Role-Based Access Control (RBAC) in Kubernetes: A Comprehensive Guide

What is RBAC

RBAC (Role-Based Access Control) is a **security mechanism** in Kubernetes that controls **who can do what** within a cluster. It restricts access based on roles assigned to users or services and ensures that only authorized actions are permitted.

Why RBAC Is Important

- **Security**: RBAC limits access to sensitive data and critical systems only to those who need it for their job, reducing the risk of accidental or malicious misuse.
- **Simplification**: It simplifies access management by grouping permissions into roles rather than managing each user separately. This makes onboarding, role changes, and offboarding easier and less error-prone.
- **Compliance**: RBAC helps organizations comply with regulations (e.g., HIPAA, GDPR) by enforcing least privilege and enabling audit trails of who accessed what and when.
- **Scalability**: Particularly useful in large organizations with many users and complex access needs, RBAC scales better than manual user-by-user permission management.

Key Components of RBAC

1.Users (Human or External Identities)

- Admins, developers, or operators accessing the cluster.
- Managed via external identity providers (e.g., IAM in AWS, LDAP, Keycloak, GitHub SSO).
- Kubernetes does not store user identities; it offloads authentication to OAuth, OpenID, or other providers.

2.Service Accounts (Machine Identities)

- Used by pods, deployments, or services to interact with the Kubernetes API.
- Every pod gets a default service account (unless specified otherwise).
- Can be misconfigured, leading to security risks (e.g., a pod deleting secrets).

3.Role

- A Role contains a set of permissions, which are rules defining what actions can be performed on which resources.
- Roles are namespaced. They apply only to resources within a specific namespace.

4.ClusterRole:

- Similar to a Role, but ClusterRoles are not namespaced. They can be used to grant permissions cluster-wide or across all namespaces.

5.RoleBinding:

- A RoleBinding grants the permissions defined in a Role to a user, group, or service account within a specific namespace.
- It links a Role to one or more subjects (users, groups, or service accounts).

6.ClusterRoleBinding:

- Similar to RoleBinding, but ClusterRoleBinding grants permissions to users, groups, or service accounts across the entire cluster.

How RBAC works

How to Set Up RBAC in Kubernetes: Step-by-Step Guide

RBAC (Role-Based Access Control) in Kubernetes ensures that **users, service accounts, and applications** have only the permissions they need. Below is a **step-by-step guide** to setting up RBAC properly.

1. Define Users

Kubernetes **does not manage** users directly. Instead, integrate with external identity providers like:

- AWS IAM (for EKS)
- LDAP / Active Directory
- OAuth (Keycloak, Google, GitHub SSO)

Example: Adding a User Manually (for Testing)

Generate a kubeconfig for a user (for demo purposes)

kubectl config set-credentials dev-user --client-certificate=dev-user.crt --client-key=dev-user.key

kubectl config set-context dev-user-context --cluster=my-cluster --user=dev-user

2. Create a Service Account (For Pods/Apps)

Service accounts are used by pods to interact with the Kubernetes API.

Example: Creating a Service Account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ci-cd-bot
  namespace: dev
```

3. Define a Role (Namespace-Scoped Permissions)

A Role restricts access to resources within a single namespace.

Example: Role for Read-Only Pod Access

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-reader
rules:
- apiGroups: [""]  # Core API group (pods, services, etc.)
  resources: ["pods"]
  verbs: ["get", "list", "watch"]  # Read-only permissions
```

4. Define a ClusterRole (Cluster-Wide Permissions)

A ClusterRole grants permissions across all namespaces (or cluster-wide resources like Nodes).

Example: ClusterRole for View-Only Access

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: global-reader
rules:
- apiGroups: [""]
  resources: ["pods", "services", "nodes"]
  verbs: ["get", "list", "watch"]  # Read-only everywhere
```

5. Bind Permissions with RoleBinding

A **RoleBinding** links a Role to a **User/ServiceAccount** in a **single namespace**.

Example: Granting Read Access to a Service Account

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ci-cd-read-pods
  namespace: dev
subjects:
- kind: ServiceAccount
  name: ci-cd-bot
  namespace: dev
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

6. Bind Permissions with ClusterRoleBinding

A **ClusterRoleBinding** links a **ClusterRole** to a **User/ServiceAccount cluster-wide.**

Example: Granting Admin Access to a User

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-access
subjects:
- kind: User
```

```
  name: admin-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin  # Built-in superuser role
  apiGroup: rbac.authorization.k8s.io
```

7. Verify RBAC Rules

Check if permissions are applied correctly:

```
# Check roles in a namespace
kubectl get roles -n dev

# Check role bindings
kubectl get rolebindings -n dev

# Check cluster roles
kubectl get clusterroles

# Test access (e.g., as a user)
kubectl get pods -n dev --as=dev-user
```

RBAC Best Practices

Follow Least Privilege Principle – Only grant necessary permissions.

Avoid Using default **Service Account** – Always create custom SAs for apps.

Use Role + RoleBinding for **Namespace-Specific Access**.

Use ClusterRole + ClusterRoleBinding **Sparingly** (security risk).

Audit RBAC Rules Regularly (kubectl get roles,rolebindings --all-namespaces).

Conclusion

Setting up RBAC involves:

1. **Defining Users/Service Accounts** (who gets access).
2. **Creating Roles/ClusterRoles** (what they can do).
3. **Binding Permissions** (linking identities to roles).

This ensures **secure, granular access control** in Kubernetes.