User Authentication & Authorization in Kubernetes

Authenticating

Authorization

Using RBAC Authorization

Role-Based Access Control (RBAC) – Detailed Explanation

RBAC (Role-Based Access Control) is the most commonly used **authorization method** in Kubernetes. It controls **which user or service account** can access **which Kubernetes resources** and what actions they can perform.

Four Key Components of RBAC:

1. **Role** – Defines permissions within a specific **Namespace**.
2. **ClusterRole** – Defines permissions **cluster-wide** (across all namespaces).
3. **RoleBinding** – Associates a **User, Group, or Service Account** with a **Role** within a specific **Namespace**.
4. **ClusterRoleBinding** – Associates a **User, Group, or Service Account** with a **ClusterRole**, applying it **cluster-wide**.

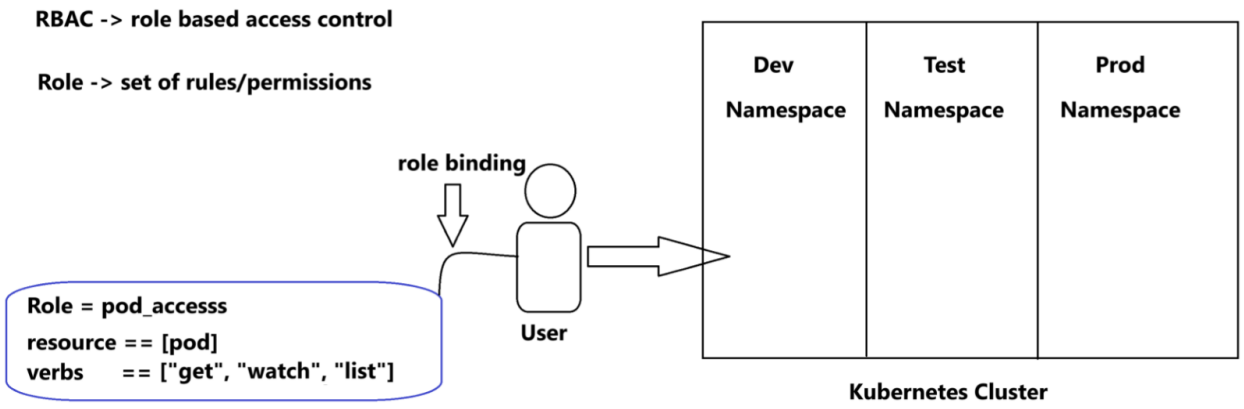
Role – Controls Access Within a Namespace

A **Role** defines **which resources** can be accessed and **what actions (verbs)** can be performed on them. However, it is limited to a **single namespace**.

Example: A "pod-reader" Role that grants Read Access to Pods

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

```
  name: pod-reader
  namespace: default  # This role applies only to the 'default' namespace
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```



RoleBinding – Assigns a Namespace-Specific Role to a User

A **RoleBinding** defines **which user** (or service account) is assigned a **Role** and in **which namespace** it applies.

Example: Assigning the "pod-reader" Role to "dev-user" (Only in the Default Namespace)

ClusterRole – Controls Access Across All Namespaces

If a user needs access **across the entire cluster** (not limited to a single namespace), a **ClusterRole** is used.

Example: "cluster-admin" ClusterRole with Full Cluster Management Permissions

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin
rules:
- apiGroups: ["*"]  # Grants access to all API groups
  resources: ["*"]  # Grants access to all resources
  verbs: ["*"]  # Grants all actions (create, delete, update, etc.)
```

Let's create role for pods

```
root@master-node:~# vim role.yaml
root@master-node:~#
root@master-node:~# kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/pod-reader created
root@master-node:~#
root@master-node:~# kubectl get role
NAME         CREATED AT
pod-reader   2025-02-27T18:34:04Z
root@master-node:~#
```

Now perform role binding

```
vim rolebind.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
```

```
  - kind: User
    name: jane  # "name" is case sensitive
    apiGroup: rbac.authorization.k8s.io
roleRef:
    kind: Role  # this must be Role or ClusterRole
  name: pod-reader  # this must match the name of the Role or ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

```
root@master-node:~# vim rolebind.yaml
root@master-node:~# kubectl apply -f rolebind.yaml
rolebinding.rbac.authorization.k8s.io/read-pods created
root@master-node:~#
root@master-node:~# kubectl get rolebinding
NAME        ROLE              AGE
read-pods   Role/pod-reader   25s
root@master-node:~#
```

```
root@master-node:~# kubectl describe rolebinding read-pods
Name:         read-pods
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  pod-reader
Subjects:
  Kind  Name  Namespace
  ----  ----  ---------
  User  jane
root@master-node:~#
```

```
root@master-node:~# kubectl auth can-i get pods
yes
root@master-node:~# kubectl auth can-i get nodes
Warning: resource 'nodes' is not namespace scoped

yes
root@master-node:~# kubectl auth can-i create pods
yes
root@master-node:~# kubectl auth can-i create pods --as jane
no
root@master-node:~# kubectl auth can-i get pods --as jane
yes
root@master-node:~#
```

Now let’s create cluster role.

vim cluster_role.yaml

aversion: rbac.authorization.k8s.io/v1

kind: ClusterRole

```
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

```
root@master-node:~# kubectl apply -f cluster
cluster.yml        cluster_role.yaml
root@master-node:~# kubectl apply -f cluster_role.yaml
clusterrole.rbac.authorization.k8s.io/secret-reader created
root@master-node:~#
root@master-node:~# kubectl get clusterrole
NAME                                                                   CREATED AT
admin                                                                  2025-02-10T18:30:0
9Z
cluster-admin                                                          2025-02-10T18:30:0
9Z
edit                                                                   2025-02-10T18:30:0
9Z
kubeadm:get-nodes                                                      2025-02-10T18:30:0
```

apiVersion: rbac.authorization.k8s.io/v1

This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.

```
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
```

apiGroup: rbac.authorization.k8s.io

```
root@master-node:~# kubectl apply -f crolebinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/read-secrets-global created
root@master-node:~#
root@master-node:~# kubectl describe clusterrolebindings read-secrets-global
Name:         read-secrets-global
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  secret-reader
Subjects:
  Kind   Name     Namespace
  ----   ----     ---------
  Group  manager
root@master-node:~#
```