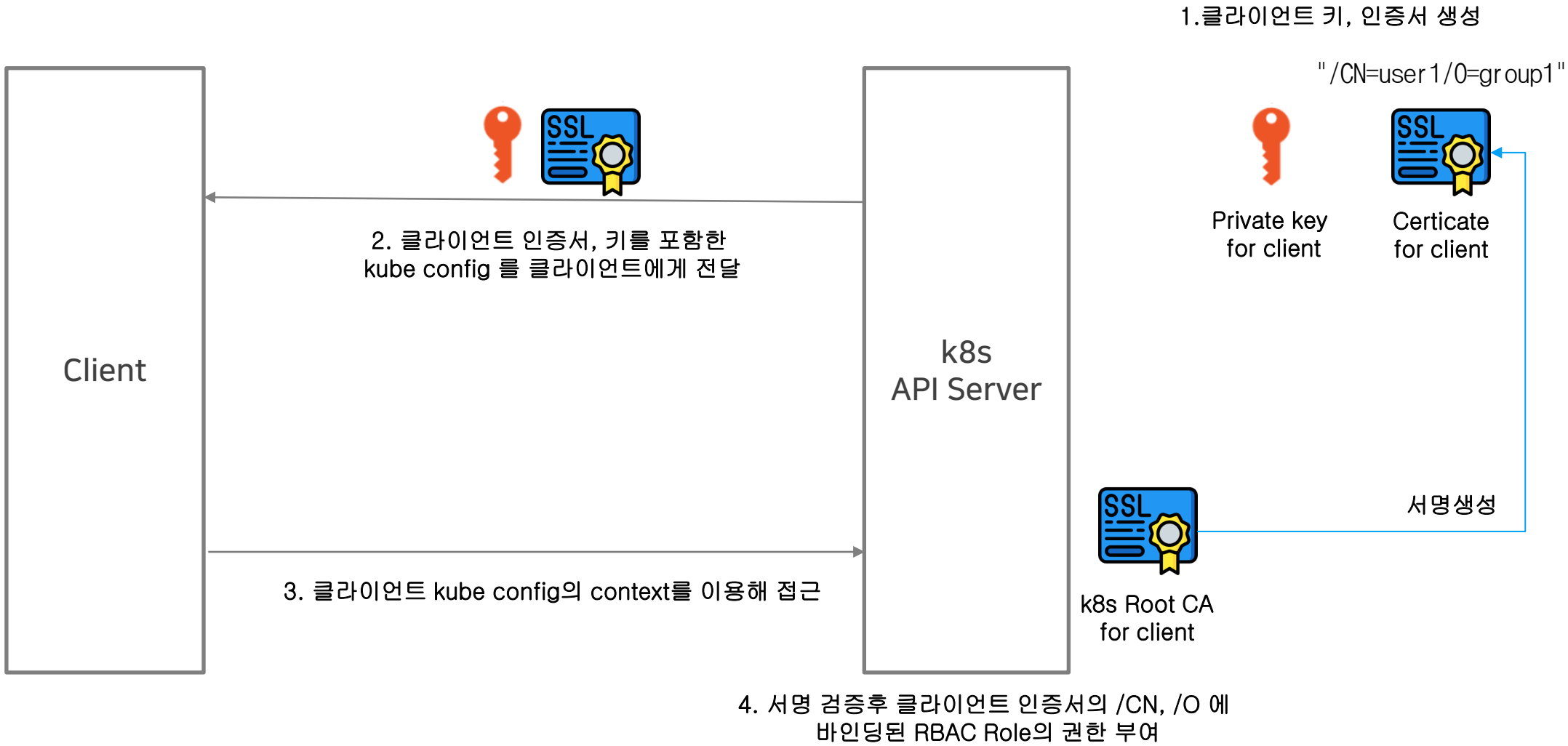
- kube config 기본 설정 예시

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJTU5DaHdQb3BJMUl3RFFZSktvWklo>
    server: https://192.168.56.201:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLVENDQWhHZ0F3SUJBZ0lJYmg4YlpUN1grRDB3RFFZSktvWklodmN>
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBc3BtWEc4clJRcHFLajJ6ejlWejRlOTRpSGEyU3l>
```

- user 정보가 클라이언트 인증서를 사용하는 방법의 예시
- 이 방식은 별도 로그인이나 토큰 없이 인증할 수 있으며 주로 클러스터 내부에서 자동생성되는 사용자(예:k8s관리자)에게 주로사용됨
- 하지만 클라이언트 인증서에 유효기간이 있으며, 유출되면 다른 사람이 관리자 권한을 행사할 위험이 존재함

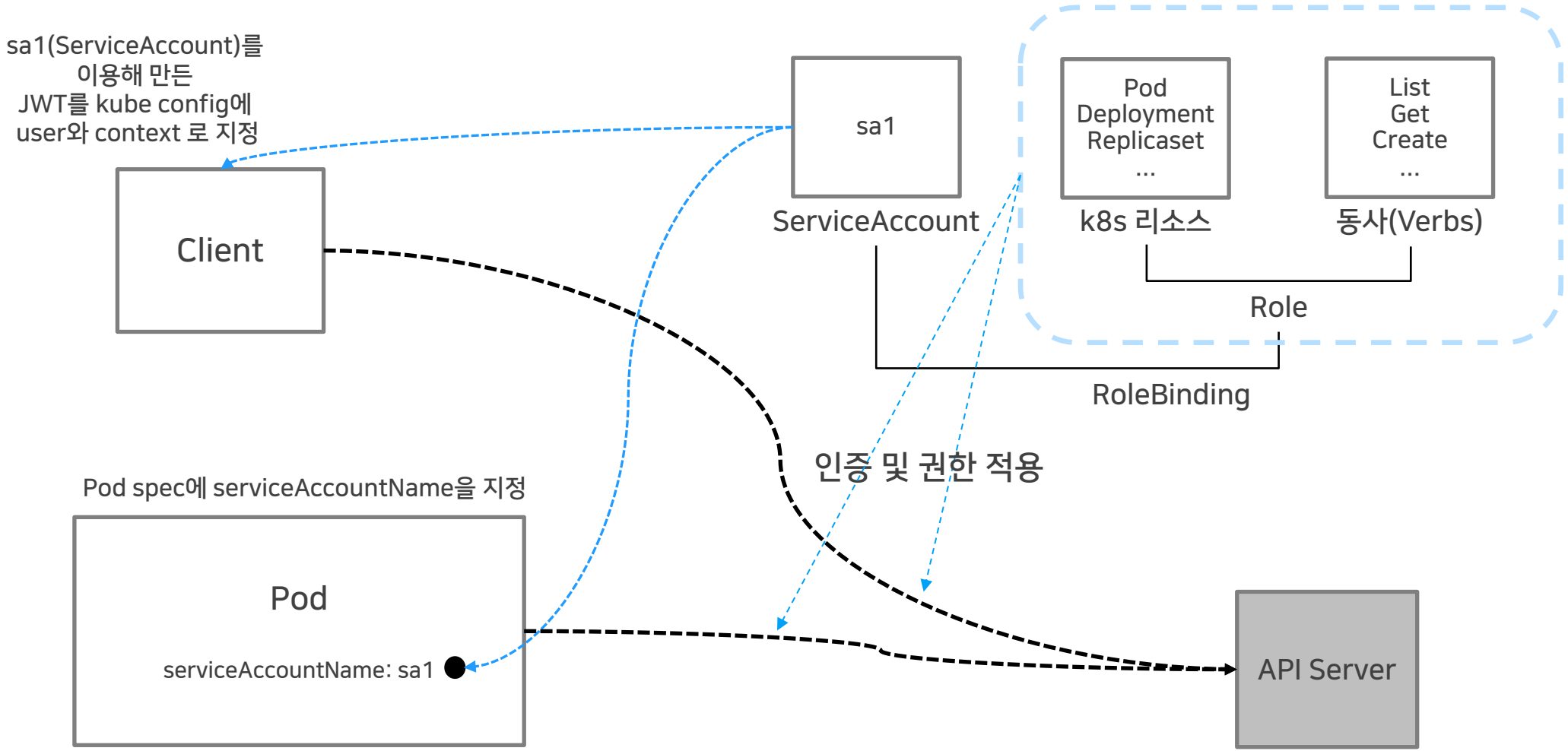
# 3. k8s의 인증 방식

## ❖클라이언트 인증서 기반의 인증



# 3. k8s의 인증 방식

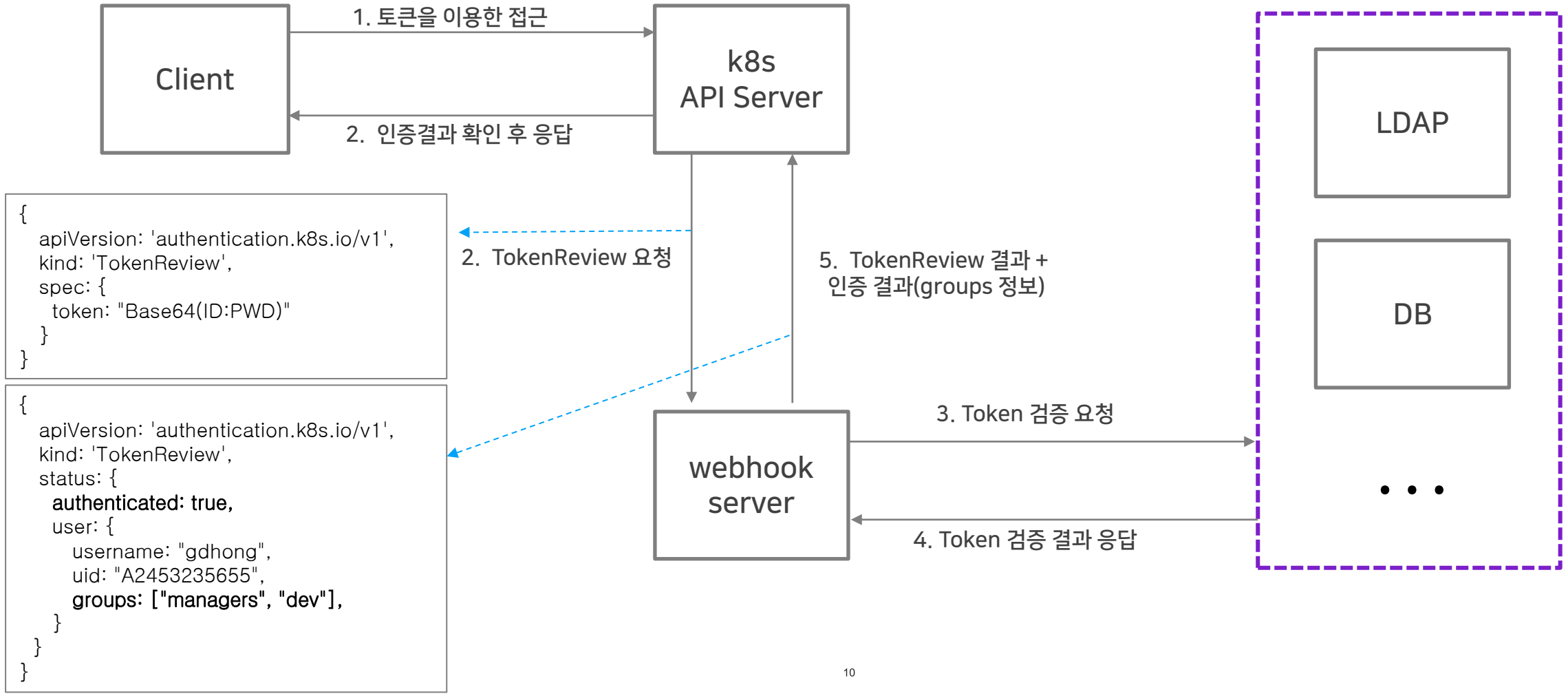
## ❖ServiceAccount의 JWT 를 이용한 인증



# 3. k8s의 인증 방식

## ❖Webhook 서버 이용 예시 1

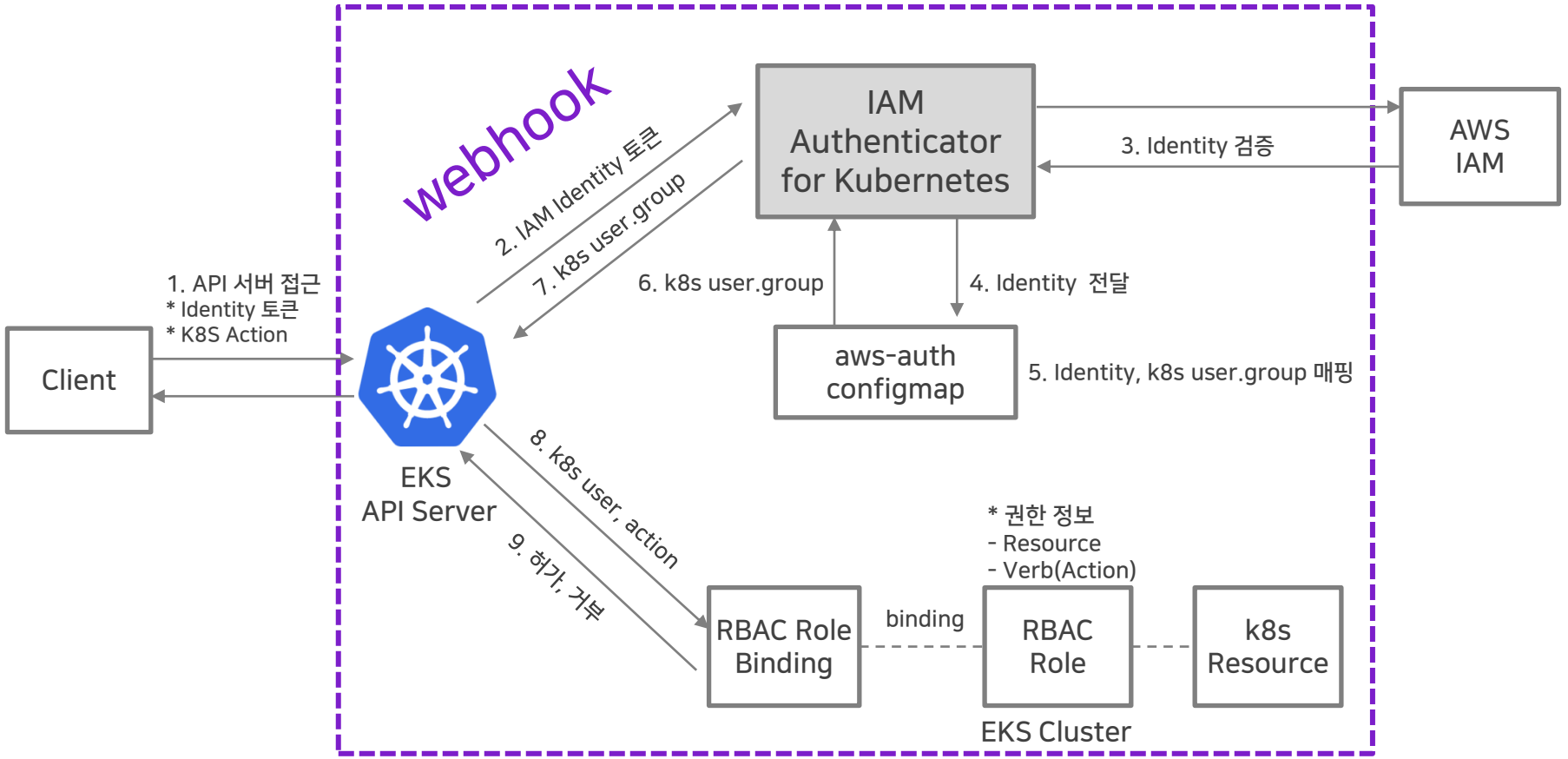
- 직접 Webhook 서버를 구성했을 때의 예시



# 3. k8s의 인증 방식

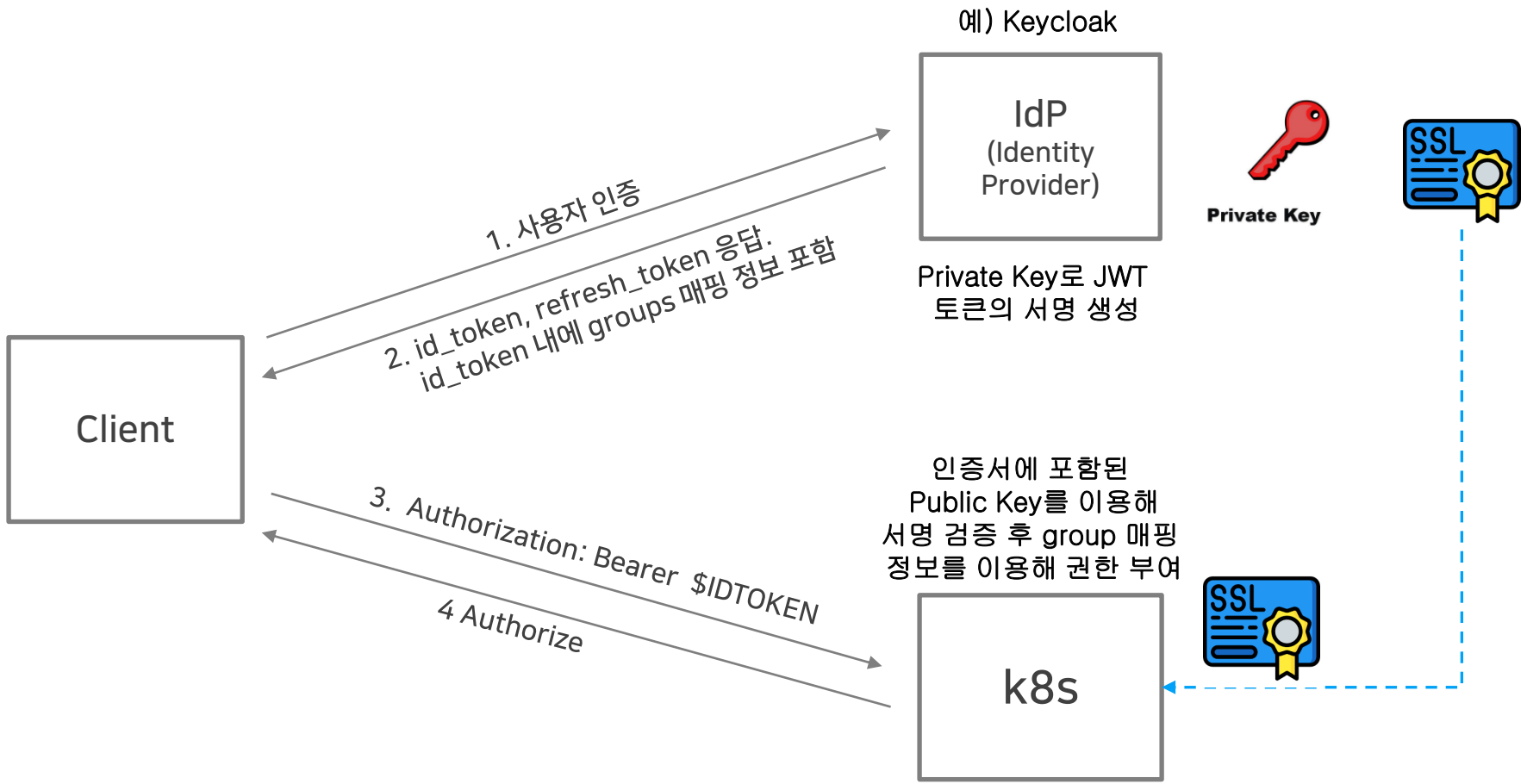
❖Webhook 서버 이용 예시 2

- AWS EKS + AWS IAM 연동 예시



# 3. k8s의 인증 방식

## ❖OIDC를 이용한 인증



# 4. ServiceAccount 토큰 이용 실습

## ❖ServiceAccount 생성

```
# dev namespace 생성하고 sa-dev ServiceAccount 생성
kubectl create ns dev
kubectl create sa sa-dev -n dev
```

## ❖Role과 RoleBinding 생성 : sa-dev-role.yaml 파일

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: sa-dev-role
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-dev-rolebinding
  namespace: dev
subjects:
  - kind: ServiceAccount
    name: sa-dev
    namespace: dev
roleRef:
  kind: Role
  name: sa-dev-role
  apiGroup: rbac.authorization.k8s.io
```

- kubectl apply -f sa-dev-role.yaml

# 4. ServiceAccount 토큰 이용 실습

❖Token 생성

- kubectl create token sa-dev -n dev --duration=86400s

❖클라이언트의 kube config 파일 수정

- 윈도우의 Ubuntu 터미널에서

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUS.....
    server: https://192.168.56.201:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes
- context:
    cluster: kubernetes
    user: sa-dev
  name: sa-dev
current-context: sa-dev
```

```
kind: Config
preferences: {}
users:
- name: sa-dev
  user:
    token: [이전단계에서 생성한 토큰]
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBD...
    client-key-data: LS0tLS1CRUdJTiBS...
```

# 4. ServiceAccount 토큰 이용 실습

## ❖접근 여부 테스트

```
# dev namespace에만 접근이 가능한지 테스트 수행
kubectl get all -n dev                          //ok
kubectl create sa test1 -n dev                  //ok

kubectl get all -n default                      //에러
kubectl get ns                                  //에러
```

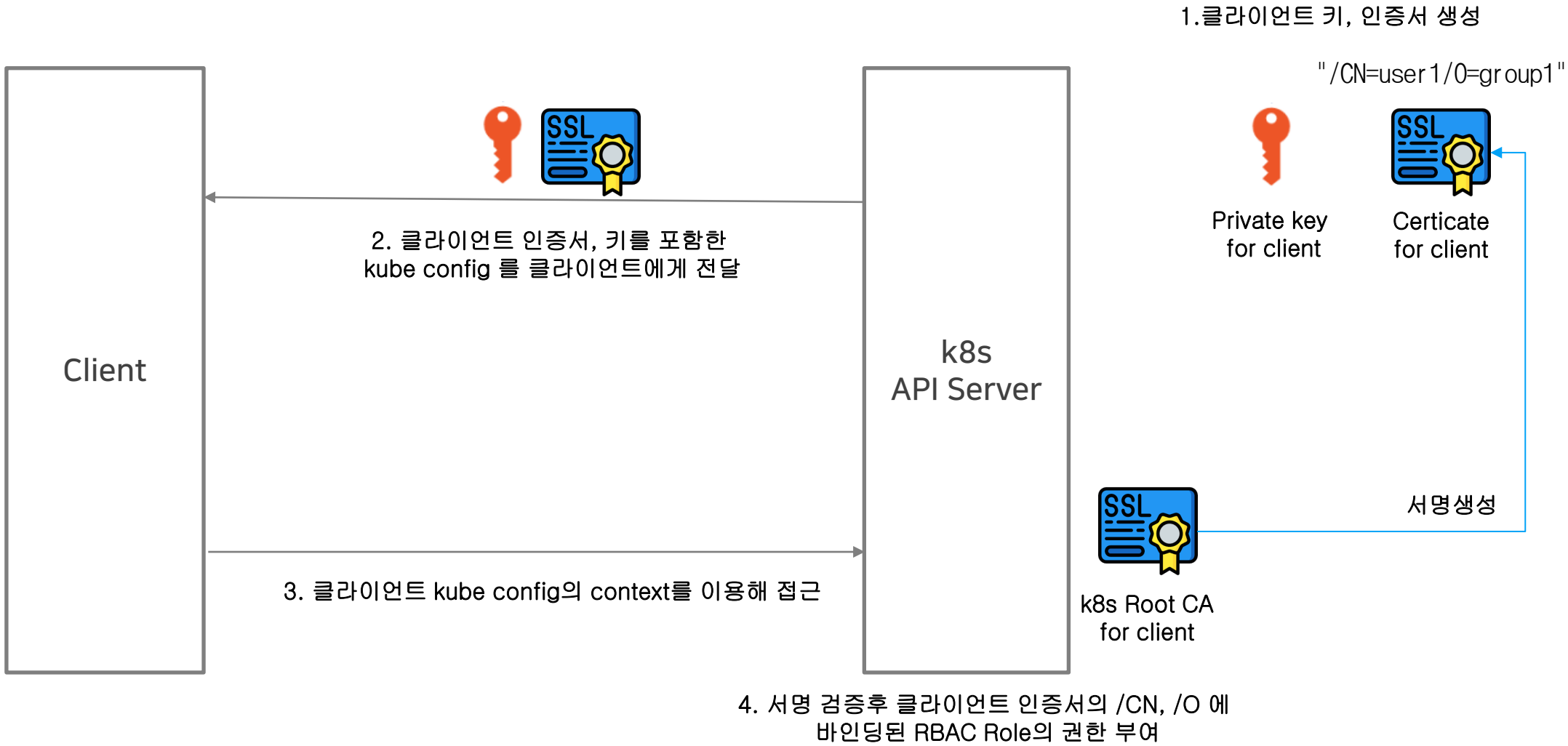
## ❖리소스 제거

- 클라이언트 컴퓨터(윈도우 우분투)에서

```
kubectl config use-context kubernetes-admin@kubernetes
kubectl config delete-context sa-dev
kubectl config delete-user sa-dev
kubectl delete ns dev
```

# 5. Client 인증서를 이용한 접근 실습

## ❖클라이언트 인증서 기반의 인증



# 5. Client 인증서를 이용한 접근 실습

❖ k8s master에서 실행

- 대상 namespace 생성

```
kubectl create ns dev2
```

- 사용자를 위한 private key, 인증서 생성

```
# 디렉토리 사용자 홈 디렉토리로 이동
cd ~

# private key 생성
openssl genrsa -out user1.key

# csr(Certificate Signing Request) 생성. CN에 사용자, O에 그룹 지정
openssl req -new -key user1.key -subj "/CN=user1/O=group1" -out user1.csr

# csr과 k8s의 CA 키와 인증서를 이용해 Client 인증서 생성
sudo openssl x509 -req -in user1.csr -CA /etc/kubernetes/pki/ca.crt ₩
  -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out user1.crt -days 3650
```

# 5. Client 인증서를 이용한 접근 실습

- Role과 RoleBinding 생성
  - group1-role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: group1-role
  namespace: dev2
rules:
  - apiGroups:
      - ""              # core api
      - "extensions"
      - "apps"
    resources:
      - "*"
    verbs:
      - "get"
      - "list"
      - "watch"
      - "create"
      - "update"
      - "patch"
      - "delete"
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: group1-rolebinding
  namespace: dev2
subjects:
  - kind: Group
    name: group1
roleRef:
  kind: Role
  name: group1-role
  apiGroup: rbac.authorization.k8s.io
```

  - kubectl apply -f group1-role.yaml

# 5. Client 인증서를 이용한 접근 실습

- 클라이언트 인증서 정보 설정 후 복사할 파일 생성

```
### master에서 수행할 것

# kube config 파일 백업
cp ~/.kube/config ~/.kube/backup.conf

# 인증을 위한 user credential을 kube config 파일에 등록
kubectl config set-credentials user1 ₩
   --client-certificate=/home/user1/user1.crt ₩
   --client-key=/home/user1/user1.key --embed-certs=true

# context 생성
# kubectl config get-clusters 명령어로 cluster 이름 확인 후 (예시: Kubernetes)
kubectl config set-context user1@k8s --cluster=kubernetes --user=user1

# kube config 파일을 다른 파일명으로 복사 후 config 내용 수정
cp ~/.kube/config ~/user1.conf

# 백업했던 kube config로 원복
cp ~/.kube/backup.conf  ~/.kube/config
```

# 5. Client 인증서를 이용한 접근 실습

- user1.conf 수정
  - user1 사용자와 user1@k8s context 만 남겨놓도록

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ......
    server: https://192.168.56.201:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    namespace: dev2
    user: user1
  name: user1@k8s
current-context: user1@k8s
kind: Config
preferences: {}
users:
- name: user1
  user:
    client-certificate-data: ......
    client-key-data: ......
```

# 5. Client 인증서를 이용한 접근 실습

- master 서버에서 윈도우 우분투로 user1.conf 파일 복사

```
# 윈도우 우분투에서 openssh-server 설치
sudo apt install openssh-server -y

# 윈도우 우분투에서 ip a 명령어로 IP 주소 확인 후 master에서 scp 명령어로 user1.conf 파일 복사
scp ~/user1.conf 우분투사용자명@우분투IP주소:~/
```

## ❖ 윈도우 우분투에서 kubectl 실행

```
# 기존 kube config 백업 후 user1.conf 파일을 우분투 사용자의 kube config 로 설정
cp  ~/.kube/config  ~/.kube/backup.conf
cp  ~/user1.conf  ~/.kube/config
kubectl config use-context user1@k8s

# 적용 여부 테스트
kubectl get pods -n dev2                    //정상 실행
kubectl get pods -n default                          //권한없음 에러
```

# 5. Client 인증서를 이용한 접근 실습

## ❖리소스 정리

```
# 윈도우 우분투에서 기존 kube config 백업으로 복구한 후 컨텍스트 변경
cp ~/.kube/backup.conf  ~/.kube/config
kubectl config use-context kubernetes-admin@kubernetes

kubectl delete ns dev2
```

# 6. keycloak을 이용한 oidc 인증

❖작업 단계

- keycloak 개요 및 설치 준비
- Postgresql DB 설정
- 인증서 생성, secret 등록
- ingress-nginx-controller 설치
- keycloak 설치
- keycloak에서 client, user 설정
- kube-apiserver 설정 변경
- 인증 테스트
- k8s 클러스터에 Role, RoleBinding 설정
- 권한 적용 여부 확인
- 클라이언트에서 kubelogin 활용하기
- 리소스 정리

# 6.1 keycloak 개요 및 설치 준비

❖ keycloak이란?
- 오픈소스 Identity and Access Management (IAM) 솔루션
- 애플리케이션이나 서비스에 SSO, 사용자 인증, 권한 부여, 계정 관리 등의 기능 제공

❖ 주요 기능
- SSO
- 외부 인증 연동 : LDAP, Active Directory, SAML 2.0, OIDC, 소셜 로그인(Facebook,Google 등)
- 자체적인 IdP 기능 제공: OAuth2, SAML, OIDC
- 웹 기반 UI 제공
- 다중 Realm 제공
- MFA 지원 : Multi Factor Authentication

❖ Use case
- 기업 내 애플리케이션에 대한 OIDC 기반 SSO 제공
- Kubernetes / ArgoCD / Jenkins 등의 관리자 인증
- OAuth2 기반 API 보안
- SaaS 서비스의 사용자 인증

# 6.1 keycloak 개요 및 설치 준비

## ❖서버 준비

- server3 가상머신 실행

## ❖helm 도구 설치 : 이미 설치되었다면 skip

```
# master 가상머신에서
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

## ❖metalLB 설치 : 이미 설치되었다면 skip

- 다음 주소의 문서 중 metalLB 설정 내용을 참조하여 설치
  - https://github.com/stepanowon/k8s-on-win
- ip 주소 풀의 범위 : 192.168.56.51~80

# 6.2 Postgresql DB 설치 및 설정

## ❖Postgresql DB 설정 : server3에서(이어서)

```
# postgresql 서버 설치
$ sudo apt install postgresql -y

# postgresql 서버 설정 : postgresql.conf 파일에서 내용을 찾아서 변경
$ sudo nano /etc/postgresql/16/main/postgresql.conf
listen_addresses = '*'
password_encryption = scram-sha-256


$ sudo nano /etc/postgresql/16/main/pg_hba.conf
# 다음열을 찾아서 볼드체 부분 변경
local   all             postgres                                trust
......
# 다음 열은 추가
host    keycloakdb      keycloak        192.168.56.0/24         scram-sha-256

# 모두 변경했다면 서비스 재시작
$ sudo systemctl restart postgresql.service
```

# 6.2 Postgresql DB 설치 및 설정

## ❖Postgresql DB 설정 : server3에서

- keycloak 서버의 설정 정보는 DB에 저장되기 때문에 DB가 필요함

```
# 루트 권한으로 psql 접속 후 사용자, DB 설정
$ psql -U postgres

# 사용자명/패스워드는 적절하게 변경하여 사용할 수 있음
CREATE ROLE keycloak WITH LOGIN PASSWORD 'asdf';

# 데이터베이스 생성. 데이머베이스명을 적절하게 변경할 수 있음
CREATE DATABASE keycloakdb WITH OWNER keycloak TEMPLATE template0 ENCODING UTF8 LC_COLLATE 'en_US.UTF-8'
LC_CTYPE 'en_US.UTF-8';

# 설정 후 psql에서 exit
exit

# 연결 테스트 후 exit
$ psql "postgres://keycloak@192.168.56.103/keycloakdb"
exit
```

# 6.3 인증서 생성, secret 등록

## ❖인증서 생성

- 주의 사항
  - 테스트를 위해 k8s api server의 CA Key, 인증서를 이용해 keycloak 서버용 인증서를 생성함
  - 하지만 실무에서는 인증기관(CA), 발급기관(RA)를 이용해 공인 인증서를 발급받아 사용할 것을 추천
- 인증서 생성 수행

```
# 인증 요청서(CSR) 생성 : master 가상머신에서
mkdir -p ~/keycloak/tls
openssl req -new -newkey rsa:2048 -nodes -days 3650 -keyout ~/keycloak/tls/keycloak.key ₩
  -out ~/keycloak/tls/keycloak.csr -subj "/CN=keycloak.ssamz.192.168.56.80.nip.io/O=ssamz"

# k8s 클러스터의 내부 CA 키,인증서를 이용해 keycloak용 인증서 생성
sudo bash -c 'openssl x509 -req -in /home/user1/keycloak/tls/keycloak.csr ₩
  -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key ₩
  -CAcreateserial -out /home/user1/keycloak/tls/keycloak.crt -days 3650 ₩
  -extensions v3_req ₩
  -extfile <(cat <<EOF
[ v3_req ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = keycloak.ssamz.192.168.56.80.nip.io
EOF
)'
```

# 6.3 인증서 생성, secret 등록

- 생성한 인증서와 Private Key를 Secret으로 등록

```
# 생성한 인증서의 소유권을 user1로 변경 --> 각 가상머신의 사용자명
sudo chown user1:user1 ~/keycloak/tls/keycloak.crt

# keycloak용 secret 등록
kubectl create ns keycloak
kubectl create secret tls keycloak-tls ₩
  --cert=/home/user1/keycloak/tls/keycloak.crt ₩
  --key=/home/user1/keycloak/tls/keycloak.key ₩
  -n keycloak
```

# 6.3 인증서 생성, secret 등록

❖참고 : nip.io 주소

- Wild card DNS 서비스
- DNS 서버에 호스트를 등록하지 않고, hosts 파일을 변경하지 않고도 유연성있는 host 명을 사용할 수 있도록 함
  - 테스트시에 편리함
- 규칙 : 다음의 모든 호스트명은 192.168.56.80에 매핑됨
  - 192.168.56.80.nip.io
  - 192-168-56-80.nip.io
  - test.com.192.168.56.80.nip.io
  - app-192-168-56-80.nip.io
  - test1.app.192.168.56.80.nip.io
  - test2-app-192-168-56-80.nip.io

# 6.4 ingress-nginx-controller 설치

## ❖helm을 이용한 ingress-nginx-controller 설치

- 주의 사항
  - ingress-nginx-controller를 설치할 때 tls 인증서를 지정하는 것이 아니라 keycloak을 생성할 때 ingress에 tls 인증서를 지정함
- 다음 명령어를 이용해 ingress-nginx-controller 설치

```
# helm repo 추가 후 업데이트 : 윈도우 우분투 터미널에서 실행
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update

# ingress-nginx 네임스페이스 ingress-controller 설치
# service 타입을 LB로 설정하고 LB의 IP 주소를 192.168.56.80으로 설정. 이를 위해 metalLB가 미리 설정되어야 함
helm install ingress-nginx ingress-nginx/ingress-nginx ₩
  --namespace ingress-nginx --create-namespace ₩
  --set controller.service.type=LoadBalancer ₩
  --set controller.service.loadBalancerIP=192.168.56.80
  --set controller.progressDeadlineSeconds=600
```

# 6.5 keycloak 설치

## ❖helm을 이용한 keycloak 설치

- 윈도우 우분투 터미널에서 진행

```
# helm repo 추가
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

- keycloak 설치를 위한 설정 파일 작성 : keycloak-values.yaml

```
auth:
  adminUser: keycloak                # 관리자 사용자명
  adminPassword: asdf                # 관리자 패스워드
ingress:
  enabled: true
  ingressClassName: nginx
  # 내부 IP 주소 사용
  hostname: keycloak.ssamz.192.168.56.80.nip.io
  # TLS 설정
  tls: true
  extraTls:
    - hosts:
        - keycloak.ssamz.192.168.56.80.nip.io
      secretName: keycloak-tls
service:
  type: ClusterIP
```

```
proxy: edge
keycloak:
  extraEnv:
    - name: KEYCLOAK_FRONTEND_URL
      value: https://keycloak.ssamz.192.168.56.80.nip.io
postgresql:
  enabled: false
externalDatabase:
  host: 192.168.56.103
  port: 5432
  user: keycloak
  password: asdf
  database: keycloakdb
```

# 6.5 keycloak 설치

- helm을 이용해 keycloak 설치

```
# 설치
helm upgrade -i keycloak bitnami/keycloak ₩
  --version 24.0.5 ₩
  --namespace keycloak --create-namespace ₩
  -f keycloak-values.yaml


# 설치 후 작동 여부 확인 : Pod가 running  1/1이 될 때까지 기다림. 3-4분 소요
$ kubectl get po -n keycloak
NAME                                                  READY   STATUS    RESTARTS   AGE
keycloak-0                                            1/1     Running   0          2m48s

# 작동 여부 확인 후 브라우저 열고 다음 주소로 접속
# 관리자 ID/PWD는 이전 페이지의 values.yaml 참조
https://keycloak.ssamz.192.168.56.80.nip.io
```

# 6.6 keycloak에서 realm,client,user 설정

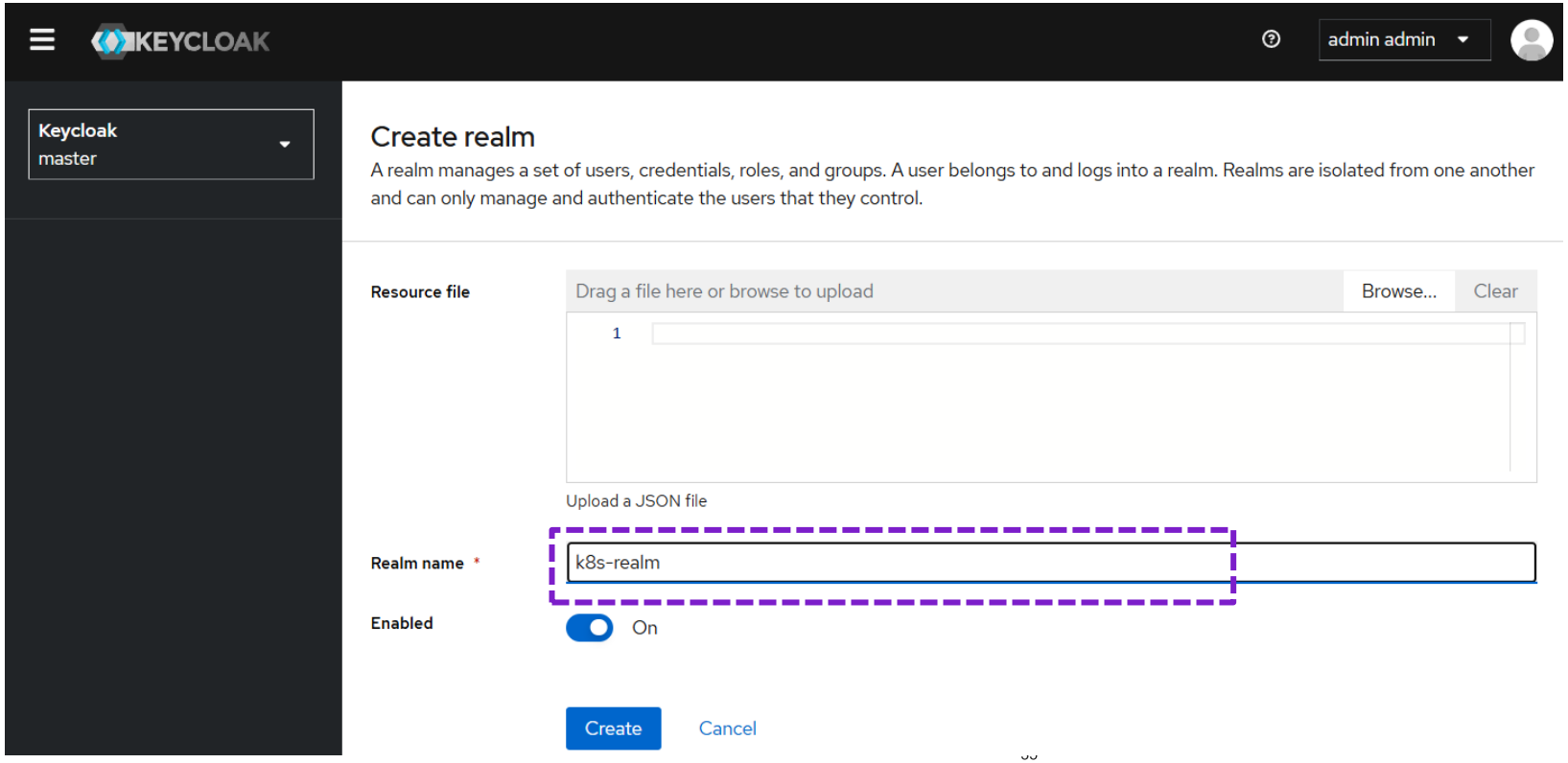
❖master realm에 새로운 관리자 추가

- keycloak/asdf 계정은 임시계정
  - keycloak 서버를 설치할 때 환경 변수로 지정한 자격증명은 임시로 지정한 것이므로 새롭게 관리자 계정을 생성
- 임시 계정 keycloak 사용자로 로그인 후 화면 왼쪽 패널의 Users 클릭
- 'Add user' 버튼 클릭하고 다음과 같이 입력 후 'Create' 클릭
  - username : admin
  - email : admin@test.com
  - firstname, lastname : admin
- 생성된 admin 사용자 이름 클릭 후 Credentials 탭에서 패스워드 설정
  - 'Set Password' 버튼 클릭 후 패스워드를 asdf로 설정
  - Temporary : off
- Role mapping 탭으로 이동 후 'Assign role' 버튼 클릭
  - Filter by clients 드롭다운을 클릭하여 'Filter by realm role' 로 변경
  - admin 체크 후 'Assign' 버튼 클릭
- 기존 계정 로그아웃 후 admin 계정으로 로그인

# 6.6 keycloak에서 realm,client,user 설정

## ❖새로운 realm 설정

- realm : 사용자, 자격증명, 클라이언트, 역할, 그룹 등을 관리하는 독립적, 논리적 영역
- 화면 왼쪽 상단의 master realm 클릭 후 'Create realm' 클릭 후 다음 설정
  - Realm name : k8s-realm
  - 'Create' 클릭



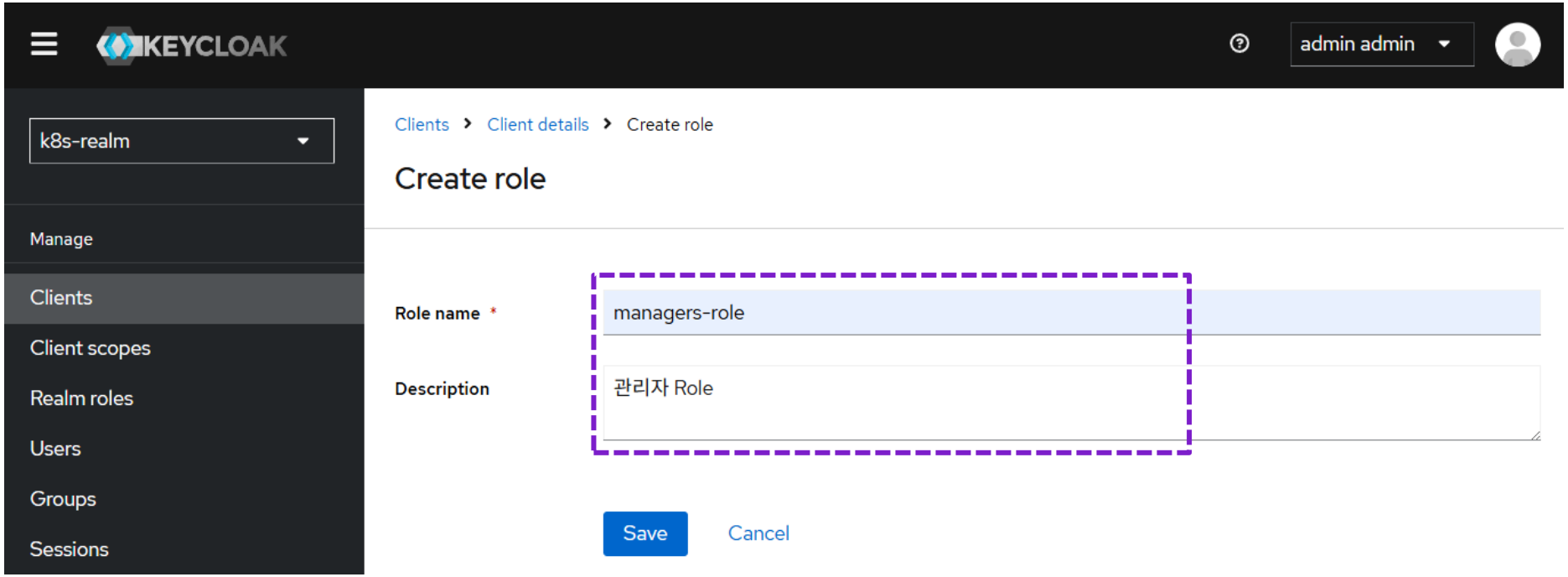
# 6.6 keycloak에서 realm,client,user 설정

- k8s-realm에 client 추가
  - 왼쪽 패널에서 Clients 클릭한 후 'Create Client' 클릭 하고 각 단계 지정
  - General Settings
    - Client ID : test-client
    - Name, Description : k8s 인증 테스트
  - Capability config
    - Client authentication : on
  - Login Settings
    - Valid redirect URIs : http://localhost:8000/*
      - kubelogin 도구를 위한 설정

# 6.6 keycloak에서 realm,client,user 설정

❖test-client 의 Roles 탭으로 이동 후 다음 설정

- 'Create role' 클릭 후 name에 managers-role 입력 후 'Save'



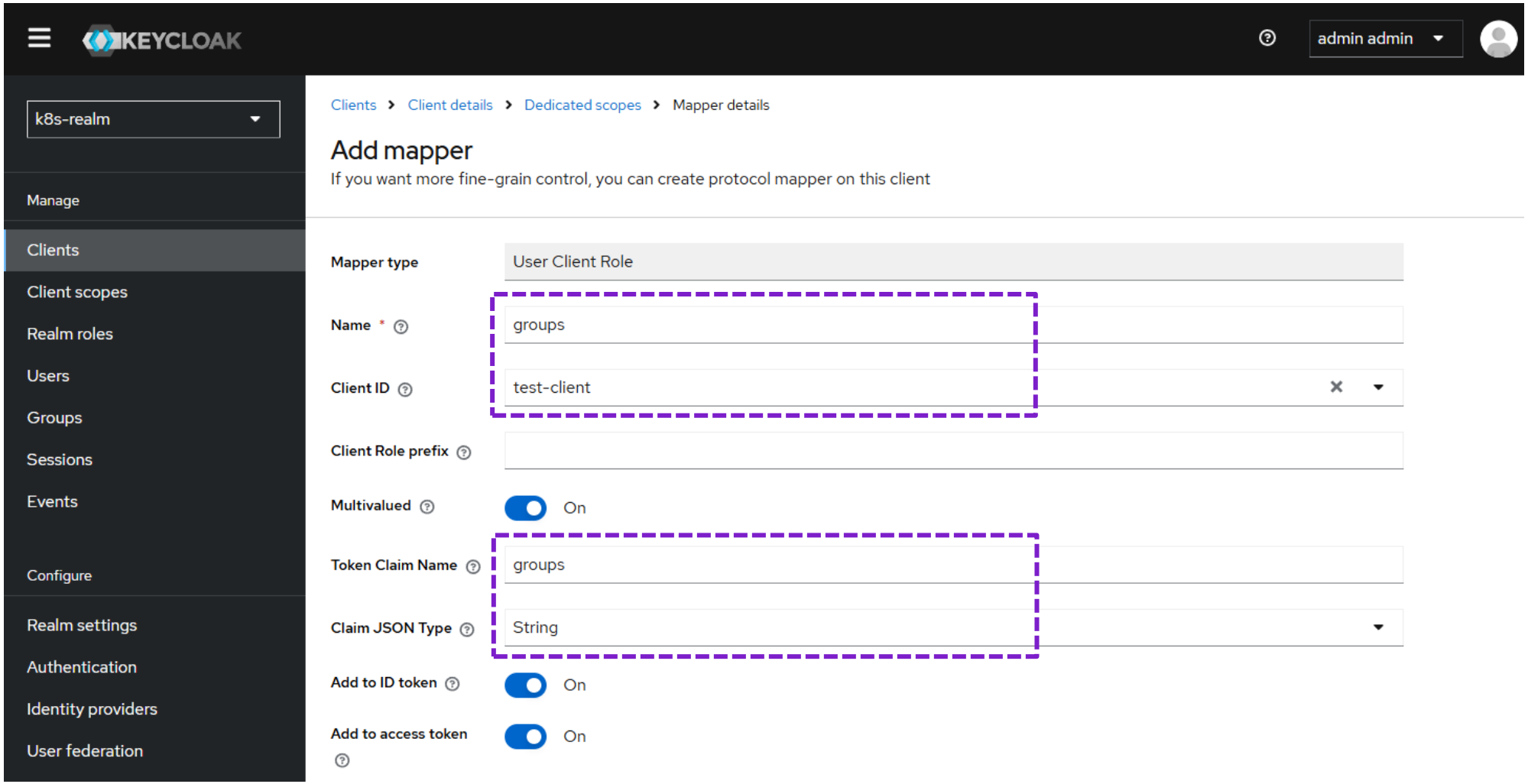
# 6.6 keycloak에서 realm,client,user 설정

❖test-client 의 Client scopes 탭으로 이동 후 다음 설정

- client scopes 항목 중 'test-client-dedicated' 를 클릭
- Scope 탭에서 Full scope allowed 를 off 로 설정
- Mapper 탭으로 이동하여 'Configure a new mapper' 클릭
- 'User Client Role'을 찾아서 클릭하고 다음과 같이 mapper 설정한 후 'Save' 클릭
  - Name : groups
  - Client ID : test-client
  - Token Claim name : groups
  - Claim JSON Type : String
- test-client에서 생성한 managers-role이 User Client Role에 매핑될 것임
  - 또한 이것이 생성된 JWT 토큰의 groups 라는 claim의 값으로 지정됨
  - 예시) groups : ["managers-role"]

# 6.6 keycloak에서 realm,client,user 설정

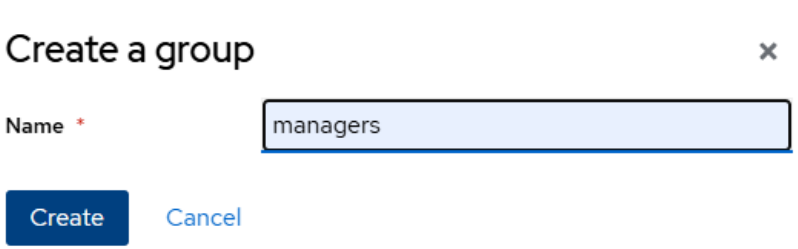
- mapper 설정 예시

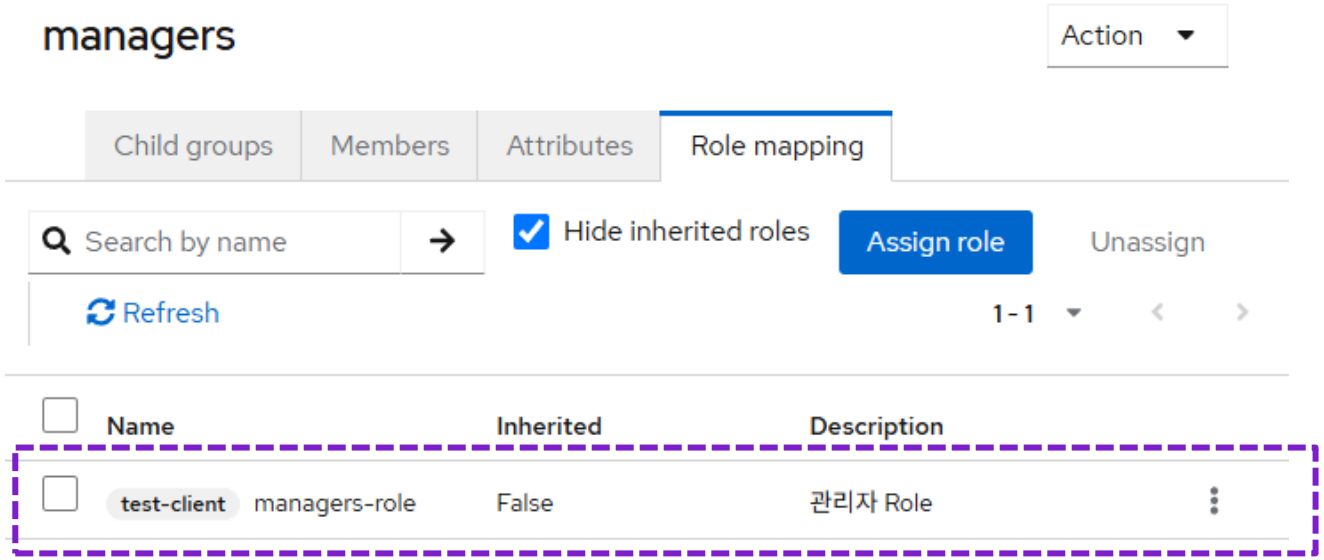


# 6.6 keycloak에서 realm,client,user 설정

❖사용할 Group 추가후 Role 매핑

- 화면 왼쪽 패널에서 Groups 클릭 후 'Create Group' 클릭
- Name 필드에 managers 입력 후 'Create'
- managers 그룹 클릭 후 Role mapping 탭으로 이동
- 'Assign role' 버튼 클릭 후 managers-role 찾아서 체크 후 'Assign' 버튼 클릭

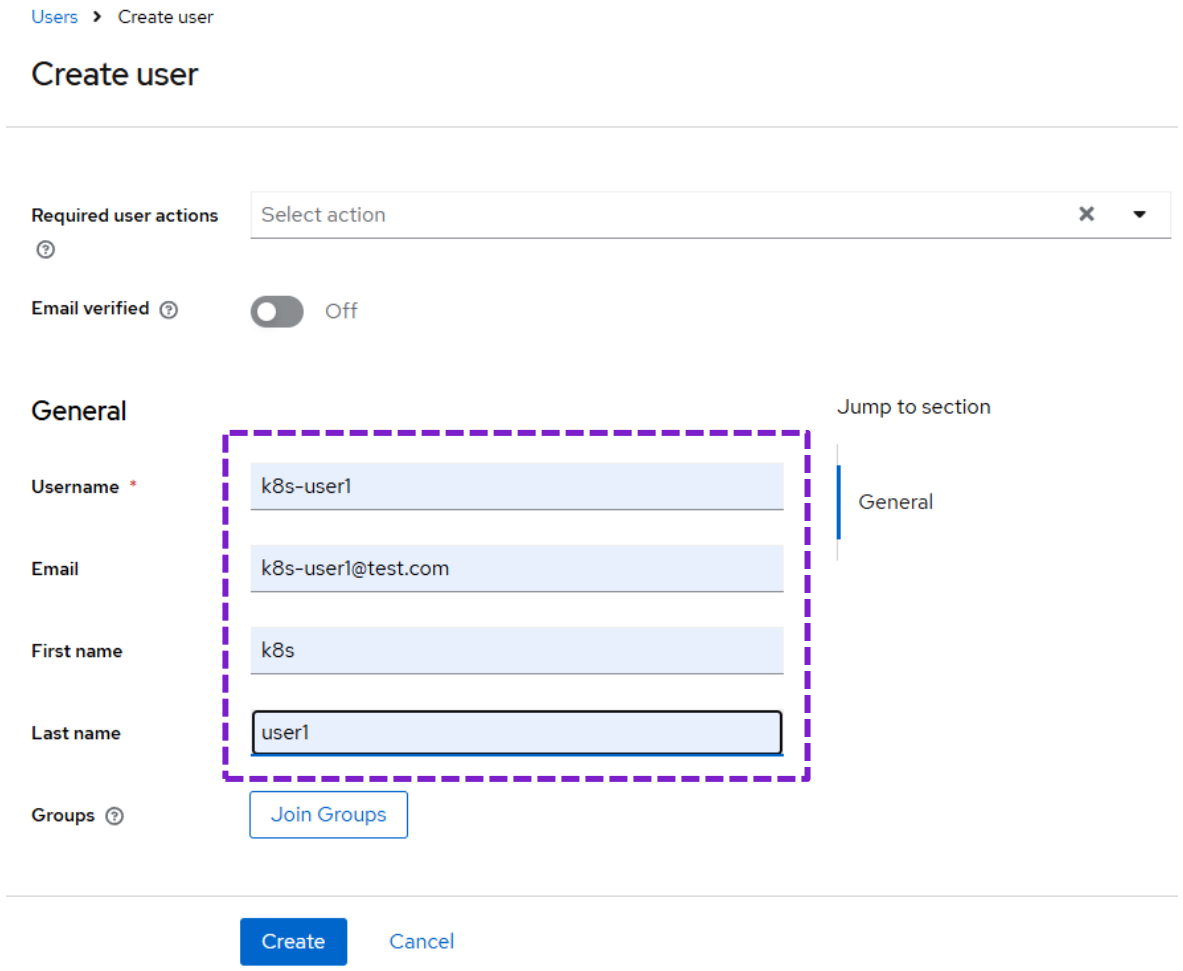


# 6.6 keycloak에서 realm,client,user 설정

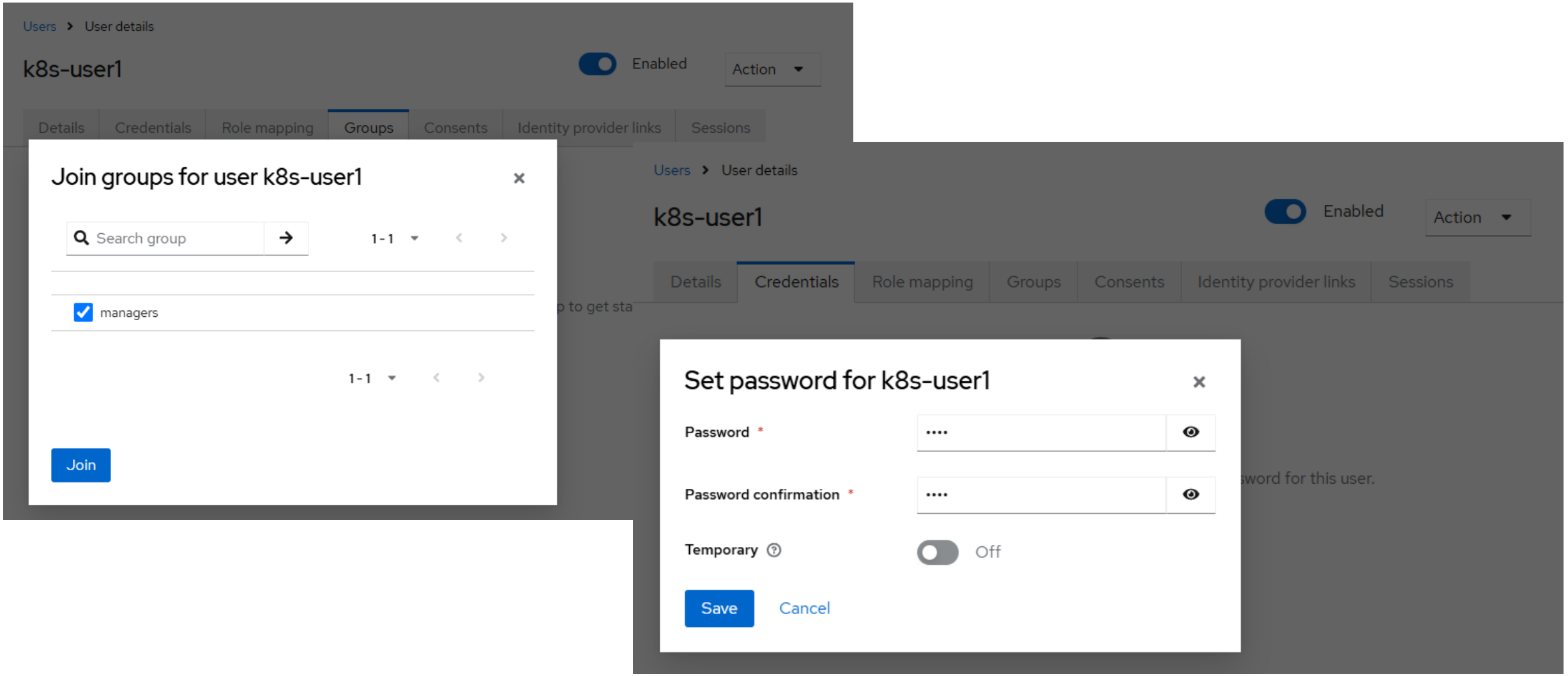
## ❖사용자 추가, 설정

- 화면 왼쪽 패널에서 Users 클릭 후 'Create new user' 버튼 클릭
- 다음과 같이 설정 후 'Create' 버튼 클릭
  - Username : k8s-user1
  - Email, FirstName, LastName 은 적절히 입력. 반드시 Email 입력해야 함
- 사용자 생성 후 나타난 화면에서 Groups 탭 클릭
  - Join Group 버튼 클릭한 후 managers 그룹 체크 후 'Join' 버튼 클릭
- Credentials 탭으로 이동한 후 Set Password를 눌러서 패스워드 설정
  - 이 실습에서는 사용자의 패스워드를 asdf 로 설정한 것으로 진행함
  - Temporary : Off 로 반드시 설정



# 6.6 keycloak에서 realm,client,user 설정

- 설정 화면 예시



# 6.6 keycloak에서 realm,client,user 설정

❖keycloak 사용자 설정 구조

- k8s-user1 사용자에 연동되는 k8s 보안주체 group은 managers-role 이다.
  - groups : [ "managers-role" ]

# 6.7 kube-apiserver 설정

❖인증서 설정 : master 가상머신에서 실행

- self-signed 인증서를 사용할 때
  - keycloak 서버가 사용중인 자체 서명 인증서를 추출함.

```
# 추출된 인증서 파일 : /etc/kubernetes/pki/keycloak-ca.crt
openssl s_client -connect keycloak.ssamz.192.168.56.80.nip.io:443 -showcerts </dev/null 2>/dev/null ₩
  | awk '/BEGIN/,/END/{ print }' | sudo tee /etc/kubernetes/pki/keycloak-ca.crt > /dev/null
```

- /etc/kubernetes/manifests/ 디렉토리 아래의 kube-apiserver.yaml을 다음과 같이 편집

```
$ sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - ......
    # 다음 내용 추가
    - --oidc-client-id=test-client
    - --oidc-groups-claim=groups
    - --oidc-issuer-url=https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s-realm
    - --oidc-username-claim=preferred_username
    - --oidc-ca-file=/etc/kubernetes/pki/keycloak-ca.crt
```

# 6.7 kube-apiserver 설정

❖ kube-apiserver.yaml 변경 후 apiserver가 정상이 될 때까지 대기

- kubectl get nodes 명령어의 결과가 정상으로 나타날 때까지 기다림

```
user1@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   5d21h   v1.30.12
worker1   Ready    <none>          5d21h   v1.30.12
worker2   Ready    <none>          5d21h   v1.30.12
worker3   Ready    <none>          5d21h   v1.30.12
```

# 6.8 인증 테스트

## ❖ token 엔드포인트 주소 확인

- 웹브라우저를 열고 다음 주소 확인

```
# k8s-realm 위치에 직접 생성한 realm 이름 지정
https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s-realm/.well-known/openid-configuration
```


## ❖test-client 의 client_secret 값 획득

- keycloak 서버의 k8s-realm에서 test-client 의 Credentials 탭으로 이동
- Client secret 값 클립보드로 복사

# 6.8 인증 테스트

❖다음 명령어를 실행해 id_token, refresh_token을 받아오는지 확인

```
# 우분투 터미널에서 실행
# 이전 단계에서 획득한 Client secret, Token endpoint 값을 지정
CLIENT_SECRET=획득한ClientSecret지정
TOKEN_ENDPOINT=획득한TokenEndpoint지정

# 사설 인증서를 사용했기 때문에 --insecure 옵션 사용. 실무에서는 사용하지 말것
curl -X POST ${TOKEN_ENDPOINT} ₩
-d grant_type=password -d client_id=test-client -d client_secret=${CLIENT_SECRET} ₩
-d username=k8s-user1 -d password="asdf" -d scope=openid  --insecure
```

```
user1@master:~$ CLIENT_SECRET=iFiPFEFqIQHsvfxL3RchBji7NxxfhvYS
TOKEN_ENDPOINT=https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s-realm/protocol/openid-connect/token

curl -X POST ${TOKEN_ENDPOINT} \
-d grant_type=password -d client_id=test-client -d client_secret=${CLIENT_SECRET} \
-d username=k8s-user1 -d password="asdf" -d scope=openid  --insecure
{"access_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJoWnd1eVJvSGdqYkhxenFOY0VPeGc2cnF6Vk9kX3JTU1F3dEdXVXYxajVrIn0.eyJleHAiOjE3NDY0OTUwNTcsI
mlhdCI6MTc0NjQ5NDc1NywianRpIjoiM2NlMjBiOGEtNmVkYS00YzQ5LTk5ZWUtMThjYWQ3YTg1NzFiIiwiaXNzIjoiaHR0cHM6Ly9rZXljbG9hay5zc2Ftei4xOTIuMTY4LjU2LjgwLm5pcC5pby9yZWF
sbXMvazhzLXJlYWxtIiwic3ViIjoiOTFlZmIyYmEtZjcxNC00Yzc1LWI3NjctNTljZWExYzllZWM2IiwidHlwIjoiQmVhcmVyIiwiYXpwIjoidGVzdC1jbGllbnQiLCJzaWQiOiIwZTJkNWQ2Yy03Zjg4L
TQzYjktYTliZi0yOTVmN2YxMjhjYzYiLCJhY3IiOiIxIiwiYWxsb3dlZC1vcmlnaW5zIjpbImh0dHA6Ly9sb2NhbGhvc3Q6ODAwMCJdLCJyZXNvdXJjZV9hY2Nlc3MiOnsidGVzdC1jbGllbnQiOnsicm9
sZXMiOlsibWFuYWdlcnMtcm9sZSJdfX0sInNjb3BlIjoib3BlbmlkIHByb2ZpbGUgZW1haWwiLCJlbWFpbF92ZXJpZmllZCI6ZmFsc2UsIm5hbWUiOiJrOHMgdXNlcjEiLCJncm91cHMiOlsibWFuYWdlc
nMtcm9sZSJdLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJrOHMtdXNlcjEiLCJnaXZlbl9uYW1lIjoiazhzIiwiZmFtaWx5X25hbWUiOiJ1c2VyMSIsImVtYWlsIjoiazhzLXVzZXIxQHRlc3QuY29tIn0.O-m
pQdsayvYc_dnyNsSWxrK18wSGu6Zt-Hm0YapfZZ8RYf87n2onp0At29e_l4qUtDRZPphSsD40nMK4C-pUxZ-Rx8tLbjmeLbshiZ3X6syWwZAhv34xOBAhQl8ZfA3qew7JFin1hfr8mkW_QxXs09LYQyYSc
ork_C4AopUJR87dWETgf58zYVCslAohUVrqPhdg4nlOEDvGSsepFHzsdP3EEM-SL-1ejrryu8W_oCwkf592yxANOy4XPSB7W844UgCzFFRutbm5TtazsGcqbhVlabFksctqfFys7iDKMf8tvd1Kzp3vKOA
xEOcdCOH3vtvvyiiMxmVIbjdS_sBCBQ","expires_in":300,"refresh_expires_in":1800,"refresh_token":"eyJhbGciOiJIUzUxMiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhN2M0NjMyY
S0zOGIyLTQwZWUtOTY2ZC0xMWZkNTQxNTA3N2YifQ.eyJleHAiOjE3NDY0OTY1NTcsImlhdCI6MTc0NjQ5NDc1NywianRpIjoiYzUwMDE2NjAtNzcxMC00ZmI1LThiYTAtN2IzMDZiNDg2YjcyIiwiaXNz
IjoiaHR0cHM6Ly9rZXljbG9hay5zc2Ftei4xOTIuMTY4LjU2LjgwLm5pcC5pby9yZWFsbXMvazhzLXJlYWxtIiwiYXVkIjoiaHR0cHM6Ly9rZXljbG9hay5zc2Ftei4xOTIuMTY4LjU2LjgwLm5pcC5pby
9yZWFsbXMvazhzLXJlYWxtIiwic3ViIjoiOTFlZmIyYmEtZjcxNC00Yzc1LWI3NjctNTljZWExYzllZWM2IiwidHlwIjoiUmVmcmVzaCIsImF6cCI6InRlc3QtY2xpZW50Iiwic2lkIjoiMGUyZDVkNmMt
N2Y4OC00M2I5LWE5YmYtMjk1ZjdmMTI4Y2M2Iiwic2NvcGUiOiJvcGVuaWQgcHJvZmlsZSBlbWFpbCByb2xlcyBhY3IgYmFzaWMgd2ViLW9yaWdpbnMifQ.X5c3TD65nmxCx7RyOSrcGoNd-xvrZT0MP8y
3U39CA5DvrsEfpj7WMgNP7Tzbm8_dZQQQtl02yiS-c-QCHiecTA","token_type":"Bearer","id_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJoWnd1eVJvSGdqYkh
xenFOY0VPeGc2cnF6Vk9kX3JTU1F3dEdXVXYxajVrIn0.eyJleHAiOjE3NDY0OTUwNTcsImlhdCI6MTc0NjQ5NDc1NywianRpIjoiMGQwNjM2YWItMzRjNS00ODRlLWFkYjItZWYzNDQzYzQwYTZhIiwia
XNzIjoiaHR0cHM6Ly9rZXljbG9hay5zc2Ftei4xOTIuMTY4LjU2LjgwLm5pcC5pby9yZWFsbXMvazhzLXJlYWxtIiwiYXVkIjoidGVzdC1jbGllbnQiLCJzdWIiOiI5MWVmYjJiYS1mNzE0LTRjNzUtYjc
2Ny01OWNlYTFjOWVlYzYiLCJ0eXAiOiJJRCIsImF6cCI6InRlc3QtY2xpZW50Iiwic2lkIjoiMGUyZDVkNmMtN2Y4OC00M2I5LWE5YmYtMjk1ZjdmMTI4Y2M2IiwiYXRfaGFzaCI6Il9NS2t6N3BiazhUc
S1pdnhpUHpxaGciLCJhY3IiOiIxIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJuYW1lIjoiazhzIHVzZXIxIiwiZ3JvdXBzIjpbIm1hbmFnZXJzLXJvbGUiXSwicHJlZmVycmVkX3VzZXJuYW1lIjoiazh
zLXVzZXIxIiwiZ2l2ZW5fbmFtZSI6Ims4cyIsImZhbWlseV9uYW1lIjoidXNlcjEiLCJlbWFpbCI6Ims4cy11c2VyMUB0ZXN0LmNvbSJ9.G89H9UaKY0dLPO0qQUVowFNoLXDG5ocHHAMQOUz_vqnw-zU5
1Oz5bCTgloOtpGRYn9-Pj3Cxygn-SJrD3VJrZ9Ia5IwlounqfRMOphcs9bhy19PA-MClTgFQTK3jtR0zW31a46_CBqKWCK4v1UYlOAioTvITrm7iobOOQGfp47uIZz28MKb3FYrEYe1mWAPi-nsCNeC87g
J06Knn7SAk05o52Sl95LZcnfdOnFT94x4S9rRC9TnNEyL8JygjcLArMjpFN4OuQ_hu46VlXS0vsPf_swEMqUa7eV_xDL_yizt0wCizi2OQspCXL0K_ZbusSLg5G2sQMM_vs0eJNVZW7A","not-before-
policy":0,"session_state":"0e2d5d6c-7f88-43b9-a9bf-295f7f128cc6","scope":"openid profile email"}user1@master:~$
```

# 6.8 인증 테스트

❖응답 받은 id_token, refresh_token을 jwt.io 사이트에서 Decode하여 확인

PAYLOAD: DATA


```
{
  "exp": 1746495057,
  "iat": 1746494757,
  "jti": "0d0636ab-34c5-484e-adb2-ef3443c40a6a",
  "iss":
"https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s
-realm",
  "aud": "test-client",
  "sub": "91efb2ba-f714-4c75-b767-59cea1c9eec6",
  "typ": "ID",
  "azp": "test-client",
  "sid": "0e2d5d6c-7f88-43b9-a9bf-295f7f128cc6",
  "at_hash": "_MKkz7pbk8Tq-ivxiPzqhg",
  "acr": "1",
  "email_verified": false,
  "name": "k8s user1",
  "groups": [
    "managers-role"
  ],
  "preferred_username": "k8s-user1",
  "given_name": "k8s",
  "family_name": "user1",
  "email": "k8s-user1@test.com"
}
```


PAYLOAD: DATA


```
{
  "exp": 1746496557,
  "iat": 1746494757,
  "jti": "c5001660-7710-4fb5-8ba0-7b306b486b72",
  "iss":
"https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s
-realm",
  "aud":
"https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s
-realm",
  "sub": "91efb2ba-f714-4c75-b767-59cea1c9eec6",
  "typ": "Refresh",
  "azp": "test-client",
  "sid": "0e2d5d6c-7f88-43b9-a9bf-295f7f128cc6",
  "scope": "openid profile email roles acr basic web-
origins"
}
```

# 6.9 k8s 클러스터에 Role, RoleBinding 설정

❖설정을 위한 yaml 파일 생성 : rbac-role.yaml, 삭제 권한 없음. (윈도우 우분투에서 실행)

- kubectl apply -f rbac-role.yaml

```
# rbac-role.yaml : 삭제 권한 없음. 윈도우 우분투에서 작성/실행
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: managers-rbac-role
rules:
  - apiGroups: [""]
    resources: ["*"]
    verbs: ["get", "list", "update", "patch", "watch", "create"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: managers-rbac-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: managers-rbac-role
subjects:
- kind: Group
  name: managers-role
  apiGroup: rbac.authorization.k8s.io
```

# 6.10 권한 적용 여부 확인

## ❖ 다음 명령어 실행

```
TOKEN=$(curl -X POST ${TOKEN_ENDPOINT} ₩
-d grant_type=password -d client_id=test-client -d client_secret=${CLIENT_SECRET} ₩
-d username=k8s-user1 -d password="asdf" -d scope=openid  --insecure | jq -r '.id_token')

curl  https://192.168.56.201:6443/api/v1/namespaces/default --header "Authorization: Bearer ${TOKEN}" --insecure
```

## ❖실행 결과 예시 : 정상적인 경우 예시

```
$ curl  https://192.168.56.201:6443/api/v1/namespaces/default --header "Authorization: Bearer ${TOKEN}" --insecure
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
     ......(생략)
  },
  "spec": {
    "finalizers": [
      "kubernetes"
    ]
  },
  "status": {
    "phase": "Active"
  }
}
```

# 6.11 클라이언트에서 kubelogin 활용하기

❖id_token, refresh_token의 유효기간이 짧기 때문에 수동으로 갱신해주기는 어려움

- keycloak에서의 id_token 의 유효기간 기본값 : 5분
- 따라서 refresh_token을 이용해 자동으로 갱신해줄 수 있어야 함
- 이것을 가능하게 하는 도구가 kubelogin!!

❖kubelogin 설치

- 클라이언트에서 수행 : 예)윈도우 컴퓨터의 Ubuntu 터미널

```
cd ~
sudo apt install unzip
curl -LO https://github.com/int128/kubelogin/releases/download/v1.32.4/kubelogin_linux_amd64.zip
unzip kubelogin_linux_amd64.zip
sudo mv kubelogin /usr/local/bin/kubelogin
sudo ln -s /usr/local/bin/kubelogin /usr/local/bin/kubectl-oidc_login
```

- keycloak 인증서를 클라이언트 컴퓨터에서 신뢰할 수 있는 인증서로 등록

```
openssl s_client -connect keycloak.ssamz.192.168.56.80.nip.io:443 -showcerts </dev/null 2>/dev/null ₩
  | awk '/BEGIN/,/END/{ print }' | tee ~/keycloak-ca.crt > /dev/null
sudo cp ./keycloak-ca.crt /usr/local/share/ca-certificates/keycloak.crt
sudo update-ca-certificates
```

# 6.11 클라이언트에서 kubelogin 활용하기

❖클라이언트에서 다음 명령 수행

```
CLIENT_SECRET=획득한ClientSecret값지정

kubectl oidc-login setup ₩
  --oidc-issuer-url=https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s-realm  ₩
  --oidc-client-id=test-client ₩
  --oidc-client-secret=${CLIENT_SECRET}
```

❖웹브라우저로 http://localhost:8000 으로 접속하여 로그인

- k8s-user1 사용자로 로그인하고 "Authenticated" 메시지 확인후 브라우저 창 닫음
- 터미널 화면에서 다음과 같은 메시지 확인하고 복사하여 실행 --〉 kube config의 user 설정

```
```
kubectl config set-credentials oidc \
  --exec-api-version=client.authentication.k8s.io/v1 \
  --exec-interactive-mode=Never \
  --exec-command=kubectl \
  --exec-arg=oidc-login \
  --exec-arg=get-token \
  --exec-arg="--oidc-issuer-url=https://keycloak.ssamz.192.168.56.80.nip.io/realms/k8s-realm" \
  --exec-arg="--oidc-client-id=test-client" \
  --exec-arg="--oidc-client-secret=iFiPFEFqIQHsvfxL3RchBji7NxxfhvYS"
```
```

# 6.11 클라이언트에서 kubelogin 활용하기

❖다음 명령어를 실행하여 context 등록 후 사용

```
kubectl config set-context oidc-context ₩
  --cluster=kubernetes ₩
  --user=oidc

kubectl config use-context oidc-context
```

❖기능 작동 여부 테스트 : 인증을 추가로 요구할 수 있음

- kubectl get pods //ok
- kubectl create ns test1 //ok
- kubectl delete ns test1 //forbidden 오류

```
stepano@laptop:~ $ kubectl get pods
error: could not open the browser: exec: "xdg-open,x-www-browser,www-browser": executable file not found in
$PATH

Please visit the following URL in your browser manually: http://localhost:8000/
No resources found in default namespace.
stepano@laptop:~ $ kubectl create ns test1
namespace/test1 created
stepano@laptop:~ $ kubectl delete ns test1
Error from server (Forbidden): namespaces "test1" is forbidden: User "https://keycloak.ssamz.192.168.56.80.n
ip.io/realms/k8s-realm#k8s-user1" cannot delete resource "namespaces" in API group "" in the namespace "test
1"
```

# 6.12 리소스 정리

## ❖master 가상머신에서

```
kubectl config use-context kubernetes-admin@kubernetes

helm uninstall keycloak -n keycloak

helm uninstall ingress-nginx -n ingress-nginx

kubectl delete ns ingress-nginx
kubectl delete ns keycloak
```

## ❖클라이언트(윈도우 우분투 터미널)에서

```
kubectl config use-context kubernetes-admin@kubernetes

kubectl config delete-context oidc-context
kubectl config delete-user oidc
```