RBAC in Kubernetes: Create Multiple Users and Attach Permissions to the EKS Cluster

Pre-requisites

1. Access to an EKS cluster.
2. Admin permissions to modify the aws-auth ConfigMap in the kube-system namespace.
3. IAM permissions to create and manage AWS users, roles, and policies.
4. Installed tools:
    - AWS CLI
    - kubectl (configured for the EKS cluster)
    - Proper network connectivity to the EKS API endpoint.

Step1: Create Namespaces

Namespaces help logically isolate resources. Create the following namespaces:

```
kubectl create namespace dev
kubectl create namespace prod
kubectl create namespace test
```

Step 2: Create IAM Users

1. In the AWS Management Console, create IAM users (e.g., user1, user2, user3) with programmatic access.
2. Generate and securely save their Access Key ID and Secret Access Key.

Step 3: Attach Policy for EKS Cluster Access

Attach the following IAM policy to each user to grant access to the specific EKS cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:eks:<region>:<account ID>:cluster/<clustername>"
      ]
    }
  ]
}
```

Replace:

- <region>: AWS region (e.g., us-east-2)
- <account_id>: AWS account ID
- <cluster_name>: Your EKS cluster name

Step 4: Map IAM Users to Kubernetes

Edit the aws-auth ConfigMap in the kube-system namespace to associate IAM users with Kubernetes usernames and groups:

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

Add the following under mapUsers:

```
mapUsers: |
 - userarn: arn:aws:iam::<AWS_ACCOUNT_ID>:user/user1
   username: user1
   groups:
     - dev-group
 - userarn: arn:aws:iam::<AWS_ACCOUNT_ID>:user/user2
   username: user2
   groups:
     - prod-group
 - userarn: arn:aws:iam::<AWS_ACCOUNT_ID>:user/user3
   username: user3
   groups:
     - test-group
```

Validate the aws-auth ConfigMap:

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

Step 5: Create RBAC Roles for Namespace Access

Define roles to grant full access to each namespace:

Role for Full Access in dev **Namespace**

dev-role.yaml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-namespace-admin
rules:
 - apiGroups: [""]
   resources: ["*"]
   verbs: ["*"]
```

Role for Full Access in prod **Namespace:**

prod-role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: prod
  name: prod-namespace-admin
rules:
  - apiGroups: [""]
    resources: ["*"]
    verbs: ["*"]
```

Role for Full Access in test **Namespace:**

test-role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: test
  name: test-namespace-admin
rules:
  - apiGroups: [""]
    resources: ["*"]
    verbs: ["*"]
```

Apply these Role configurations:

```
kubectl apply -f dev-role.yaml
kubectl apply -f prod-role.yaml
kubectl apply -f test-role.yaml
```

Step 6: Bind Users to Their Namespaces

Create RoleBinding for each user to bind them to their respective namespaces.

Bind user1 **to** dev **Namespace:**

user1-binding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user1-dev-binding
  namespace: dev
subjects:
  - kind: User
    name: user1                # make sure to match the username in aws-auth ConfigMap
    apiGroup: rbac.authorization.k8s.io
```

```
roleRef:
  kind: Role
  name: dev-namespace-admin
  apiGroup: rbac.authorization.k8s.io
```

Bind user2 **to** prod **Namespace:**

User2-binding.yaml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user2-prod-binding
  namespace: prod
subjects:
  - kind: User
    name: user2
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: prod-namespace-admin
  apiGroup: rbac.authorization.k8s.io
```

Bind user3 **to** test **Namespace:**

User3-binding.yaml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user3-test-binding
  namespace: test
subjects:
  - kind: User
    name: user3
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: test-namespace-admin
  apiGroup: rbac.authorization.k8s.io
```

Apply the RoleBindings:

```
kubectl apply -f user1-binding.yaml
kubectl apply -f user2-binding.yaml
kubectl apply -f user3-binding.yaml
```

Step 7: Test User Permissions

For each user:

1. Configure **kubectl** to use their credentials:

```
aws eks --region <region> update-kubeconfig --name <cluster_name>
aws configure
```

Enter the **Access Key ID** and **Secret Access Key** specific to the user.

2. Verify namespace-specific access:

For user1 **in** dev **Namespace**:

```
kubectl auth can-i list pods -n dev
kubectl get pods -n dev
kubectl auth can-i list pods -n prod      # Should return "no"
kubectl auth can-i list pods -n test      # Should return "no"
```

```
ubuntu@ip-172-31-10-176:~$ aws eks --region us-east-2 update-kubeconfig --name three-tier
Updated context arn:aws:eks:us-east-2:842676005617:cluster/three-tier in /home/ubuntu/.kube/config
ubuntu@ip-172-31-10-176:~$ aws configure
AWS Access Key ID [****************BZ6H]: AKIA4IM[redacted]
AWS Secret Access Key [****************cjle]: VrCDEdm8D[redacted]XWdch
Default region name [us-east-2]:
Default output format [None]:
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n dev
yes
ubuntu@ip-172-31-10-176:~$ kubectl get pods -n dev
No resources found in dev namespace.
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n prod
no
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n test
no
ubuntu@ip-172-31-10-176:~$ kubectl get pods -n prod
Error from server (Forbidden): pods is forbidden: User "eks-user1" cannot list resource "pods" in API group "" in the namespace "prod"
ubuntu@ip-172-31-10-176:~$ kubectl get pods -n test
Error from server (Forbidden): pods is forbidden: User "eks-user1" cannot list resource "pods" in API group "" in the namespace "test"
```

```
ubuntu@ip-172-31-10-176:~$ kubectl run ecommerce-frontend --image=madeep2669/ecommerce-frontend:latest -n dev
pod/ecommerce-frontend created
ubuntu@ip-172-31-10-176:~$ kubectl get pods -n dev
NAME                 READY   STATUS    RESTARTS   AGE
ecommerce-frontend   1/1     Running   0          10s
```

For user2 **in** prod **Namespace:**

```
kubectl auth can-i list pods -n prod
kubectl get pods -n prod
kubectl auth can-i list pods -n dev      # Should return "no"
kubectl auth can-i list pods -n test     # Should return "no"
```

```
ubuntu@ip-172-31-10-176:~$ aws eks --region us-east-2 update-kubeconfig --name three-tier
Updated context arn:aws:eks:us-east-2:842676005617:cluster/three-tier in /home/ubuntu/.kube/config
ubuntu@ip-172-31-10-176:~$ aws configure
AWS Access Key ID [****************JJJQ]: AK[redacted]3Z6H
AWS Secret Access Key [****************YwPq]: XfcvUC[redacted]C/O7pDNjK[redacted]yeaI1Scjle
Default region name [us-east-2]:
Default output format [None]:
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n prod
yes
ubuntu@ip-172-31-10-176:~$ kubectl get pods -n prod
No resources found in prod namespace.
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n dev
no
ubuntu@ip-172-31-10-176:~$ kubectl auth can-i list pods -n test
no
```

For user3 **in** test **Namespace:**

```
kubectl auth can-i list pods -n test
kubectl get pods -n test
kubectl auth can-i list pods -n dev       # Should return "no"
kubectl auth can-i list pods -n prod      # Should return "no"
```