# Managing Cluster with AWS IAM authentication and Kubernetes RBAC authorization

* Kubernete RBAC
* Kubernetes Default Roles
* RoleBinding VS ClusterRoleBinding
* Amazon EKS
* Integrate IAM users to K8S RBAC

# Kubernetes RBAC

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise.

Kubernetes RBAC is an authorization and access control specification where you define the actions (GET, UPDATE, DELETE, etc) that Kubernetes subjects (i.e. users, applications, kubelets) are allowed to perform over Kubernetes entities (i.e. pods, secrets, nodes).

A Role can only be used to grant access to resources within a single namespace.

A **ClusterRole** can be used to grant the same permissions as a **Role**, but because they are cluster-scoped

# Kubernetes Default Roles

Roles intended to be granted cluster-wide using ClusterRoleBindings (**cluster-status**), and roles intended to be granted within particular namespaces using RoleBindings (**admin**, **edit**, **view**).

|  |  |  |
| --- | --- | --- |
| Default ClusterRole | Default ClusterRoleBinding | Description |
| **cluster-admin** | **system:masters** group | Allows super-user access to perform any action on any  Resource.  it gives full control over every resource in the  cluster and in all namespaces. |
| **admin** | None | Allows admin access, intended to be granted within a  Namespace.  It allows read/write access to most resources in a namespace,  It does not allow write access to resource quota or to the namespace itself. |
| **edit** | None | Allows read/write access to most objects in a namespace.  It does not allow viewing or modifying roles or rolebindings. |
| **view** | None | Allows read-only access to see most objects in a  namespace.  It does not allow viewing roles or rolebindings.  It does not allow viewing secrets. |

# RoleBinding VS ClusterRoleBinding

A role binding grants the role’s permission to a user or group. Permissions can be granted within a namespace or cluster-wide.

A **RoleBinding** grant the **Role** in the namespace. The following **RoleBinding** grants the “view” role to the group “aiqweb-read” within the “aiqweb” namespace. This allows the user within ‘aiqweb-read” have view permission in the “aiqweb” namespace.

***# This role binding allows the kubernets group "aiqweb-read" to read pods in the "aiqweb" namespace.***

**kind: RoleBinding**

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

**metadata:**

**name: aiq-dev-view-rolebinding**

**namespace: aiqweb**

**subjects:**

**- kind: Group**

**name: aiqweb-read *# Name is case sensitive***

**apiGroup: rbac.authorization.k8s.io**

**roleRef:**

**kind: ClusterRole *#this must be Role or ClusterRole***

**name: view *# this must match the name of the Role or ClusterRole you wish to bind to***

**apiGroup: rbac.authorization.k8s.io**

A **ClusterRoleBinding** grant permission at the cluster level and in all namespaces. The following **ClusterRoleBinding** allows the user in the group “ops-cluster-admin” to have full controls in any namespace.

***# This cluster role binding allows anyone in the "ops" group to have full control in any namespace.***

**kind: ClusterRoleBinding**

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

**metadata:**

**name: ops-cluster-admin**

**subjects:**

**- kind: Group**

**name: ops-cluster-admin *# Name is case sensitive***

**apiGroup: rbac.authorization.k8s.io**

**roleRef:**

**kind: ClusterRole**

**name: cluster-admin**

**apiGroup: rbac.authorization.k8s.io**

### kubectl create rolebinding

### Grants a Role or ClusterRole within a specific namespace.

* Grant the admin ClusterRole to the group named “aiqweb-admin” in the namespace “aiqweb”:

kubectl create rolebinding aiqweb-dev-admin-rolebinding –-namespace=aiqweb -–group=aiqweb-admin –clusterrole=admin

* Grant the view ClusterRole to the group named “aiqvision-read” in the namespace “aiqvision”:

kubectl create rolebinding aiqweb-dev-view-rolebinding --clusterrole=view --group =aiqvision-view --namespace=aiqvision

### kubectl create clusterrolebinding

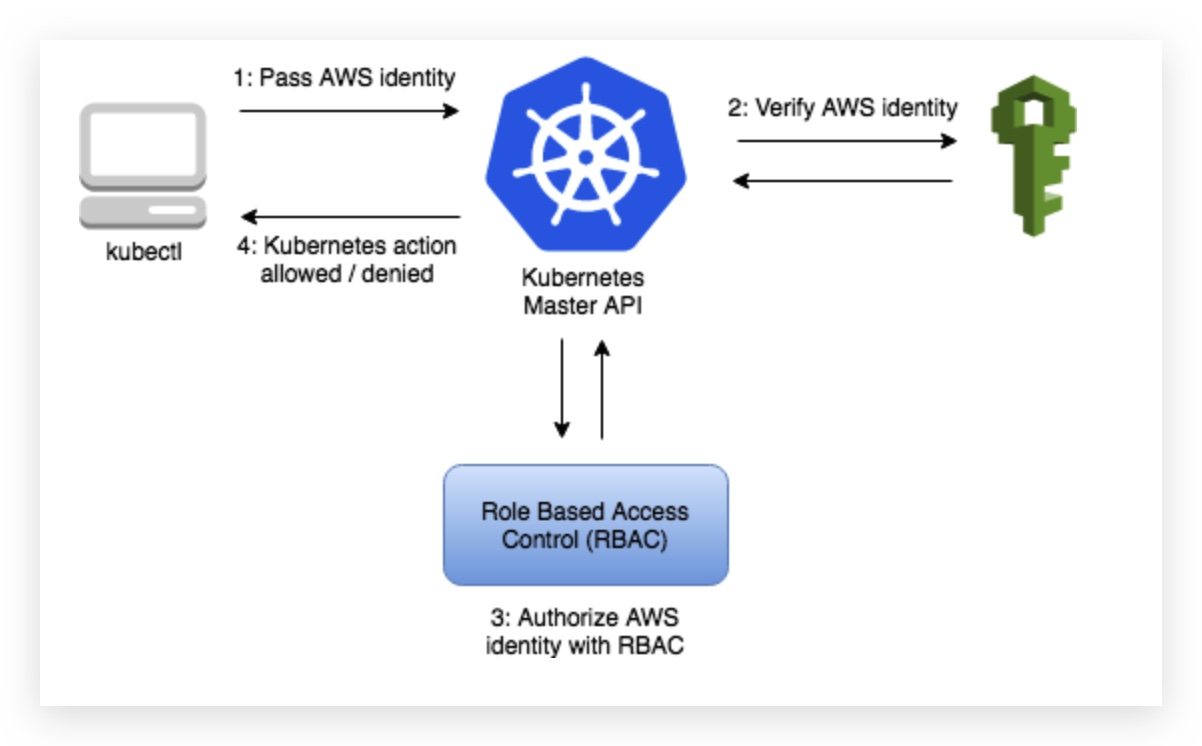
### Grants a ClusterRole across the entire cluster, including all namespaces

* Grant the cluster-admin ClusterRole to the group named “ops” across the entire cluster:

kubectl create clusterrolebinding ops-cluster-admin-cluster-rolebinding --clusterrole=cluster-admin --group=ops

# Amazon EKS

Amazon EKS uses IAM to provide authentication to the Kubernetes cluster through AWS IAM Authenticator for Kubernetes, but it still relies on native Kubernetes RBAC for authorization. This means that IAM is only used for authentication of valid IAM entities. All permissions for interacting with your Amazon EKS cluster’s Kubernetes API is managed through the native Kubernetes RBAC system.



# Integrate IAM User with K8S RBAC

Assuming you have a cluster running and you want to add AWS IAM Authenticator for Kubernetes support, you need to:

1. Create an IAM role you'll use to identify users.
2. Run the Authenticator server as a DaemonSet. (Skipped it if you are running K8S on Amazon EKS)
3. Configure Roles.
4. Set up kubectl to use Authenticator tokens.

### 1. Create AWS IAM role

First, you must create one or more IAM roles that will be mapped to users/groups inside your Kubernetes cluster.

* Choose the "Role for cross-account access" / "Provide access between AWS accounts you own" option.
* Paste in your AWS account ID number
* Your role does not need any additional policies attached.

This will create an IAM role with no permissions that can be assumed by authorized users/roles in your account.

### 2. Server side setup for kops

If you want to turn on AWS IAM Authenticator, you can add this block to your cluster running Kubernetes 1.10 or newer:

authentication:

aws: {}

For example:

apiVersion: kops/v1alpha2

kind: Cluster

metadata:

name: cluster.example.com

spec:

authentication:

aws: {}

authorization:

rbac: {}

**3. Map IAM Role to RBAC**

Configmap example

|  |
| --- |
| *# 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: |  - rolearn: arn:aws:iam::*111122223333*:role/doc-test-worker-nodes-NodeInstanceRole-WDO5P42N3ETB  username: system:node:{{EC2PrivateDNSName}}  groups:  - system:bootstrappers  - system:nodes  mapUsers: |  - userarn: *arn:aws:iam::555555555555:user/admin*  username: *admin*  groups:  - *system:masters*  - userarn: *arn:aws:iam::111122223333:user/ops-user*  username: *ops-user*  groups:  - *system:masters*  kind: ConfigMap  metadata:  annotations:  kubectl.kubernetes.io/last-applied-configuration: |  {"apiVersion":"v1","data":{"mapRoles":"- rolearn: arn:aws:iam::*111122223333*:role/doc-test-worker-nodes-NodeInstanceRole-WDO5P42N3ETB\n username: system:node:{{EC2PrivateDNSName}}\n groups:\n - system:bootstrappers\n - system:nodes\n"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"aws-auth","namespace":"kube-system"}}  creationTimestamp: 2018-04-04T18:49:10Z  name: aws-auth  namespace: kube-system  resourceVersion: "780"  selfLink: /api/v1/namespaces/kube-system/configmaps/aws-auth  uid: dcc31de5-3838-11e8-af26-02e00430057c |

Add your IAM users, roles, or AWS accounts to the configMap.

* **To add an IAM user:**
  + **userarn**: The ARN of the IAM user to add.
  + **username**: The user name within Kubernetes to map to the IAM user arn.
  + **groups**: A list of groups within Kubernetes to which the user is mapped .
* **To add an IAM role:** 
  + **rolearn**: The ARN of the IAM role to add.
  + **username**: The user name within Kubernetes to map to the IAM role arn.
  + **groups**: A list of groups within Kubernetes to which the role is mapped.

**4. Client side setup**

First make sure you have the aws-iam-authenticator binary installed. To authenticate, run kubectl --kubeconfig /path/to/kubeconfig kubectl will exec the aws-iam-authenticator binary with the supplied params in your kubeconfig which will generate a token and pass it to the apiserver. The token is valid for 15 minutes (the shortest value AWS permits) and can be reused multiple times.

# [...]

users:

- name: kubernetes-admin

user:

exec:

apiVersion: client.authentication.k8s.io/v1alpha1

command: aws-iam-authenticator

args:

- "token"

- "-i"

- "CLUSTER\_ID"

- "-r"

- "ROLE\_ARN"

env:

- name: AWS\_PROFILE

value: <AWS\_PROFILE>

# no client certificate/key needed here!

For EKS:

Use the AWS CLI **update-kubeconfig** command to create or update your kubeconfig for your cluster.

|  |
| --- |
| **aws eks update-kubeconfig --name *cluster\_name*** |

# Troubleshooting

If your client fails with an error like could not get token: AccessDenied [...], you can try assuming the role with the AWS CLI directly:

# AWS CLI version of `aws-iam-authenticator token -r arn:aws:iam::ACCOUNT:role/ROLE`:

$ aws sts assume-role --role-arn arn:aws:iam::ACCOUNT:role/ROLE --role-session-name test

If that fails, there are a few possible problems to check for:

* Make sure your base AWS credentials are available in your shell (aws sts get-caller-identity can help troubleshoot this).
* Make sure the target role allows your source account access (in the role trust policy).
* Make sure your source principal (user/role/group) has an IAM policy that allows sts:AssumeRole for the target role.
* Make sure you don't have any explicit deny policies attached to your user, group, or in AWS Organizations that would prevent the sts:AssumeRole.

When you execute a kubectl command it does a REST call to Kubernetes’s API server and sends the token generated by aws-iam-authenticator in the Authentication header. When you kubectl auth can-i get pods, the client does more or less this:

|  |
| --- |
| cat ~/.kube/config.d/my-cluster \  | shyaml get-value 'clusters.0.cluster.certificate-authority-data' \  | base64 -D > ~/.kube/config.d/my-cluster-ca-data  # Generate the authentication token  TOKEN=`heptio-authenticator-aws token -i my-cluster | jq -r .status.token`  REQUEST='{"kind": "SelfSubjectAccessReview", "apiVersion": "authorization.k8s.io/v1", "spec":{"resourceAttributes":{"name":"pod","verb":"get"}}}'  echo "$REQUEST" \  | http --verify=~/.kube/config.d/my-cluster-ca-data \  post https://[REDACTED].[REDACTED].us-west-2.eks.amazonaws.com/apis/authorization.k8s.io/v1/selfsubjectaccessreviews \  Authorization:"Bearer $TOKEN"  HTTP/1.1 201 Created  Content-Length: 196  Content-Type: application/json  Date: Wed, 11 Jul 2018 01:21:27 GMT    {  [...]  "status": {  "allowed": true  }  } |

On the server side Kubernetes passes the token to a webhook to the aws-iam-authenticator process running on EKS host; if all goes well the authenticator returns a “normal user” identity consisting ofusername (a string) and groups (a list of strings containing group names). If that works, that is when authentication ends.

If it fails then it means that you attempted to execute the request using an IAM User or Role that the aws-iam-authenticator running on your cluster does not know how to map to a “normal user” in Kubernetes space.

There are essentially two cases here:

**The IAM User or Role that was used to create the cluster** is hardwired in aws-iam-authenticatorconfiguration to map to a “normal user” who belongs to the system:masters group. This part is not visible anywhere in data you can reach via kubectl and aws commands, but that also means you can’t modify it.

For **other IAM Users or Roles** you must configure aws-iam-authenticator by setting the aws-authconfigmap.

There is nothing EKS-specific in authorization process; in particular, any permissions or policies attached to IAM Users don’t mean anything, you’re talking directly to Kubernetes API server, and it uses IAM only for authentication.

EKS clusters run in RBAC mode, meaning you grant permissions by binding roles to identities.