**Implementing RBAC on EKS with IAM Users**

A stack of shipping containers

Description automatically generated

**What is RBAC?**

**Role-Based Access Control**(RBAC) is a cornerstone of Kubernetes security. When working with Amazon EKS, integrating RBAC with AWS Identity and Access Management (IAM) offers significant advantages. IAM centralizes your user directory, letting you easily control who can access what resources within your cluster — all while aligning with your existing AWS security practices.

**Why we need to integrate RBAC with IAM?**

But the issue arises when in a complex environment with numerous users and resources, the number of roles required for fine-grained permissions can quickly become overwhelming. Over time, permissions might get added to roles in an ad-hoc manner, without proper review or cleanup.

IAM acts as a central source of truth for user identities. Instead of defining and managing roles for each system separately (including Kubernetes), you can map existing IAM users and groups to Kubernetes roles, reducing redundancy and the potential for inconsistencies.

**Let’s start the practical**

Now, I’ll start to demonstrate on how to set up an IAM role for authentication and assign a RBAC role (authorization)to scope the API calls allowed.

**Step 1: IAM Task**

The following activities are being performed using an already logged in user.

* Go to the IAM Console and click on create user.
* Give the required details and a new user is created. I have given **admin** name for better understanding in the future tasks. Now do the above task again to create a user named: **developer.**

A screenshot of a computer

Description automatically generated

* Now we’ll create a policy.

{  
 "Version": "2012-10-17",  
 "Statement": [  
 {  
 "Sid": "VisualEditor0",  
 "Effect": "Allow",  
 "Action": [  
 "eks:ListEksAnywhereSubscriptions",  
 "eks:DescribeFargateProfile",  
 "eks:ListTagsForResource",  
 "eks:DescribeInsight",  
 "eks:ListAccessEntries",  
 "eks:ListAddons",  
 "eks:DescribeEksAnywhereSubscription",  
 "eks:DescribeAddon",  
 "eks:ListAssociatedAccessPolicies",  
 "eks:DescribeNodegroup",  
 "eks:ListUpdates",  
 "eks:DescribeAddonVersions",  
 "eks:ListIdentityProviderConfigs",  
 "eks:ListNodegroups",  
 "eks:DescribeAddonConfiguration",  
 "eks:DescribeAccessEntry",  
 "eks:DescribePodIdentityAssociation",  
 "eks:ListInsights",  
 "eks:ListPodIdentityAssociations",  
 "eks:ListFargateProfiles",  
 "eks:DescribeIdentityProviderConfig",  
 "eks:DescribeUpdate",  
 "eks:AccessKubernetesApi",  
 "eks:DescribeCluster",  
 "eks:ListClusters",  
 "eks:ListAccessPolicies"  
 ],  
 "Resource": "\*"  
 }  
 ]  
}

* Attach the policy to the two users.

This read-only policy will be used to talk to API server to authenticate.

* Create AWS credentials of the users and download them.

Go to the IAM users-> Select the user-> Go to security credentials. Click on create access key. And then choose the CLI option. After creating the keys, download them.

**Note:** Do for all the users.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Summing up:** We have in total 3 users. One is me named Rajat who has admin access and administrator access policy is already attached to it.

Admin and Developer users have EKS, list and read policy attached to it.

**Step 2: Cluster Creation**

* Install [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"  
unzip awscliv2.zip  
sudo ./aws/install

* Setting up the user: Use the credentials of main user which has administrator access has policy, as that will be the one creating eks cluster.

aws configure

* Install [eksctl](https://docs.aws.amazon.com/eks/latest/userguide/setting-up.html): Will use this to create cluster

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp  
sudo mv /tmp/eksctl /usr/local/bin  
eksctl version

* Install [kubectl](https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html):

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.0/2024-01-04/bin/linux/amd64/kubectl  
chmod +x ./kubectl  
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH  
kubectl version --client

* Create cluster

eksctl create cluster --name=<your-eks-cluster-name> --region=<your-region-code>

* Once the cluster is created successfully, verify it with the below command.

eksctl get cluster

* Add a node group.

eksctl create nodegroup \   
 --cluster=<your-eks-cluster-name> \   
 --region=<your-region-code> \   
 --name=<your-nodegroup-name> \   
 --node-type=<instance-type> \   
 --nodes=<desired-nodes> \  
 --nodes-min=<minimum-num-of-nodes> \  
 --nodes-max=<maximum-num-of-nodes> \  
 --ssh-access \  
 --ssh-public-key=<your-ssh-key>

* To interact with the EKS cluster using kubectl, update your kubeconfig:

eksctl utils write-kubeconfig --cluster=<your-cluster-name> --set-kubeconfig-context=true

* Run any command to check

kubcectl get all

**Step 3: Creating users in cluster.**

* Create a folder where we’ll store the everything related to RBAC.

mkdir role

* Now will create the cluster-role for admin.

vi role/admin.yml

apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: admin-clusterrole  
rules:  
 - apiGroups: [""]  
 resources: ["nodes", "namespaces", "pods"]  
 verbs: ["get", "list", "create"]  
 - apiGroups: ["apps"]  
 resources: ["deployments" ,"daemonsets" ,"statefulsets" ,"replicasets"]  
 verbs: ["get", "list", "create"]  
 - apiGroups: [ "batch"]  
 resources: ["jobs"]  
 verbs: ["get", "list", "create"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: admin-clusterrole-binding  
subjects:  
 - kind: User  
 name: admin  
 apiGroup: rbac.authorization.k8s.io  
roleRef:  
 kind: ClusterRole  
 name: admin-clusterrole  
 apiGroup: rbac.authorization.k8s.io

kubectl apply -f role/admin.yml

* Now will create a role for developer.

vi role/developer.yml

apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: developer-role  
rules:  
 - apiGroups: [""]  
 resources: ["pods"]  
 verbs: ["get", "list", "watch["]  
 - apiGroups: ["apps"]  
 resources: ["deployments" ,"daemonsets" ,"statefulsets" ,"replicasets"]  
 verbs: ["get", "list", "create"]  
 - apiGroups: [ "batch"]  
 resources: ["jobs"]  
 verbs: ["get", "list"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: developer-role-binding  
subjects:  
 - kind: User  
 name: developer  
 apiGroup: rbac.authorization.k8s.io  
roleRef:  
 kind: Role  
 name: developer-role  
 apiGroup: rbac.authorization.k8s.io

kubectl apply -f role/developer.yml

**Setting up the connection**

By default, whoever creates an EKS cluster, that IAM User or IAM Role will have full admin access to it. In order to add additional Users/Roles there is a ConfigMap which is used for management of Users or IAM Roles for the Cluster.

To see the default config-map.

kubectl get configmap aws-auth -n kube-system -o yaml

* This will show that the user who created the cluster is mapped. Now we’ll map the admin and developer user.

kubectl edit configmap aws-auth -n kube-system -o yaml

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: aws-auth  
 namespace: kube-system  
data:  
 mapRoles: |  
 - groups:  
 - system:bootstrappers  
 - system:nodes  
 rolearn: arn:aws:iam::00000000000:role/eksctl-my-cluster-node-NodeInstanceRole-TUKH4Z187ANC  
 username: system:node:{{EC2PrivateDNSName}}  
 - groups:  
 - system:bootstrappers  
 - system:nodes  
 rolearn: arn:aws:iam::00000000000:role/eksctl-my-cluster-node-NodeInstanceRole-2RY0KK33CGIG  
 username: system:node:{{EC2PrivateDNSName}}  
 mapUsers: |  
 - userarn: arn:aws:iam::00000000000:user/admin  
 username: admin  
 - userarn: arn:aws:iam::00000000000:user/developer  
 username: developer

Now, the users are connected and can be used to communicate with EKS cluster.

**CONCLUSION**

By integrating IAM with RBAC for your EKS cluster, you gain a centralized and scalable way to manage user access and permissions. This improves security and reduces administrative overhead compared to managing Kubernetes roles independently. Remember to regularly review your RBAC policies as your team and application requirements evolve.