Gmail

Sandhya Babu <sandhya.bca7@gmail.com>

Day23 Challenge: Securing Santa's Workshop: Mastering Kubernetes RBAC and AWS IAM Integration

1 message

Sagar Utekar <getfitwithsagar2366@gmail.com> Wed, Dec 25, 2024 at 8:00 AM
Bcc: sandhya.bca7@gmail.com

Hello Learners,

Welcome back to another thrilling episode of the **DevOps SRE Daily Challenge!** 🎄 🎅

In the spirit of the holiday season, today's challenge focuses on securing your Kubernetes clusters with a **Christmas and Santa Claus theme**. We will explore the integration of AWS IAM with Kubernetes **Role-Based Access Control (RBAC)** and the management of users and groups created using **OpenSSL**.

Your mission? To ensure that **Santa's workshop** is secure, allowing only the right elves (users) to access the resources they need while keeping out any naughty ones!

What You'll Learn:

- How to create and manage Kubernetes RBAC for users and groups created with OpenSSL.
- How to integrate AWS IAM users and roles with Kubernetes RBAC in an Amazon EKS cluster.
- How to utilize Kubernetes service accounts for applications running in pods.
- Best practices for managing access permissions securely in your Kubernetes environment.

Challenge Tasks:

The RBAC and IAM Theory Challenge:

Before we dive into practical tasks, let's set the scene:

1. **What is Kubernetes RBAC?**

- Define RBAC and explain its role in managing access to Kubernetes resources.
- Discuss why integrating AWS IAM with Kubernetes RBAC is crucial for securing access during the holiday rush.

2. **Understanding AWS IAM Integration:**

- Explain how AWS IAM roles and users interact with Kubernetes RBAC.
- Describe the purpose of the `aws-auth` ConfigMap in mapping IAM identities to Kubernetes roles.

3. **Key Components of RBAC:**

- What are Roles and ClusterRoles in the context of Santa's workshop?
- Explain the difference between RoleBindings and ClusterRoleBindings—who gets what access?

4. **Service Accounts:**

- What are service accounts, and how do they differ from user accounts?
- Explain how service accounts can be used to manage permissions for applications running in pods.

5. **Best Practices for Securing Santa's Workshop:**

- List essential best practices for managing RBAC effectively while integrating with AWS IAM.
- Discuss the importance of regularly reviewing permissions to prevent unauthorized access by mischievous elves.

Implementing RBAC with OpenSSL Users, AWS IAM, and Service Accounts

Time to roll up your sleeves and secure Santa's operations!

Task 1: Grant Access to Users and Groups Created with OpenSSL

Setup: Use a Kubernetes cluster deployed via Kubeadm or an EKS cluster for this task.

1. **Create a User:**

   Use OpenSSL to generate a certificate for the user.

   ```
   openssl genrsa -out user.key 2048

   openssl req -new -key user.key -out user.csr -subj "/CN=user"

   openssl x509 -req -in user.csr -signkey user.key -out user.crt
   ```

2. **Create a Group:**

   Similarly, create a certificate for the group.

   ```
   openssl genrsa -out group.key 2048

   openssl req -new -key group.key -out group.csr -subj "/CN=group"

   openssl x509 -req -in group.csr -signkey group.key -out group.crt
   ```

3. Create a Role:

Define a Role that grants specific permissions within a namespace (e.g., `development`).

```
apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

  namespace: development

  name: developer-role

rules:

- apiGroups: [""]

  resources: ["pods"]

  verbs: ["get", "list", "create"]
```

4. Create RoleBindings:

Bind the Role to the user and the group.

- **RoleBinding for User:**

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: user-role-binding
  namespace: development
subjects:
- kind: User
  name: user  # Replace with actual username from certificate CN
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
```

- **RoleBinding for Group:**

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: group-role-binding
  namespace: development
subjects:
- kind: Group
  name: group  # Replace with actual group name from certificate CN
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
```

5. Apply the Configurations:

```
kubectl apply -f role.yaml
```

```
kubectl apply -f user-role-binding.yaml

kubectl apply -f group-role-binding.yaml
```

Task 2: Grant Access to IAM Users and Roles in EKS

Setup: Continue using the same EKS cluster.

1. **Create an IAM User in AWS Management Console or CLI.**
2. Update `aws-auth` ConfigMap:
   Edit the `aws-auth` ConfigMap to map the IAM user to a Kubernetes user.

```
kubectl edit configmap aws-auth -n kube-system
```

Add entries under `mapUsers`:

```
mapUsers: |
  - userarn: arn:aws:iam::123456789012:user/santa-helper
    username: santa-helper
    groups:
```

```
    - system:masters
```

3. **Create an IAM Role that can be assumed by EC2 instances.**
4. **Update `aws-auth` ConfigMap for IAM Role:**
   Add entries under `mapRoles` in the `aws-auth` ConfigMap:

```
mapRoles: |
  - rolearn: arn:aws:iam::123456789012:role/santa-ec2-role
    username: santa-ec2-user
    groups:
      - system:nodes
```

5. **Create ClusterRole and ClusterRoleBinding (if needed):**
   Define ClusterRoles and bind them to the IAM users or roles if they need access across namespaces.

- Create a ClusterRole:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
  name: deployment-viewer
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list"]
```

- Create ClusterRoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: deployment-viewer-binding
subjects:
- kind: User
  name: santa-helper
```

```
roleRef:
  kind: ClusterRole
  name: deployment-viewer
```

6. Apply the Configurations:

```
kubectl apply -f clusterrole.yaml
kubectl apply -f clusterrolebinding.yaml
```

Task 3: Utilize Service Accounts in Your Applications

1. Create a Service Account for an application running in your cluster (e.g., an application that needs access to pods).

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: app-sa
  namespace: development
```

2. Bind Permissions to Service Account using a Role or ClusterRole.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-sa-binding
  namespace: development
subjects:
- kind: ServiceAccount
  name: app-sa
roleRef:
  kind: Role
  name: developer-role
```

3. Apply the Service Account Configuration:

```
kubectl apply -f serviceaccount.yaml

kubectl apply -f app-sa-binding.yaml
```

Task 4: Deploy Applications and Validate Access

1. Deploy an Application Using Service Account:

Deploy a simple application that uses the service account created earlier.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: example-app-deployment

spec:

  replicas: 1

  selector:
```

```
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
    spec:
      serviceAccountName: app-sa
      containers:
      - name: example-app-container
        image: nginx
        ports:
        - containerPort: 80
```

Apply this deployment configuration.

```
kubectl apply -f deployment.yaml
```

2. Validate Access Using kubectl auth can-i:

You can use `kubectl auth can-i` commands to validate whether specific actions are allowed for different users, groups, or service accounts.

- Check if `santa-helper` can list pods in `development` namespace as follows:

```
kubectl auth can-i list pods --as=santa-helper --namespace=development
```

- Check if `user` can create pods:

```
kubectl auth can-i create pods --as=user --namespace=development
```

- Check if `app-sa` (service account) can get pods:

```
kubectl auth can-i get pods --as=system:addserviceaccount --
namespace=development
```

3. Check Application Logs (for debugging):

You can check logs of your application pod to ensure it is running correctly:

```
kubectl logs <example-app-pod-name> --namespace=development
```

Bonus Task:

Implement Best Practices for RBAC Management in Santa's Workshop:

- Develop a strategy for regularly reviewing and auditing RBAC configurations in your cluster, especially focusing on mappings between AWS IAM roles/users, OpenSSL users/groups, and Kubernetes roles/service accounts.
- Create documentation outlining roles, bindings, service accounts, and their purposes within your festive organization.

Report Observations:

- Note any challenges encountered during role creation and binding.
- Share insights on how implementing RBAC improved security within Santa's workshop.

Submission Guidelines:

Submit the following:

- Answers to the theory section.
- Screenshots of:
- OpenSSL user/group creation steps.
- AWS IAM user/role creation steps.
- Service account creation steps.

- Role, RoleBinding, ClusterRole, and ClusterRoleBinding creation steps.
- Screenshots or outputs showing successful validation of access permissions through application deployments using `kubectl auth can-i`.
- Documentation of your RBAC management strategy.
- Your insights and observations from the tasks.
- Post your progress with the hashtags: #KubernetesRBAC, #DevOpsChallenge, #HolidayCheer

Resources to Help You:

- Kubernetes RBAC Documentation
- AWS EKS Authentication with IAM
- Integrating AWS IAM with Kubernetes
- Kubernetes RBAC Step by step guide
- Kubernetes Authn & Authz

If you missed any previous challenges, you can catch up by reviewing the problem statements on GitHub.

Best regards,
Sagar Utekar