Kubernetes RBAC

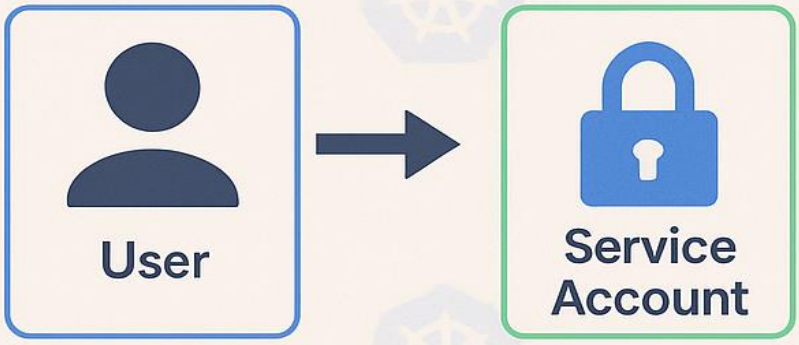
Why RBAC is Used

RBAC is a method of regulating access to resources based on the roles of individual users within a Kubernetes cluster.

When RBAC is Used

RBAC is used to enforce rules when a user attempts to access a Kubernetes resource.

Two key objests in RBAC



Identity Providers

External systems (e g., Keycloak, Dex) that authenticate users; typically indirectly integrated with Kubernetes

Core RBAC Resources

- **Role** Defines permissions within a namespace
- **ClusterRole** Defines permissions cluster-wide
- **RoleBinding** Grants a role to a user or service account within a namespace
- **ClusterRoleBinding** Grants a cluster role to a user

What is RBAC (Role-Based Access Control) in Kubernetes?

RBAC is a **security mechanism in Kubernetes** that regulates **who (user/service account)** can **do what (verbs)** on **which resources (pods, secrets, etc.)** in a namespace or cluster-wide.

RBAC ensures **fine-grained access control** over Kubernetes resources.

Why Is RBAC Used?

Without RBAC:

1. Anyone with cluster access can perform **unrestricted operations**

2. **Accidental deletion** of critical resources (like deleting all pods or secrets)

3. **Security risk**: Unauthorized access to secrets, configs, logs, etc.

With RBAC:

1. Define **least privilege** access

2. Allow only **specific actions** (e.g., view pods, but not delete them)

3. **Isolate teams**, environments, and components

When Is RBAC Used?

In **multi-tenant clusters** (dev/prod teams)

For **CI/CD pipelines** (service accounts need controlled access)

To enforce **principle of least privilege**

To protect **sensitive operations** (e.g., managing secrets, deployments)

With **external identity providers** like Okta, LDAP for enterprise SSO

Problems Solved by RBAC

Problem	Solved by RBAC
Everyone has admin access	Assign roles with limited permissions
Lack of auditability	Only authorized users perform sensitive actions
No environment separation	RBAC can be namespace-scoped
Misuse of Service Accounts	Restrict SA to minimum required access

---

Two Key Subjects in RBAC

Users – Human users (devs, admins)

Authenticated through **identity providers** (SSO, Okta, LDAP)

Not created in Kubernetes directly (you configure external auth)

ServiceAccounts – Non-human identities

Used by **Pods/Controllers** to interact with the API server

Auto-mounted inside Pods

How RBAC Works – Step-by-Step Flow

Scenario: Alice wants to list Pods in the dev namespace.

Step-by-Step RBAC Flow:

Authentication (Who is this?)

Kubernetes checks the user identity (from token/certificate)

Identity could be from an **Identity Provider** (IdP)

Authorization (Are they allowed to?)

Kubernetes checks **RBAC policies**:

Does Alice have a Role or ClusterRole?

Is she bound using a RoleBinding or ClusterRoleBinding?

Decision

If allowed → request proceeds

If denied → HTTP 403 Forbidden

RBAC = Authorization layer (not authentication).

Identity Providers (for Authentication)

Kubernetes doesn't manage user accounts. You integrate with:

Identity Provider	Use Case
LDAP	Traditional enterprise directory
Okta	Cloud-based SSO, MFA
GitHub	Useful for dev teams
Google Workspace (SSO)	Google-authenticated access
Active Directory	Windows-based enterprise environments

These providers handle **who you are,** then Kubernetes uses RBAC to decide **what you can do**.

Core RBAC Resources

1. Role

Grants permissions within a **specific namespace**

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

---

2. ClusterRole

Grants **cluster-wide** or **non-namespaced** resource permissions

Can also be used in a namespace if bound appropriately

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

---

3. RoleBinding

> Binds a **Role** to a **user, group, or ServiceAccount** in a **namespace**

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bind-alice
  namespace: dev
subjects:
- kind: User
  name: alice@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

---

4. ClusterRoleBinding

Binds a **ClusterRole** to **subjects** across the entire cluster

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: admin-binding
subjects:
- kind: User
  name: bob@example.com
```

```
    apiGroup: rbac.authorization.k8s.io
roleRef:
    kind: ClusterRole
    name: cluster-admin
    apiGroup: rbac.authorization.k8s.io
```

Real-World Example

Scenario: You want to allow a **CI/CD pipeline** (viaServiceAccount) to only **create and delete Deployments** in the dev namespace.

Step 1: Create Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
    namespace: dev
    name: cicd-deploy
rules:
- apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["create", "delete"]
```

Step 2: Bind the Role

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    name: bind-cicd-sa
    namespace: dev
subjects:
- kind: ServiceAccount
    name: cicd-pipeline
    namespace: dev
roleRef:
    kind: Role
    name: cicd-deploy
    apiGroup: rbac.authorization.k8s.io
```

Advantages of RBAC in Kubernetes

Advantage	Description
Least Privilege	Enforce fine-grained, minimal permissions
Extensible	Integrates with Identity Providers
Auditable	Easy to track who can do what
Separation of Concerns	Users, teams, apps, and environments isolated
Protects Critical Resources	Avoid accidental damage
Declarative	Define and manage using YAML (GitOps-friendly)
Namespace Scoped	Easily restrict users to environments (dev, staging, prod)

Limitations / Considerations

Limitation	Workaround
No built-in user management	Use external IdPs
Debugging access issues can be hard	Use kubectl auth can-i
Complex in large teams	Use RBAC generators, policies
Static rules	Use tools like OPA/Gatekeeper for dynamic policies

RBAC = Authorization framework in K8s

Users & ServiceAccounts are the subjects

Roles/ClusterRoles define permissions

RoleBinding/ClusterRoleBinding assign roles to users/SA

Identity Providers (SSO, LDAP, Okta) are used for authentication

RBAC allows **secure, scalable, and compliant** access control in K8s

---