Chapter 9

Customizing Kubernetes API

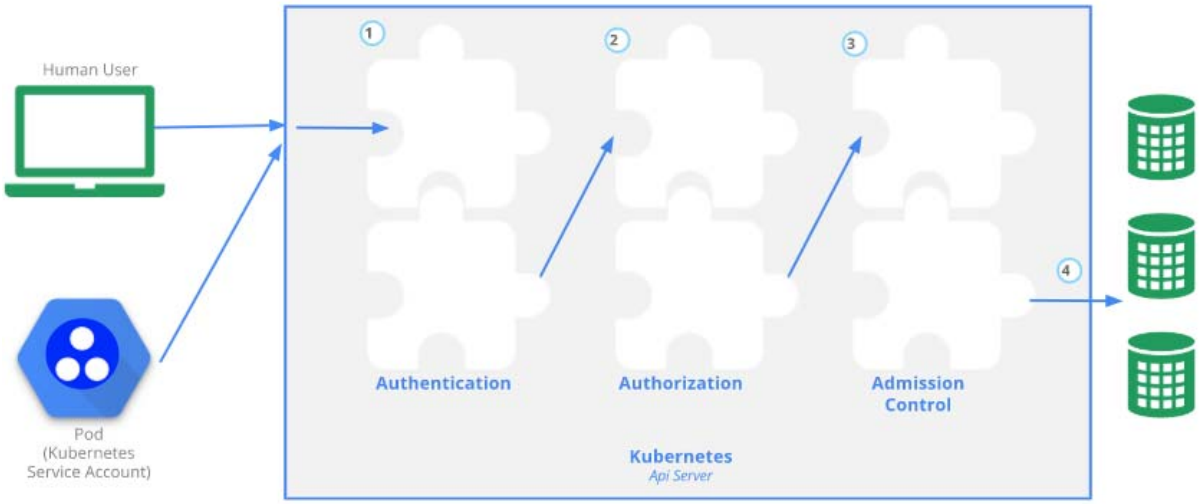
API 인증

- **API 서버에 접근하기 위해서는 인증작업 필요**
  - 일반사용자
  - ServiceAccount
- **일반 사용자 및 그룹**
  - 클러스터 외부에서 쿠버네티스를 조작하는 사용자
  - 다양한 방법의 인증을 거친다.
- **Service Account**
  - 쿠버네티스 내부적으로 관리되는 Account
  - 포드가 쿠버네티스 API를 다룰 때 사용하는 사용자

API 인증

쿠버네티스를 운영하면서 보안을 확보하는 방법이 여러 가지 있다.

사용자가 쿠버네티스의 API 에 접근하기 위해서는 인증(Authentication)을 거쳐야 한다. API server 는 테스트 목적으로 로컬호스트의 8080 포트에 http 포트를 띄우고, 외부에서 접근할 수 있는 기본 포트는 6443 으로 TLS 인증이 적용되어 있다. 일반적인 https 인증은 접근하는 클라이언트에서는 인증서가 필요 없지만, 쿠버네티스의 API server 에 열려 있는 포트에 접근하기 해서는 APIserver 의 인증서를 가지고 접근해야 통신이 가능하다.



https://kubernetes.io/docs/reference/access-authn-authz/controlling-access/

쿠버네티스에서 인증을 요청하기 위한 사용자는 일반적인 사용자(Human user)와 서비스어카운트(service account) 두 가지 개념을 가진다.

명칭	내용
일반사용자 (Human User)	클러스터 **외부에서 쿠버네티스를 조작하는 사용자**로, 다양한 방법으로 인증을 거친다.
서비스계정 (serviceaccoount)	쿠버네티스 내부적으로 관리되며 **포드가 쿠버네티스 API를 다룰 때 사용하는 사용자**

일반 사용자(Human user)는 개발자 및 운영 실무자가 쿠버네티스를 조작하기 위해 사용한다. 쿠버네티스 클러스터 외부로부터 들어오는 접근을 관리하기 위한 사용자다. 일반 사용자를 분류하는 그룹 개념도 있어서 이 그룹 단위로 권한을 부여할 수도 있다.

서비스 계정(service account)은 쿠버네티스 리소스의 일종이다. 쿠버네티스 클러스터 내부에서 권한을 관리하는 역할을 한다. 서비스 계정과 연결된 파드는 주어진 권한에 따라 쿠버네티스 리소스를 다룰 수 있다.

서비스 계정 및 일반 사용자의 권한은 **RBAC(Role-Based Access Control)**이라는 메커니즘을 통해 제어된다. RBAC 는 롤에 따라 리소스에 대한 권한을 제어하는 기능이자 개념이다. RBAC 를 적절히 사용해 쿠버네티스 리소스의 보안을 확보할 수 있다.

배포와 관련된 서비스나 디플로이먼트의 접근 권한을 일부 일반 사용자에게만 허용하거나, 포드의 로그 열람 권한을 다른 일반 사용자에게도 허용하는 등이 정책을 **일반 사용자 권한 부여**로 실현할 수 있다.

서비스 계정은 애플리케이션을 통해 쿠버네티스 조작을 통제할 수 있다는 점이 장점이다. 클러스터 안에서 봇을 동작시키는 파드에 권한을 부여해두고, 이 봇으로 기존 디플로이먼트를 업데이트하거나 Replicas 의 수를 조절하는 식으로 활용할 수 있다.

일반 사용자 및 그룹

- **클러스터 외부에서 쿠버네티스를 조작하는 사용자(예:root, guru)**
- **쿠버네티스는 role이 연결될 일반 사용자 또는 일반 사용자 그룹을 지정**
- **일반 사용자는 API에 인증서를 이용해 인증 받음**
  # cat ~/.kube/config

일반 사용자 및 일반 사용자 그룹

쿠버네티스는 role 이 연결될 일반 사용자 또는 일반 사용자 그룹을 지정할 수 있다. 일반 사용자의 인증은 다음과 같은 방법으로 할 수 있다.

- 서비스 계정 토큰 방식
- 정적 토큰 방식
- 패스워드 파일 방식 : 사용자와 패스워드를 파일에 정의하는 방식
- X509 클라이언트 인증서 방식: 클라이언트 인증서를 이용한 인증 방식
- OpenID 방식 : OpenID 프로바이더(구글 등)을 이용한 인증방식

일반 사용자 인증 : 인증서를 이용한 인증

쿠버네티스는 apiserver 와 통신하기 위해서 사용하는 기본 인증 방법으로 TLS(Transport Layer Security)인증을 사용한다. apiserver 쪽에는 있는 인증서와 매치되는 클라이언트 인증서를 가지고 접속을 시도하면 된다. 그동안 kubectl 을 설치한 후에 별다른 문제없이 명령어들을 사용해 왔는데, 이것은 이미 kubectl 설정에 이 인증서가 들어 있기 때문에 가능한 것이었다.

```
# cat ~/.kube/config
  apiVersion: v1
  clusters:
  - cluster:
      certificate-authority-data: LS0t....
      server: https://10.100.0.50:6443
    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: LS0tL...
    client-key-data: LS0tLS1C...
```

```
# kubectl config view
```

cluster 설정에는 클러스터 이름을 나타내는 name 과 접속 주소를 나타내는 server 가 있다. 외부에 있는 쿠버네티스에 접근하려면 여기 있는 주소를 외부 쿠버네티스의 apiserver 주소로 변경해 주면 된다. cluster 다음에는 하단의 Users 가 있다. 여러 명의 user 를 명시해 두고 사용하는 것이 가능한데, 여기서는 TLS 인증을 사용하는 유저이기 때문에 client-certificate-data 와 client-key-data 를 사용한다.

contexts 부분을 보면 context 는 실제로 어떤 cluster 에 접근하기 위해 필요한 user 가 누구인지를 연결해 주고 있다. 그리고 여기에는 생략되어 있지만 context 에는 namespace 까지 지정할 수 있어서 default 네임스페이스가 아닌 특정 네임스페이스에 대한 작업을 할 수 있게 한다.

마지막으로 **current-context** 를 보면 설정에 있는 여러 가지의 컨텍스트 중에서 현재 어떤 컨텍스트를 사용해서 클러스터에 접근할지를 결정한다. ~/.kube/config 파일의 내용을 직접 편집해도 되고 kubectl config set-context 를 이용해서 컨텍스트를 변경하면서 다양한 클러스터에 다양한 유저로 접근해서 사용하는 것이 가능하다.

Service Account

- **Pod, Secret, Configmap등과 같은 리소스**
- **하나의 namespace 접근만 허용됨(네임스페이스 격리가 보장됨)**
- **Pod, Secret, Configmap등은 정확히 하나의 ServiceAccount와 연관됨**
- **각 namespace에는 default 라는 기본 ServiceAccount가 존재**
- **Pod 생성 시 ServiceAccount를 설정하지 않으면 default로 정의됨**
- **Pod에 ServiceAccount를 할당하면 Pod에 접근할 수 있는 리소스를 제어함**

  # kubectl get serviceaccounts

  # kubectl get pod

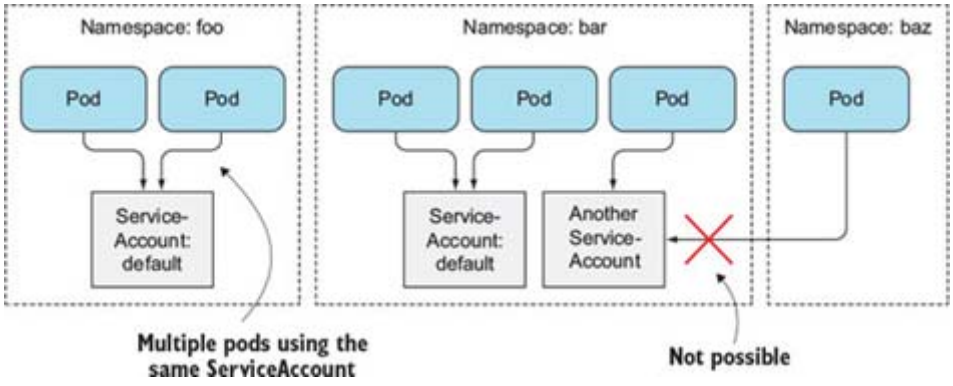
ServiceAccount : 토큰을 이용한 인증

ServiceAccount 는 Pod, Secret, Configmap 등과 같은 리소스이며 하나의 네임스페이스로 범위가 지정된다. API Server 가 Pod 의 요청을 수신할 때마다 자체 네임스페이스만 자격증명이 적용되기 때문에 네임스페이스의 격리가 보장된다.

쿠버네티스는 Pod 를 대신하여 ServiceAccount 를 관리한다. 쿠버네티스가 Pod 를 인스턴스할 때마다 해당 Pod 에 서비스 계정을 할당한다. ServiceAccount 는 API 서버와 상호작용을 할 때 모든 Pod 프로세스를 식별한다. 각각의 서비스 계정은 일련의 자격증명의 집합을 가지고 있으며 Secret Volume 에 탑재되어 있다. 각각의 네임스페이스는 **default** 라는 기본 ServiceAccount 가 있다. Pod 를 만들 때 다른 ServiceAccount 를 지정하지 않으면 default 서비스 계정이 할당된다.

```
# kubectl get serviceaccounts
  NAME       SECRETS    AGE
  default    1          6d3h
```

각 Pod 는 정확히 하나의 ServiceAccount 와 연관되어 있으나 여러 pod 에서 같은 Service Account 를 사용할 수 있다. Pod 는 동일 네임스페이스의 service account 만 사용할 수 있다.



특별히 지정하지 않으면 기본으로 Pod 는 자신이 속한 namespace 의 default 서비스어카운트를 사용한다. Pod 에 각기 다른 ServiceAccount 를 할당하면 각 Pod 에 접근할 수 있는 리소스를 제어할 수 있다.

```
# kubectl get pod testpod -o yaml
  ...
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
```

default 사용자의 account token 이 있다. 이것을 인증에 사용하는 것이다.

```
# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-57gkx   kubernetes.io/service-account-token   3      6d3h
```

ServiceAccount 생성

- **default Account 외 다른 Account 생성 가능**
- **클러스터 접근을 제한하기 위해 생성**

  # kubectl create serviceaccount foo

  # kubectl get serviceaccounts

ServiceAccounts 생성

모든 namespace 에는 자체 디폴트 serviceaccount 가 있으나 필요한 경우 추가할 수 있다.

serviceaccount 를 생성하는 이유는 보안 때문이다. 클러스터 메타 데이터를 읽을 필요가 없는 포드는 클러스터에 배포된 리소스를 검색하거나 수정할 수 없는 제한된 계정에서 실행해야한다.

리소스 메타데이터를 검색해야 하는 포드는 해당 객체의 메타 데이터를 읽을 수 있는 서비스 어카운트 아래에서 실행해야하지만 해당 객체를 수정해야 하는 포드는 API 객체를 수정할 수 있는 자체 서비스어카운트에서 실행되어야 한다.

Creating a ServiceAccount:

```
# kubectl create serviceaccount foo
```

```
# kubectl get serviceaccounts
NAME      SECRETS   AGE
default   1         6d3h
foo       1         15s
```

생성된 foo serviceaccount 의 secret 정보를 확인해보자.

```
# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-57gkx   kubernetes.io/service-account-token   3      6d3h
foo-token-6c8vl       kubernetes.io/service-account-token   3      8m20s
```

foo ServiceAccont 의 정보를 자세히 보면 사용하는 secret Token 을 확인할 수 있다.

```
# kubectl describe serviceaccounts foo
Name:                foo
Namespace:           default
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   foo-token-6c8vl
Tokens:              foo-token-6c8vl
Events:              <none>
```

serviceaccount 가 생성되고, secret 이 생성되어 foo ServiceAccount 와 연결되었다. secret 에는 인증서, 네임스페이스, 토큰이 포함되어 있다.

```
# kubectl describe secrets foo-token-6c8vl
Name:         foo-token-6c8vl
Namespace:    default
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: foo
              kubernetes.io/service-account.uid: 36e58304-9356-11e9-b848-5254003462c1

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  7 bytes
token:      eyJhb.....
```

사용자 지정 ServiceAccount 할당

앞서 생성한 foo ServiceAccount 를 Pod 에 할당해본다. 해당 Pod 에 ServiceAccount 를 default 가 아닌 foo 로 설정하면 foo 서비스어카운트로 API 서버와 통신 할 수 있다.

smlinux/curl	API의 REST 인터페이스를 탐색
smlinux/kubectl-proxy	pod의 service account token을 사용해 API 서버로 인증하기 위해 사용

```
# cat curl-custom-sa.yaml
apiVersion: v1
kind: Pod
metadata:
  name: curl-custom-sa
spec:
```

```
    serviceAccountName: foo
    containers:
    - name: main
      image: smlinux/curl
      command: ["sleep", "9999999"]
    - name: ambassador
      image: smlinux/kubectl-proxy:1.6.2
```

```
# kubectl create -f curl-custom-sa.yaml
# kubectl get pods
NAME             READY   STATUS      RESTARTS    AGE
curl-custom-sa   2/2     Running     0           23s
```

생성한 ServiceAccount 의 토큰이 두 컨테이너에 마운트 되었는지 확인하려면 아래의 명령을 실행하여 토큰을 확인해보면 된다.

```
# kubectl exec  -it curl-custom-sa  -c main -- ls /var/run/secrets/kubernetes.io/serviceaccount -l
  lrwxrwxrwx 1 root root 13 Jun 20 12:48 ca.crt -> ..data/ca.crt
  lrwxrwxrwx 1 root root 16 Jun 20 12:48 namespace -> ..data/namespace
  lrwxrwxrwx 1 root root 12 Jun 20 12:48 token -> ..data/token
```

```
# kubectl exec  -it curl-custom-sa  -c main ₩
    -- cat /var/run/secrets/kubernetes.io/serviceaccount/token
  eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3N
```

생성한 토큰으로 API 서버와 통신을 해보자. localhost:8001 번의 proxy 를 통해서 들어간다.

```
# kubectl exec -it curl-custom-sa -c main curl localhost:8001/api/v1/pods
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "pods is forbidden: User \"system:serviceaccount:default:foo\" cannot lis
 resource \"pods\" in API group \"\" at the cluster scope",
  "reason": "Forbidden",
  "details": {
    "kind": "pods"
  },
  "code": 403
}[root@master yaml]#
```

정확한 결과가 표시되지 않고 403 forbidden 에러가 표시된다. 쿠버네티스 1.6 버전부터 클러스터 보안이 상당히 향상되었다. 이전버전에서는 포드 중 하나에서 인증 토큰을 얻는데 성공하면 클러스터에서 원하는 작업을 모두 수행할 수 있었다. 1.8 이후버전에서는 RBAC 승인 플러그인 GA(General Availability)로 승격하여 이제는 많은 클러스터에서 기본적으로 활성화되었다.

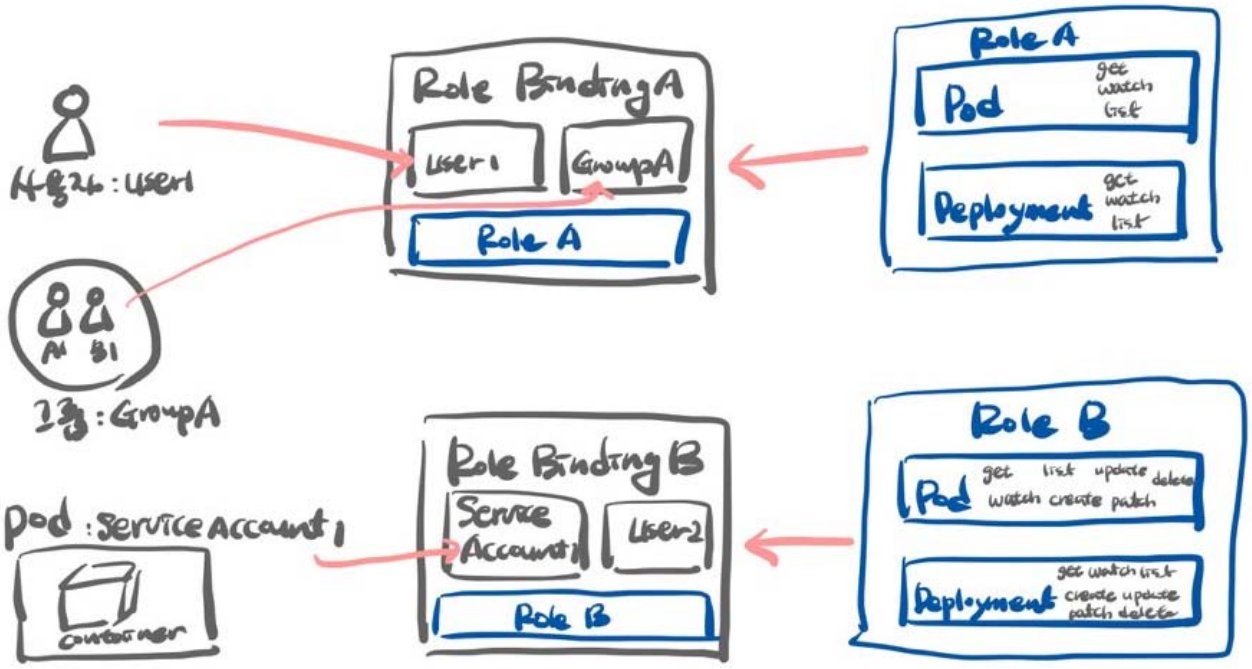
RBAC를 이용한 권한제어

- **권한제어**
- **Role**
  - 어떤 API를 이용할 수 있는지의 정의
  - 쿠버네티스의 사용권한을 정의
  - 지정된 네임스페이스에서만 유효
- **RoleBinding**
  - 사용자/그룹 또는 Service Account와 role을 연결

RBAC를 이용한 권한제어

RBAC 를 이용한 권한제어는 크게 두가지 요소로 구현된다.

- 어떤 쿠버네티스 API를 사용할 수 있는지가 정의된 role
- 이 role을 일반 사용자(그룹) 또는 service account와 연결해주는 Binding이다.



사용자/그룹/서비스계정을 role 과 연결

RBAC authorization plugin

- **RBAC : 권한 없는 사용자가 클러스터 상태를 보거나 수정할 수 없게 제한**
  - 서비스어카운트 정보 상태 보기 불가능
  - 추가 권한 할당받지 않고는 클러스터 수정할 수 없음
- **ACTION**
  - GET, POST, PUT, DELETE
- **RBAC 플러그인**
  - 사용자가 액션을 수행할 수 있는지 여부를 결정하는 핵심요소

RBAC authorization plugin

RBAC 는 권한이 없는 사용자가 클러스터 상태를 보거나 수정할 수 없도록 한다.
default 서비스어카운트는 추가 권한을 부여하지 않는 한 어떤 방식으로든 수정하지 않고는 클러스터 상태를 볼 수 없다. 쿠버네티스 API 서버와 통신하는 애플리케이션을 작성하려면 RBAC 관련 리소스를 이용한 권한 부여 관리방법을 이해해야한다.

ACTION

REST 클라이언트는 GET, POST, PUT, DELETE 및 기타 HTTP 요청을 특정 REST 리소스를 나타내는 특정 URL 로 보낸다. 쿠버네티스에서 이런 리소스는 pod, service, secret 등이 있다.

쿠버네티스에서 취할 수 있는 action 은 다음과 같다.

- 포드가져오기
- 서비스 생성하기
- 시크릿 업데이트하기
- 기타

RBAC 플러그인

사용자의 역할을 보고 사용자가 액션을 수행할 수 있는지 여부를 결정할 때 핵심 요소로 사용한다. 이런 역할을 가진 사용자는 자신의 역할이 허용하는 모든 작업을 수행할 수 있다.

예를들어 SECRET UPDATE 와 같은 역할이 사용자에게 없는 경우 API 서버는 사용자가 시크릿에 PUT 또는 PATCH 요청을 수행하지 못하게 한다.

- example

RBAC 리소스가 API 서버와 통신하는지 알아보자.

1. API 서버와 통신할 수 있는 포드를 실행하면 되는데, 이전에 네임스페이스 보안이 어떻게 되는지 알아보자. 다른 namespace 공간에서 pod 를 생성하자

```
# kubectl create ns foo
# kubectl create ns bar
# kubectl get namespaces

# kubectl run test --image=smlinux/kubectl-proxy -n foo
# kubectl run test --image=smlinux/kubectl-proxy -n bar
  deployment "test" created

# kubectl get deployment -n foo
  NAME    READY    UP-TO-DATE    AVAILABLE    AGE
  test      0/1      0               0          14s

# kubectl get po -n foo
  NAME                      READY    STATUS     RESTARTS    AGE
  test-5f89f97665-99bdf      1/1     Running     0          13s

# kubectl get po -n bar
  NAME                      READY    STATUS     RESTARTS    AGE
  test-7fcbd9fbbc-l49vk      1/1     Running     0          7m
```

2. 포드를 실행하자. 두개의 터미널을 열고 각 포트에 접속한다.

```
T3# kubectl exec -it test-7fcbd9fbbc-l49vk –n bar -- sh
  / #

T2# kubectl exec -it test-7fcbd9fbbc-4w4xg –n foo -- sh
  / #
```

RBAC 가 활성화 되었어도 포드가 클러스터 상태를 읽지 못함을 확인해보자. curl 을 이용해 foo 네임스페이스의 서비스를 나열해보자

```
T2 /# curl localhost:8001/api/v1/namespaces/foo/services
  ..
  "status": "Failure",
```

"message": "services is forbidden: User ₩"system:serviceaccount:foo:defaultcannot list services in the namespace ₩"foo
"reason": "Forbidden",
"details": {
"kind": "services"
},
"code": 403

API 서버는 포드가 동일한 네임스페이스에서 실행중이더라도 서비스어카운트가 foo 네임스페이스에서 default 서비스어카운트가 foo 네임스페이스의 서비스의 리스트 요청을 허용하지 않았다. 잘 작동하고 있는 것이다.

3. 서비스어카운트를 사용해보자

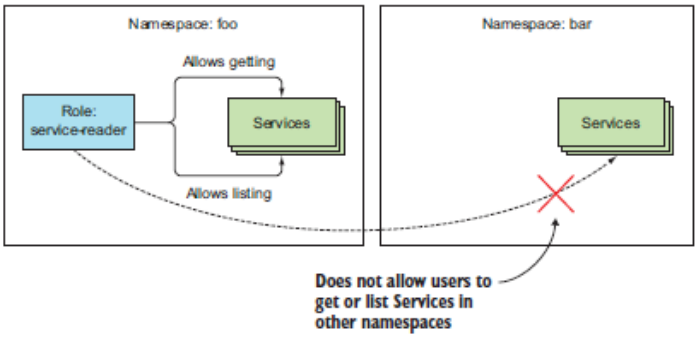
1) 먼저 롤 리소스를 만들어야 한다. 롤 리소스는 어떤 리소스에서 수행할 수 있는 액션을 정의한다. 다음 예제는 foo 네임스페이스에서 서비스를 get 하고 list 수행을 허용할 롤을 정의한다.

```
# cat service-reader.yaml
        apiVersion: rbac.authorization.k8s.io/v1
        kind: Role
        metadata:
          namespace: foo                    ## role 은 foo 네임스페이스에 적용 된다.
          name: service-reader
        rules:
        - apiGroups: [""]
          verbs: ["get", "list"]
          resources: ["services"]           ## role 은 service 에서 get, list 를 허용한다.
```

롤 리소스를 생성한다. 이제 foo 네임스페이스에 이전 롤을 생성하자

```
# kubectl create -f service-reader.yaml -n foo
  role.rbac.authorization.k8s.io/service-reader created
```

service-reader 롤은 foo 네임스페이스에서 서비스 getting 과 listing 을 허용한다.

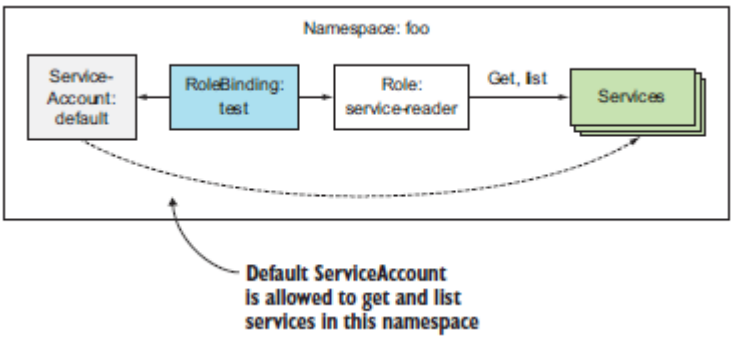


yaml 이 아닌 명령어를 통해 롤 생성할 수 있다. bar 롤을 생성해본다.

```
# kubectl create role service-reader --verb=get --verb=list --resource=services -n bar
  role.rbac.authorization.k8s.io/service-reader created
```

4. 서비스어카운트에 롤을 바인딩

롤은 수행할 수 있는 액션을 정의하지만 누가 수행할 수 있는지는 지정하지 않는다. 이를 수행하려면 롤을 사용자, 서비스어카운트 또는 그룹이 될 수 있는 주체에 바인딩 해야 한다. 롤바인딩 리소스를 작성해 주체에 롤을 바인딩 할 수 있다.



service-reader 롤을 가진 default 서비스어카운트를 바인딩하는 test 롤바인딩

롤을 기본 서비스어카운트에 바인딩하려면 다음 명령을 실행한다. foo 네임스페이스의 default 서비스어카운트가 service-reader 롤을 바인딩하는 rolebinding 을 생성한다.

```
# kubectl create rolebinding test --role=service-reader --serviceaccount=foo:default -n foo
  rolebinding.rbac.authorization.k8s.io/test created

# kubectl get rolebindings test -n foo -o yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: test
    namespace: foo
    ..
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role                          ## 롤바인딩은 service-reader 롤을 참조한다.
    name: service-reader
  subjects:
  - kind: ServiceAccount
    name: default                       ## 네임스페이스 상에서 기본 서비스어카운트에 바인드한다.
    namespace: foo
```

서비스어카운트에 롤을 바인드 했으니 해당 포드에서 서비스를 나열한다.

T2/# curl localhost:8001/api/v1/namespaces/foo/services


```
"kind": "ServiceList",
"apiVersion": "v1",
"metadata": {
"selfLink": "/api/v1/namespaces/foo/services",
"resourceVersion": "1552995"
},
"items": []
}
```


5. 롤 바인딩에서 다른 네임스페이스의 서비스어카운트 포함하기

아래와 같이 subjects 라인 아래에 bar 네임스페이스의 default 서비스어카운트를 추가한다.

 # kubectl edit rolebindings test -n foo

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2019-07-11T23:53:36Z"
  name: test
  namespace: foo
  resourceVersion: "1423251"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/foo/rolebindings/test
  uid: 66944110-fb7c-40c7-b668-e1959a472d9d
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: service-reader
subjects:
- kind: ServiceAccount
  name: default
  namespace: foo
- kind: ServiceAccount
  name: default
  namespace: bar
```

6. bar namespace 의 Pod 에서 foo namespace 의 service 정보를 조회 해본다.

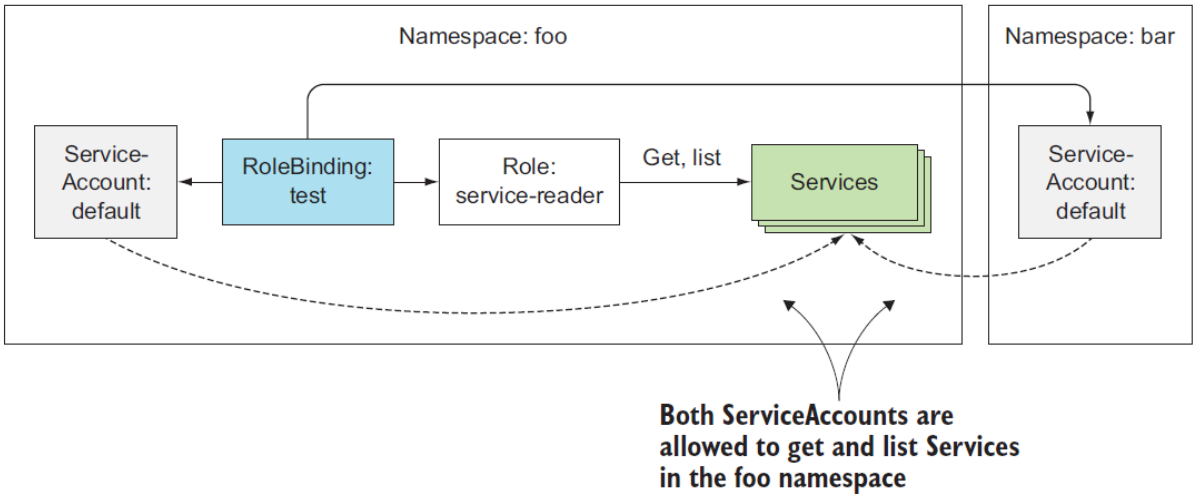
T3/ # curl localhost:8001/api/v1/namespaces/bar/services


```
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "services is forbidden: User ₩"system:serviceaccount:bar:default₩" cannot list resource ₩"services₩" in API group ₩"₩" in the namespace ₩"bar₩"",
  "reason": "Forbidden",
  "details": {
    "kind": "services"
  },
  "code": 403
```


/ # curl localhost:8001/api/v1/namespaces/foo/services


```
{
  "kind": "ServiceList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/foo/services",
    "resourceVersion": "1427054"
  },
  "items": []
}
/#
```

클러스터롤과 클러스터롤바인딩

- **클러스터롤**
  - 각 쿠버네티스 API의 사용 권한을 정의한다.
  - 클러스터 전체에 유효함
- **클러스터롤바이딩**
  - 일반 사용자/그룹 또는 서비스 계정을 클러스터롤과 연결

클러스터롤과 클러스터롤바인딩

롤과 롤바인딩은 단일 네임스페이스의 리소스에 상주하고 적용한다.
클러스터롤과 클러스터롤 바인딩은 네임스페이스가 아닌 클러스터의 전체(네임스페이스에 없는 PV 접근 등)에 접근할 수 있다.

1. 클러스터 레벨 리소스에 접근 허용

```
# kubectl create clusterrole pv-reader --verb=get,list --resource=persistentvolumes
# kubectl get clusterrole pv-reader
NAME           AGE
pv-reader      22s

# kubectl get clusterrole pv-reader -o yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    creationTimestamp: "2019-07-12T00:18:27Z"
    name: pv-reader
  ...
  rules:
  - apiGroups:
    - ""
    resources:
    - persistentvolumes
    verbs:
    - get
    - list
```

2. 클러스터 수준 리소스에 대한 접근을 부여하려면 항상 클러스터 롤바인딩을 사용해야한다.
foo 사용자에게 클러스터 롤바인딩을 생성하자

 # kubectl create clusterrolebinding pv-test --clusterrole=pv-reader --serviceaccount=foo:default

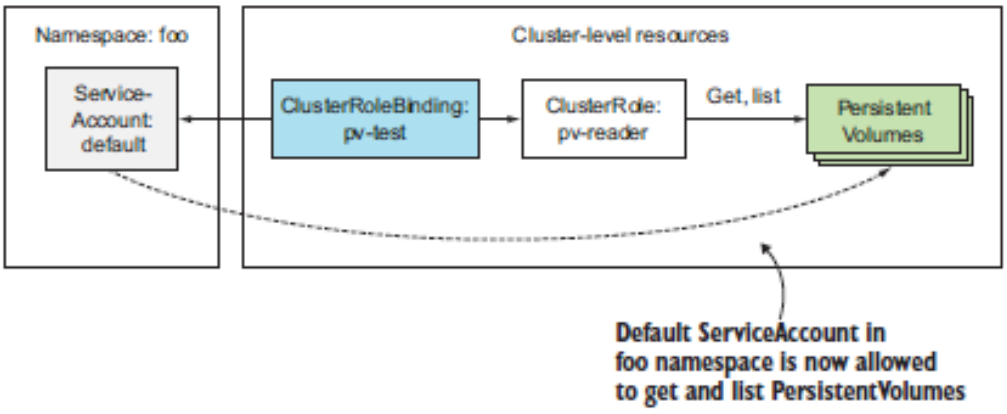
 # kubectl get clusterrolebindings pv-test

```
NAME          AGE
pv-test       66s
```

3. Persistentvolume 을 나열해보자.

T2 /# curl localhost:8001/api/v1/persistentvolumes

```
{
"kind": "PersistentVolumeList",
...
        "spec": {
          "capacity": {
            "storage": "2Gi"
          },
          "nfs": {
            "server": "10.100.0.254",
            "path": "/disk1"
          },
          "accessModes": [
            "ReadWriteOnce",
            "ReadOnlyMany"
...
```



foo 사용자에게 클러스터 전체 네임스페이스의 정보를 볼 수 있도록 권한(view)을 할당하자.

 # kubectl create clusterrolebinding view-test --clusterrole=view --serviceaccount=foo:default

clusterrolebinding "view-test" created

pod 정보보기

T2/# curl localhost:8001/api/v1/namespaces/foo/pods

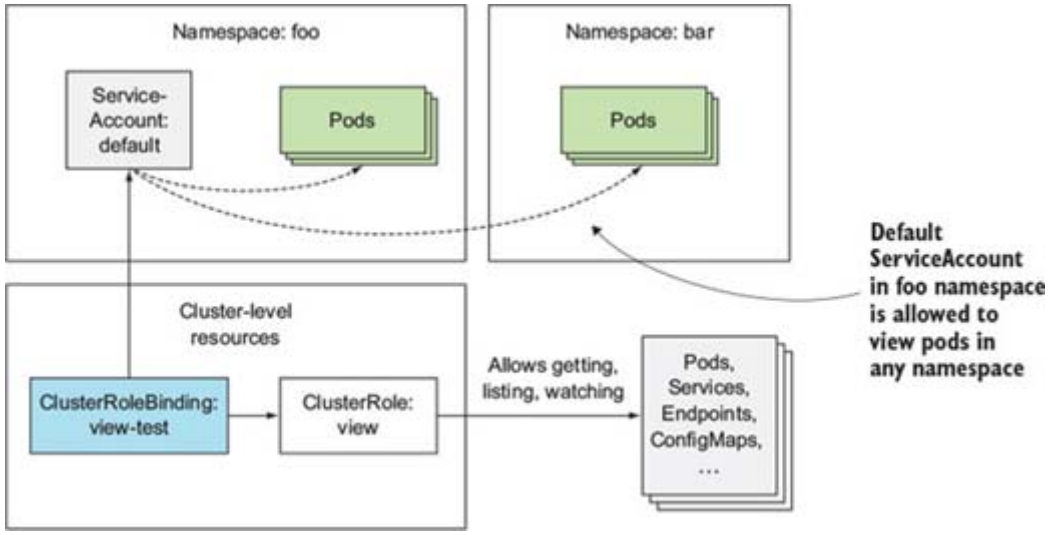
```
"kind": "PodList",
"apiVersion": "v1",
"metadata": {
"selfLink": "/api/v1/namespaces/foo/pods",
"resourceVersion": "895449"
},
```

전체 네임스페이스확인

/# curl localhost:8001/api/v1/pods

/# curl localhost:8001/api/v1/namespaces/bar/pods

예상대로 포드는 클러스터의 모든 포드 목록을 가져올 수 있다. 요약하면 네임스페이스 리소스를 참조하고 있는 클러스터롤바인딩과 클러스터롤을 조합하면 아래 그램과 같이 포드가 어떤 네임스페이스의 네임스페이스 리소스에 접근한다 해도 허용할 수 있다.



클러스터 롤을 롤 바인딩으로 대체하면 어떤 일이 발생하는지 보자. 먼저 ClusterRoleBinding 을 삭제한다.

 # kubectl delete clusterrolebinding pv-test

다음 RoleBinding 을 작성하고, RoleBinding 이 네임스페이스가 되므로 네임스페이스내에 생성하려면 네임스페이스를 지정해야한다. foo namespace 내에 rolebinding 을 생성하자.

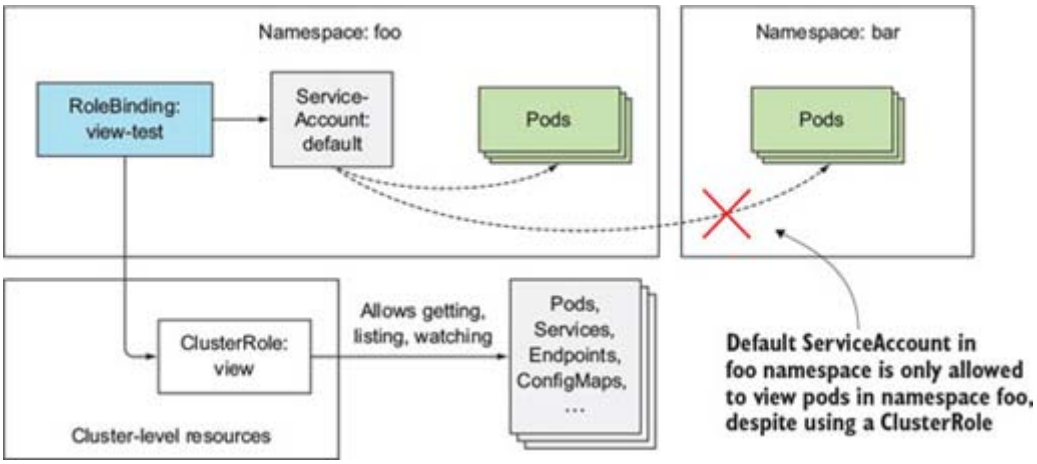
 # kubectl create rolebinding view-test --clusterrole=view --serviceaccount=foo:default -n foo

이제 foo 네임스페이스에 RoleBinding 이 생성되었다. view 클러스터롤과 동일한 네임스페이스내에 default 서비스 어카운트를 바인딩했다. 이제 포드로 접근할 수 있을까?

```
T2 / # curl localhost:8001/api/v1/namespaces/foo/pods
```

```
T2 / # curl localhost:8001/api/v1/namespaces/bar/pods
T2 / # curl localhost:8001/api/v1/pods
```

포드는 foo 네임스페이스에 포드를 나열할 수 있지만 다른 특정 네임스페이스나 모든 네임스페이스에서는 포드를 나열할 수 없다.



롤, 클러스토롤, 롤바인딩, 클러스터롤 바인딩 조합 요약

다양한 조합을 다루어 봤으므로 각 조합을 언제 사용해야 할지 기억하기 어려울수 있다. 이런 모든 조합을 특정 사용 사례별로 분류해 이해할 수 있는지 알아보겠다.

롤과 바인딩 타입의 특정 조합을 사용하는 경우

접근	롤 타입	사용할 바인딩 타입
클러스터 수준 리소스 (Nodes, PersistentVolumes, ...)	ClusterRole	ClusterRoleBinding
Non-resource URLs (/api, /healthz, ...)	ClusterRole	ClusterRoleBinding
모든 Namespace 및 namespace로 지정된 리소스	ClusterRole	ClusterRoleBinding
특정 Namespace의 namespace로 지정된 리소스(여러 namespace에 동일한 ClusterRole 사용)	ClusterRole	RoleBinding
특정 Namespace의 namespace로 지정된 리소스 (각 namespace의 롤을 정의해야 함)	Role	RoleBinding

Default CluserRole과 ClusterRolebinding 이해

쿠버네티스는 API 서버가 시작될 때 마다 업데이트되는 클러스터롤 및 클러스터롤바인딩의 기본 세트를 제공한다. 때문에 실수로 삭제하거나 새로운 버전의 쿠버네티스가 클러스터롤 및 바인딩의 다른 구성을 사용하는 경우 모든 디폴트롤과 바인딩이 다시 만들어진다.

기본으로 제공되는 클러스터롤, 클러스터롤바인딩을 확인해본다.

```
# kubectl get clusterrole view -o yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: view
    ...
  rules:
  - apiGroups:
    - ""
    resources:
    - configmaps
    - endpoints
    - persistentvolumeclaims
    - pods
    - replicationcontrollers
    - replicationcontrollers/scale
    - serviceaccounts
    - services
    verbs:
    - get
    - list
    - watch
  ...
```

이 중 가장 중요한 롤은 view, edit, admin, cluster-admin 클러스터롤이다. 이들은 사용자 정의 포드를 사용하는 서비스어카운트에 바운딩 되기 위한 것이다.

- view 클러스터롤을 통한 리소스 읽기 전용 접근 허용
- edit 클러스터롤을 통한 리소스 수정 허용
- admin 클러스터롤을 통한 네임스페이스 전체 통제 권한 허용
- cluster-admin 클러스터롤을 통한 완전한 통제허용

인증권한 부여

- **기본으로 Pod는 클러스터 상태를 볼 수 없다**
  - 모든 service account에 cluster-admin 클러스터롤을 부여하는 것은 좋지 않다.
  - 모든 사람이 업무를 수행하는데 필요한 권한만 제공한다.
- **각 포드에 특정한 서비스 어카운트를 생성**
  - 각 포드에 특정 serviceaccount를 작성한후 rolebinding을 통해 맞춤형 롤을 적용시키는 것이 좋다.
- **애플리케이션 취약점 예상하기**
  - serviceaccount에 제약을 걸어 예기치 못하는 상황에 인증토큰을 노출시키지 않아야한다.

인증권한 부여하기

기본적으로 네임스페이스의 디폴트 서비스어카운트에는 인증되지 않은 사용자의 권한이외에는 권한 설정이 없다. 따라서 기본적으로 포드는 클러스터 상태를 볼 수 없다. 적절한 권한을 부여하는 것은 전적으로 관리자에게 달려있다.

분명히 모든 serviceaccount 에 cluster-admin 클러스터롤을 부여하는 것은 좋지 않은 생각이다. 보안성을 유지하기 위해 항상 그렇듯이 모든 사람에게 업무를 수행하는데 필요한 권한만 제공하고 단일 권한 이상을 부여하지 않는 것이 가장 좋다.

각 포드에 특정한 서비스어카운트 생성

각 포드의 특정 서비스어카운트를 작성한 다음 롤바인딩을 통해 맞춤형 롤과 연관시키는 것이 좋다. 하나의 포드가 포드를 읽을 필요가 있는 반면 다른 포드도 포드를 수정해야 하는 경우 두 개의 서비스어카운트를 작성하고 포드 스펙에서 serviceAccountName 특정을 지정해 해당 포드가 사용할 수 있게 하라. 두 포드에 필요한 모든 권한을 네임스페이스의 디폴트 서비스어카운트에게 추가하지 말자

애플리케이션 취약점 예상하기

애플리케이션의 취약점을 예상하여 원하지 않는 사람이 서비스어카운트의 인증토큰을 손에 넣지 않도록 serviceaccount 에 항상 제약을 두어야한다.