10 보안을 위한 인증과 인가 : ServiceAccount와 RBAC

클라우드 플랫폼은 보통 수많은 사용자와 애플리케이션이 동시에 사용하는 것이 일반적입니다. 사내 개발 조직에 속해 있는 여러 명의 개발자가 동시에 클라우드를 사용할 수도 있고, 필요에 따라서는 수십 개의 애플리케이션이 클라우드상에서 동시에 실행될 수도 있습니다. 쿠버네티스 또한 예외는 아니며, 여러 명의 개발자가 쿠버네티스에 접근할 수도 있고, 각 개발자가 kubectl과 같은 명령어를 통해 애플리케이션을 동시에 배포하는 일도 빈번할 것입니다.

이처럼 여러 개발자와 애플리케이션이 쿠버네티스를 동시에 사용할 때 깊이 있게 고려해야 할 부분 중 하나는 바로 보안입니다. 쿠버네티스는 보안 측면에서도 다양한 기능을 제공하고 있는데, 그중에서 가장 자주 사용되는 것은 RBAC(Role Based Access Control)를 기반으로 하는 서비스 어카운트(Service Account)라는 기능입니다. 서비스 어카운트는 사용자 또는 애플리케이션 하나에 해당하며, RBAC라는 기능을 통해 특정 명령을 실행할 수 있는 권한을 서비스 어카운트에 부여합니다. 권한을 부여받은 서비스 어카운트는 해당 권한에 해당하는 기능만 사용할 수 있게 됩니다.

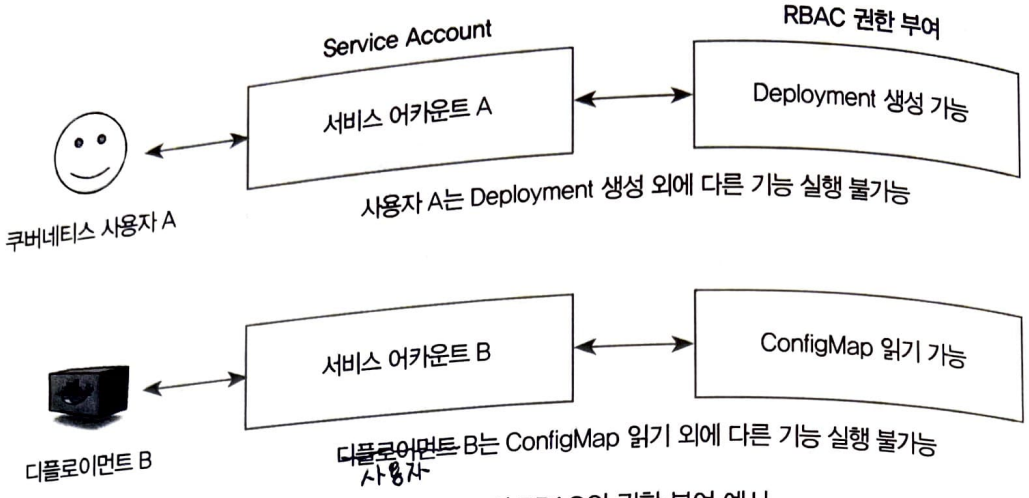

그림 10.1 서비스 어카운트와 RBAC의 권한 부여 예시

간단히 생각해서 리눅스에서 root 유저와 일반 유저를 나누는 기능을 쿠버네티스에서도 유사하게 사용할 수 있다고 생각하면 됩니다. root 유저는 최고 권한으로 모든 기능을 사용할 수 있지만, 보통 root 유저를 사용하는 것은 보안상의 이유로 권장되지 않습니다. 따라서 리눅스 일반 유저를 생성한 다음, 특정 명령어만 실행할 수 있도록 /etc/sudoer에서 설정해 사용하는 방법이 일반적입니다.

쿠버네티스도 리눅스와 매우 유사한 원리로 사용자 및 애플리케이션의 권한을 관리합니다. 뒤에서 조금 더 자세히 설명하겠지만, 지금까지 kubectl 명령어를 사용해왔던 권한은 사실 최상위에 해당하는, 마치 리눅스의 root 사용자와 같은 권한을 가지고 있습니다. 쿠버네티스를 학습하기 위한 용도라면 상관이 없지만, 쿠버네티스의 API에 접근하는 애플리케이션을 운영 환경에 배포하거나, 여러 명의 사용자가 동시에 쿠버네티스를 사용해야 한다면 최상위 권한을 사용하지 않는 것이 좋습니다. 사용자에게 필요한 권한만을 최소한으로 부여함으로써 실행할 수 있는 기능을 제한하는 것이 바람직할 것입니다.

이번 장에서는 서비스 어카운트(ServiceAccount) 및 RBAC를 사용하기 위한 롤(Role), 클러스터 롤(ClusterRole) 등을 먼저 사용해본 뒤, 사용자를 추상화한 유저(User) 및 그룹(Group), OIDC(Open ID Connect)에 대해서 알아보겠습니다.

10.1 쿠버네티스의 권한 인증 과정

쿠버네티스는 kube-apiserver, kube-controller, kube-scheduler, etcd 등과 같은 컴포넌트들로 구성돼 있습니다. 지금까지 살펴본 쿠버네티스의 기능을 익히는 데에는 이러한 컴포넌트들을 반드시 알 필요가 없기 때문에 자세히 설명하지는 않았습니다. 이러한 컴포넌트 중에서도 여러분이

가장 먼저 접하게 될, 그리고 자주 사용하게 될 컴포넌트가 있는데, 바로 쿠버네티스의 API 서버에 해당하는 kube-apiserver 컴포넌트입니다.

쿠버네티스 컴포넌트들은 kube-system 네임스페이스에서 실행되고 있으므로 직접 목록을 확인할 수도 있습니다.

```
$ kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
...
etcd-ip…                            1/1     Running   49         64d
kube-apiserver-ip…                  1/1     Running   15         7d17h
kube-controller-manager-ip…         1/1     Running   52         64d
...
```

여러분이 kubectl 명령어를 사용해 쿠버네티스의 기능을 실행하면 쿠버네티스 내부에서는 어떠한 일이 일어날까요? 사용자가 kubectl apply -f와 같은 간단한 명령어를 사용하더라도 쿠버네티스에서는 다음과 같은 복잡한 절차를 걸쳐 실제 기능을 실행합니다.

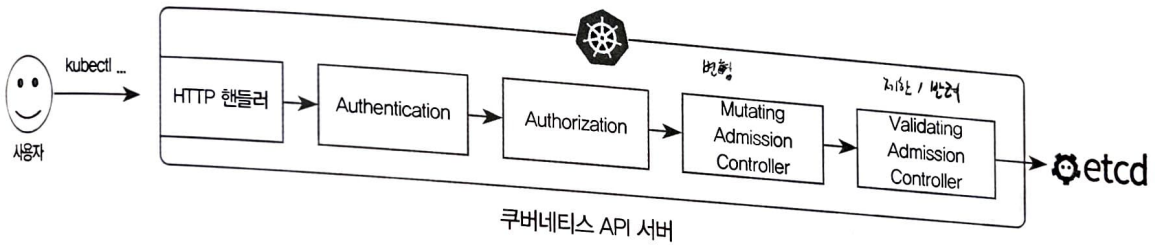

그림 10.2 kubectl 명령어를 사용할 때 내부적으로 처리되는 절차

가장 먼저 kubectl 명령어는 쿠버네티스 API 서버의 HTTP 핸들러에 요청을 전송합니다. API 서버는 해당 클라이언트가 쿠버네티스의 사용자가 맞는지(Authentication : 인증), 해당 기능을 실행할 권한이 있는지(Authorization : 인가) 확인합니다. 인증과 인가에는 서비스 어카운트 외에도 서드파티 인증(Open ID Connect: OAuth), 인증서 등과 같이 다양한 방법이 사용될 수 있습니다. 그 뒤에는 어드미션 컨트롤러(Admission Controller)라는 별도의 단계를 거친 뒤 비로소 요청받은 기능을 수행합니다.[1]

그렇지만 지금까지 kubectl 명령어를 사용할 때는 이러한 단계들을 전혀 신경 쓰지 않았다는 점이 의아할 수도 있습니다. 여러분은 인증을 위한 계정 같은 것을 쿠버네티스에서 생성한 적도 없고,

1 어드미션 컨트롤러에 대해서는 이번 장에서 다루지 않으며, 다음 장의 11.1절에서 간단히 설명합니다.

그러한 계정에 권한을 부여한 적도 없습니다. 사실 이는 설치 도구를 이용해 쿠버네티스를 설치할 때 설치 도구가 자동으로 kubectl이 관리자 권한을 갖도록 설정해 두기 때문인데, 그러한 설정은 ~/.kube/config라는 파일에서 확인할 수 있습니다.

```
# ~/.kube/config 파일의 예시
apiVersion: v1

clusters:
- cluster:
    certificate-authority-data: ...
    server: https://10.43.0.20:6443
  name: kubernetes

contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes

users:
- name: kubernetes-admin
  user:
    client-certificate-data: ...
    client-key-data: ...

kind: Config
preferences: {}
```

kubectl을 사용할 때는 기본적으로 ~/.kube/config라는 파일에 저장된 설정을 읽어 들여 쿠버네티스 클러스터를 제어합니다. 이 파일에 저장된 내용 중에서 users라는 항목에는 인증을 위한 데이터가 설정돼 있습니다. client-certificate-data와 client-key-data에 설정된 데이터는 base64로 인코딩된 인증서(공개키와 비밀키)인데, 이 키 쌍은 쿠버네티스에서 최고 권한(cluster-admin)을 갖습니다. 그렇기 때문에 지금까지 여러분은 아무런 문제 없이 쿠버네티스의 모든 명령어를 사용할 수 있었던 것입니다.

~/.kube/config 파일 내용에 대해서는 지금 당장 이해하지 않아도 괜찮습니다. users 항목에는 인증을 위한 정보를, clusters 항목에는 클러스터에 접근하기 위한 정보를 저장한다는 정도만 알고 넘어가면 됩니다.

기본적으로 설정된 ~/.kube/config 파일에서는 인증서 키 쌍을 사용해 API 서버에 인증하지만, 이 인증 방법은 비교적 절차가 복잡하고 관리하기가 어렵기 때문에 자주 사용하는 방법은 아닙니다.[2] 쿠버네티스에서는 인증을 위해 인증서 키 쌍뿐만 아니라 여러 가지 방법을 사용할 수 있으며, 그중 하나가 이번 장에서 주로 다뤄볼 서비스 어카운트(ServiceAccount)입니다.

10.2 서비스 어카운트와 롤(Role), 클러스터 롤(Cluster Role)

서비스 어카운트는 체계적으로 권한을 관리하기 위한 쿠버네티스 오브젝트입니다. 서비스 어카운트는 한 명의 사용자나 애플리케이션에 해당한다고 생각하면 이해하기 쉽습니다. 서비스 어카운트는 네임스페이스에 속하는 오브젝트로, serviceaccount 또는 sa라는 이름으로 사용할 수 있습니다.

```
$ kubectl get sa # sa라는 이름으로도 사용 가능
$ kubectl get serviceaccount
NAME      SECRETS   AGE
default   1         64d
```

여러분이 서비스 어카운트를 생성하지 않았더라도 각 네임스페이스에는 기본적으로 default라는 이름의 서비스 어카운트가 존재합니다. kubectl create나 delete를 통해 간단하게 서비스 어카운트를 생성하거나 삭제할 수도 있습니다.

```
$ kubectl create sa alicek106
serviceaccount/alicek106 created
```

지금까지 여러분이 kubectl 명령어를 사용했을 때에는 ~/.kube/config 파일에 저장돼 있던 관리자 권한의 인증 정보를 사용했지만, 이번에는 방금 생성한 alicek106이라는 이름의 서비스 어카운트를 이용해 kubectl 명령어를 사용해 보겠습니다. --as 옵션을 사용하면 임시로 특정 서비스 어카운트를 사용할 수 있습니다.[3,4]

```
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   64d

$ kubectl get services --as system:serviceaccount:default:alicek106
Error from server (Forbidden): services is forbidden: User "system:serviceaccount:default:alic
ek106" cannot list resource "services" in API group "" in the namespace "default"
```

2 인증서를 이용한 사용자 인증 방법은 이번 장의 뒷쪽에서 다시 다룹니다.

3 이처럼 특정 사용자로 가장해 API를 사용하는 것을 쿠버네티스 공식 문서에서는 Impersonate라고 표현합니다.

4 Docker Desktop에서는 모든 서비스 어카운트(system:serviceaccount)에 대해 클러스터 관리자 권한(cluster-admin이라는 이름의 클러스터 롤)이 할당되어 있습니다. 이 권한이 연결되어 있는 클러스터 롤 바인딩의 이름은 docker-for-desktop-binding이며, 원활한 실습을 위해 이 클러스터 롤 바인딩을 삭제해두는 것도 나쁘지 않습니다. (https://github.com/docker/for-mac/issues/3694)

--as 옵션에 사용된 system:serviceaccount는 인증을 위해 서비스 어카운트를 사용한다는 것을 나타내며, default:alicek106은 default 네임스페이스의 alicek106 서비스 어카운트를 의미합니다. 뒤에서 다시 설명하겠지만, 쿠버네티스는 서비스 어카운트 외에도 다양한 인증 방법을 제공하기 때문에 이러한 형식의 이름을 사용합니다.

방금 생성한 alicek106 서비스 어카운트로 서비스의 목록을 조회했더니 API 서버로부터 에러가 반환됐습니다. 이 서비스 어카운트는 default 네임스페이스에서 서비스 목록을 조회할 수 있는 권한이 아직 부여되지 않았다는 뜻입니다. 따라서 서비스 어카운트에 적절한 권한을 부여해야만 쿠버네티스의 기능을 제대로 사용할 수 있습니다.

쿠버네티스에서 권한을 부여하는 방법은 크게 두 가지가 있습니다. 롤(Role)과 클러스터 롤(Cluster Role)을 이용해 권한을 설정하는 것입니다.

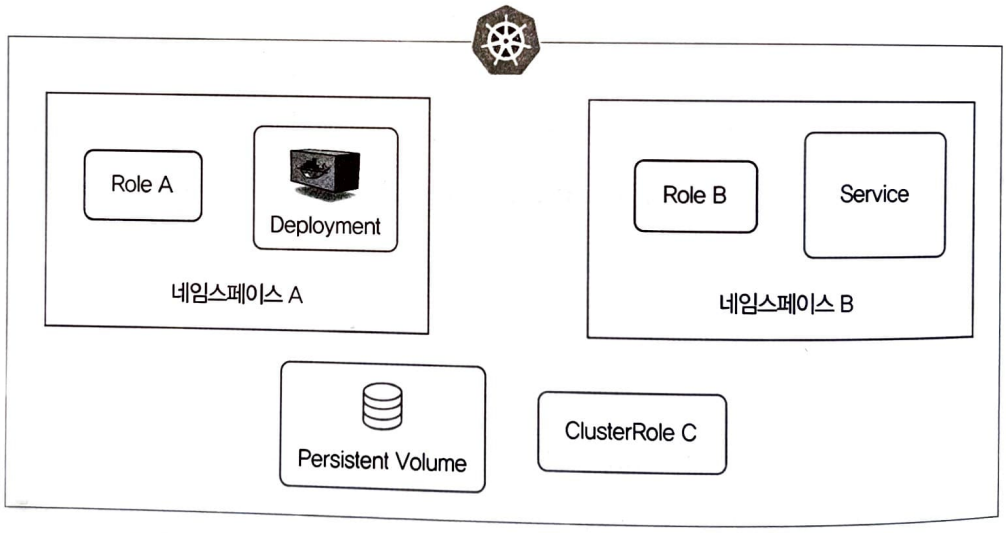

그림 10.3 롤은 네임스페이스에 속하며, 클러스터 롤은 네임스페이스에 속하지 않음

롤과 클러스터 롤은 부여할 권한이 무엇인지를 나타내는 쿠버네티스 오브젝트입니다. 예를 들어 '디플로이먼트를 생성할 수 있다'라는 것도 하나의 롤이 될 수 있고, '서비스의 목록을 조회한다'라는 것도 롤이 될 수 있습니다. 단, 롤은 네임스페이스에 속하는 오브젝트이므로 디플로이먼트나 서비스처럼 네임스페이스에 속하는 오브젝트들에 대한 권한을 정의할 때 쓰입니다.

롤와 비슷한 쿠버네티스 오브젝트로는 클러스터 롤이 있는데, 클러스터 롤은 말 그대로 클러스터 단위의 권한을 정의할 때 사용합니다. 예를 들어 '퍼시스턴트 볼륨의 목록을 조회할 수 있다'라는 권한

은 클러스터 롤로 정의할 수 있습니다.[5] 또한 네임스페이스에 속하지 않는 오브젝트뿐만 아니라 클러스터 전반에 걸친 기능을 사용하기 위해서도 클러스터 롤을 정의할 수 있으며, 여러 네임스페이스에서 반복적으로 사용되는 권한을 클러스터 롤로 만들어 재사용하는 것도 가능합니다.

가장 먼저 롤과 클러스터 롤을 사용해 보겠습니다. 롤은 네임스페이스에 속하지만, 클러스터 롤은 네임스페이스에 속하지 않는 전역적인 쿠버네티스 오브젝트입니다. 따라서 kubectl get role 명령어는 현재 네임스페이스의 롤 목록만을 출력하지만, kubectl get clusterrole 명령어는 클러스터 자체에 존재하는 모든 클러스터 롤의 목록을 출력합니다.

```
$ kubectl get role
No resources found.

$ kubectl get clusterrole
NAME                                                   AGE
admin                                                  64d
calico-node                                            64d
cluster-admin                                          64d
edit                                                   64d
nginx-ingress-clusterrole                              22d
...
```

클러스터 롤은 쿠버네티스 컴포넌트가 사용하는 권한도 포함하기 때문에 꽤 많은 수의 클러스터 롤이 미리 생성돼 있습니다. 그중에는 이전에 Nginx 인그레스 컨트롤러를 사용할 때 함께 생성됐던 nginx-ingress-clusterrole이나, 쿠버네티스에서 관리자 권한으로 모든 기능을 사용할 수 있는 cluster-admin이라는 클러스터 롤도 있습니다.

~/.kube/config 파일에 기본적으로 설정돼 있던 인증서에는 cluster-admin 클러스터 롤이 부여돼 있습니다. 이처럼 인증서에 권한을 부여하는 방법은 이번 장의 뒤쪽에서 다시 설명합니다.

지금은 롤을 먼저 사용해 보겠습니다. 아래의 내용으로 YAML 파일을 작성합니다.

예제 10.1 chapter10/service-reader-role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
```

5 이전에 설명했던 것처럼 퍼시스턴트 볼륨은 네임스페이스에 속하지 않는 클러스터 수준의 오브젝트입니다.

```
  name: service-reader
rules:
- apiGroups: [""]                    # 1. 대상이 될 오브젝트의 API 그룹
  resources: ["services"]            # 2. 대상이 될 오브젝트의 이름
  verbs: ["get", "list"]             # 3. 어떠한 동작을 허용할 것인지 명시
```

위 YAML 파일은 서비스의 목록을 읽을 수 있는 롤을 정의합니다. metadata 항목의 namespace는 롤이 생성될 네임스페이스를, name은 롤의 이름을 설정했습니다. 가장 중요한 부분인 rules 항목의 내용을 조금 자세히 살펴보겠습니다.

- apiGroups : 어떠한 API 그룹에 속하는 오브젝트에 대해 권한을 지정할지 설정합니다. API 그룹은 쿠버네티스의 오브젝트가 가지는 목적에 따라 분류되는 일종의 카테고리입니다. 이 예시에서는 ""로 설정했는데, 이는 포드, 서비스 등이 포함된 코어 API 그룹을 의미합니다. kubectl api-resources 명령어를 사용하면 특정 쿠버네티스 오브젝트가 어떤 API 그룹에 속하는지 확인할 수 있습니다.

```
$ kubectl api-resources
NAME                SHORTNAMES   APIGROUP   NAMESPACED   KIND
...
pods                po                      true         Pod
...
services            svc                     true         Service
...
deployments         deploy       apps       true         Deployment
```

  포드나 서비스 등은 코어 API 그룹에 속하기 때문에 아무것도 표시되지 않습니다. 디플로이먼트나 레플리카셋 등은 apps라는 이름의 API 그룹에 속합니다.

- resources : 어떠한 쿠버네티스 오브젝트에 대해 권한을 정의할 것인지 입력합니다. 위 예시에서는 서비스(services)를 다룰 수 있는 권한을 정의할 것이기 때문에 ["services"]와 같이 설정했습니다. resources 항목 또한 kubectl api-resources에 출력되는 오브젝트의 이름을 적절히 사용하면 됩니다.
- verbs : 이 롤을 부여받은 대상이 resources에 지정된 오브젝트들에 대해 어떤 동작을 수행할 수 있는지 정의합니다. 위 예시에서는 get과 list 동작을 명시했으므로 kubectl get services 명령어로 개별 서비스의 정보를 가져오거나 모든 서비스 목록을 확인할 수 있도록 권한이 부여됩니다.

YAML 파일에서 ["1", "2"..]와 같이 대괄호를 이용한 표현은 배열과 같은 기능을 합니다. verbs: ["get", "list"] 항목은 아래와 같이 표현할 수도 있습니다.

```
...
resources: ["services"]
  verbs:
  - get
    list
```

따라서 이 YAML 파일에 정의된 apiGroup과 resources, verbs를 종합하면 "코어 API 그룹("")에 속하는 서비스 리소스에 대해 get과 list를 실행할 수 있다"라고 해석할 수 있습니다. 이 YAML 파일을 이용해 롤을 생성해 보겠습니다.

```
$ kubectl apply -f service-reader-role.yaml
role.rbac.authorization.k8s.io/service-reader created

$ kubectl get roles
NAME             AGE
service-reader   105s
```

그렇지만 롤은 특정 기능에 대한 권한만을 정의하는 오브젝트이기 때문에 롤을 생성하는 것만으로는 서비스 어카운트나 사용자에게 권한이 부여되지 않습니다. 이 롤을 특정 대상에게 부여하려면 롤 바인딩(RoleBinding)이라는 오브젝트를 통해 특정 대상과 롤을 연결해야 합니다. 예를 들어 서비스 어카운트에 롤에 정의된 권한을 부여하려면 아래와 같은 롤 바인딩을 생성하면 됩니다.

예제 10.2 chapter10/rolebinding-service-reader.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-reader-rolebinding
  namespace: default
subjects:
- kind: ServiceAccount      # 권한을 부여할 대상이 ServiceAccount입니다.
  name: alicek106           # alicek106이라는 이름의 서비스 어카운트에 권한을 부여합니다.
  namespace: default
roleRef:
  kind: Role                # Role에 정의된 권한을 부여합니다.
  name: service-reader      # service-reader라는 이름의 Role을 대상(subjects)에 연결합니다.
  apiGroup: rbac.authorization.k8s.io
```

롤 바인딩에서는 어떠한 대상을 어떠한 롤에 연결할 것인지 정의합니다. 위 예시에서는 subjects 항목에 alicek106이라는 이름의 서비스 어카운트를, roleRef 항목에 service-reader 롤을 지정했습니다. 따라서 alicek106 서비스 어카운트는 service-reader 롤에 정의된 권한을 사용할 수 있게 됩니다.

롤 바인딩을 생성한 뒤 다시 alicek106 서비스 어카운트로 명령어를 실행해 보겠습니다.

```
$ kubectl apply -f rolebinding-service-reader.yaml
rolebinding.rbac.authorization.k8s.io/service-reader-rolebinding created

$ kubectl get services --as system:serviceaccount:default:alicek106
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3d21h
```

서비스의 목록을 확인할 수 있는 권한을 부여받았기 때문에 정상적으로 kubectl get services 명령어를 사용할 수 있습니다. 그렇지만 서비스 어카운트에 부여되지 않은 다른 기능들은 여전히 사용할 수 없는 상태입니다.

```
$ kubectl get deployment --as system:serviceaccount:default:alicek106
Error from server (Forbidden): deployments.extensions is forbidden: User "system:serviceaccou
nt:default:alicek106" cannot list resource "deployments" in API group "apps" in the namespace
"default"
```

롤 바인딩과 롤, 서비스 어카운트는 모두 1:1 관계가 아니라는 점에 유의해야 합니다. 하나의 롤은 여러 개의 롤 바인딩에 의해 참조될 수도 있고, 하나의 서비스 어카운트는 여러 개의 롤 바인딩에 의해 권한을 부여받을 수도 있습니다. 즉, 롤은 권한을 부여하기 위한 일종의 템플릿과 같은 역할을, 롤 바인딩은 롤과 서비스 어카운트를 연결하기 위한 중간 다리 역할을 하는 셈입니다.

롤이나 클러스터 롤에서 사용되는 verbs 항목에는 get, list, watch, create, update, patch, delete 등에서 선택해 사용할 수 있지만, 와일드카드를 의미하는 *를 사용할 수도 있습니다.
단, 특정 리소스에 한정된 기능을 사용할 때는 서브 리소스(sub resource)를 명시해야 할 수도 있습니다. 예를 들어 kubectl exec 명령어로 포드 내부에 들어가기 위한 권한을 생성하려면 포드의 하위 리소스인 pod/exec을 resources 항목에 정의해야 합니다.[5]

```
...
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create"]
...
```

6 https://kubernetes.io/docs/reference/access-authn-authz/rbac/#referring-to-resources