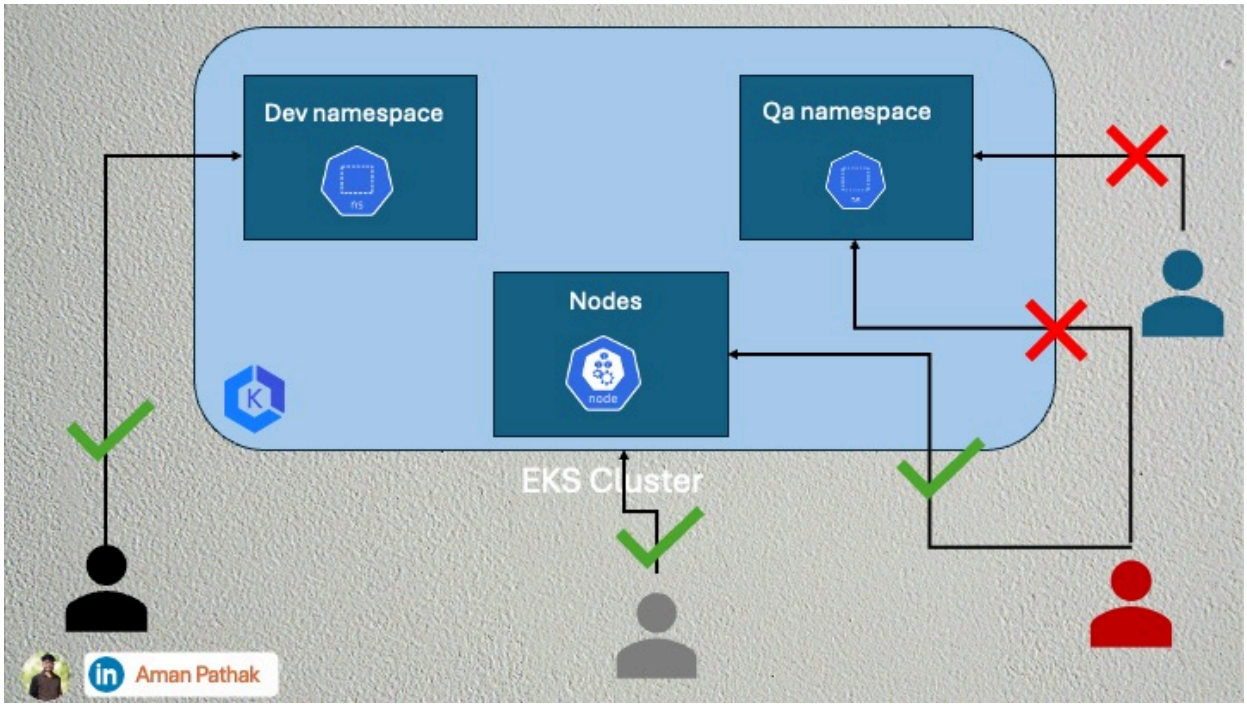
Configure RBAC on Production Ready EKS Cluster



Introduction

Managing user roles and permissions is a critical part of maintaining a secure and efficient Kubernetes environment on Amazon EKS. In this blog, we'll walk through the setup and configuration of Role-Based Access Control (RBAC) for multiple users on an EKS cluster. With these steps, you'll gain a hands-on understanding of how to secure different levels of access for users across namespaces.

Prerequisites

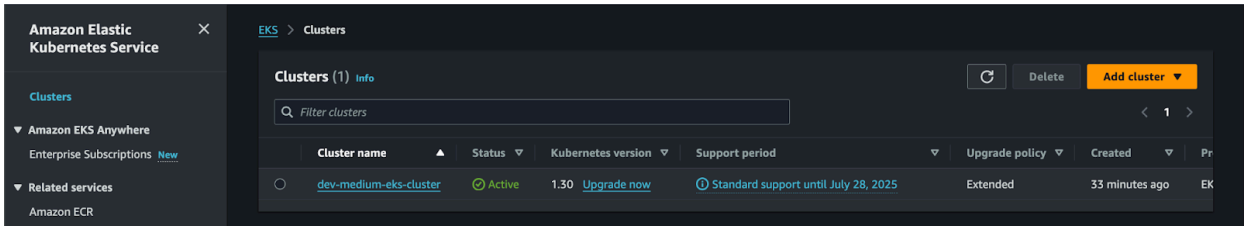
To follow along, make sure you have:

- An existing EKS cluster in your AWS account. If you need help setting one up, check out this guide(Configuring Production-Ready EKS Clusters with Terraform and GitHub Actions | by Aman Pathak | Stackademic) on deploying a production-ready EKS cluster using Terraform and GitHub Actions.
- Basic knowledge of Kubernetes concepts like namespaces and roles.
- AWS CLI, kubectl, and the necessary permissions on your AWS account to access and configure EKS resources.

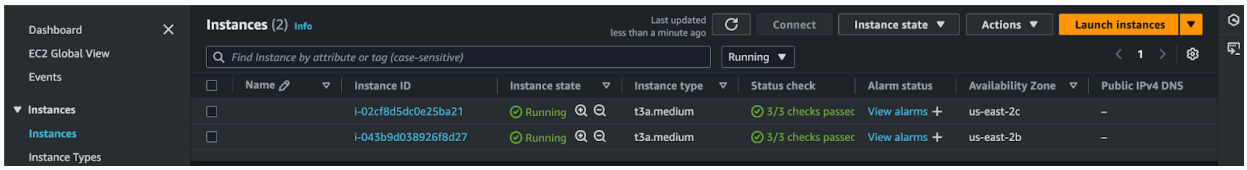
Objective

In this guide, we'll configure RBAC on an EKS cluster to set up different permissions for multiple users. This will allow us to control access at both the namespace and cluster levels, ensuring that each user has the necessary permissions without over-privileging.

EKS Cluster has been created

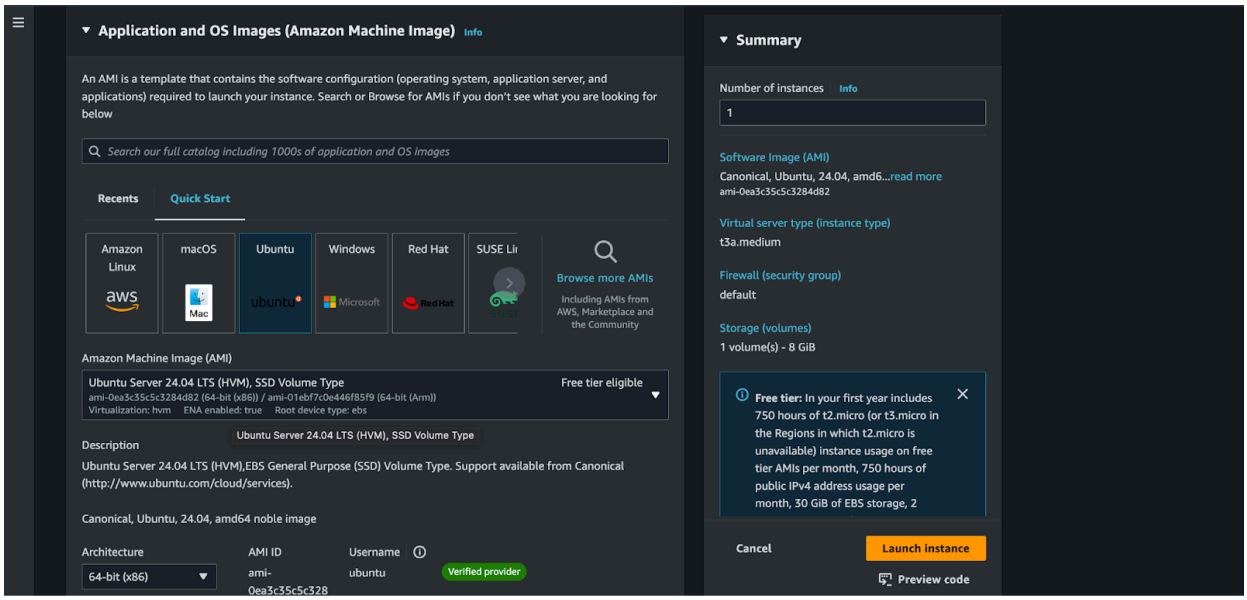


Both nodes are running.

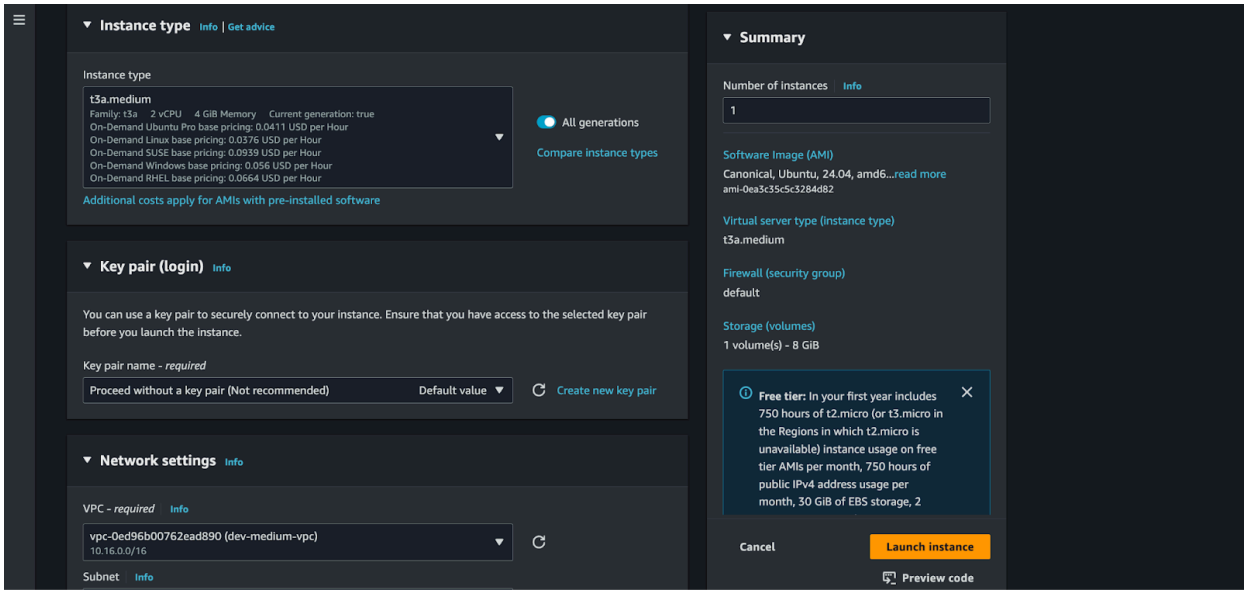


Our EKS Cluster is Private, so we can't access our Kubernetes cluster outside of the network(VPC). So, we will create a new EC2 within the same VPC to access the Kubernetes cluster.

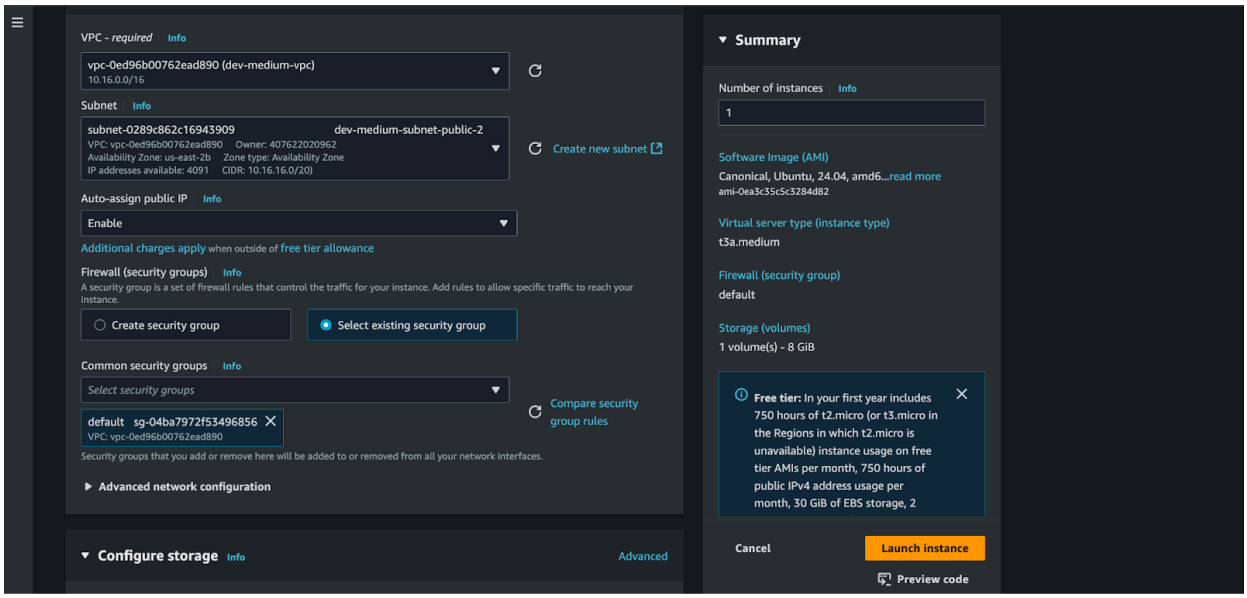
You can check out the configurations for EKS-Jump-Server to access the Kubernetes cluster.



Be sure to add the same VPC where the EKS cluster and other services have been created. You can also add a Key pair if you are not going to use Session Manager to log into EKS-Jump-Server.



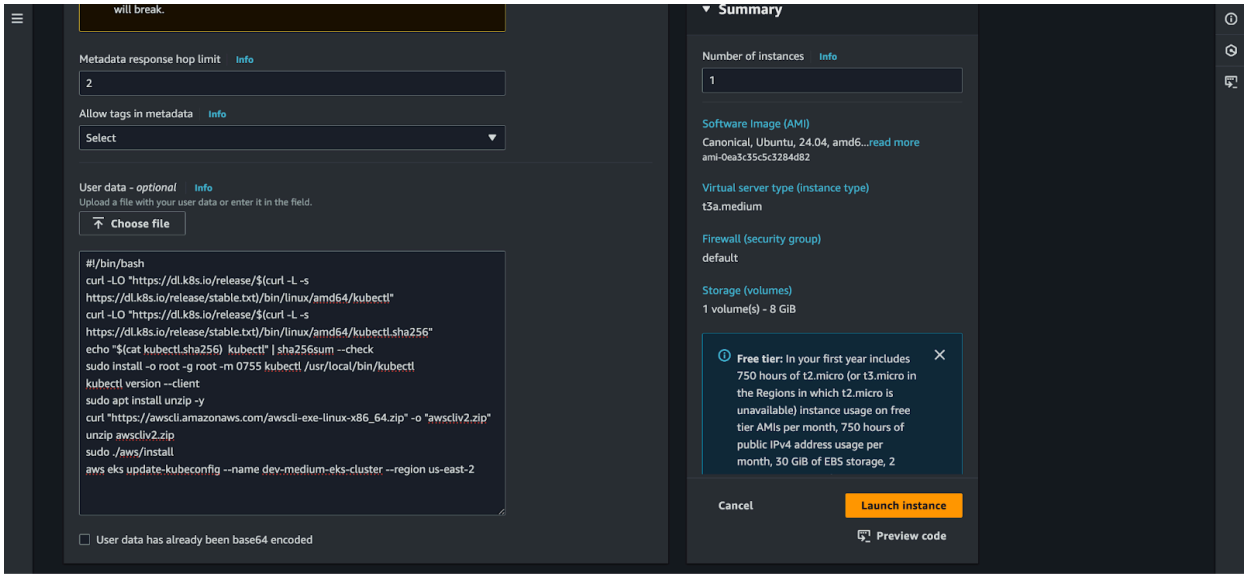
Ensure to keep your EKS-Jump-Server in the Public Subnet and Auto-assign public IP must be Enabled.



As we need to install a few tools. Use the below script to install on your EKS-Jump-Server

```
#!/bin/bash
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
curl -LO "https://dl.k8s.io/release/$(curl -L -s
```

```
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
sudo apt install unzip -y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```



Now, Login to EKS-Jump-Server and validate that both tools have been installed by running the below command.

```
kubectl version
aws --version
```

```
ubuntu@ip-10-16-21-138:~$ kubectl version
Client Version: v1.31.2
Kustomize Version: v5.4.2
The connection to the server localhost:8080 was refused - did you specify the right host or port?
ubuntu@ip-10-16-21-138:~$
ubuntu@ip-10-16-21-138:~$ aws --version
aws-cli/2.19.1 Python/3.12.6 Linux/6.8.0-1016-aws exe/x86_64.ubuntu.24
ubuntu@ip-10-16-21-138:~$
```

Configure your AWS credentials by entering the access key and secret access key id. Make sure that your AWS credentials have EKS-related permissions. Once you configured AWS credentials, you are good to go for kubectl commands

```
aws configure
aws eks update-kubeconfig --name <cluster-name> --region
<aws-region>
kubectl get nodes
```

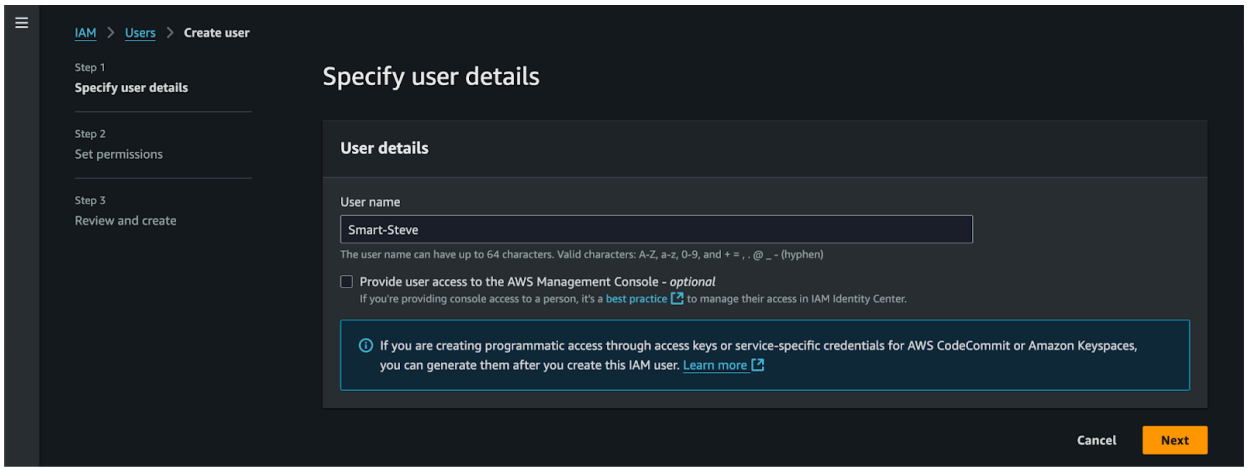
```
ubuntu@ip-10-16-21-138:~$ aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-2
Default output format [None]: json
ubuntu@ip-10-16-21-138:~$
ubuntu@ip-10-16-21-138:~$ aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2
Updated context arn:aws:eks:us-east-2:407622020962:cluster/dev-medium-eks-cluster in /home/ubuntu/.kube/config
ubuntu@ip-10-16-21-138:~$
ubuntu@ip-10-16-21-138:~$
ubuntu@ip-10-16-21-138:~$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-16-158-181.us-east-2.compute.internal   Ready    <none>   43m   v1.30.4-eks-a737599
ip-10-16-172-243.us-east-2.compute.internal   Ready    <none>   43m   v1.30.4-eks-a737599
ubuntu@ip-10-16-21-138:~$
```

After running the command **aws sts get-caller-identity,** you can see the user under the Arn section. Currently, Arn has admin-level permission for the EKS cluster. So, we will define multiple roles for multiple users according to the requirements.

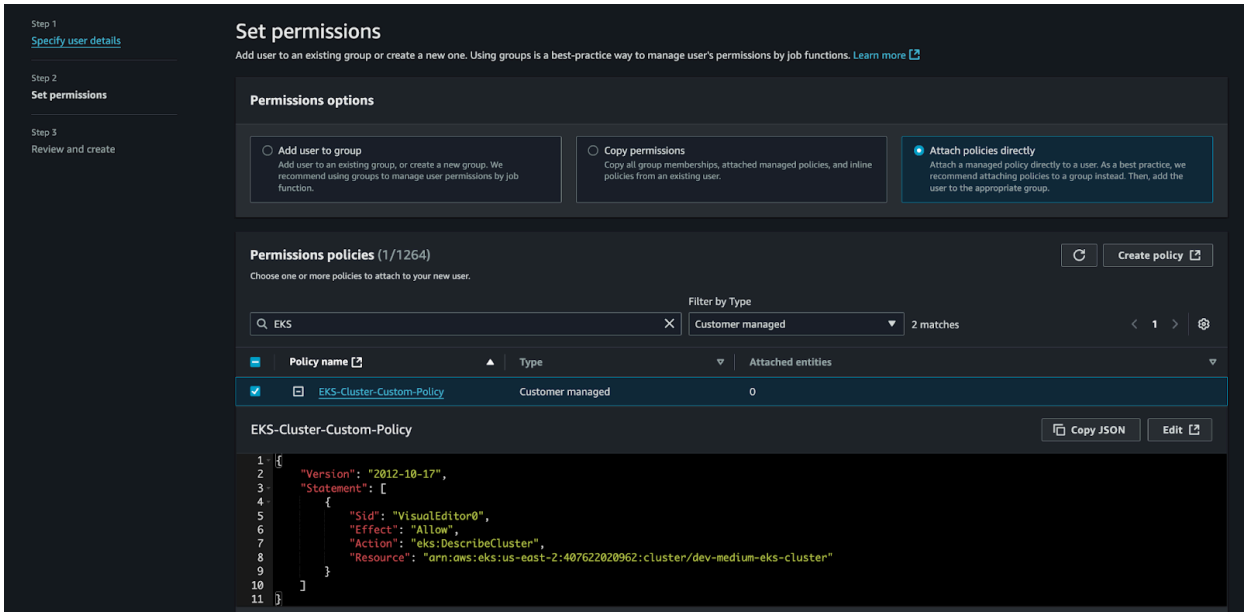
```
ubuntu@ip-10-16-21-138:~$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE   VERSION
ip-10-16-158-181.us-east-2.compute.internal   Ready    <none>   65m   v1.30.4-eks-a737599
ip-10-16-172-243.us-east-2.compute.internal   Ready    <none>   66m   v1.30.4-eks-a737599
ubuntu@ip-10-16-21-138:~$
ubuntu@ip-10-16-21-138:~$ aws sts get-caller-identity
{
    "UserId": "AIDAV52BKN5RNSWYD2H4V",
    "Account": "407622020962",
    "Arn": "arn:aws:iam::407622020962:user/Adorable-Aman"
}
ubuntu@ip-10-16-21-138:~$
```

So, first of all, we will be going to create users that are shown below in the snippet
P.S.: The first user has already been created.

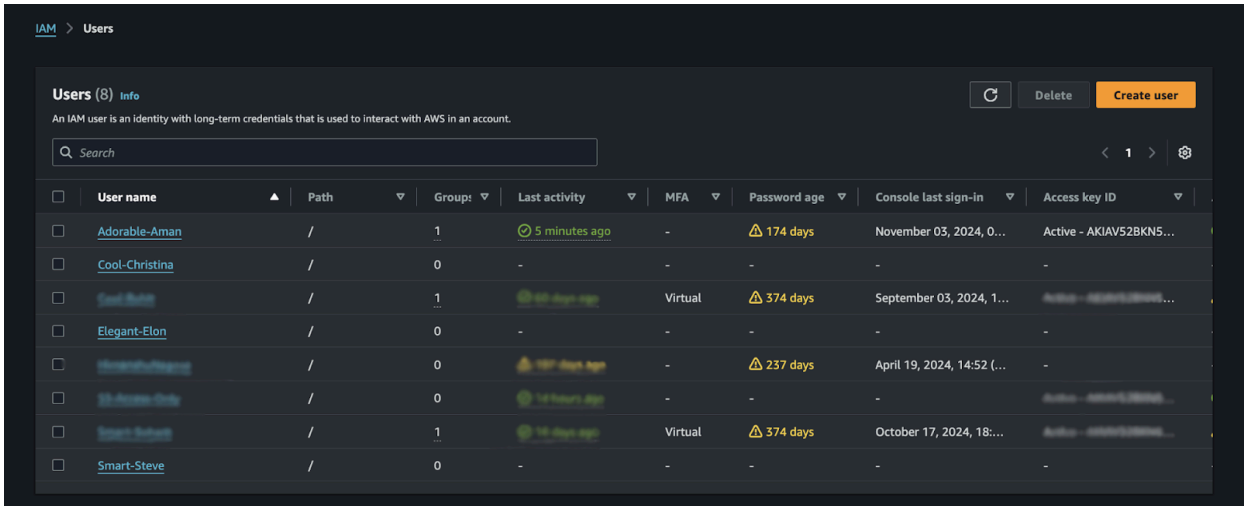
Go to IAM



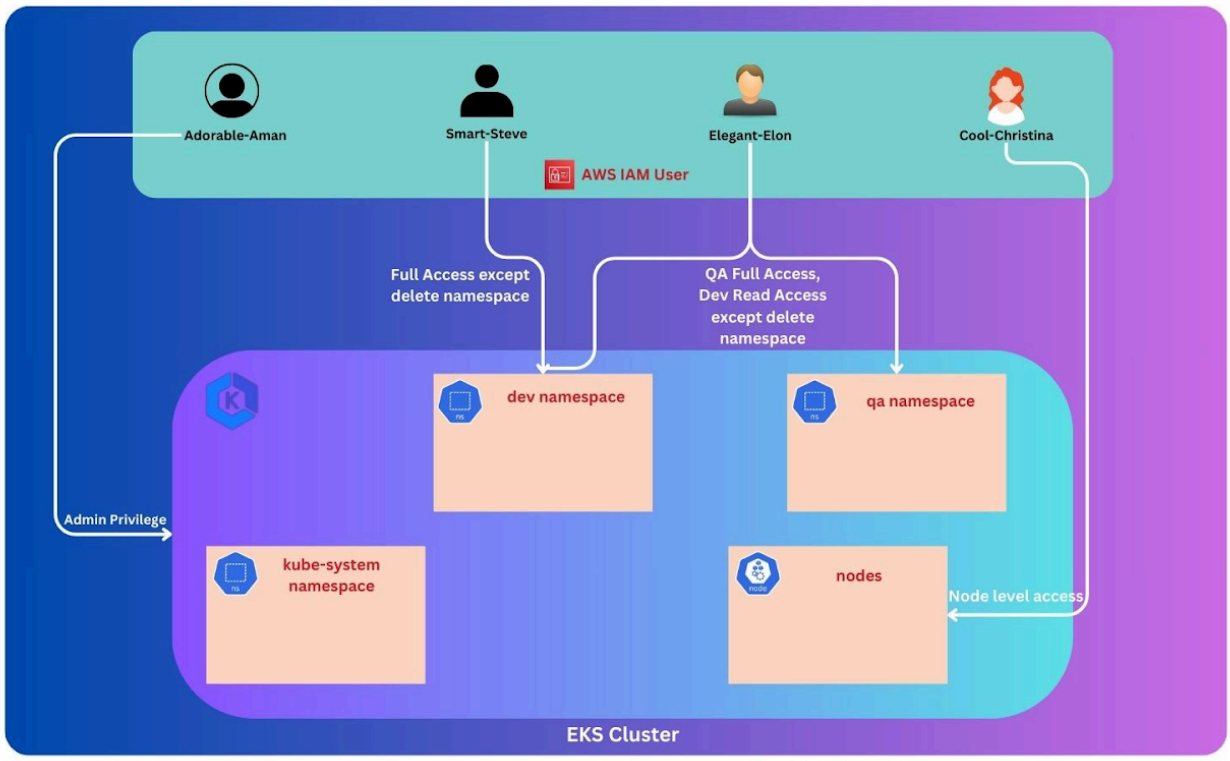

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "eks:DescribeCluster",
            "Resource":
"arn:aws:eks:us-east-2:<account_id>:cluster/dev-medium-eks-clust
er"
        }
    ]
}
```



Need to blue irrelevant usernames.

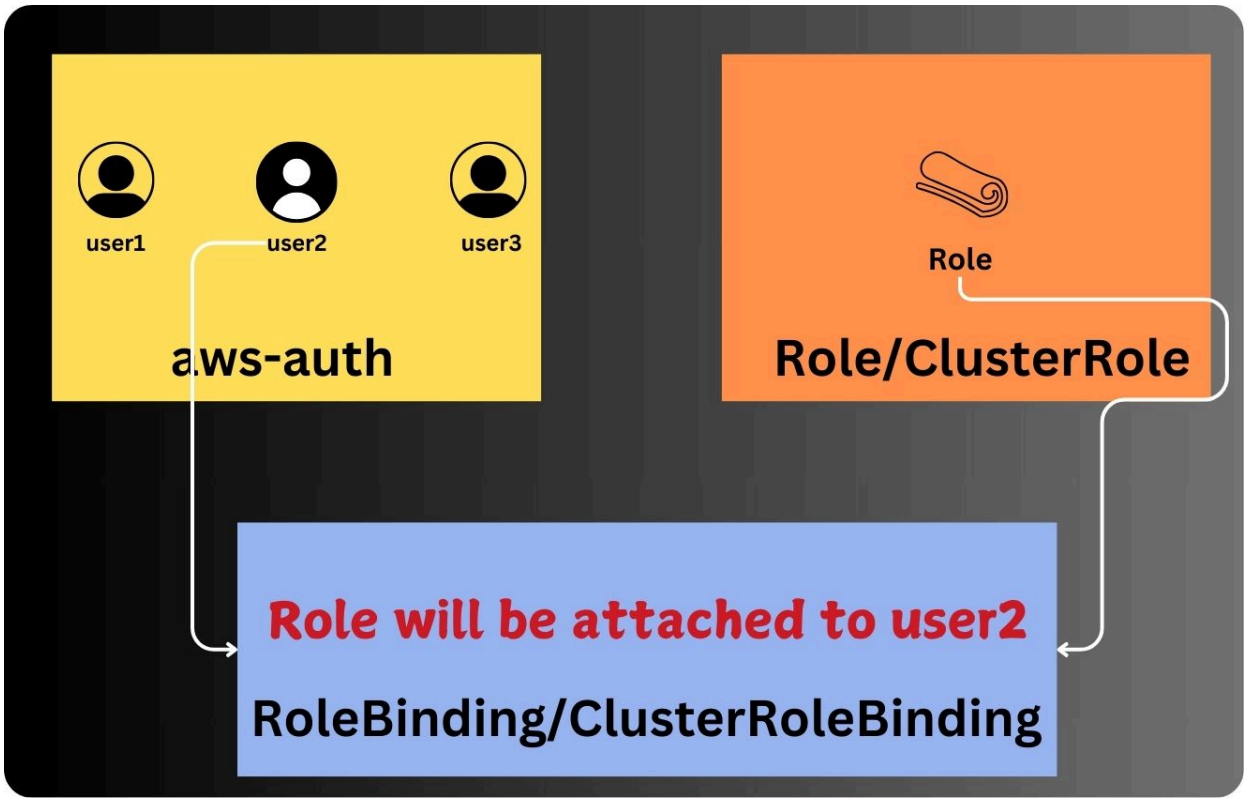


The basic diagram to understand the flow of today's HandsOn



Before going on HandsOn, we need to understand a small theory for three important things

- aws-auth
- Role/ClusterRole
- RoleBinding/ClusterRoleBinding



aws-auth

The **aws-auth** is a configmap in Kubernetes(EKS) which comes predefined but you need it to set up RBAC on your EKS cluster. The aws-auth contains the name of the username and user arn which is present in the IAM user.

Role vs ClusterRole

As the name suggests, the **Role**s are bound to specific namespace which means the role will be intended for namespaces. So, if you want to set up a role on a particular namespace instead of the entire Cluster you should consider Role.

The **ClusterRole** binds to the entire Cluster where if you want to set up a role on a cluster level you should consider ClusterRole.

RoleBinding vs ClusterRoleBinding

In **RoleBinding**, we generally attach a created role with the user that is provided in aws-auth configmap

In **ClusterRoleBinding**, we generally attach a created cluster role with the user that is provided in the aws-auth configmap
For a detailed understanding refer to the official documentation-
https://kubernetes.io/docs/reference/access-authn-authz/rbac/

RBAC HandsOn

Configure RBAC for Smart-Steve user

We need to provide access to the **dev** namespace to the user Smart-Steve, we need to create a dev namespace being an admin.
kubectl create ns dev

```
ubuntu@ip-10-16-45-162:~$ kubectl create ns dev
namespace/dev created
ubuntu@ip-10-16-45-162:~$
ubuntu@ip-10-16-45-162:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   76m
dev               Active   7s
kube-node-lease   Active   76m
kube-public       Active   76m
kube-system       Active   76m
ubuntu@ip-10-16-45-162:~$
```

Before going to create role and role binding, we need to add our IAM users in the auth config maps
List the aws-auth as we are using AWS EKS

```
kubectl get cm -n kube-system
```

```
ubuntu@ip-10-16-45-162:~$ kubectl get cm -n kube-system
NAME                                                 DATA   AGE
amazon-vpc-cni                                       7      85m
aws-auth                                             1      80m
coredns                                              1      85m
extension-apiserver-authentication                   6      88m
kube-apiserver-legacy-service-account-token-tracking 1      88m
kube-proxy                                           1      85m
kube-proxy-config                                    1      85m
kube-root-ca.crt                                     1      88m
ubuntu@ip-10-16-45-162:~$
```

We need to get the existing yaml file for the aws-auth config map as we need to add our new users to the existing config map

```
kubectl get cm aws-auth -n kube-system -o yaml
```

```
ubuntu@ip-10-16-45-162:~$ kubectl get cm aws-auth -n kube-system -o yaml
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::407622020962:role/dev-medium-eks-cluster-nodegroup-role-5561
      username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2024-11-03T02:59:46Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "2111"
  uid: 276cde84-23cf-463b-8b9c-3806f29a0331
ubuntu@ip-10-16-45-162:~$
```

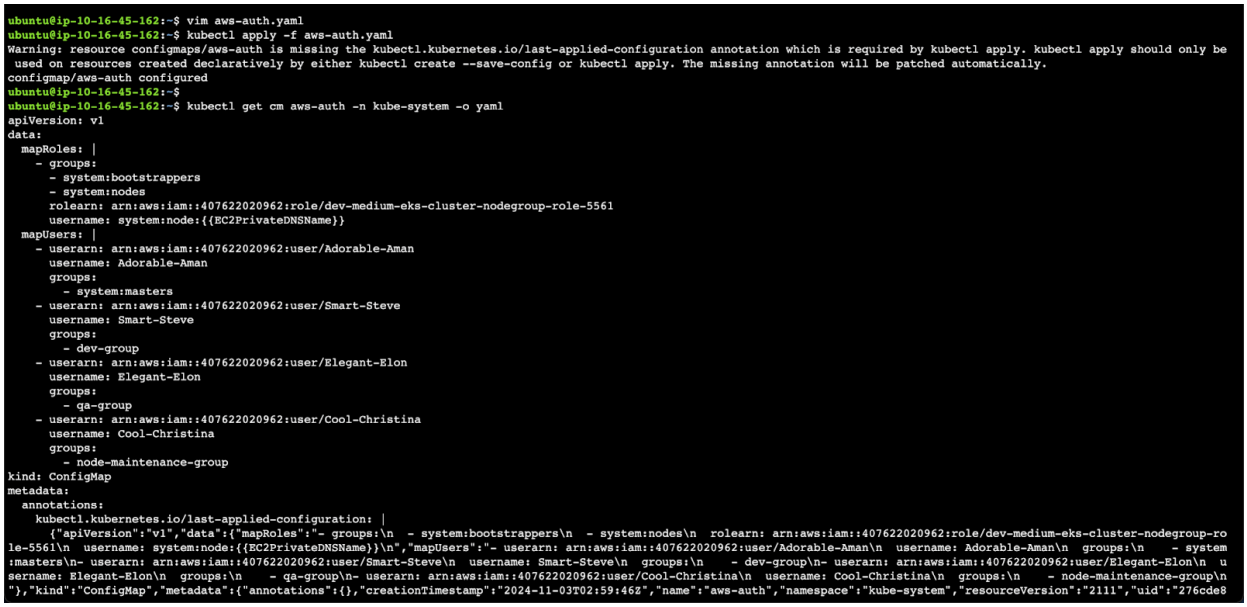
Once you get the existing aws-auth yaml file, paste it into your VSCode or any code editor. Then, add your users in aws-auth looking like the below snippet.

Once you add your usernames and other configs, you have to apply the configurations through the kubectl command

```
kubectl apply -f aws-auth.yaml
```

Also, you can validate whether the users have been added to your aws-auth config map or not by running the below command

```
kubectl get cm aws-auth -n kube-system -o yaml
```



Now, we will be going to create a Role.

I have created a dedicated directory where role and role binding are present as it won't create any confusion for all of us
You can see the role below as we are creating a dedicated role for the dev namespace with full privilege except delete the namespace

```yaml
# Role
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-full-access
rules:
  - apiGroups: [""]
    resources: ["pods", "services", "configmaps"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources:  ["dev"]
    verbs: ["get", "list"]
```

```
ubuntu@ip-10-16-45-162:~$ mkdir Smart-Steve
ubuntu@ip-10-16-45-162:~$ cd Smart-Steve/
ubuntu@ip-10-16-45-162:~/Smart-Steve$ ls
ubuntu@ip-10-16-45-162:~/Smart-Steve$ vim role.yaml
ubuntu@ip-10-16-45-162:~/Smart-Steve$ cat role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-full-access
rules:
  - apiGroups: [""]
    resources: ["pods", "services", "configmaps"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources:  ["dev"]
    verbs: ["get", "list"]
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, we will be going to create the role by applying the config

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/dev-full-access created
ubuntu@ip-10-16-45-162:~/Smart-Steve$
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get role -n dev
NAME              CREATED AT
dev-full-access   2024-11-03T04:15:01Z
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

In the role binding, we will attach our role to the user Smart-Steve. You can refer to the yaml file below

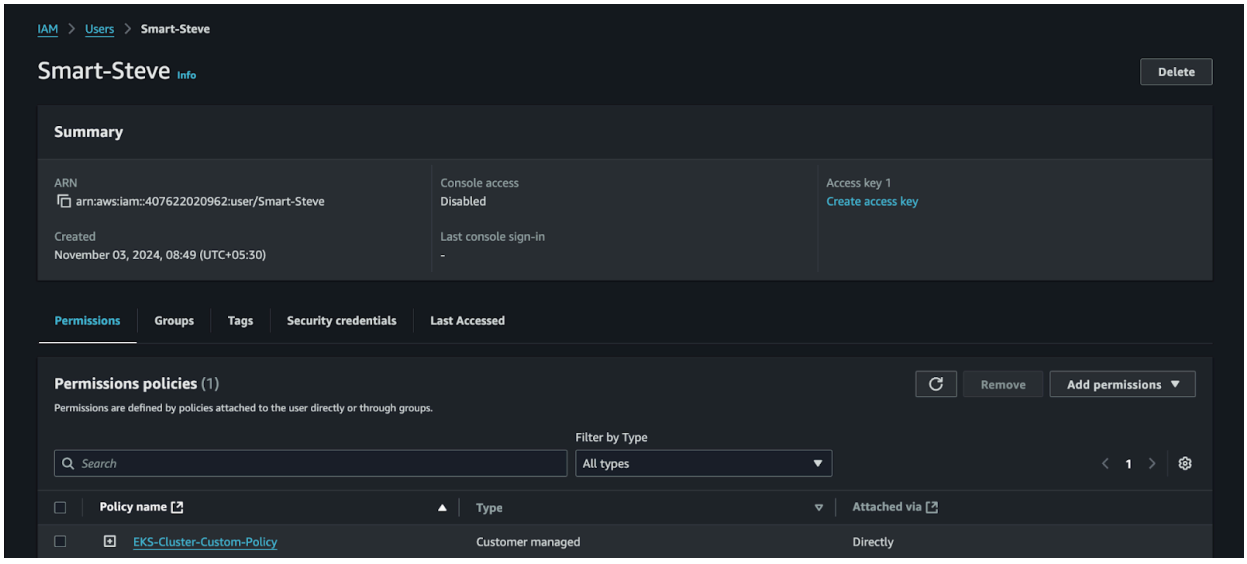
```yaml
# Role Binding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: dev-user-full-access
 namespace: dev
subjects:
 - kind: User
   name: Smart-Steve
   apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: Role
 name: dev-full-access
 apiGroup: rbac.authorization.k8s.io
```

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ vim rolebinding.yaml
ubuntu@ip-10-16-45-162:~/Smart-Steve$
ubuntu@ip-10-16-45-162:~/Smart-Steve$ cat rolebinding.yaml
# Role Binding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-user-full-access
  namespace: dev
subjects:
  - kind: User
    name: Smart-Steve
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: dev-full-access
  apiGroup: rbac.authorization.k8s.io
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

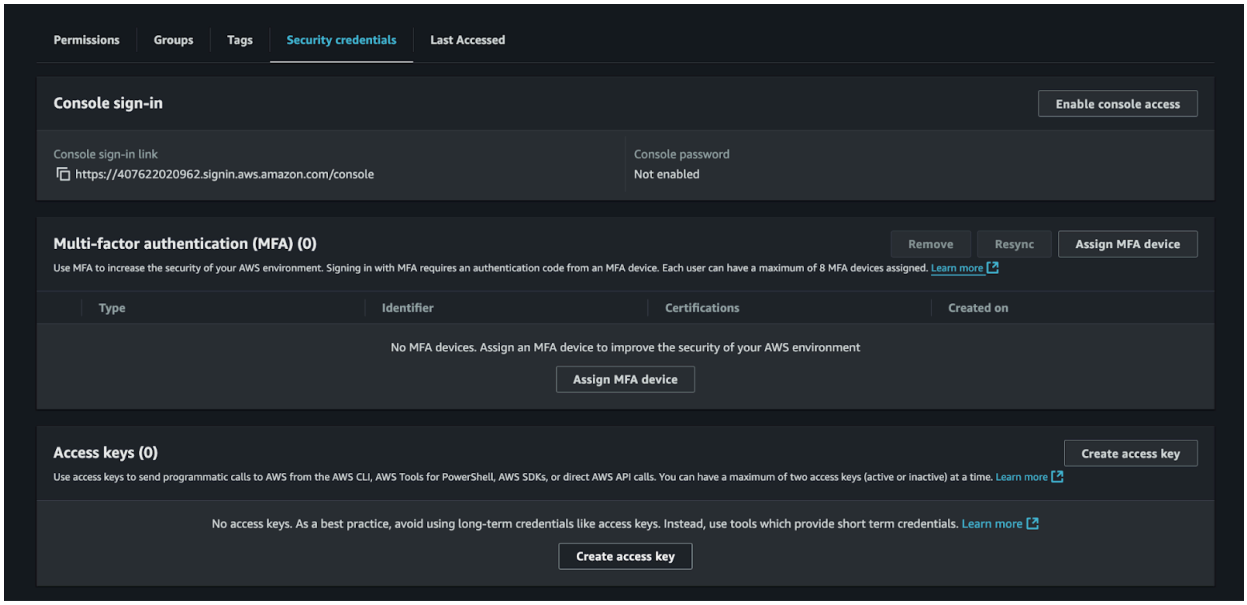
Now, we will be going to create the role binding by applying the config

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl apply -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/dev-user-full-access created
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get rolebinding -n dev
NAME                   ROLE                   AGE
dev-user-full-access   Role/dev-full-access   13s
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

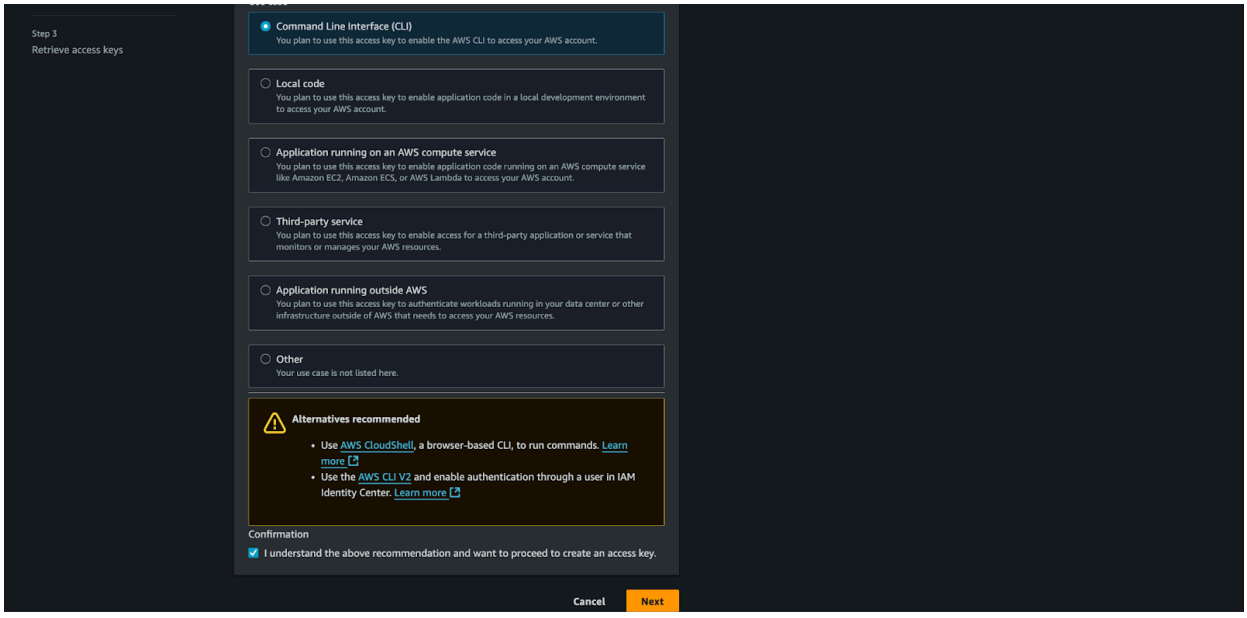
As we have configured all three things including aws-auth, role and rolebinding. Now, the next step is to generate the AWS access keys as we need to check whether the Kubernetes RBAC is working or not.
Navigate to your IAM user for that you have to create role and role binding in AWS.



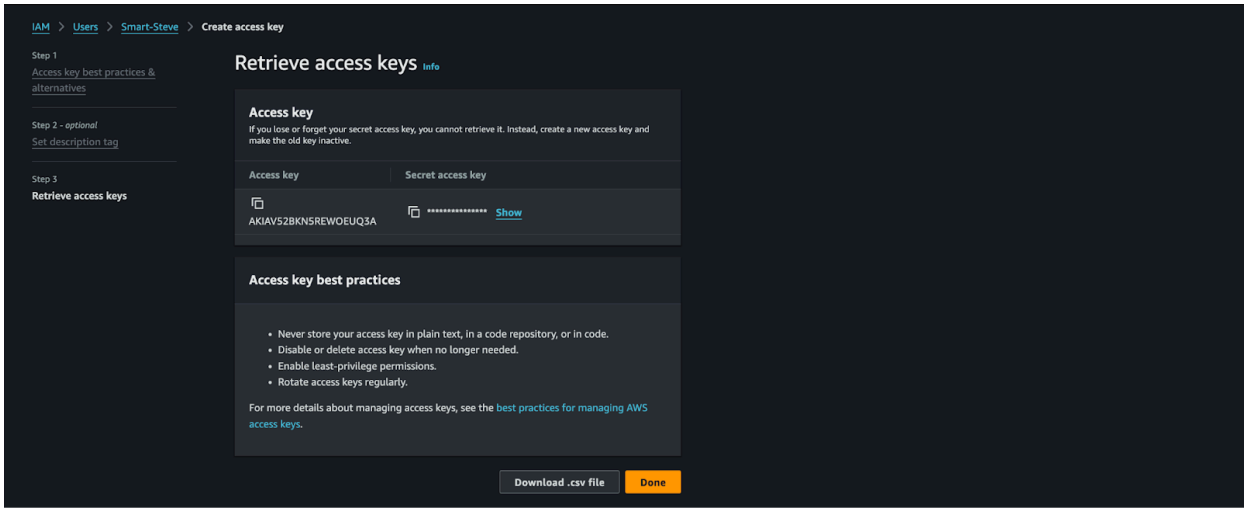
Click on Security credentials and then, click on Create access key to generate access keys.



Select CLI and tick the checkbox then click on Next.



Here are your credentials and keep it somewhere.



Now, go back to your EKS-Jump-Server and configure aws with a different profile name.

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ aws configure --profile steve
AWS Access Key ID [None]: AKIAV52BKN5REWOEUQ3A
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, you have to update the kubeconfig context as we are using the same server with different users we need to provide the profile name which is Steve in the current demonstration

```
aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2 --profile steve
```

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2 --profile steve
Updated context arn:aws:eks:us-east-2:407622020962:cluster/dev-medium-eks-cluster in /home/ubuntu/.kube/config
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, try to run kubectl commands by creating one sample deployment

```
kubectl create deployment Kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 -n dev
```

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 -n dev
deployment.apps/kubernetes-bootcamp created
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

I am trying to list down the pods within the dev namespace and we can do that which means we have permission for it.

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get pods -n dev
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-644c5687f4-js47n   1/1     Running   0          2m46s
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, let's try to delete the deployment and we should be able to do this
Yes, we can delete the deployment

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get deploy -n dev
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           4m21s
ubuntu@ip-10-16-45-162:~/Smart-Steve$
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl delete deploy kubernetes-bootcamp -n dev
deployment.apps "kubernetes-bootcamp" deleted
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, let's try to delete the dev namespace and as per the role we should not be able to do this.
Yes, The user Smart-Steve is not able to delete the namespace which is great.

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl delete ns dev
Error from server (Forbidden): namespaces "dev" is forbidden: User "Smart-Steve" cannot delete resource "namespaces" in API group "" in the namespace "dev"
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

So far, we have demonstrated to the user1**(Smart-Steve)** how RBAC works. But still, if you didn't understand you can continue this blog and see how we are going to configure other users with other required roles

Now, Let's move to the user **Elegant-Elon**
To configure the RBAC for user Elegant-Elon, we need to go back to our Admin user which is Adorable-Admin as it has full access to the EKS cluster

```
aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2
```

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2
Updated context arn:aws:eks:us-east-2:407622020962:cluster/dev-medium-eks-cluster in /home/ubuntu/.kube/config
ubuntu@ip-10-16-45-162:~/Smart-Steve$
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get ns
NAME              STATUS   AGE
default           Active   140m
dev               Active   64m
kube-node-lease   Active   140m
kube-public       Active   140m
kube-system       Active   140m
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, we have to create a role for user Elegant-Elon where it should have qa namespace full access and read-only access for the dev namespace
To do that, make sure you have created qa namespace

```
kubectl create ns qa
```

```
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl create ns qa
namespace/qa created
ubuntu@ip-10-16-45-162:~/Smart-Steve$ kubectl get ns
NAME              STATUS   AGE
default           Active   142m
dev               Active   66m
kube-node-lease   Active   142m
kube-public       Active   142m
kube-system       Active   142m
qa                Active   6s
ubuntu@ip-10-16-45-162:~/Smart-Steve$
```

Now, we have to configure Roles for two namespaces. So, either we need to use **ClusterRole** which is not bound to any specific namespace or **Role** which binds to a specific namespace such as dev or qa.

So, we are going to use Simple Role & RoleBinding as we need to give access to namespace-based

If you understand the above concept then we need to understand one more concept that lies under Role & RoleBinding objects

Now, we have a user named **Elegant-Elon** which needs two types of access where first one is dev namespace **dev-read-only** access and the second one is **qa-full-access**.

Create a dedicated directory for **Elegant-Elon** user.

```
ubuntu@ip-10-16-45-162:~$ mkdir Elegant-Elon
ubuntu@ip-10-16-45-162:~$ cd Elegant-Elon/
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Now, we will create two Role & RoleBinding each in both namespaces.

Let's start with **dev-read-only** Role and RoleBinding

```
# Role
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: dev
 name: dev-read-access-only
rules:
 - apiGroups: [""]
   resources: ["pods", "services", "configmaps", "secrets",
"persistentvolumeclaims"]
   verbs: ["get", "list"]
 - apiGroups: ["apps"]
   resources: ["deployments", "replicasets", "statefulsets"]
   verbs: ["get", "list"]
```

```yaml
# RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: dev-user-read-access-only
 namespace: dev
subjects:
 - kind: User
   name: Elegant-Elon
   apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: Role
 name: dev-read-access-only
 apiGroup: rbac.authorization.k8s.io
```

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ vim dev-role.yaml
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ vim dev-rolebinding.yaml
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ cat dev-role.yaml && cat dev-rolebinding.yaml
# Role
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-read-access-only
rules:
  - apiGroups: [""]
    resources: ["pods", "services", "configmaps", "secrets", "persistentvolumeclaims"]
    verbs: ["get", "list"]
  - apiGroups: ["apps"]
    resources: ["deployments", "replicasets", "statefulsets"]
    verbs: ["get", "list"]
# RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-user-read-access-only
  namespace: dev
subjects:
  - kind: User
    name: Elegant-Elon
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: dev-read-access-only
  apiGroup: rbac.authorization.k8s.io
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Now, we will apply both the role and rolebinding for dev namespace

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl apply -f dev-role.yaml
role.rbac.authorization.k8s.io/dev-read-access-only created
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl apply -f dev-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/dev-user-read-access-only created
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get role -n dev
NAME                 CREATED AT
dev-full-access      2024-11-03T04:15:01Z
dev-read-access-only 2024-11-03T06:23:37Z
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get rolebinding -n dev
NAME                        ROLE                       AGE
dev-user-full-access        Role/dev-full-access       105m
dev-user-read-access-only   Role/dev-read-access-only  19s
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Let’s forward with **qa-full-access** Role and RoleBinding

```yaml
# Roles
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: qa
 name: qa-full-access
rules:
 - apiGroups: [""]
   resources: ["pods", "services", "configmaps", "secrets",
"persistentvolumeclaims"]
   verbs: ["get", "list", "create", "update", "delete"]
 - apiGroups: ["apps"]
   resources: ["deployments", "replicasets", "statefulsets"]
   verbs: ["get", "list", "create", "update", "delete"]
```

```yaml
# RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: qa-user-full-access
 namespace: qa
subjects:
 - kind: User
   name: Elegant-Elon
   apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: Role
 name: qa-full-access
 apiGroup: rbac.authorization.k8s.io
```

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ vim qa-role.yaml
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ vim qa-rolebinding.yaml
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ vim qa-rolebinding.yaml
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ cat qa-role.yaml  && cat qa-rolebinding.yaml
# Roles
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: qa
  name: qa-full-access
rules:
  - apiGroups: [""]
    resources: ["pods", "services", "configmaps", "secrets", "persistentvolumeclaims"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: ["apps"]
    resources: ["deployments", "replicasets", "statefulsets"]
    verbs: ["get", "list", "create", "update", "delete"]
# RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: qa-user-full-access
  namespace: qa
subjects:
  - kind: User
    name: Elegant-Elon
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: qa-full-access
  apiGroup: rbac.authorization.k8s.io
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Now, we will apply both the role and role binding for qa namespace.

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl apply -f qa-role.yaml
role.rbac.authorization.k8s.io/qa-full-access created
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl apply -f qa-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/qa-user-full-access created
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubect get role -n qa
Command 'kubect' not found, did you mean:
  command 'kubectx' from deb kubectx (0.9.5-1ubuntu0.2)
Try: sudo apt install <deb name>
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get role -n qa
NAME             CREATED AT
qa-full-access   2024-11-03T06:30:12Z
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get rolebinding -n qa
NAME                  ROLE                  AGE
qa-user-full-access   Role/qa-full-access   34s
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Now, you have to generate the AWS credentials for user **Elegant-Elon**, configure AWS with the different profile in EKS-Jump-Server and set the kubeconfig context with the same profile

```
aws configure --profile elon
aws eks update-kubeconfig --name dev-medium-eks-cluster --region
us-east-2 --profile elon
```

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ aws configure --profile elon
AWS Access Key ID [None]: AKIAV52BKN5RHYUZDDIF
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2 --profile elon
Updated context arn:aws:eks:us-east-2:407622020962:cluster/dev-medium-eks-cluster in /home/ubuntu/.kube/config
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Now, try to run kubectl commands by creating one sample deployment.

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get ns
Error from server (Forbidden): namespaces is forbidden: User "Elegant-Elon" cannot list resource "namespaces" in API group "" at the cluster scope
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get pod -n dev
No resources found in dev namespace.
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get pod -n qa
No resources found in qa namespace.
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Also, you can check whether you have permissions for particular resources in Kubernetes or not by running the below command.
If you get no, it means you don't have permission for the particular Kubernetes resource.

```
kubectl auth can-i <action> <resource> -n <namespace>
```

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl auth can-i get nodes
Warning: resource 'nodes' is not namespace scoped

no
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl auth can-i get pods
no
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl auth can-i get pods -n dev
yes
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl auth can-i get cm -n qa
yes
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

Let's try to deploy sample applications in both namespaces. According to our requirement, the user can't have access to do anything except read in the dev namespace and in the qa namespace, it should have full access.
P.S: Our Role is working as expected

```
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 -n qa
deployment.apps/kubernetes-bootcamp created
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl get pods -n qa
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-644c5687f4-pp9pf   1/1     Running   0          8s
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
ubuntu@ip-10-16-45-162:~/Elegant-Elon$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 -n dev
error: failed to create deployment: deployments.apps is forbidden: User "Elegant-Elon" cannot create resource "deployments" in API group "apps" in the namespace "dev"
ubuntu@ip-10-16-45-162:~/Elegant-Elon$
```

The demonstration is completed for User2(Elegant-Elon).
Now, we will head toward the last demonstration where User3(Cool-Christina) will have only access related to the node as she will be handling EKS in maintenance work.

Create a dedicated directory for User **Cool-Christina** as we will write ClusterRole and ClusterRoleBinding in it.

```
ubuntu@ip-10-16-45-162:~$ mkdir Cool-Christina
ubuntu@ip-10-16-45-162:~$ cd Cool-Christina/
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

You can check the ClusterRole and ClusterRoleBinding where we provide the nodes-related permission such as list or get to user **Cool-Christina**

```
# Cluster Role
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: list-nodes
rules:
 - apiGroups: [""]
   resources: ["nodes"]
   verbs: ["list", "get"]
```

```
# ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: list-nodes-binding
subjects:
 - kind: User
   name: Cool-Christina
   apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: ClusterRole
 name: list-nodes
 apiGroup: rbac.authorization.k8s.io
```

```
ubuntu@ip-10-16-45-162:~$ mkdir Cool-Christina
ubuntu@ip-10-16-45-162:~$ cd Cool-Christina/
ubuntu@ip-10-16-45-162:~/Cool-Christina$ vim clusterrole.yaml
ubuntu@ip-10-16-45-162:~/Cool-Christina$ vim clusterrolebinding.yaml
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ cat clusterrole.yaml && cat clusterrolebinding.yaml
# Cluster Role
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: list-nodes
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["list", "get"]
# ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: list-nodes-binding
subjects:
  - kind: User
    name: Attractive-Ajay
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: list-nodes
  apiGroup: rbac.authorization.k8s.io
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

Now, we will be going to apply both the configs

Note: Make sure you logged in as admin in Kubernetes to configure RBAC

```
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl apply -f clusterrole.yaml

clusterrole.rbac.authorization.k8s.io/list-nodes created
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl apply -f clusterrolebinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/list-nodes-binding created
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get clusterrole | grep list-nodes
list-nodes                                                        2024-11-03T07:05:52Z
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

Now, you have to generate the AWS credentials for user **Cool-Christina** and, configure aws with the different profile in EKS-Jump-Server and set the kubeconfig context with the same profile

```
aws configure --profile christina
aws eks update-kubeconfig --name dev-medium-eks-cluster --region
us-east-2 --profile christina
```

```
ubuntu@ip-10-16-45-162:~$ aws configure --profile christina
AWS Access Key ID [None]: AKIAV52BKN5RMSGO5XXX
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-2
Default output format [None]: json
ubuntu@ip-10-16-45-162:~$
ubuntu@ip-10-16-45-162:~$ aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-2 --profile christina
Updated context arn:aws:eks:us-east-2:407622020962:cluster/dev-medium-eks-cluster in /home/ubuntu/.kube/config
ubuntu@ip-10-16-45-162:~$
```

Now, try to run the command to list down the nodes.
As you can see, I can view my nodes which is expected.

```
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE     VERSION
ip-10-16-133-196.us-east-2.compute.internal   Ready    <none>   5h14m   v1.30.4-eks-a737599
ip-10-16-137-19.us-east-2.compute.internal    Ready    <none>   5h14m   v1.30.4-eks-a737599
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

Let’s try to list other resources except nodes such as pods or namespace.
As you can see, Kubernetes denied me listing both resources which is expected.

```
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get pods
Error from server (Forbidden): pods is forbidden: User "Cool-Christina" cannot list resource "pods" in API group "" in the namespace "default"
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get ns
Error from server (Forbidden): namespaces is forbidden: User "Cool-Christina" cannot list resource "namespaces" in API group "" at the cluster scope
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

You can describe the nodes as per the permission.

```
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE     VERSION
ip-10-16-133-196.us-east-2.compute.internal   Ready    <none>   5h15m   v1.30.4-eks-a737599
ip-10-16-137-19.us-east-2.compute.internal    Ready    <none>   5h15m   v1.30.4-eks-a737599
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl describe node ip-10-16-133-196.us-east-2.compute.internal
Name:               ip-10-16-133-196.us-east-2.compute.internal
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/instance-type=t3a.medium
                    beta.kubernetes.io/os=linux
                    eks.amazonaws.com/capacityType=ON_DEMAND
                    eks.amazonaws.com/nodegroup=dev-medium-eks-cluster-on-demand-nodes
                    eks.amazonaws.com/nodegroup-image=ami-05836d8573f341724
                    failure-domain.beta.kubernetes.io/region=us-east-2
                    failure-domain.beta.kubernetes.io/zone=us-east-2a
                    k8s.io/cloud-provider-aws=3a5276ffded8c9c28162ad247d515f6f
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=ip-10-16-133-196.us-east-2.compute.internal
                    kubernetes.io/os=linux
                    node.kubernetes.io/instance-type=t3a.medium
                    topology.ebs.csi.aws.com/zone=us-east-2a
                    topology.k8s.aws/zone-id=use2-az1
                    topology.kubernetes.io/region=us-east-2
                    topology.kubernetes.io/zone=us-east-2a
                    type=ondemand
Annotations:        alpha.kubernetes.io/provided-node-ip: 10.16.133.196
```

But we can’t delete the nodes as we did not provide the delete permission to user **Cool-Christina.**

```
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE     VERSION
ip-10-16-133-196.us-east-2.compute.internal   Ready    <none>   5h16m   v1.30.4-eks-a737599
ip-10-16-137-19.us-east-2.compute.internal    Ready    <none>   5h16m   v1.30.4-eks-a737599
ubuntu@ip-10-16-45-162:~/Cool-Christina$
ubuntu@ip-10-16-45-162:~/Cool-Christina$ kubectl delete node ip-10-16-133-196.us-east-2.compute.internal
Error from server (Forbidden): nodes "ip-10-16-133-196.us-east-2.compute.internal" is forbidden: User "Cool-Christina" cannot delete resource "nodes" in API group "" at the clus
ter scope
ubuntu@ip-10-16-45-162:~/Cool-Christina$
```

Conclusion

By the end of this guide, you should have a working EKS setup with RBAC configured for multiple users. You'll have set up roles, permissions, and bindings for users with varying access needs, applying best practices to balance security and usability on your Kubernetes cluster.

Follow for more

Join Discord Community: https://lnkd.in/dsEdxpst

Follow on GitHub: https://github.com/AmanPathak-DevOps/

Follow on LinkedIn- https://www.linkedin.com/in/aman-devops/