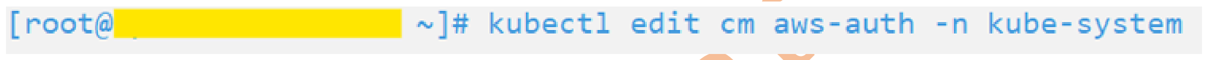
Kubernetes EKS RBAC with IAM

Kubernetes RBAC stands for Role Based Access Control. It is used when multiple teams have the access of Kubernetes cluster, in those scenarios a user will get the restricted access by the Kubernetes Administrator using RBAC.

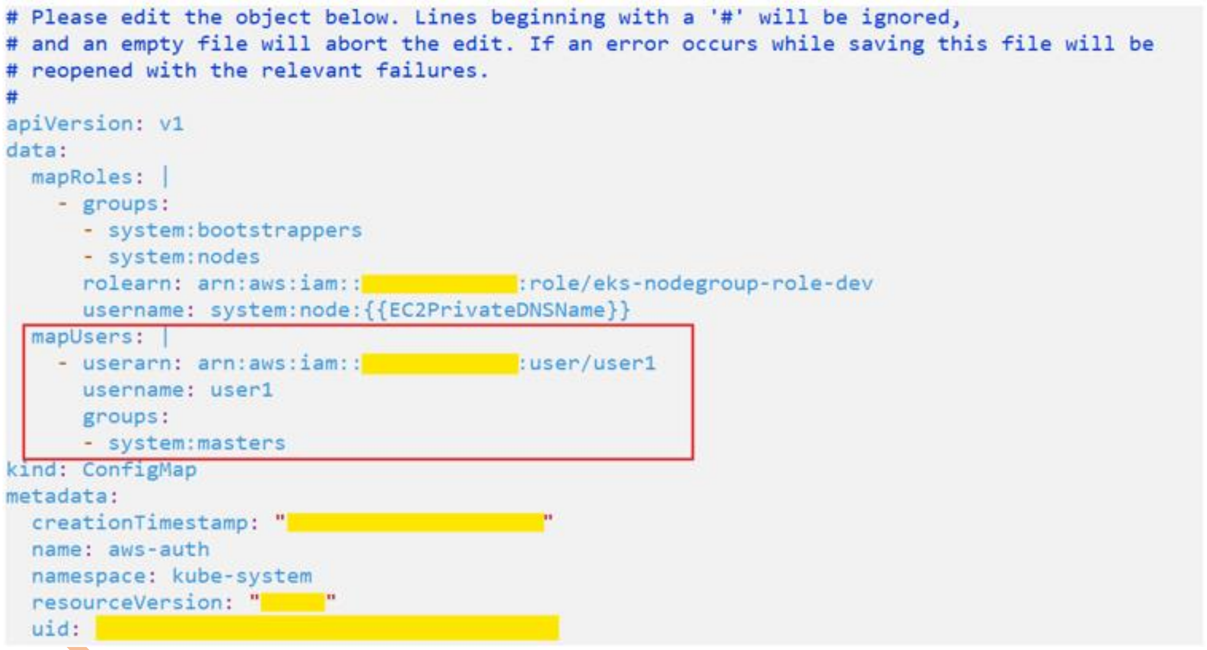
1. IAM user based RBAC in EKS

Here Kubernetes cluster has been created in AWS using EKS (Elastic Kubernetes Cluster), a managed Kubernetes Service. I have created two IAM users user1 and user2. No need to assign any IAM policy to these users as I have created these users for accessing only Kubernetes cluster. **User1 will be provided with administrator access** and **user2 will have pod reader access**. Generate Access Key and Secret Key ID for users user1 and user2. As per industry standard rotate Access Key and Secret access key ID after 90 days.

To list the configmap in kube-system namespace of EKS based Kubernetes cluster run the command **kubectl get cm -n kube-system.** After running this command, you will find a configmap with the name of aws-auth in the namespace kube-system. Now I have edited the configmap aws-auth for user1 which is present in the kube-system namespace as show in the screenshot attached below.



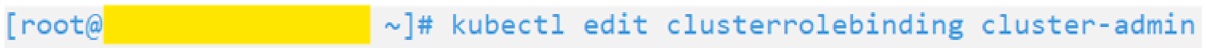
```
[root@[redacted] ~]# kubectl edit cm aws-auth -n kube-system
```



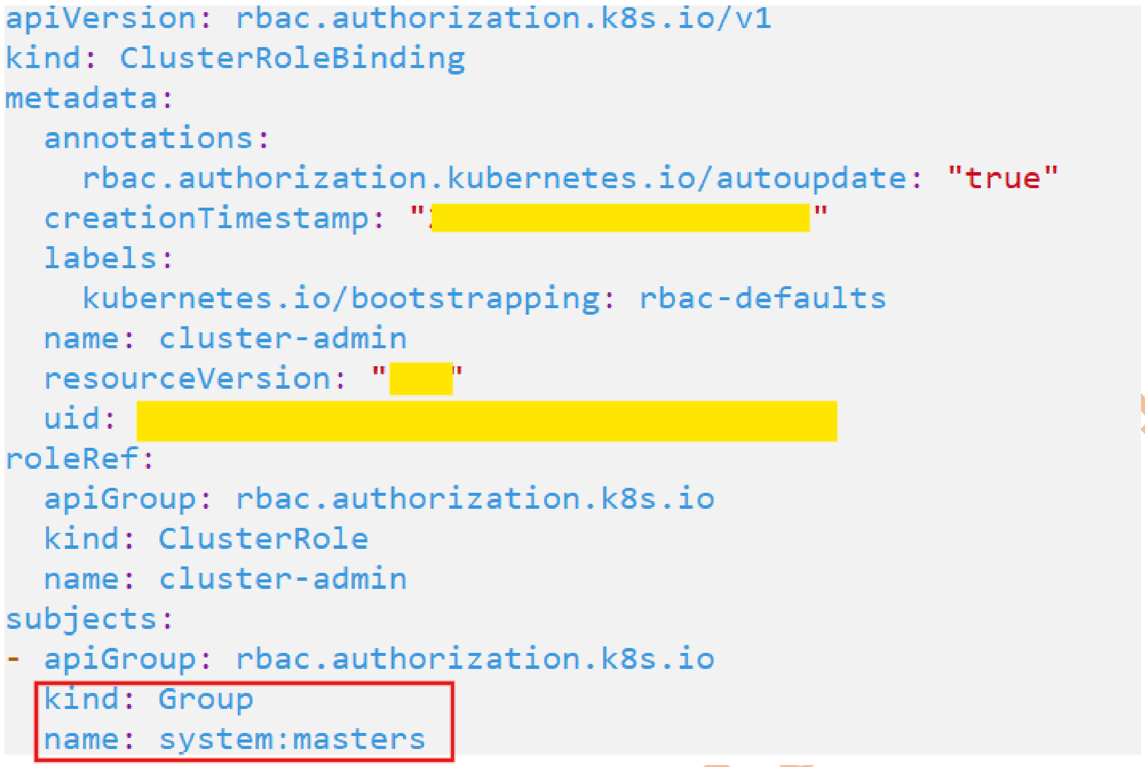
```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::[redacted]:user/user1
      username: user1
      groups:
      - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: [redacted]
```

As shown in the screenshot attached above to provide administrator privilege to user1 I have added the user user1 to the Kubernetes group **system:masters.** When a user will be added to the Kubernetes group system:masters they will get the administrator privilege.

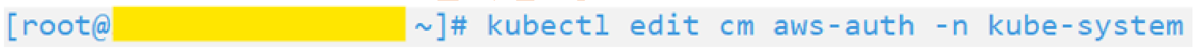
Check the cluster-admin ClusterRoleBinding, the cluster-admin ClusterRole will be added to the Kubernetes group system:masters as shown in the screenshot attached below.

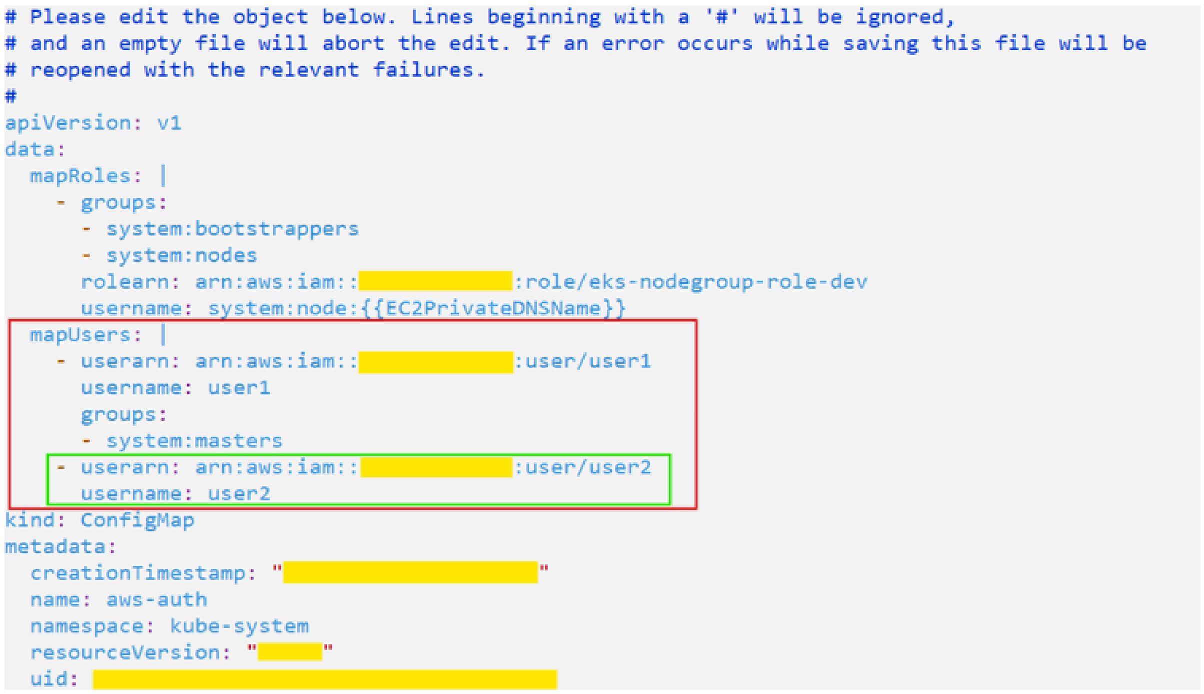


```
[root@[redacted] ~]# kubectl edit clusterrolebinding cluster-admin
```

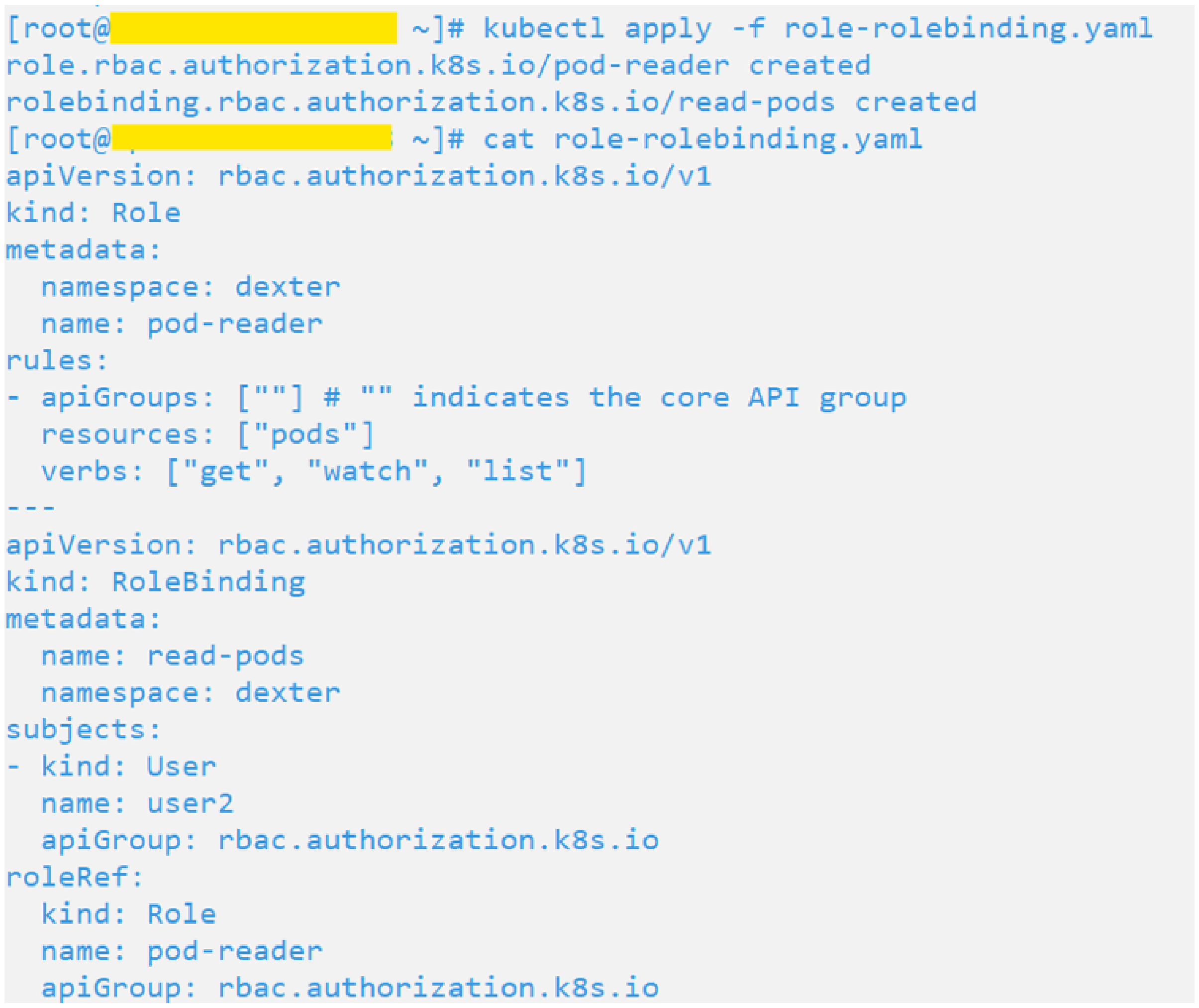

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: "[redacted]"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: cluster-admin
  resourceVersion: "[redacted]"
  uid: [redacted]
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:masters
```

As per the requirement now I am adding user2 to the configmap aws-auth. To do so I have edit the configmap aws-auth present in the namespace kube-system as shown in the screenshot attached below.


```
[root@[redacted] ~]# kubectl edit cm aws-auth -n kube-system
```


```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::[redacted]:user/user1
      username: user1
      groups:
      - system:masters
    - userarn: arn:aws:iam::[redacted]:user/user2
      username: user2
kind: ConfigMap
metadata:
  creationTimestamp: "[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: [redacted]
```

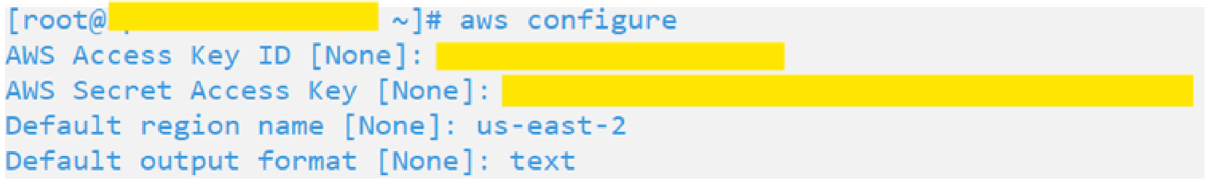
Created pod-reader Role and read-pods RoleBinding to provide pod-reader access to the user2 for the namespace dexter as shown in the screenshot attached below.



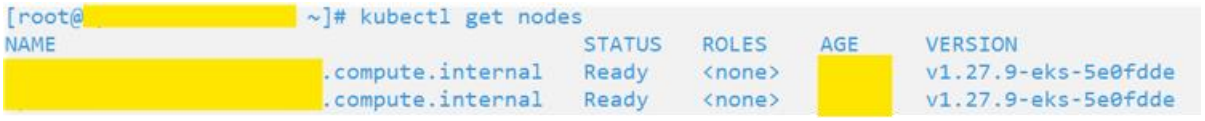
```
[root@[redacted] ~]# kubectl apply -f role-rolebinding.yaml
role.rbac.authorization.k8s.io/pod-reader created
rolebinding.rbac.authorization.k8s.io/read-pods created
[root@[redacted] ~]# cat role-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dexter
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: dexter
subjects:
- kind: User
  name: user2
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Now share the kubeconfig file present at the path ~/.kube/config to user1 and user2 along with their Access Key and Secret Key IDs. User1 and user2 logged-in on their own system and used his Access Key and Secret Key IDs after that they tried to access the Kubernetes cluster as shown in the screenshot attached below.

For User1

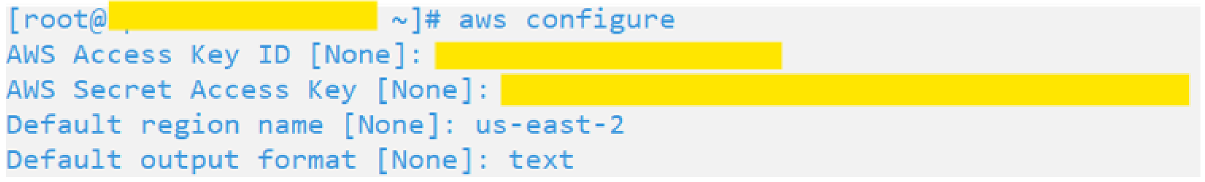


```
[root@[redacted] ~]# aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-2
Default output format [None]: text
```

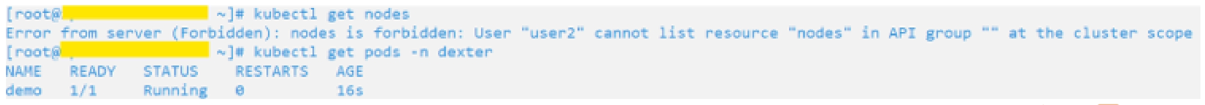


```
[root@[redacted] ~]# kubectl get nodes
NAME                                          STATUS   ROLES    AGE         VERSION
[redacted].compute.internal   Ready    <none>   [redacted]  v1.27.9-eks-5e0fdde
           .compute.internal   Ready    <none>              v1.27.9-eks-5e0fdde
```

For User2



```
[root@[redacted] ~]# 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]: text
```

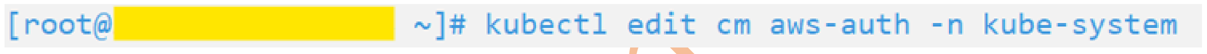


```
[root@[redacted] ~]# kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "user2" cannot list resource "nodes" in API group "" at the cluster scope
[root@[redacted] ~]# kubectl get pods -n dexter
NAME   READY   STATUS    RESTARTS   AGE
demo   1/1     Running   0          16s
```

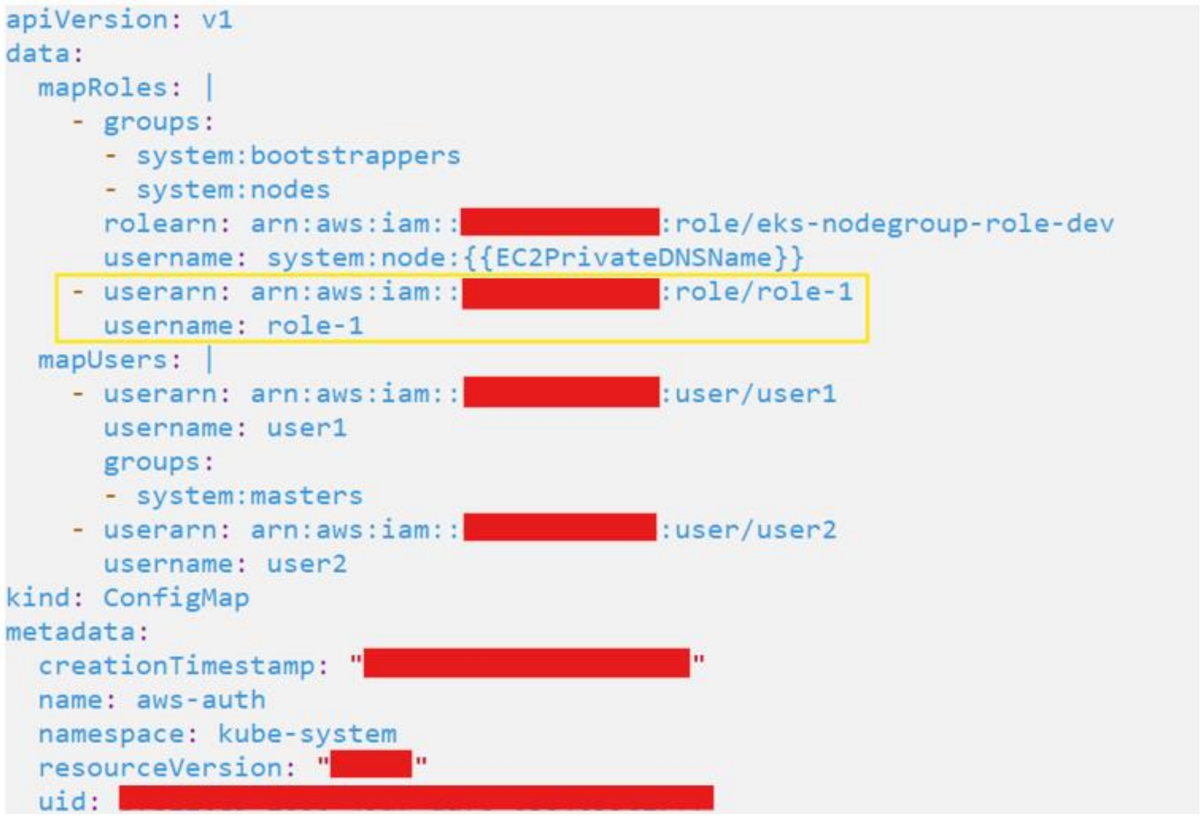
2. IAM Role based RBAC in EKS

For demonstration of IAM Role based RBAC in EKS I have created three IAM Roles named as role1, role2 and role3. I am providing full access to IAM Role role1 to the namespace mederma using Role and RoleBinding and providing administrator access to the IAM Role role2 using the Kubernetes group system:masters. Finally providing administrator access to the IAM Role role3 using the ClusterRole and ClusterRoleBinding as shown in the screenshot attached below.

For IAM Role role1



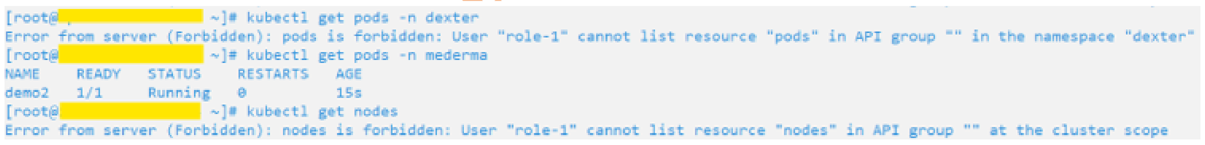
```
[root@[redacted] ~]# kubectl edit cm aws-auth -n kube-system
```



```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
    - userarn: arn:aws:iam::[redacted]:role/role-1
      username: role-1
  mapUsers: |
    - userarn: arn:aws:iam::[redacted]:user/user1
      username: user1
      groups:
      - system:masters
    - userarn: arn:aws:iam::[redacted]:user/user2
      username: user2
kind: ConfigMap
metadata:
  creationTimestamp: "[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: [redacted]
```

```
[root@[redacted] ~]# kubectl apply -f role-rolebinding2.yaml
role.rbac.authorization.k8s.io/full_access created
rolebinding.rbac.authorization.k8s.io/full_access_role_binding created
[root@[redacted] ~]# cat role-rolebinding2.yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: mederma
  name: full_access
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: full_access_role_binding
  namespace: mederma
subjects:
- kind: User
  name: role-1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: full_access
  apiGroup: rbac.authorization.k8s.io
```

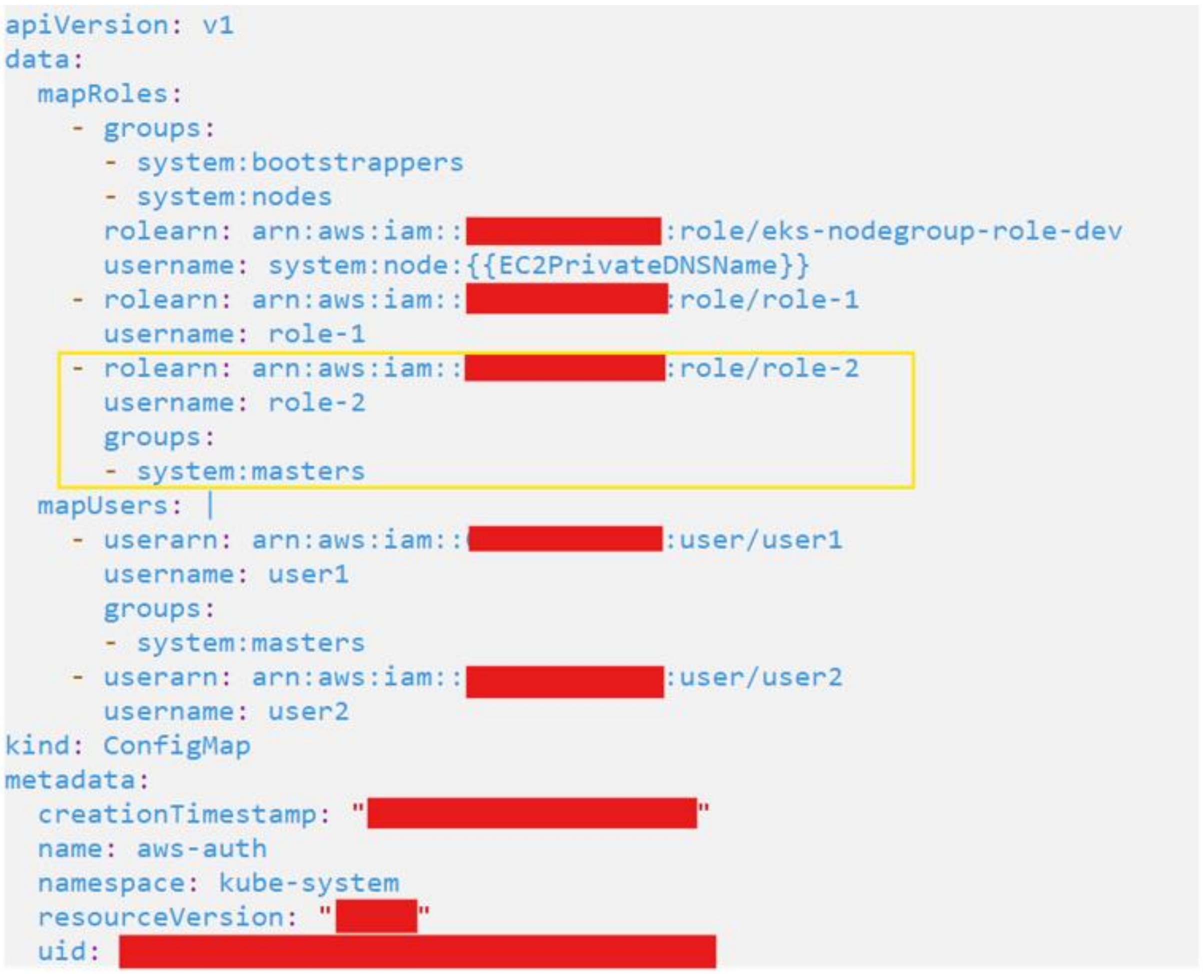
Now attach the IAM Role role1 to the EC2 Instance from specific users/team will access the Kubernetes cluster as shown in the screenshot attached below.



```
[root@[redacted] ~]# kubectl get pods -n dexter
Error from server (Forbidden): pods is forbidden: User "role-1" cannot list resource "pods" in API group "" in the namespace "dexter"
[root@[redacted] ~]# kubectl get pods -n mederma
NAME    READY   STATUS    RESTARTS   AGE
demo2   1/1     Running   0          15s
[root@[redacted] ~]# kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "role-1" cannot list resource "nodes" in API group "" at the cluster scope
```

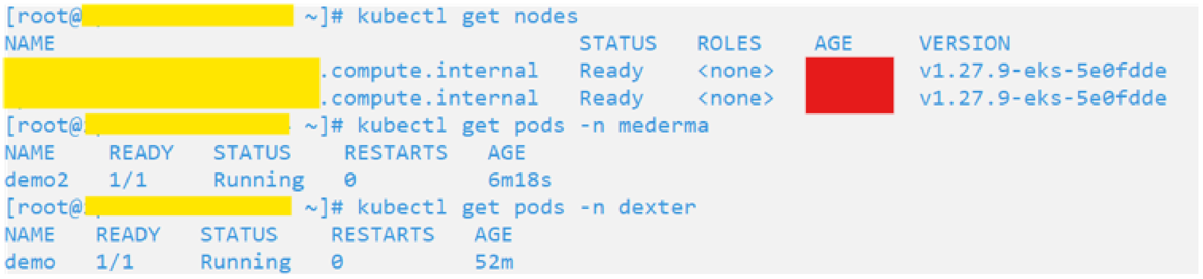
For IAM Role role2

```
[root@[redacted] ~]# kubectl edit cm aws-auth -n kube-system
```



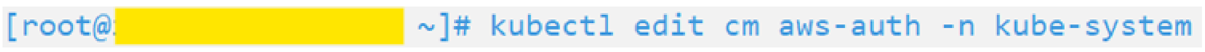
```
apiVersion: v1
data:
  mapRoles:
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
    - rolearn: arn:aws:iam::[redacted]:role/role-1
      username: role-1
    - rolearn: arn:aws:iam::[redacted]:role/role-2
      username: role-2
      groups:
      - system:masters
  mapUsers: |
    - userarn: arn:aws:iam::[redacted]:user/user1
      username: user1
      groups:
      - system:masters
    - userarn: arn:aws:iam::[redacted]:user/user2
      username: user2
kind: ConfigMap
metadata:
  creationTimestamp: "[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: [redacted]
```

Now Attach the IAM Role role2 to the EC2 Instance from which users/team wants to access the Kubernetes Cluster. They have the administrator privilege as shown in the screenshot attached below.

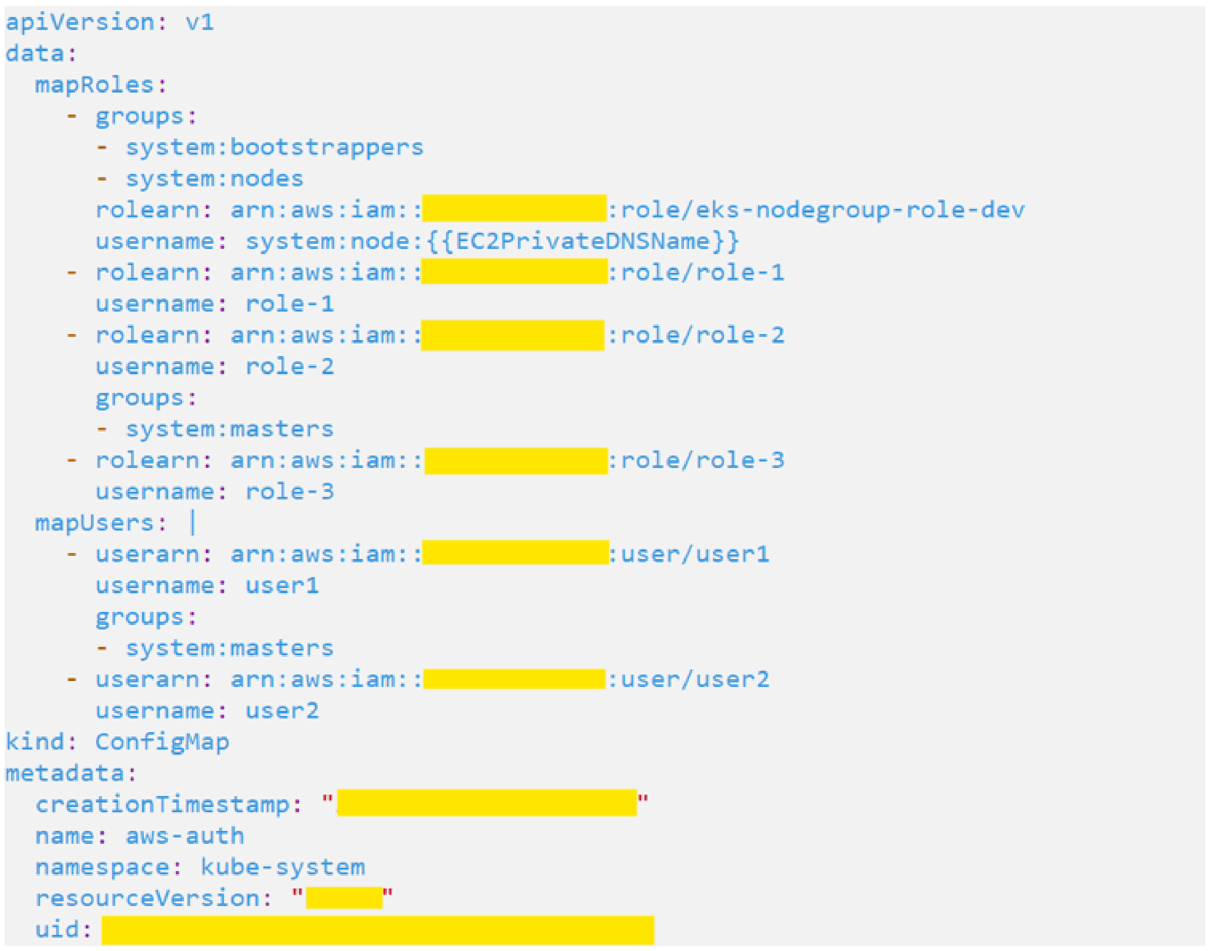


```
[root@[redacted] ~]# kubectl get nodes
NAME                                            STATUS   ROLES    AGE        VERSION
[redacted].compute.internal                     Ready    <none>   [redacted] v1.27.9-eks-5e0fdde
.compute.internal                               Ready    <none>              v1.27.9-eks-5e0fdde
[root@[redacted] ~]# kubectl get pods -n mederma
NAME     READY   STATUS    RESTARTS   AGE
demo2    1/1     Running   0          6m18s
[root@[redacted] ~]# kubectl get pods -n dexter
NAME    READY   STATUS    RESTARTS   AGE
demo    1/1     Running   0          52m
```

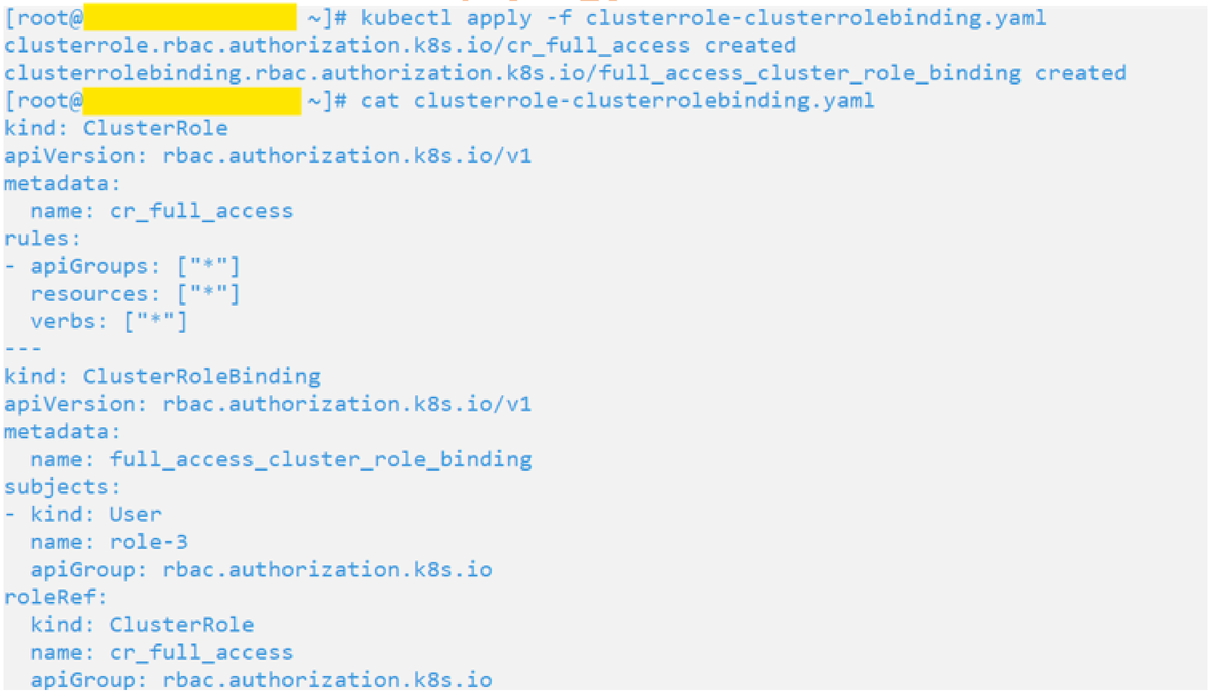
For IAM Role role3



```
[root@[redacted] ~]# kubectl edit cm aws-auth -n kube-system
```



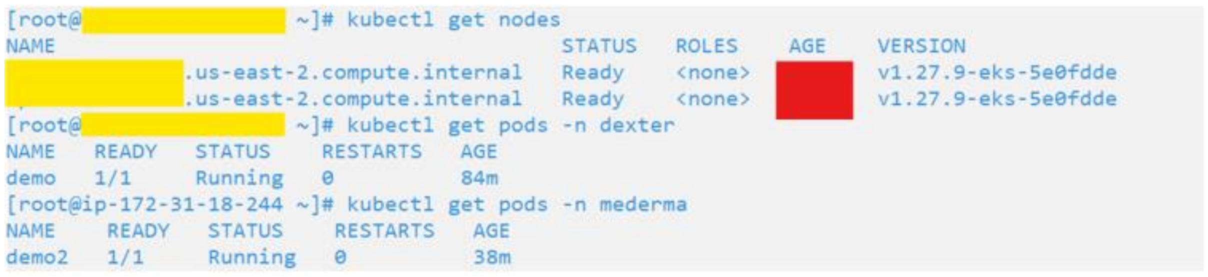
```
apiVersion: v1
data:
  mapRoles:
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::[redacted]:role/eks-nodegroup-role-dev
      username: system:node:{{EC2PrivateDNSName}}
    - rolearn: arn:aws:iam::[redacted]:role/role-1
      username: role-1
    - rolearn: arn:aws:iam::[redacted]:role/role-2
      username: role-2
      groups:
      - system:masters
    - rolearn: arn:aws:iam::[redacted]:role/role-3
      username: role-3
  mapUsers: |
    - userarn: arn:aws:iam::[redacted]:user/user1
      username: user1
      groups:
      - system:masters
    - userarn: arn:aws:iam::[redacted]:user/user2
      username: user2
kind: ConfigMap
metadata:
  creationTimestamp: "[redacted]"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "[redacted]"
  uid: [redacted]
```



```
[root@[redacted] ~]# kubectl apply -f clusterrole-clusterrolebinding.yaml
clusterrole.rbac.authorization.k8s.io/cr_full_access created
clusterrolebinding.rbac.authorization.k8s.io/full_access_cluster_role_binding created
[root@[redacted] ~]# cat clusterrole-clusterrolebinding.yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cr_full_access
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: full_access_cluster_role_binding
subjects:
- kind: User
  name: role-3
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cr_full_access
  apiGroup: rbac.authorization.k8s.io
```

Using the ClusterRole and ClusterRoleBinding (as shown in the screenshot attached above) the IAM Role role3 will get the administrator privilege.

Attach this IAM Role role3 to the EC2 Instance from which the users/team wants to access the Kubernetes Cluster as shown in the screenshot attached below.



```
[root@[redacted] ~]# kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
[redacted].us-east-2.compute.internal   Ready    <none>   [redacted]   v1.27.9-eks-5e0fdde
            .us-east-2.compute.internal   Ready    <none>         v1.27.9-eks-5e0fdde
[root@[redacted] ~]# kubectl get pods -n dexter
NAME   READY   STATUS    RESTARTS   AGE
demo   1/1     Running   0          84m
[root@ip-172-31-18-244 ~]# kubectl get pods -n mederma
NAME    READY   STATUS    RESTARTS   AGE
demo2   1/1     Running   0          38m
```