Kubernetes RBAC and Ingress Student Guide

Table of Contents

Overview

This lab demonstrates two important Kubernetes concepts:

- **RBAC (Role-Based Access Control)**: Managing permissions for users and service accounts
- **Ingress**: Exposing HTTP/HTTPS routes from outside the cluster to services within the cluster

Prerequisites

- Minikube or any Kubernetes cluster
- kubectl CLI tool
- Basic understanding of Kubernetes pods, services, and namespaces

Part 1: RBAC (Role-Based Access Control)

What is RBAC?

RBAC is a security mechanism that restricts access to Kubernetes resources based on roles assigned to users or service accounts. It follows the principle of least privilege.

Key RBAC Components:

1. **Service Account**: An identity for processes running in pods
2. **Role**: Defines permissions within a namespace
3. **RoleBinding**: Binds a role to a subject (user, group, or service account)
4. **ClusterRole**: Defines permissions cluster-wide
5. **ClusterRoleBinding**: Binds a cluster role to a subject

Step 1: Set Up the Project Directory

bash

```bash
# Create and navigate to the project directory
mkdir rbac_k8s
cd rbac_k8s
```

Step 2: Create the Required YAML Files

2.1 Service Account (`sa.yaml`)

yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dev-user
  namespace: default
```

2.2 Role (`role.yaml`)

yaml

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

2.3 RoleBinding (`rolebinding.yaml`)

yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-reader-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: dev-user
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

2.4 Test Pod (`nginxpod.yaml`)

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: default
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Step 3: Apply the RBAC Configuration

```bash
# Create the test pod first
kubectl apply -f nginxpod.yaml

# Create the service account
kubectl apply -f sa.yaml

# Verify service account creation
kubectl get sa

# Create the role
kubectl apply -f role.yaml

# Verify role creation
kubectl get role

# Create the role binding
kubectl apply -f rolebinding.yaml

# Verify role binding creation
kubectl get rolebinding
```

Step 4: Test RBAC Permissions

4.1 Create a Token for the Service Account

```bash
# Generate a token for the dev-user service account
kubectl create token dev-user --namespace default

# Save the token to a file for easier handling
kubectl create token dev-user --namespace default > token.txt

# Set the token as an environment variable
TOKEN=$(cat token.txt)
echo $TOKEN
```

4.2 Configure kubectl to Use the Service Account

```bash
# Set credentials for the dev-user
kubectl config set-credentials dev-user --token=$TOKEN

# Create a context for the dev-user
kubectl config set-context dev-user-context --cluster=minikube --user=dev-user

# Switch to the dev-user context
kubectl config use-context dev-user-context
```

4.3 Test Permissions

```bash
# This should work (dev-user has get/list permissions for pods)
kubectl get pods

# This should fail (dev-user doesn't have delete permissions)
kubectl delete pod nginx-pod
```

Expected error message:

```
Error from server (Forbidden): pods "nginx-pod" is forbidden: User "system:serviceaccount:default:dev-user" cannot delete resource "pods" in API group "" in the namespace "default"
```

4.4 Clean Up RBAC Test

```bash
# Switch back to admin context
kubectl config use-context minikube

# Delete the test pod
kubectl delete pod nginx-pod
```

Part 2: Ingress Configuration

What is Ingress?

Ingress is an API object that manages external access to services in a cluster, typically HTTP/HTTPS. It provides load balancing, SSL termination, and name-based virtual hosting.

Step 1: Set Up Ingress Controller

```bash
# Clone the lab repository
git clone git@github.com:erangcy/k8s-labs.git
cd k8s-labs/lab-ingress

# Create the ingress-nginx namespace
kubectl create namespace ingress-nginx

# Install the NGINX Ingress Controller
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.1/deploy/static/provide
```

Step 2: Examine the Application Files

The lab includes three simple applications:

2.1 Cats Application (`cats.yaml`)

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cats-app
  labels:
    app: cats
spec:
  containers:
  - name: cats
    image: hashicorp/http-echo
    args:
    - "-text=cats"
    - "-listen=:8080"
    ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: cats-service
spec:
  selector:
    app: cats
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

2.2 Dogs Application (`dogs.yaml`)

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dogs-app
  labels:
    app: dogs
spec:
  containers:
  - name: dogs
    image: hashicorp/http-echo
    args:
    - "-text=dogs"
    - "-listen=:8080"
    ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: dogs-service
spec:
  selector:
    app: dogs
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

2.3 Birds Application (`birds.yaml`)

yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: birds-app
  labels:
    app: birds
spec:
  containers:
  - name: birds
    image: hashicorp/http-echo
    args:
    - "-text=birds"
    - "-listen=:8080"
    ports:
    - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: birds-service
spec:
  selector:
    app: birds
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

2.4 Ingress Configuration (`ingress.yaml`)

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: app-ingress
 annotations:
  nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 ingressClassName: nginx
 rules:
 - http:
    paths:
    - path: /cats
      pathType: Prefix
      backend:
       service:
         name: cats-service
         port:
           number: 80
    - path: /dogs
      pathType: Prefix
      backend:
       service:
         name: dogs-service
         port:
           number: 80
    - path: /birds
      pathType: Prefix
      backend:
       service:
         name: birds-service
         port:
           number: 80
```

Step 3: Deploy the Applications

```bash
# Deploy all applications
kubectl create -f cats.yaml
kubectl create -f dogs.yaml
kubectl create -f birds.yaml

# Create the ingress resource
kubectl create -f ingress.yaml

# Verify all pods are running
kubectl get pods

# Check ingress status
kubectl get ingress
```

Step 4: Test the Ingress

```bash
# Get the ingress controller service URL
minikube service ingress-nginx-controller --url -n ingress-nginx
```

This will output URLs like:

```
http://192.168.49.2:31672
http://192.168.49.2:30425
```

Test the applications by accessing:

- `http://192.168.49.2:31672/cats` - Should display "cats"
- `http://192.168.49.2:31672/dogs` - Should display "dogs"
- `http://192.168.49.2:31672/birds` - Should display "birds"
- `http://192.168.49.2:31672/notexist` - Should return 404

Cleanup

```bash
# Delete all resources created in this lab
kubectl delete -f ingress.yaml
kubectl delete -f cats.yaml
kubectl delete -f dogs.yaml
kubectl delete -f birds.yaml

# Delete the ingress controller
kubectl delete namespace ingress-nginx

# For complete cleanup, delete the minikube cluster
minikube delete
```

Key Concepts Summary

RBAC Concepts:

- **Service Account**: Provides identity for processes in pods
- **Role**: Defines what actions can be performed on which resources
- **RoleBinding**: Links roles to subjects (users, groups, service accounts)
- **Principle of Least Privilege**: Grant only the minimum required permissions

Ingress Concepts:

- **Ingress Controller**: Implements the ingress rules (e.g., NGINX, Traefik)
- **Ingress Resource**: Defines the routing rules
- **Path-based Routing**: Routes traffic based on URL paths
- **Backend Services**: The actual services that handle the requests

Troubleshooting

Common RBAC Issues:

1. **Permission Denied**: Check if the role has the required verbs and resources
2. **Token Expired**: Regenerate the token using `kubectl create token`
3. **Wrong Context**: Ensure you're using the correct kubectl context

Common Ingress Issues:

1. **404 Errors**: Check if the ingress rules match the request path

2. **503 Errors**: Verify that backend services are running and healthy
3. **Ingress Controller Not Running**: Check if the ingress controller pods are running in the ingress-nginx namespace

Useful Commands:

```bash
# Check current context
kubectl config current-context

# View all contexts
kubectl config get-contexts

# Check pod logs
kubectl logs <pod-name>

# Describe resources for detailed information
kubectl describe ingress app-ingress
kubectl describe role pod-reader
kubectl describe rolebinding pod-reader-binding
```

Additional Exercises

1. **Extend RBAC**: Create a role that allows creating and deleting pods
2. **Host-based Routing**: Modify the ingress to route based on different hostnames
3. **SSL/TLS**: Add SSL certificates to the ingress configuration
4. **Resource Quotas**: Implement resource quotas with RBAC
5. **Namespace Isolation**: Create separate namespaces with different RBAC policies

Best Practices

RBAC Best Practices:

- Follow the principle of least privilege
- Use service accounts for applications, not personal accounts
- Regularly audit and review permissions
- Use namespaces to isolate resources
- Implement network policies along with RBAC

Ingress Best Practices:

- Use meaningful annotations for ingress controller configuration
- Implement proper SSL/TLS termination
- Set up monitoring and logging for ingress traffic
- Use resource limits for ingress controllers
- Implement proper health checks for backend services