Amazon EKS Cluster Access Management Controls

**[Access Methods for Amazon EKS cluster](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "access-methods-for-amazon-eks-cluster)**

There are 2 types of identities to allow access to Amazon EKS cluster:

1. **An AWS Identity and Access Management (IAM) principal (role or user)**: it requires authentication to IAM. The IAM Identity
   * **can be assigned** to Kubernetes permissions to work with Kubernetes objects on cluster. For this, there are 2 methods:
     1. **access entries**: Use access entries to manage the Kubernetes permissions of IAM principals from outside the cluster. You can add and manage access to the cluster by using same tools that you created the cluster with.
     2. **aws-auth ConfigMap**: Use aws-auth ConfigMap to manage the Kubernetes permissions of IAM principals from inside the cluster. You can't migrate entries that Amazon EKS added to the ConfigMap however, such as entries for IAM roles used with managed node groups or Fargate profiles
   * **can be assigned** to IAM permissions to work with Amazon EKS cluster and its resources using the Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console, or eksctl.

Nodes join cluster by assuming an IAM role. The ability to access cluster using IAM principals is provided by the [AWS IAM Authenticator](https://github.com/kubernetes-sigs/aws-iam-authenticator#readme) for Kubernetes, which runs on the Amazon EKS control plane.

1. **A user in your own OpenID Connect (OIDC) provider**: It requires authentication to OIDC provider. The OIDC Identity
   * **can be assigned** to Kubernetes permissions to work with Kubernetes objects on cluster.
   * **can't be assigned** to IAM permissions so that they can work with your Amazon EKS cluster and its resources using the Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console, or eksctl.

You can use both types of identities with cluster. Users need to configure their kubectl config file to access Kubernetes objects on cluster.

**[Introduction to Amazon EKS Cluster IAM Access Management Controls](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "introduction-to-amazon-eks-cluster-iam-access-management-controls)**

Amazon EKS [supports simplified configuration of AWS IAM users and roles](https://aws.amazon.com/about-aws/whats-new/2023/12/amazon-eks-controls-iam-cluster-access-management/) with Kubernetes clusters, through a new set of APIs that tightly integrate IAM identities with Kubernetes authentication and authorization controls.

EKS already supports IAM identity authentication to Kubernetes clusters. This integration enables administrators to leverage IAM security features such as audit logging and multi-factor authentication. EKS access management controls simplify the process of mapping IAM to Kubernetes identities, by allowing administrators to fully define authorized IAM principals and their associated Kubernetes permissions directly through an EKS API during or after cluster creation.

The IAM identity used to create a EKS cluster can have its Kubernetes permissions removed or scoped down to comply with security requirements, and control of a cluster can always be restored to an AWS account administrator. Other AWS services can use EKS access management controls to automatically obtain permissions to run applications on EKS clusters. EKS access management controls simplify the amount of work administrators need to do in order to create and manage clusters that are shared by multiple users and other AWS services.

EKS cluster administrators can now grant AWS IAM principals access to all supported versions (v1.23 and beyond) of Amazon EKS clusters and Kubernetes objects directly through Amazon EKS APIs.

EKS access management controls are supported in all AWS regions for newly created clusters using Kubernetes version 1.23 or later. Existing clusters need to be updated to a supported EKS platform version before using this feature. For more details, visit the [EKS documentation](https://docs.aws.amazon.com/eks/latest/userguide/access-entries.html).

**[How does EKS Cluster IAM Access Management Controls work?](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "how-does-eks-cluster-iam-access-management-controls-work)**

This functionality relies on two new concepts:

1. **access entries**: An access entry is a cluster identity and is directly linked to an AWS IAM principal user or role, that is used to authenticate to an Amazon EKS cluster.
2. **access policies**: Access policies are Amazon EKS-specific policies that authorizes an access entry to perform specific cluster actions. At launch, Amazon EKS supports only predefined AWS managed policies. Access policies are not AWS IAM entities and are defined and managed by Amazon EKS. Amazon EKS access policies include permission sets that support common use cases of administration, editing, or read-only access to Kubernetes resources.

Run the following command and output provide an up-to-date list of supported access policies for managing cluster access.

aws eks list-access-policies

{

"accessPolicies": [

{

"name": "AmazonEKSAdminPolicy",

"arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"

},

{

"name": "AmazonEKSClusterAdminPolicy",

"arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

},

{

"name": "AmazonEKSEditPolicy",

"arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy"

},

{

"name": "AmazonEKSViewPolicy",

"arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"

}

]

}

The following Amazon EKS access policies are based on these [user-facing roles](https://kubernetes.io/docs/reference/access-authn-authz/rbac/#user-facing-roles) published in the Kubernetes documentation:

**AmazonEKSClusterAdminPolicy** – cluster-admin

**AmazonEKSAdminPolicy** – admin

**AmazonEKSEditPolicy** – edit

**AmazonEKSViewPolicy** – view

**[Cluster access management API](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "cluster-access-management-api)**

The new cluster access management API objects and commands allow administrators to define access management configurations either during cluster creation or later, using familiar infrastructure as code (IaC) tools such as [AWS CloudFormation](https://docs.aws.amazon.com/cloudformation/?icmpid=docs_homepage_mgmtgov), [Terraform](https://www.terraform.io/), or the [AWS Cloud Development Kit (CDK)](https://docs.aws.amazon.com/cdk/v2/guide/getting_started.html).

Cluster access management using the access entry API is an opt-in feature for Amazon EKS v1.23 and either new or existing clusters.

**[Kubernetes authorizers](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "kubernetes-authorizers)**

Kubernetes allows different AuthZ services (i.e. authorizers) chained together in a sequence to make AuthZ decisions about inbound API server requests. This allows custom AuthZ services to be used with the Kubernetes API server. The new feature allows to use upstream RBAC (Role-based access control) in combination with access policies. Both the upstream RBAC and Amazon EKS authorizer support allow and pass (but not deny) on AuthZ decisions. When creating an access entry with Kubernetes usernames or groups, the upstream RBAC evaluates and immediately returns a AuthZ decision upon an allow outcome. If the RBAC authorizer can’t determine the outcome, then it passes the decision to the Amazon EKS authorizer. If both authorizers pass, then a deny decision is returned.

With the cluster access management controls, only AWS IAM principals with the appropriate permissions can authorize other AWS IAM principals to access Amazon EKS clusters. Only the AWS IAM principal and the applied Amazon EKS access entry policies are used by the cluster access management authorizer. The following diagram illustrates the workflow.



**[Cluster authentication modes](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management" \l "cluster-authentication-modes)**

The authentication mode determines methods to allow IAM principals to access Kubernetes objects on cluster. There are 3 cluster authentication modes.

1. **CONFIG\_MAP**: aws-auth ConfigMap Only. This is the original authentication mode for Amazon EKS clusters. The IAM principal that created the cluster is the initial user that can access the cluster by using kubectl.
2. **API\_AND\_CONFIG\_MAP**: With this, you can use both methods (i.e. EKS API and ConfigMap) to add IAM principals to the cluster. Note that each method stores separate entries.
3. **API**: Access entries only. With this, you can use the EKS API, AWS Command Line Interface, AWS SDKs, AWS CloudFormation, and AWS Management Console to manage access to the cluster for IAM principals.

Each access entry has a type and you can use the combination of an access scope to limit the principal to a specific namespace and an access policy to set preconfigured reusable permissions policies. Alternatively, you can use the Standard type and Kubernetes RBAC groups to assign custom permissions.

Switching between Authentication Modes

**[Switching between Authentication Modes](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/1-switching-modes" \l "switching-between-authentication-modes)**

In this section, let us explore possible options to switch between different Authentication modes.

**[Prerequisites](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/1-switching-modes" \l "prerequisites)**

Before procedding further, ensure that the EKS cluster must have a [platform version that is the same or later than the version listed in the table, or a Kubernetes version that is later than the versions listed in the table](https://docs.aws.amazon.com/eks/latest/userguide/access-entries.html).

Check the platform version of the EKS cluster.

export EKS\_CLUSTER\_NAME="eksworkshop-eksctl"

aws eks describe-cluster --name $EKS\_CLUSTER\_NAME --query 'cluster.{"Kubernetes Version": version, "Platform Version": platformVersion}'

Check Output

{

"Kubernetes Version": "1.28",

"Platform Version": "eks.6"

}

**[Default Cluster authentication mode](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/1-switching-modes" \l "default-cluster-authentication-mode)**

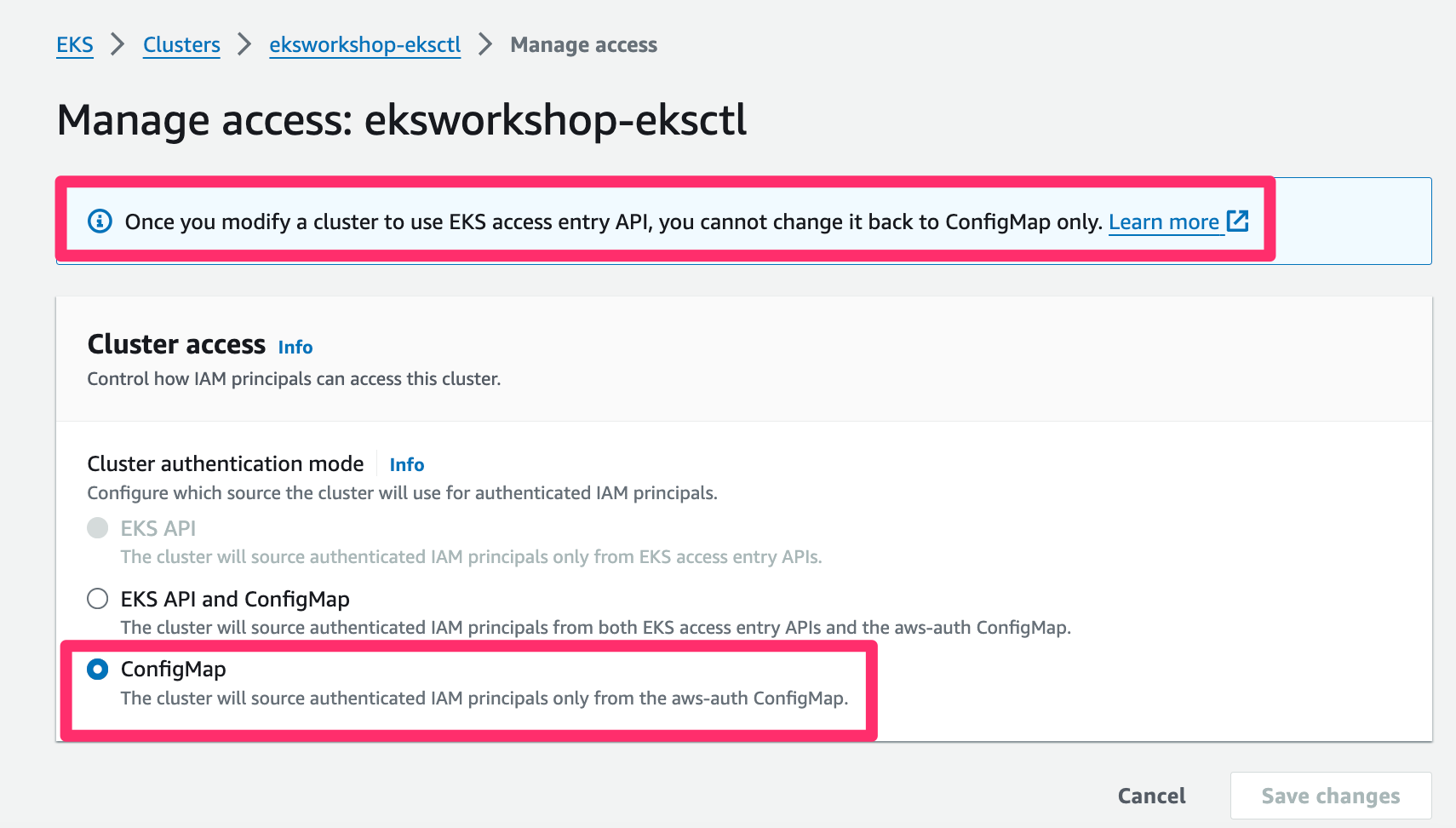
The default Cluster authentication is CONFIG\_MAP.

Run below command to get the current cluster Authentication mode.

aws eks describe-cluster --name $EKS\_CLUSTER\_NAME --query 'cluster.accessConfig'

Check Output

You can also see that in the EKS Console, under the **Access** Tab.



**[Configure API\_AND\_CONFIG\_MAP Mode](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/1-switching-modes" \l "configure-api_and_config_map-mode)**

Switching authentication modes on an existing cluster is a one-way operation. You can switch from CONFIG\_MAP to API\_AND\_CONFIG\_MAP but cannot directly switch to API mode. You can then switch from API\_AND\_CONFIG\_MAP to API. You **cannot** revert these operations in the opposite direction. Meaning you cannot switch back to CONFIG\_MAP or API\_AND\_CONFIG\_MAP from API. And you cannot switch back to CONFIG\_MAPfrom API\_AND\_CONFIG\_MAP.

Let's first ensure there are no access entries exists by default.

aws eks list-access-entries --cluster-name $EKS\_CLUSTER\_NAME

Check Output

An error occurred (InvalidRequestException) when calling the ListAccessEntries operation: The cluster's authentication mode must be set to one of [API, API\_AND\_CONFIG\_MAP] to perform this operation.

The error is expected since the EKS cluster is currently in the default mode i.e. CONFIG\_MAP

Let us now try to switch directly to API mode from CONFIG\_MAP.

export AUTHENTICATION\_MODE="API"

aws eks update-cluster-config \

--name $EKS\_CLUSTER\_NAME \

--access-config authenticationMode=$AUTHENTICATION\_MODE

Check Output

An error occurred (InvalidParameterException) when calling the UpdateClusterConfig operation: Unsupported authentication mode update from CONFIG\_MAP to API

The error is expected as explained before.

Let us also see content of the aws-auth ConfigMap.

k -n kube-system get cm aws-auth -oyaml

Check Output

apiVersion: v1

data:

mapRoles: |

- groups:

- system:bootstrappers

- system:nodes

rolearn: arn:aws:iam::ACCOUNT\_ID:role/eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa

username: system:node:{{EC2PrivateDNSName}}

kind: ConfigMap

metadata:

creationTimestamp: "2024-01-26T07:46:50Z"

name: aws-auth

namespace: kube-system

resourceVersion: "1400"

uid: 58b427de-975b-4b37-8b5e-e64e736ac4ed

Let us now try to switch to API\_AND\_CONFIG\_MAP mode from CONFIG\_MAP.

export AUTHENTICATION\_MODE="API\_AND\_CONFIG\_MAP"

aws eks update-cluster-config \

--name $EKS\_CLUSTER\_NAME \

--access-config authenticationMode=$AUTHENTICATION\_MODE

Check Output

{

"update": {

"id": "f710c23d-8ae9-4972-818a-76780287b18d",

"status": "InProgress",

"type": "AccessConfigUpdate",

"params": [

{

"type": "AuthenticationMode",

"value": "\"API\_AND\_CONFIG\_MAP\""

}

],

"createdAt": 1703060401.088,

"errors": []

}

}

It takes few minutes to change the Authentication mode.

Run below command to get the current cluster Authentication mode.

aws eks describe-cluster --name $EKS\_CLUSTER\_NAME --query 'cluster.accessConfig'

Check Output

{

"authenticationMode": "API\_AND\_CONFIG\_MAP"

}

Let us try switching the Authentication mode to CONFIG\_MAP

export AUTHENTICATION\_MODE="CONFIG\_MAP"

aws eks update-cluster-config \

--name $EKS\_CLUSTER\_NAME \

--access-config authenticationMode=$AUTHENTICATION\_MODE

Check Output

An error occurred (InvalidParameterException) when calling the UpdateClusterConfig operation: Unsupported authentication mode update from API\_AND\_CONFIG\_MAP to CONFIG\_MAP

The error is expected as explained before.

Let us again list if any access entries are automatically created when we changed to Authentication mode API\_AND\_CONFIG\_MAP

aws eks list-access-entries --cluster-name $EKS\_CLUSTER\_NAME

Check Output

{

"accessEntries": [

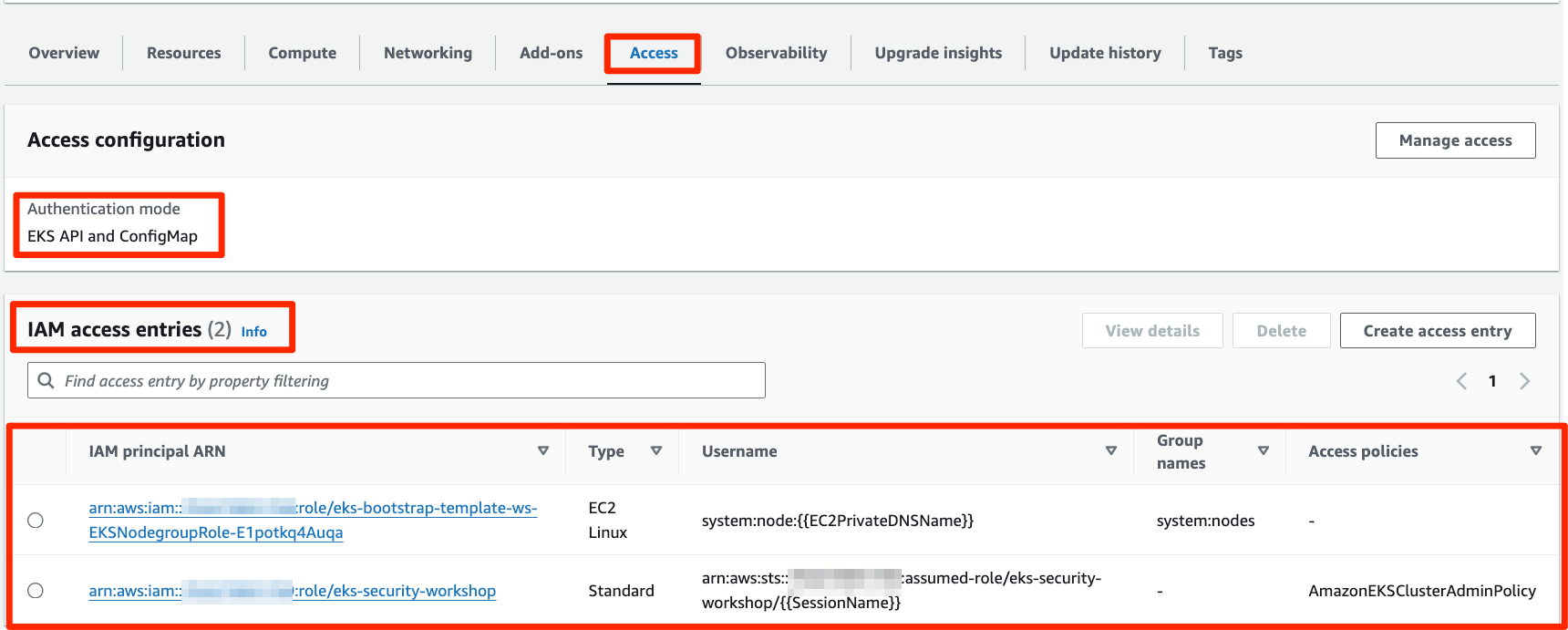
"arn:aws:iam::ACCOUNT\_ID:role/eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa",

"arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop"

]

}

You can also view these access entries in the EKS Console under the **Access** Tab.



Notice there are 2 access entries in the above output. The first one arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop is the IAM Role used to create the EKS cluster. Note this IAM Role cannot be seen anywhere in the default Authentication Mode CONFIG\_MAP

Run the below command to see the current IAM Role assigned to the Cloud9 EC2 Instance. This will be the same IAM Role as first access entry listed above.

aws sts get-caller-identity

Check Output

{

"UserId": "AROA26YVAA7X6WCNE47I3:i-0099e96934253a907",

"Account": "ACCOUNT\_ID",

"Arn": "arn:aws:sts::ACCOUNT\_ID:assumed-role/eks-security-workshop/i-0099e96934253a907"

}

Removing the default cluster administrator

**[Removing the default cluster administrator](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/2-remove-default-admin" \l "removing-the-default-cluster-administrator)**

**[View existing access policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/2-remove-default-admin" \l "view-existing-access-policy)**

Earlier, in the default Mode CONFIG\_MAP, when an Amazon EKS cluster was created, the principal used to provision the cluster was permanently granted Kubernetes cluster-admin privileges. It was not possible to remove Kubernetes cluster-admin privileges from this IAM principal.

Now, with authentication modes API\_AND\_CONFIG\_MAP and API, it is possible to remove the Kubernetes cluster-admin privileges from this IAM principal.

In this section, let us test removing Kubernetes cluster-admin privileges from the IAM principal and re-attaching it.

Let us first describe the access entry for the EKS cluster creation role eks-security-workshop and see what access policy is assigned to it.

export ACCOUNT\_ID=$(aws sts get-caller-identity --output text --query Account)

export ACCESS\_ENTRY="arn:aws:iam::$ACCOUNT\_ID:role/eks-security-workshop"

echo "ACCESS\_ENTRY=$ACCESS\_ENTRY"

Let us describe the access entry.

aws eks describe-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $ACCESS\_ENTRY

Check Output

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop",

"kubernetesGroups": [],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/eks-security-workshop/cec6a258-37a5-8a88-c509-d66bf99078c9",

"createdAt": "2024-01-26T07:33:50.099000+00:00",

"modifiedAt": "2024-01-26T07:33:50.099000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/eks-security-workshop/{{SessionName}}",

"type": "STANDARD"

}

}

Let us get the associated access policies for the above IAM principal in the above access entry.

export IAM\_PRINCIPAL\_ARN=$(aws eks describe-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $ACCESS\_ENTRY --query 'accessEntry.principalArn' --output text)

echo "IAM\_PRINCIPAL\_ARN=$IAM\_PRINCIPAL\_ARN"

export ACCESS\_POLICY\_ARN=$(aws eks list-associated-access-policies --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --query 'associatedAccessPolicies[0].policyArn' --output text)

echo "ACCESS\_POLICY\_ARN=$ACCESS\_POLICY\_ARN"

Check Output

IAM\_PRINCIPAL\_ARN=arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop

ACCESS\_POLICY\_ARN=arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy

As you see the IAM principal arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop is associated with access policy AmazonEKSClusterAdminPolicy which is mapped to Kubernetes cluster-admin role.

Before removing Kubernetes cluster-admin privileges from the IAM principal arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop, Let us list Kubernetes permissions for authenticated user.

Test cluster access now.

kubectl get node

Check Output

NAME STATUS ROLES AGE VERSION

ip-192-168-105-62.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

ip-192-168-158-255.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

ip-192-168-184-154.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

**[Remove the access policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/2-remove-default-admin" \l "remove-the-access-policy)**

Run the below command to disassociate the access policy from the IAM principal.

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --policy-arn $ACCESS\_POLICY\_ARN

Let us check the list of associated access policies for the IAM principal.

aws eks list-associated-access-policies --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN

Check Output

{

"associatedAccessPolicies": [],

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop"

}

As expected, the associatedAccessPolicies is empty.

**Note**

Notice that we only removed access policy from the IAM principal but the access entry still exists for this IAM principal.

Let us list the access entries avalable for the cluster.

aws eks list-access-entries --cluster-name $EKS\_CLUSTER\_NAME

Check Output

{

"accessEntries": [

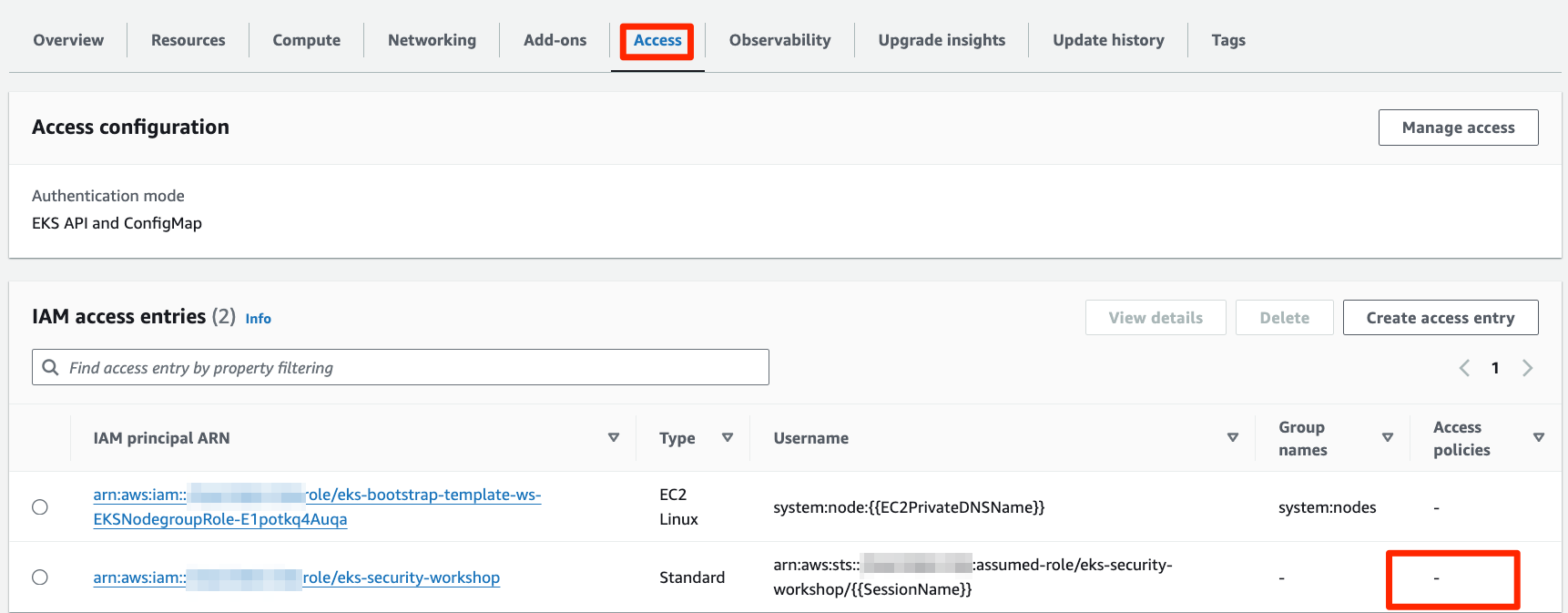
"arn:aws:iam::ACCOUNT\_ID:role/eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa",

"arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop"

]

}

You can also view these access entries in the EKS Console under the **Access** Tab.



Test cluster access now.

kubectl get node

Check Output

Error from server (Forbidden): nodes is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/eks-security-workshop/EKSGetTokenAuth" cannot list resource "nodes" in API group "" at the cluster scope

This indicates that the IAM principal, which originally created the EKS cluster does not have any Kubernetes permissions assigned now.

**[Add the access policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/2-remove-default-admin" \l "add-the-access-policy)**

Let us associate the access policy back to the IAM principal.

export ACCESS\_SCOPE="cluster"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --policy-arn $ACCESS\_POLICY\_ARN --access-scope type=$ACCESS\_SCOPE

Check Output

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy",

"accessScope": {

"type": "cluster",

"namespaces": []

},

"associatedAt": "2024-01-30T11:43:51.815000+00:00",

"modifiedAt": "2024-01-30T11:43:51.815000+00:00"

}

}

Let us check again for the list of associated access policies for the IAM principal.

aws eks list-associated-access-policies --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN

Check Output

{

"associatedAccessPolicies": [

{

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy",

"accessScope": {

"type": "cluster",

"namespaces": []

},

"associatedAt": "2024-01-30T11:43:51.815000+00:00",

"modifiedAt": "2024-01-30T11:43:51.815000+00:00"

}

],

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/eks-security-workshop"

}

Test cluster access now.

kubectl get node

Check Output

NAME STATUS ROLES AGE VERSION

ip-192-168-105-62.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

ip-192-168-158-255.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

ip-192-168-184-154.ec2.internal Ready <none> 172m v1.28.3-eks-e71965b

**Note**

EKS clusters can be created with the AWS IAM principal with no cluster administrator access permissions at all using bootstrapClusterCreatorAdminPermissions flag. For example, aws eks create-cluster --name CLUSTER\_NAME --role-arn CLUSTER\_ROLE\_ARN --resources-vpc-config subnetIds=value,securityGroupIds=value --access-config authenticationMode=API\_AND\_CONFIG\_MAP,bootstrapClusterCreatorAdminPermissions=false

Create Access entries and policies

In this section, let us create access entries and associate required access policies for different persona.

* Users from the IAM group k8sClusterAdmin are cluster administrators and needs the complete administrator access to the cluster
* Users from the IAM group k8sTeamADev are developers for the project A and needs the administrator access to their Namespace team-a
* Users from the IAM group k8sTeamATest are testers for the project A and needs the read-only access to their Namespace team-a

So, we will create the following IAM Roles in this section.

* **k8sClusterAdmin** role assumed by users in the IAM group k8sClusterAdmin and is associated with **AmazonEKSClusterAdminPolicy** Kubernetes permissions on the cluster
* **k8sTeamADev** role assumed by users in the IAM group k8sTeamADev and is associated with **AmazonEKSAdminPolicy** Kubernetes permissions on the Namespace team-a
* **k8sTeamATest** role assumed by users in the IAM group k8sTeamATest and is associated with **AmazonEKSViewPolicy** Kubernetes permissions on the Namespace team-a

Create AWS IAM Roles

Set below environment variables

1

2

export ACCOUNT\_ID=$(aws sts get-caller-identity --output text --query Account)

export AWS\_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')

Create the IAM Roles:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

POLICY=$(echo -n '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":{"AWS":"arn:aws:iam::'; echo -n "$ACCOUNT\_ID"; echo -n ':root"},"Action":"sts:AssumeRole","Condition":{}}]}')

export IAM\_ROLE="k8sClusterAdmin"

export ROLE\_DESCRIPTION="Kubernetes administrator role."

export IAM\_ROLE\_ARN=$(aws iam get-role --role-name $IAM\_ROLE | jq -r '.Role.Arn')

if [ -z "$IAM\_ROLE\_ARN" ]

then

IAM\_ROLE\_ARN=$(aws iam create-role \

--role-name $IAM\_ROLE \

--description "$ROLE\_DESCRIPTION" \

--assume-role-policy-document "$POLICY" \

--output text \

--query 'Role.Arn')

echo "IAM Role ${IAM\_ROLE} created. IAM\_ROLE\_ARN=$IAM\_ROLE\_ARN"

else

echo "IAM Role ${IAM\_ROLE} already exist..."

fi

export IAM\_ROLE="k8sTeamADev"

export ROLE\_DESCRIPTION="Kubernetes Admin role for develoeprs for Namespace team-a"

export IAM\_ROLE\_ARN=$(aws iam get-role --role-name $IAM\_ROLE | jq -r '.Role.Arn')

if [ -z "$IAM\_ROLE\_ARN" ]

then

IAM\_ROLE\_ARN=$(aws iam create-role \

--role-name $IAM\_ROLE \

--description "$ROLE\_DESCRIPTION" \

--assume-role-policy-document "$POLICY" \

--output text \

--query 'Role.Arn')

echo "IAM Role ${IAM\_ROLE} created. IAM\_ROLE\_ARN=$IAM\_ROLE\_ARN"

else

echo "IAM Role ${IAM\_ROLE} already exist..."

fi

export IAM\_ROLE="k8sTeamATest"

export ROLE\_DESCRIPTION="ubernetes Viewer role for testers for Namespace team-a"

export IAM\_ROLE\_ARN=$(aws iam get-role --role-name $IAM\_ROLE | jq -r '.Role.Arn')

if [ -z "$IAM\_ROLE\_ARN" ]

then

IAM\_ROLE\_ARN=$(aws iam create-role \

--role-name $IAM\_ROLE \

--description "$ROLE\_DESCRIPTION" \

--assume-role-policy-document "$POLICY" \

--output text \

--query 'Role.Arn')

echo "IAM Role ${IAM\_ROLE} created. IAM\_ROLE\_ARN=$IAM\_ROLE\_ARN"

else

echo "IAM Role ${IAM\_ROLE} already exist..."

fi

Check Output

1

2

3

4

5

6

An error occurred (NoSuchEntity) when calling the GetRole operation: The role with name k8sClusterAdmin cannot be found.

IAM Role k8sClusterAdmin created. IAM\_ROLE\_ARN=arn:aws:iam::ACCOUNT\_ID:role/k8sClusterAdmin

An error occurred (NoSuchEntity) when calling the GetRole operation: The role with name k8sTeamADev cannot be found.

IAM Role k8sTeamADev created. IAM\_ROLE\_ARN=arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev

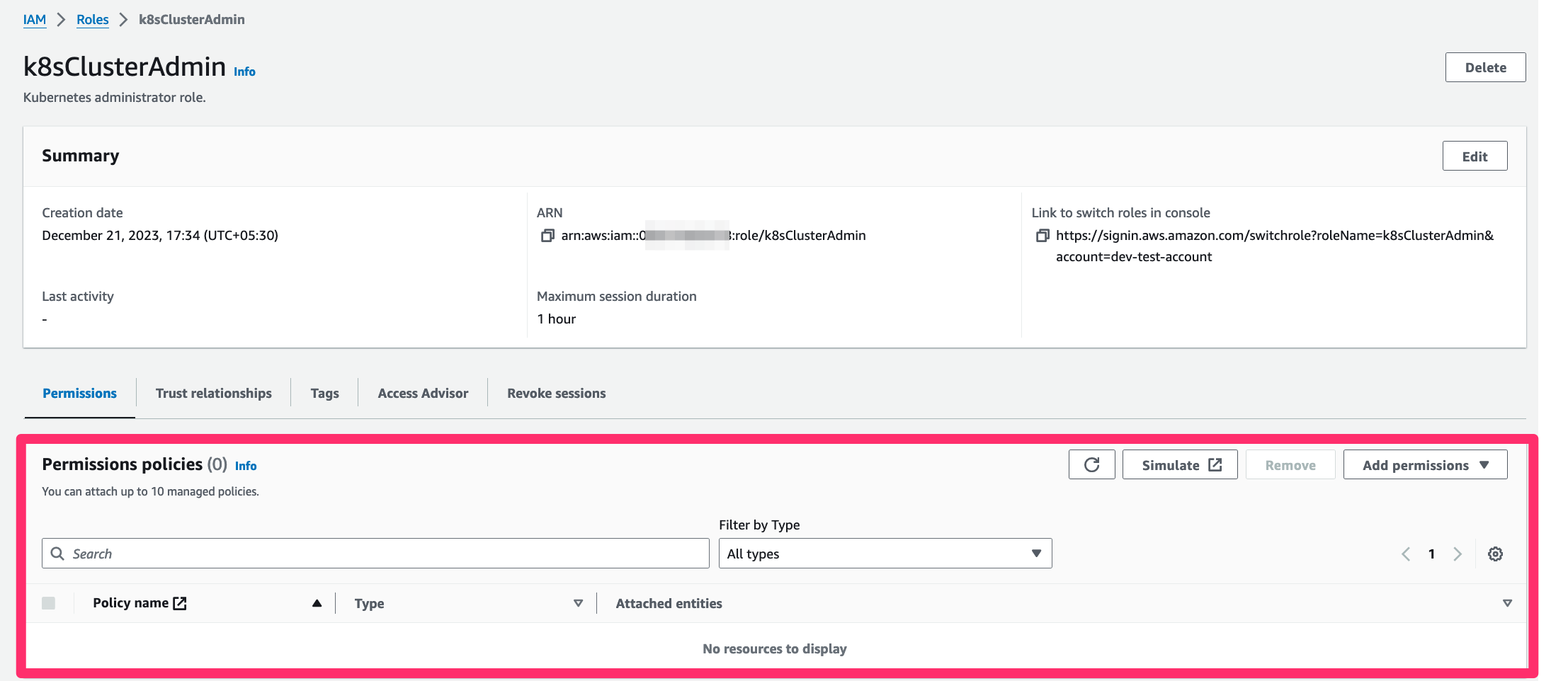
An error occurred (NoSuchEntity) when calling the GetRole operation: The role with name k8sTeamATest cannot be found.

IAM Role k8sTeamATest created. IAM\_ROLE\_ARN=arn:aws:iam::ACCOUNT\_ID:role/k8sTeamATest

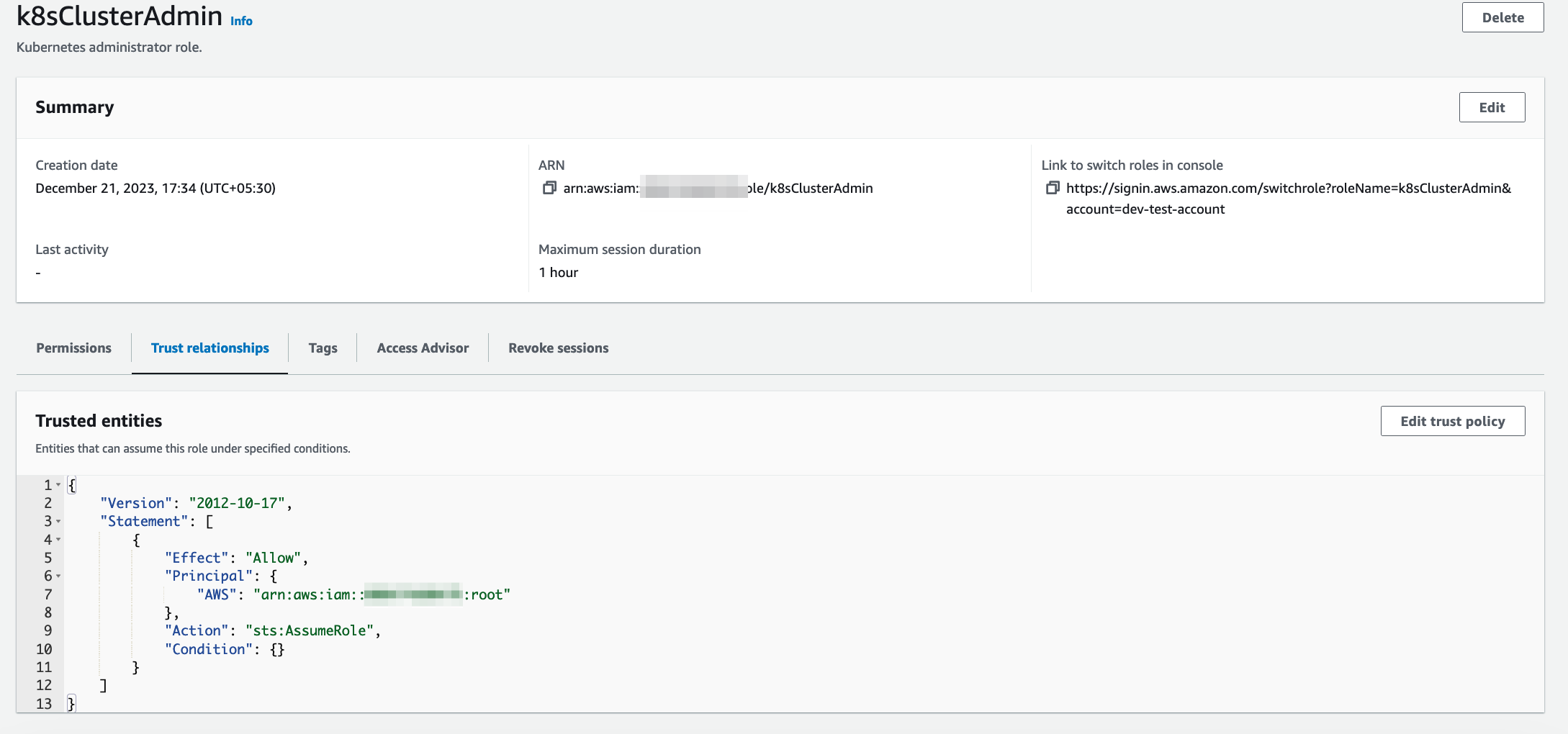
*In this example, the assume-role-policy allows the root account to assume the role. We are going to allow specific groups to also be able to assume those roles. Check the*[official documentation](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts-technical-overview.html)*for more information.*

Because the above roles are only used to authenticate within our Amazon EKS cluster, they don't need to have AWS permissions. We will only use them to allow some IAM groups to assume this role in order to have access to our EKS cluster.

Let's go to the AWS IAM Console and check one of the above IAM Role and see that there are no IAM permissions attached to the Role.



And also let's see trust policy of the IAM Role that allows the root account to assume the role, which means any IAM principal (user or role) can now assume the role.



# Create AWS IAM Groups

In this section let's create 3 IAM groups and attach IAM permission policy on these IAM groups to assume the IAM roles created earlier for Kubernetes role.

We want to have different IAM users which will be added to specific IAM groups in order to have different rights in the kubernetes cluster.

We will define 3 groups:

* **k8sClusterAdmin** - users from this group will have **AmazonEKSClusterAdminPolicy** Kubernetes permissions on the cluster
* **k8sTeamADev** - users from this group will have **AmazonEKSAdminPolicy** Kubernetes permissions on the Namespace ns-a
* **k8sTeamATest** - users from this group will have **AmazonEKSViewPolicy** Kubernetes permissions on the Namespace ns-a

#### [Create k8sClusterAdmin IAM Group](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/2-create-iam-groups" \l "create-k8sclusteradmin-iam-group)

The **k8sClusterAdmin** Group will be allowed to assume the **k8sClusterAdmin** IAM Role.

IAM\_GROUP="k8sClusterAdmin"

export IAM\_GROUP\_ARN=$(aws iam get-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

if [ -z "$IAM\_GROUP\_ARN" ]

then

IAM\_GROUP\_ARN=$(aws iam create-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

echo "IAM Group ${IAM\_GROUP} created. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

else

echo "IAM Group ${IAM\_GROUP} already exist. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

fi

Check Output

An error occurred (NoSuchEntity) when calling the GetGroup operation: The group with name k8sClusterAdmin cannot be found.

IAM Group k8sClusterAdmin created. IAM\_GROUP\_ARN=arn:aws:iam::ACCOUNT\_ID:group/k8sClusterAdmin

Let's add a Policy on our group which will allow users from this group to assume our k8sClusterAdmin Role:

ADMIN\_GROUP\_POLICY=$(echo -n '{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowAssumeOrganizationAccountRole",

"Effect": "Allow",

"Action": "sts:AssumeRole",

"Resource": "arn:aws:iam::'; echo -n "$ACCOUNT\_ID"; echo -n ':role/k8sClusterAdmin"

}

]

}')

echo ADMIN\_GROUP\_POLICY=$ADMIN\_GROUP\_POLICY

aws iam put-group-policy \

--group-name k8sClusterAdmin \

--policy-name k8sClusterAdmin-policy \

--policy-document "$ADMIN\_GROUP\_POLICY"

#### [Create k8sTeamADev IAM Group](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/2-create-iam-groups" \l "create-k8steamadev-iam-group)

The **k8sTeamADev** Group will be allowed to assume the **k8sTeamADev** IAM Role.

IAM\_GROUP="k8sTeamADev"

export IAM\_GROUP\_ARN=$(aws iam get-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

if [ -z "$IAM\_GROUP\_ARN" ]

then

IAM\_GROUP\_ARN=$(aws iam create-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

echo "IAM Group ${IAM\_GROUP} created. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

else

echo "IAM Group ${IAM\_GROUP} already exist. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

fi

Check Output

An error occurred (NoSuchEntity) when calling the GetGroup operation: The group with name k8sTeamADev cannot be found.

IAM Group k8sDev created. IAM\_GROUP\_ARN=arn:aws:iam::ACCOUNT\_ID:group/k8sTeamADev

Let's add a Policy on our group which will allow users from this group to assume our k8sTeamADev Role:

DEV\_GROUP\_POLICY=$(echo -n '{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowAssumeOrganizationAccountRole",

"Effect": "Allow",

"Action": "sts:AssumeRole",

"Resource": "arn:aws:iam::'; echo -n "$ACCOUNT\_ID"; echo -n ':role/k8sTeamADev"

}

]

}')

echo DEV\_GROUP\_POLICY=$DEV\_GROUP\_POLICY

aws iam put-group-policy \

--group-name k8sTeamADev \

--policy-name k8sTeamADev-policy \

--policy-document "$DEV\_GROUP\_POLICY"

#### [Create k8sTeamATest IAM Group](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/2-create-iam-groups" \l "create-k8steamatest-iam-group)

IAM\_GROUP="k8sTeamATest"

export IAM\_GROUP\_ARN=$(aws iam get-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

if [ -z "$IAM\_GROUP\_ARN" ]

then

IAM\_GROUP\_ARN=$(aws iam create-group --group-name $IAM\_GROUP | jq -r '.Group.Arn')

echo "IAM Group ${IAM\_GROUP} created. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

else

echo "IAM Group ${IAM\_GROUP} already exist. IAM\_GROUP\_ARN=$IAM\_GROUP\_ARN"

fi

Check Output

An error occurred (NoSuchEntity) when calling the GetGroup operation: The group with name k8sTeamATest cannot be found.

IAM Group k8sInteg created. IAM\_GROUP\_ARN=arn:aws:iam::ACCOUNT\_ID:group/k8sTeamATest

Let's add a Policy on our group which will allow users from this group to assume our k8sTeamATest Role:

TEST\_GROUP\_POLICY=$(echo -n '{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowAssumeOrganizationAccountRole",

"Effect": "Allow",

"Action": "sts:AssumeRole",

"Resource": "arn:aws:iam::'; echo -n "$ACCOUNT\_ID"; echo -n ':role/k8sTeamATest"

}

]

}')

echo TEST\_GROUP\_POLICY=$TEST\_GROUP\_POLICY

aws iam put-group-policy \

--group-name k8sTeamATest \

--policy-name k8sTeamATest-policy \

--policy-document "$TEST\_GROUP\_POLICY"

You now should have your 3 groups

aws iam list-groups

The output will look like below.

{

"Groups": [

{

"Path": "/",

"GroupName": "k8sClusterAdmin",

"GroupId": "AGPAZRV3OHPJZGT2JKVDV",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sClusterAdmin",

"CreateDate": "2020-04-07T13:32:52Z"

},

{

"Path": "/",

"GroupName": "k8sTeamADev",

"GroupId": "AGPAZRV3OHPJUOBR375KI",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sTeamADev",

"CreateDate": "2020-04-07T13:33:15Z"

},

{

"Path": "/",

"GroupName": "k8sTeamATest",

"GroupId": "AGPAZRV3OHPJR6GM6PFDG",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sTeamATest",

"CreateDate": "2020-04-07T13:33:25Z"

}

]

}

Create AWS IAM Users

In order to test our scenarios, we will create 3 users, one for each groups we created :

IAM\_USERS=("User1Admin" "User1TeamADev" "User1TeamATest")

for IAM\_USER in ${IAM\_USERS[@]}; do

export IAM\_USER\_ARN=$(aws iam get-user --user-name $IAM\_USER | jq -r '.User.Arn')

if [ -z "$IAM\_USER\_ARN" ]

then

IAM\_USER\_ARN=$(aws iam create-user --user-name $IAM\_USER | jq -r '.User.Arn')

echo "IAM User ${IAM\_USER} created. IAM\_USER\_ARN=$IAM\_USER\_ARN"

else

echo "IAM User ${IAM\_USER} already exist..."

fi

done

Check Output

An error occurred (NoSuchEntity) when calling the GetUser operation: The user with name User1Admin cannot be found.

IAM User User1Admin created. IAM\_USER\_ARN=arn:aws:iam::ACCOUNT\_ID:user/User1Admin

An error occurred (NoSuchEntity) when calling the GetUser operation: The user with name User1TeamADev cannot be found.

IAM User User1TeamADev created. IAM\_USER\_ARN=arn:aws:iam::ACCOUNT\_ID:user/User1TeamADev

An error occurred (NoSuchEntity) when calling the GetUser operation: The user with name User1TeamATest cannot be found.

IAM User User1TeamATest created. IAM\_USER\_ARN=arn:aws:iam::ACCOUNT\_ID:user/User1TeamATest

Add users to associated groups:

aws iam add-user-to-group --group-name k8sClusterAdmin --user-name User1Admin

aws iam add-user-to-group --group-name k8sTeamADev --user-name User1TeamADev

aws iam add-user-to-group --group-name k8sTeamATest --user-name User1TeamATest

Check users are correctly added in their groups:

aws iam get-group --group-name k8sClusterAdmin

aws iam get-group --group-name k8sTeamADev

aws iam get-group --group-name k8sTeamATest

Check Output

{

"Users": [

{

"Path": "/",

"UserName": "User1Admin",

"UserId": "AIDAYGIGGNX6DQ3VBNWSP",

"Arn": "arn:aws:iam::ACCOUNT\_ID:user/User1Admin",

"CreateDate": "2023-03-14T09:38:58+00:00"

}

],

"Group": {

"Path": "/",

"GroupName": "k8sClusterAdmin",

"GroupId": "AGPAYGIGGNX6INPRF5C7E",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sClusterAdmin",

"CreateDate": "2023-03-14T09:33:25+00:00"

}

}

{

"Users": [

{

"Path": "/",

"UserName": "User1TeamADev",

"UserId": "AIDAYGIGGNX6KG5ALPI65",

"Arn": "arn:aws:iam::ACCOUNT\_ID:user/User1TeamADev",

"CreateDate": "2023-03-14T09:38:59+00:00"

}

],

"Group": {

"Path": "/",

"GroupName": "k8sTeamADev",

"GroupId": "AGPAYGIGGNX6GRTEAJQE3",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sTeamADev",

"CreateDate": "2023-03-14T09:35:00+00:00"

}

}

{

"Users": [

{

"Path": "/",

"UserName": "User1TeamATest",

"UserId": "AIDAYGIGGNX6EF5ELOVZ4",

"Arn": "arn:aws:iam::ACCOUNT\_ID:user/User1TeamATest",

"CreateDate": "2023-03-14T09:39:00+00:00"

}

],

"Group": {

"Path": "/",

"GroupName": "k8sTeamATest",

"GroupId": "AGPAYGIGGNX6KBNORQ3GN",

"Arn": "arn:aws:iam::ACCOUNT\_ID:group/k8sTeamATest",

"CreateDate": "2023-03-14T09:35:55+00:00"

}

}

**Note** For the sake of simplicity, in this chapter, we will save credentials to a file to make it easy to toggle back and forth between users. Never do this in production or with credentials that have priviledged access; It is not a security best practice to store credentials on the filesystem.

Retrieve Access Keys for our fake users:

aws iam create-access-key --user-name User1Admin | tee /tmp/User1Admin.json

aws iam create-access-key --user-name User1TeamADev | tee /tmp/User1TeamADev.json

aws iam create-access-key --user-name User1TeamATest | tee /tmp/User1TeamATest.json

Check Output

{

"AccessKey": {

"UserName": "User1Admin",

"AccessKeyId": "XXXXXXXXX",

"Status": "Active",

"SecretAccessKey": "XXXXXXXX",

"CreateDate": "2023-03-14T09:52:30+00:00"

}

}

{

"AccessKey": {

"UserName": "User1TeamADev",

"AccessKeyId": "XXXXXXX",

"Status": "Active",

"SecretAccessKey": "XXXXXXXX",

"CreateDate": "2023-03-14T09:52:31+00:00"

}

}

{

"AccessKey": {

"UserName": "User1TeamATest",

"AccessKeyId": "XXXXXXX",

"Status": "Active",

"SecretAccessKey": "XXXXXXX",

"CreateDate": "2023-03-14T09:52:32+00:00"

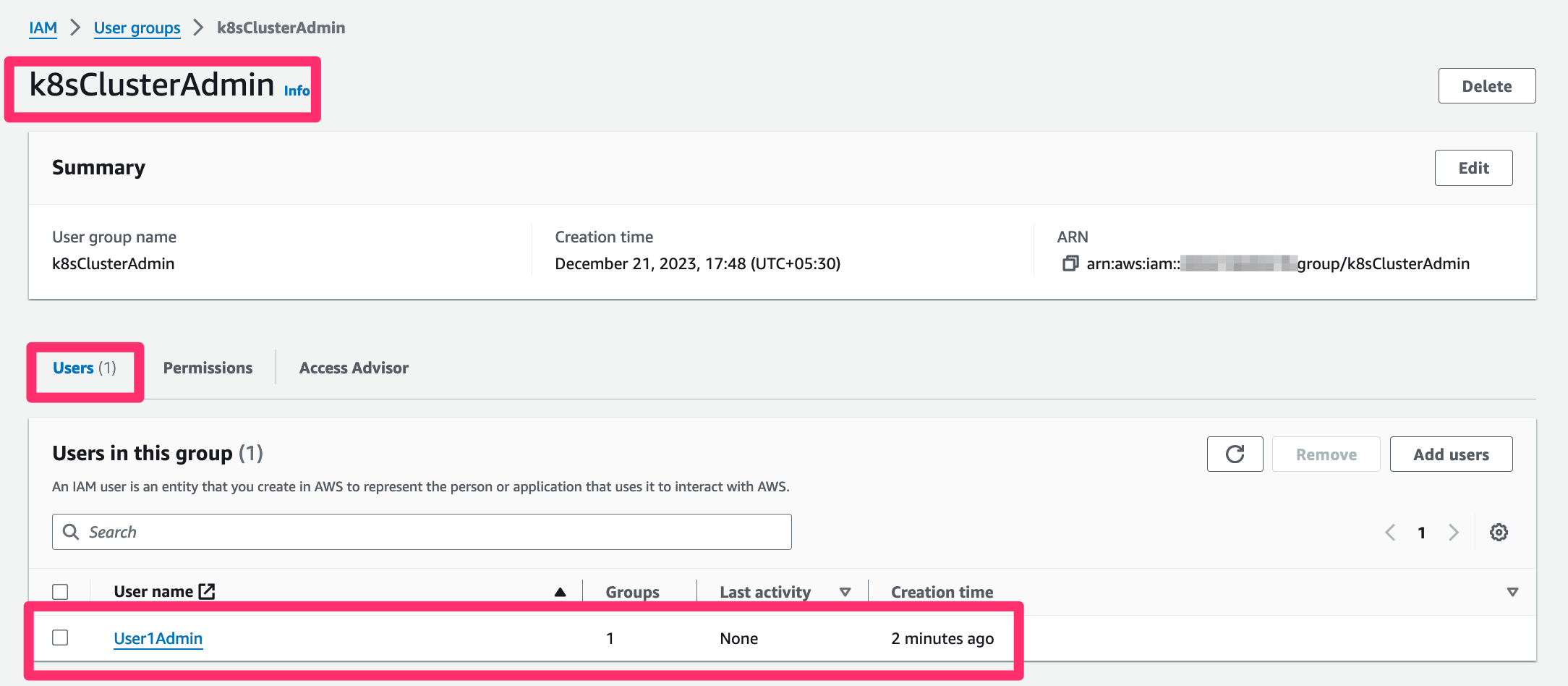
}

}

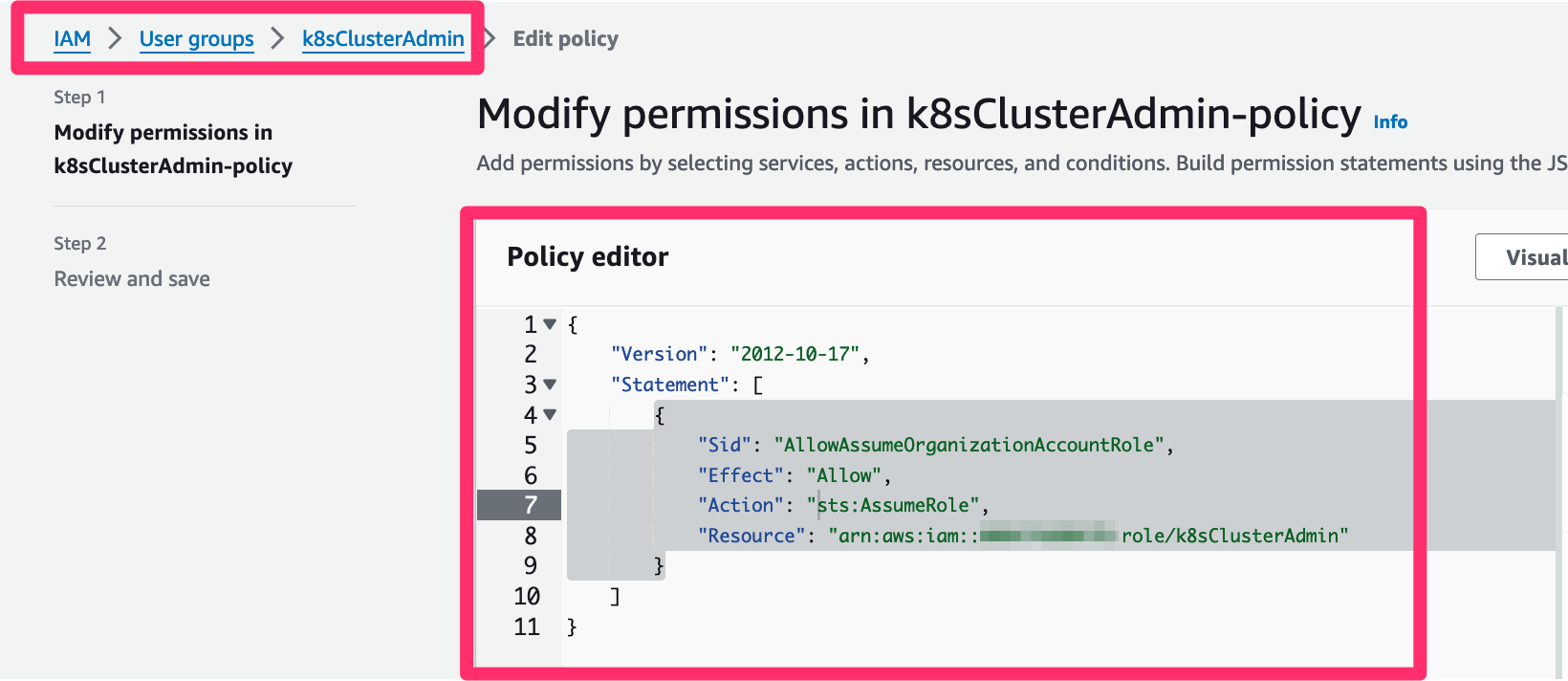
Recap:

* **User1Admin** is in the **k8sClusterAdmin** group and will be able to assume the **k8sClusterAdmin** role.
* **User1TeamADev** is in **k8sTeamADev** Group and will be able to assume IAM role **k8sTeamADev**
* **User1TeamATest** is in **k8sTeamATest** group and will be able to assume IAM role **k8sTeamATest**

Let's go to the [AWS IAM Console](https://console.aws.amazon.com/iamv2/home#/home) and check one of the above IAM Groups and see that there are IAM users part of the group.



And also let's see trust policy of the IAM Group that allows users from this group to assume an IAM Role:



Create Access entries and associate access policies

**[Create Access entries](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/4-create-access-entries" \l "create-access-entries)**

So far we created **IAM Roles/Groups/Users** in AWS. In this section, let's create access entries for the IAM Roles.

1

2

3

4

5

6

7

8

9

10

11

export EKS\_CLUSTER\_NAME="eksworkshop-eksctl"

export TYPE="STANDARD"

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

aws eks create-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --type $TYPE

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

aws eks create-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --type $TYPE

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamATest"

aws eks create-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --type $TYPE

Expand for Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sClusterAdmin",

"kubernetesGroups": [],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sClusterAdmin/90c6ad21-e9df-a837-6b39-6ef67f1bd03d",

"createdAt": "2024-01-30T12:06:51.206000+00:00",

"modifiedAt": "2024-01-30T12:06:51.206000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/{{SessionName}}",

"type": "STANDARD"

}

}

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev",

"kubernetesGroups": [],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sTeamADev/f8c6ad21-ebb9-d2aa-335c-d8af833bb77e",

"createdAt": "2024-01-30T12:06:52.268000+00:00",

"modifiedAt": "2024-01-30T12:06:52.268000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/{{SessionName}}",

"type": "STANDARD"

}

}

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamATest",

"kubernetesGroups": [],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sTeamATest/96c6ad21-edac-ba70-daaf-bd4f626d04d0",

"createdAt": "2024-01-30T12:06:53.092000+00:00",

"modifiedAt": "2024-01-30T12:06:53.092000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamATest/{{SessionName}}",

"type": "STANDARD"

}

}

**[Associate Access policies](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/4-create-access-entries" \l "associate-access-policies)**

Let us assoicate the required access policy to each of the access entries created above.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

# Associate AmazonEKSClusterAdminPolicy access policy to IAM Role k8sClusterAdmin

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

export ACCESS\_SCOPE="cluster"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN \

--access-scope type=$ACCESS\_SCOPE

# Associate AmazonEKSAdminPolicy access policy to IAM Role k8sTeamADev

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

export ACCESS\_SCOPE="namespace"

export NAMESPACES="team-a"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN \

--access-scope type=$ACCESS\_SCOPE,namespaces=$NAMESPACES

# Associate AmazonEKSViewPolicy access policy to IAM Role k8sTeamATest

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamATest"

export ACCESS\_SCOPE="namespace"

export NAMESPACES="team-a"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN \

--access-scope type=$ACCESS\_SCOPE,namespaces=$NAMESPACES

Expand for Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sClusterAdmin",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy",

"accessScope": {

"type": "cluster",

"namespaces": []

},

"associatedAt": 1703388453.491,

"modifiedAt": 1703388453.491

}

}

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy",

"accessScope": {

"type": "namespace",

"namespaces": [

"team-a"

]

},

"associatedAt": 1703402093.134,

"modifiedAt": 1703402093.134

}

}

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamATest",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy",

"accessScope": {

"type": "namespace",

"namespaces": [

"team-a"

]

},

"associatedAt": 1703402181.222,

"modifiedAt": 1703402181.222

}

}

# Test Amazon EKS access

## [Install kubectl whoami plugin](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "install-kubectl-whoami-plugin)

Before we start testing access to EKS cluster with using different IAM Roles, let us install this useful plugin to see who is authenticating to the EKS cluster.

1

2

3

4

5

cd ~/environment

wget https://github.com/rajatjindal/kubectl-whoami/releases/download/v0.0.46/kubectl-whoami\_v0.0.46\_linux\_amd64.tar.gz

tar xvf kubectl-whoami\_v0.0.46\_linux\_amd64.tar.gz

chmod +x kubectl-whoami

sudo cp kubectl-whoami /usr/local/bin/

Run the command to see who is currently authenticating to EKS cluster.

1

kubectl whoami

Check Output

1

arn:aws:sts::ACCOUNT\_ID:assumed-role/eks-security-workshop/EKSGetTokenAuth

As expected we are currently using the IAM Role eks-security-workshop which is used to create the EKS cluster.

## [Automate Assume Role with AWS CLI](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "automate-assume-role-with-aws-cli)

It is possible to automate the retrieval of temporary credentials for the assumed role by configuring the AWS CLI in the files ~/.aws/config and ~/.aws/credentials. As an example, we will define three profiles.

#### [Add in](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awsconfig:)**[~/.aws/config](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awsconfig:)**[:](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awsconfig:)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

mkdir -p ~/.aws

export AWS\_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')

export ADMIN\_ROLE="k8sClusterAdmin"

export ADMIN\_PROFILE="eksAdmin"

export DEV\_ROLE="k8sTeamADev"

export DEV\_PROFILE="eksDev"

export TEST\_ROLE="k8sTeamATest"

export TEST\_PROFILE="eksTest"

# function to insert profile

insert\_profile () {

cat << EoF >> ~/.aws/config

[profile ${1}]

role\_arn=arn:aws:iam::${ACCOUNT\_ID}:role/${2}

source\_profile=${3}

EoF

}

if test -f ~/.aws/config; then

#test profile admin

if ! grep -q "profile admin" ~/.aws/config; then

insert\_profile "admin" $ADMIN\_ROLE $ADMIN\_PROFILE

echo "added profile admin"

fi

#test profile dev

if ! grep -q "profile dev" ~/.aws/config; then

insert\_profile "dev" $DEV\_ROLE $DEV\_PROFILE

echo "added profile dev"

fi

#test profile test

if ! grep -q "profile test" ~/.aws/config; then

insert\_profile "test" $TEST\_ROLE $TEST\_PROFILE

echo "added profile test"

fi

echo "all profiles added correctly..."

else

cat << EoF >> ~/.aws/config

[default]

region = ${AWS\_REGION}

EoF

insert\_profile "admin" $ADMIN\_ROLE $ADMIN\_PROFILE

insert\_profile "dev" $DEV\_ROLE $DEV\_PROFILE

insert\_profile "test" $TEST\_ROLE $TEST\_PROFILE

echo "config file with profiles added..."

fi

#### [Add in](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awscredentials:)**[~/.aws/credentials](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awscredentials:)**[:](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "add-in-~.awscredentials:)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

if ! test -f ~/.aws/credentials; then

cat << EoF >> ~/.aws/credentials

[eksAdmin]

aws\_access\_key\_id=$(jq -r .AccessKey.AccessKeyId /tmp/User1Admin.json)

aws\_secret\_access\_key=$(jq -r .AccessKey.SecretAccessKey /tmp/User1Admin.json)

[eksDev]

aws\_access\_key\_id=$(jq -r .AccessKey.AccessKeyId /tmp/User1TeamADev.json)

aws\_secret\_access\_key=$(jq -r .AccessKey.SecretAccessKey /tmp/User1TeamADev.json)

[eksTest]

aws\_access\_key\_id=$(jq -r .AccessKey.AccessKeyId /tmp/User1TeamATest.json)

aws\_secret\_access\_key=$(jq -r .AccessKey.SecretAccessKey /tmp/User1TeamATest.json)

EoF

else

echo "AWS Credentials file ~/.aws/credentials already exists..."

fi

#### [Test AWS Identity this with admin and dev profiles:](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "test-aws-identity-this-with-admin-and-dev-profiles:)

1

aws sts get-caller-identity --profile admin

The output looks like below.

1

2

3

4

5

{

"UserId": "AROA26YVAA7XXGCMO6D5W:botocore-session-1706617342",

"Account": "ACCOUNT\_ID",

"Arn": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1706617342"

}

The assumed-role is k8sClusterAdmin, so we achieved our goal.

When specifying the **--profile admin** parameter we automatically ask for temporary credentials for the role k8sClusterAdmin. You can test this with **dev** and **test** also.

1

aws sts get-caller-identity --profile dev

The output looks like below.

1

2

3

4

5

{

"UserId": "AROA26YVAA7X4JDJXGLQX:botocore-session-1706617378",

"Account": "ACCOUNT\_ID",

"Arn": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/botocore-session-1706617378"

}

*When specifying the****--profile dev****parameter we automatically ask for temporary credentials for the role k8sTeamADev*

## [Using AWS profiles with the Kubectl config file](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "using-aws-profiles-with-the-kubectl-config-file)

### [Install yq for yaml processing](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "install-yq-for-yaml-processing)

1

2

3

echo 'yq() {

docker run --rm -i -v "${PWD}":/workdir mikefarah/yq "$@"

}' | tee -a ~/.bashrc && source ~/.bashrc

It is also possible to specify the AWS\_PROFILE to use with the aws-iam-authenticator in the ~/.kube/config file, so that it will use the appropriate profile.

### [With admin profile](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "with-admin-profile)

Create a new KUBECONFIG file to test this:

1

2

export KUBECONFIG=/tmp/kubeconfig-admin && eksctl utils write-kubeconfig -c eksworkshop-eksctl

cat $KUBECONFIG | yq e '.users.[].user.exec.args += ["--profile", "admin"]' - -- | sed 's/eksworkshop-eksctl./eksworkshop-eksctl-admin./g' | sponge $KUBECONFIG

*Note: this assume you uses yq >= version 4. you can reference to*[this page](https://mikefarah.gitbook.io/yq/upgrading-from-v3)*to adapt this command for another version.*

We added the --profile admin parameter to our kubectl config file, so that this will ask kubectl to use our IAM role associated to our admin profile, and we rename the context using suffix **-admin**.

Let us look at the kubeconfig file created and see how we are using profile admin to get token and then authenticating to the cluster.

1

cat $KUBECONFIG

Check Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https://C7D065E5990AA2C6E0772CD22937F4FC.gr7.us-east-1.eks.amazonaws.com

name: eksworkshop-eksctl-admin.us-east-1.eksctl.io

contexts:

- context:

cluster: eksworkshop-eksctl-admin.us-east-1.eksctl.io

user: i-08b0a08412575c115@eksworkshop-eksctl-admin.us-east-1.eksctl.io

name: i-08b0a08412575c115@eksworkshop-eksctl-admin.us-east-1.eksctl.io

current-context: i-08b0a08412575c115@eksworkshop-eksctl-admin.us-east-1.eksctl.io

kind: Config

preferences: {}

users:

- name: i-08b0a08412575c115@eksworkshop-eksctl-admin.us-east-1.eksctl.io

user:

exec:

apiVersion: client.authentication.k8s.io/v1beta1

args:

- eks

- get-token

- --output

- json

- --cluster-name

- eksworkshop-eksctl

- --region

- us-east-1

- --profile

- admin

command: aws

env:

- name: AWS\_STS\_REGIONAL\_ENDPOINTS

value: regional

provideClusterInfo: **false**

Run the command again to see who is now authenticating to the EKS cluster.

1

kubectl whoami

Check Output

1

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703478796

This shows that we are now authenticating to the cluster using IAM Role k8sClusterAdmin using the profile admin

Since the IAM Principal k8sClusterAdmin is associated with access policy AmazonEKSClusterAdminPolicy, it has all the Kubernetes permissions on the cluster.

Let us run few kubectl commands.

1

2

kubectl get pod -A

kubectl get node

Let's create a Namespace test and create a pod in that namespace.

1

2

kubectl create ns team-a

kubectl run nginx-admin --image=nginx -n team-a

Check Output

1

2

namespace/team-a created

pod/nginx-admin created

We can list the pods:

1

kubectl get pods -n team-a

The output looks like below

1

2

NAME READY STATUS RESTARTS AGE

nginx-admin 1/1 Running 0 14s

### [With dev profile](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "with-dev-profile)

Create a new KUBECONFIG file to test this:

1

2

export KUBECONFIG=/tmp/kubeconfig-dev && eksctl utils write-kubeconfig -c eksworkshop-eksctl

cat $KUBECONFIG | yq e '.users.[].user.exec.args += ["--profile", "dev"]' - -- | sed 's/eksworkshop-eksctl./eksworkshop-eksctl-dev./g' | sponge $KUBECONFIG

Check Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

2023-03-14 10:16:28 [✔] saved kubeconfig as "/tmp/kubeconfig-dev"

Unable to find image 'mikefarah/yq:latest' locally

latest: Pulling from mikefarah/yq

63b65145d645: Pulling fs layer

865242c25e72: Pulling fs layer

48f2cb577b3c: Pulling fs layer

6b38082b4af1: Pulling fs layer

0a8c5b7f3b42: Pulling fs layer

6b38082b4af1: Waiting

0a8c5b7f3b42: Waiting

48f2cb577b3c: Verifying Checksum

48f2cb577b3c: Download complete

63b65145d645: Verifying Checksum

63b65145d645: Download complete

865242c25e72: Verifying Checksum

865242c25e72: Download complete

63b65145d645: Pull complete

6b38082b4af1: Verifying Checksum

6b38082b4af1: Download complete

0a8c5b7f3b42: Verifying Checksum

0a8c5b7f3b42: Download complete

865242c25e72: Pull complete

48f2cb577b3c: Pull complete

6b38082b4af1: Pull complete

0a8c5b7f3b42: Pull complete

Digest: sha256:29ebb32f7d89a6b8e102a9cf1fb1c073d7154c17e5eda8a584f60f036b11f655

Status: Downloaded newer image for mikefarah/yq:latest

*Note: this assume you uses yq >= version 4. you can reference to*[this page](https://mikefarah.gitbook.io/yq/upgrading-from-v3)*to adapt this command for another version.*

We added the --profile dev parameter to our kubectl config file, so that this will ask kubectl to use our IAM role associated to our dev profile, and we rename the context using suffix **-dev**.

Run the command again to see who is now authenticating to the EKS cluster.

1

kubectl whoami

Check Output

1

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/botocore-session-1703482786

This shows that we are now authenticating to the cluster using IAM Role k8sTeamADev using the profile dev

Since the IAM Principal k8sTeamADev is associated with access policy AmazonEKSAdminPolicy for the Namespace team-a, it allows read/write access to most resources in a namespace, including the ability to create roles and role bindings within the namespace.

Let's create a pod:

1

kubectl run nginx-dev --image=nginx -n team-a

Check Output

1

pod/nginx-dev created

We can list the pods:

1

kubectl get pods -n team-a

The output looks like below

1

2

3

NAME READY STATUS RESTARTS AGE

nginx-admin 1/1 Running 0 5m54s

nginx-dev 1/1 Running 0 13s

... but not in other namespaces:

1

kubectl get pods -n default

The output looks like below

1

Error from server (Forbidden): pods is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/botocore-session-1703482786" cannot list resource "pods" in API group "" in the namespace "default"

#### [Test with test profile](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "test-with-test-profile)

1

2

export KUBECONFIG=/tmp/kubeconfig-test && eksctl utils write-kubeconfig -c eksworkshop-eksctl

cat $KUBECONFIG | yq e '.users.[].user.exec.args += ["--profile", "test"]' - -- | sed 's/eksworkshop-eksctl./eksworkshop-eksctl-test./g' | sponge $KUBECONFIG

Check Output

1

2023-03-14 10:24:31 [✔] saved kubeconfig as "/tmp/kubeconfig-test"

*Note: this assume you uses yq >= version 4. you can reference to*[this page](https://mikefarah.gitbook.io/yq/upgrading-from-v3)*to adapt this command for another version.*

We added the --profile test parameter to our kubectl config file, so that this will ask kubectl to use our IAM role associated to our test profile, and we rename the context using suffix **-test**.

Run the command again to see who is now authenticating to the EKS cluster.

1

kubectl whoami

Check Output

1

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamATest/botocore-session-1703484518

This shows that we are now authenticating to the cluster using IAM Role k8sTeamATest using the profile test

Since the IAM Principal k8sTeamATest is associated with access policy AmazonEKSViewPolicy for the Namespace team-a, it allows read-only access to see most objects in a namespace.

Let's create a pod:

1

kubectl run nginx-test --image=nginx -n team-a

Check Output

1

Error from server (Forbidden): pods is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamATest/botocore-session-1703484518" cannot create resource "pods" in API group "" in the namespace "team-a"

The error is expected since it is a read-only access.

We can list the pods:

1

kubectl get pods -n team-a

The output looks like below

NAME READY STATUS RESTARTS AGE

nginx-admin 1/1 Running 0 5m54s

nginx-dev 1/1 Running 0 13s

... but not in other namespaces:

1

kubectl get pods -n default

The output looks like below

1

Error from server (Forbidden): pods is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamATest/botocore-session-1703484518" cannot list resource "pods" in API group "" in the namespace "default"

## [Conclusion](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/5-test-eks-access" \l "conclusion)

In this module, we have seen how to configure Amazon EKS to provide finer access to users combining IAM Groups, IAM Roles and EKS access policies. You can create different groups depending on your needs, configure their associated Kubernetes access in your cluster, and simply add or remove users from the group to grant or revoke access to your cluster.

Users will only have to configure their AWS CLI in order to automatically retrieve their associated rights in your cluster.

# AWS console access to EKS Cluster

This step is optional, as nearly all of the workshop content is CLI-driven. But, if you'd like full access to your workshop cluster in the EKS console this step is recommended.

The EKS console allows you to see not only the configuration aspects of your cluster, but also to view Kubernetes cluster objects such as Deployments, Pods, and Nodes. For this type of access, the console IAM User or Role needs to be granted permission within the cluster.

By default, the credentials used to create the cluster are automatically granted these permissions. Following along in the workshop, you've created a cluster using temporary IAM credentials from within Cloud9. This means that you'll need to add your AWS Console credentials to the cluster.

#### [Import your EKS Console credentials to your new cluster:](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/3-create-access-entrie-and-policies/6-console-credentials" \l "import-your-eks-console-credentials-to-your-new-cluster:)

Set the environment variable for the IAM role used to login into the AWS Console.

1

export rolearn="arn:aws:iam::$ACCOUNT\_ID:role/WSParticipantRole"

With your ARN in hand, you can issue the command to associate access policy for the above IAM role.

export EKS\_CLUSTER\_NAME="eksworkshop-eksctl"

export TYPE="STANDARD"

aws eks create-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $rolearn --type $TYPE

export ACCESS\_SCOPE="cluster"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $rolearn \

--policy-arn $ACCESS\_POLICY\_ARN \

--access-scope type=$ACCESS\_SCOPE

Check Output

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/WSOpsRole",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy",

"accessScope": {

"type": "cluster",

"namespaces": []

},

"associatedAt": "2024-01-30T13:03:45.241000+00:00",

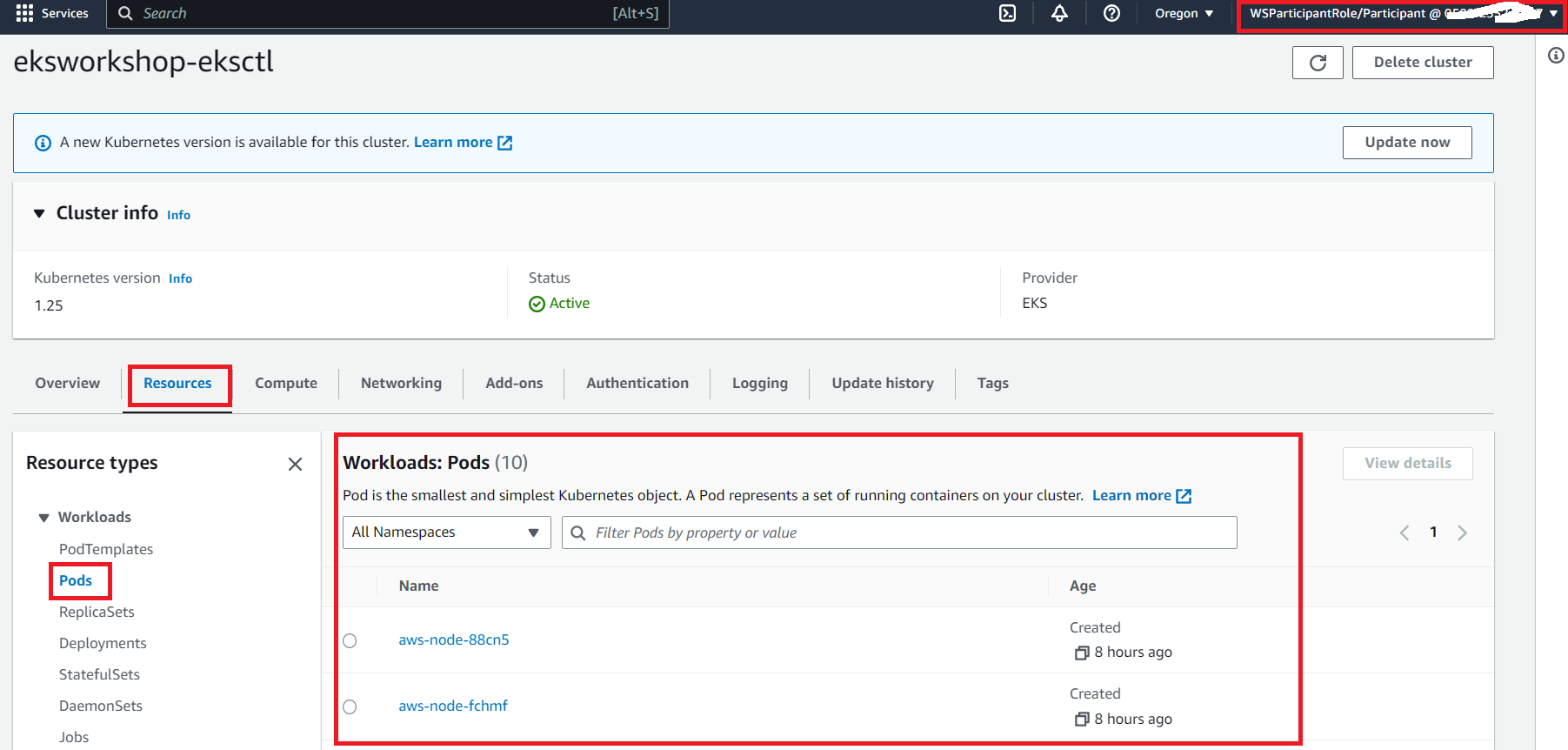
"modifiedAt": "2024-01-30T13:03:45.241000+00:00"

}

}

Note that permissions can be restricted and granular but as this is a workshop cluster, you're adding your console credentials as administrator.

You can now view various Kubernetes Objects in the Amazon EKS Cluster in the [AWS Console for Amazon EKS](https://console.aws.amazon.com/eks/home?#/clusters/eksworkshop-eksctl?selectedTab=cluster-resources-tab&selectedResourceId=pods).



For more information, check out the [EKS documentation](https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html) on this topic.

Using access entries with Kubernetes RBAC

**[Using access entries with Kubernetes RBAC](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "using-access-entries-with-kubernetes-rbac)**

The cluster access management controls and associated APIs don’t replace the existing RBAC authorizer in Amazon EKS. Rather, Amazon EKS access entries can be combined with the RBAC authorizer to grant cluster access to an AWS IAM principal while relying on Kubernetes RBAC to apply desired permissions.

In this section, instead of using EKS access policy AmazonEKSClusterAdminPolicy, we will use Kubernetes RBAC for administrator access to the EKS cluster.

**[dis-associate access policy from IAM Role.](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "dis-associate-access-policy-from-iam-role.)**

Let us first dis-associate the access policy AmazonEKSClusterAdminPolicy from the IAM Role k8sClusterAdmin.

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN

Use IAM Role k8sClusterAdmin to access EKS cluster.

export KUBECONFIG=/tmp/kubeconfig-admin

kubectl whoami

Check Output

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059

Test access to cluster.

kubectl get node

Check Output

1

Error from server (Forbidden): pods is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059" cannot list resource "pods" in API group "" in the namespace "default"```

This shows that the IAM Role k8sClusterAdmin does not have any access to the cluster.

**[Apply Kubernetes RBAC to access entry](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "apply-kubernetes-rbac-to-access-entry)**

export EKS\_CLUSTER\_NAME="eksworkshop-eksctl"

export EKS\_ADMIN\_RBAC\_GROUP="k8s-rbac-group-admin"

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

aws eks update-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --kubernetes-groups $EKS\_ADMIN\_RBAC\_GROUP

Check Output

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sClusterAdmin",

"kubernetesGroups": [

"k8s-rbac-group-admin"

],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sClusterAdmin/90c6ad21-e9df-a837-6b39-6ef67f1bd03d",

"createdAt": "2024-01-30T12:06:51.206000+00:00",

"modifiedAt": "2024-01-30T12:54:00.663000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/{{SessionName}}",

"type": "STANDARD"

}

}

Let us apply Kubernetes RBAC using ClusterRoleBinding Resource

cd ~/environment

cat << EOF > rbac-for-access-entry.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: cluster-admin-ae

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: $EKS\_ADMIN\_RBAC\_GROUP

EOF

unset KUBECONFIG

kubectl apply -f rbac-for-access-entry.yaml

Check Output

clusterrolebinding.rbac.authorization.k8s.io/cluster-admin-ae created

Use IAM Role k8sClusterAdmin to access EKS cluster.

export KUBECONFIG=/tmp/kubeconfig-admin

kubectl whoami

Check Output

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059

Test access to cluster.

kubectl get node

Check Output

NAME STATUS ROLES AGE VERSION

ip-192-168-105-62.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

ip-192-168-158-255.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

ip-192-168-184-154.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

Using access entries with Kubernetes RBAC

**[Using access entries with Kubernetes RBAC](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "using-access-entries-with-kubernetes-rbac)**

The cluster access management controls and associated APIs don’t replace the existing RBAC authorizer in Amazon EKS. Rather, Amazon EKS access entries can be combined with the RBAC authorizer to grant cluster access to an AWS IAM principal while relying on Kubernetes RBAC to apply desired permissions.

In this section, instead of using EKS access policy AmazonEKSClusterAdminPolicy, we will use Kubernetes RBAC for administrator access to the EKS cluster.

**[dis-associate access policy from IAM Role.](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "dis-associate-access-policy-from-iam-role.)**

Let us first dis-associate the access policy AmazonEKSClusterAdminPolicy from the IAM Role k8sClusterAdmin.

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN

Use IAM Role k8sClusterAdmin to access EKS cluster.

export KUBECONFIG=/tmp/kubeconfig-admin

kubectl whoami

Check Output

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059

Test access to cluster.

kubectl get node

Check Output

Error from server (Forbidden): pods is forbidden: User "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059" cannot list resource "pods" in API group "" in the namespace "default"```

This shows that the IAM Role k8sClusterAdmin does not have any access to the cluster.

**[Apply Kubernetes RBAC to access entry](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/4-access-entry-with-rbac" \l "apply-kubernetes-rbac-to-access-entry)**

export EKS\_CLUSTER\_NAME="eksworkshop-eksctl"

export EKS\_ADMIN\_RBAC\_GROUP="k8s-rbac-group-admin"

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

aws eks update-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --kubernetes-groups $EKS\_ADMIN\_RBAC\_GROUP

Check Output

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sClusterAdmin",

"kubernetesGroups": [

"k8s-rbac-group-admin"

],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sClusterAdmin/90c6ad21-e9df-a837-6b39-6ef67f1bd03d",

"createdAt": "2024-01-30T12:06:51.206000+00:00",

"modifiedAt": "2024-01-30T12:54:00.663000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/{{SessionName}}",

"type": "STANDARD"

}

}

Let us apply Kubernetes RBAC using ClusterRoleBinding Resource

cd ~/environment

cat << EOF > rbac-for-access-entry.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: cluster-admin-ae

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- apiGroup: rbac.authorization.k8s.io

kind: Group

name: $EKS\_ADMIN\_RBAC\_GROUP

EOF

1

2

unset KUBECONFIG

kubectl apply -f rbac-for-access-entry.yaml

Check Output

1

clusterrolebinding.rbac.authorization.k8s.io/cluster-admin-ae created

Use IAM Role k8sClusterAdmin to access EKS cluster.

export KUBECONFIG=/tmp/kubeconfig-admin

kubectl whoami

Check Output

1

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sClusterAdmin/botocore-session-1703490059

Test access to cluster.

1

kubectl get node

Check Output

NAME STATUS ROLES AGE VERSION

ip-192-168-105-62.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

ip-192-168-158-255.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

ip-192-168-184-154.ec2.internal Ready <none> 4d1h v1.28.3-eks-e71965b

Deleting IAM principal from access entry

**[Deleting IAM principal from access entry](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/5-delete-iam-principal-from-ae" \l "deleting-iam-principal-from-access-entry)**

The reference of a cluster access entry to its underlying AWS IAM principal is unique, as seen in the accessEntryArn in the following create-access-entry output snippet.

Let us get the accessEntryArn for the IAM Principal k8sTeamADev

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

export ACCESS\_ENTRY\_ARN=$(aws eks describe-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --query 'accessEntry.accessEntryArn' --output text)

echo "ACCESS\_ENTRY\_ARN=$ACCESS\_ENTRY\_ARN"

Check Output

ACCESS\_ENTRY\_ARN=arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sTeamADev/f8c6ad21-ebb9-d2aa-335c-d8af833bb77e

Once an access entry is created, the underlying AWS IAM principal cannot be changed, while keeping the cluster access. The access entry and associated access policies must be recreated

Use IAM Role k8sTeamADev to access EKS cluster.

export KUBECONFIG=/tmp/kubeconfig-dev

kubectl whoami

Check Output

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/botocore-session-1703494771

Test access to cluster.

kubectl get pod -n team-a

Check Output

NAME READY STATUS RESTARTS AGE

nginx-admin 1/1 Running 0 78m

nginx-dev 1/1 Running 0 3s

Before deleting the IAM Principal k8sTeamADev. let us find the Role Id.

export IAM\_ROLE="k8sTeamADev"

export ROLE\_ID=$(aws iam get-role --role-name $IAM\_ROLE --query 'Role.RoleId' --output text)

echo "ROLE\_ID=$ROLE\_ID"

Check Output

ROLE\_ID=AROAQAHCJ2QPJTOAB3E4V

**[Delete and re-create IAM Role](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/5-delete-iam-principal-from-ae" \l "delete-and-re-create-iam-role)**

Let us delete the IAM Role k8sTeamADev

aws iam delete-role --role-name $IAM\_ROLE

Let us re-create the IAM Role k8sTeamADev

export IAM\_ROLE="k8sTeamADev"

export ROLE\_DESCRIPTION="Kubernetes Admin role for develoeprs for Namespace team-a"

export IAM\_ROLE\_ARN=$(aws iam get-role --role-name $IAM\_ROLE | jq -r '.Role.Arn')

if [ -z "$IAM\_ROLE\_ARN" ]

then

IAM\_ROLE\_ARN=$(aws iam create-role \

--role-name $IAM\_ROLE \

--description "$ROLE\_DESCRIPTION" \

--assume-role-policy-document "$POLICY" \

--output text \

--query 'Role.Arn')

echo "IAM Role ${IAM\_ROLE} created. IAM\_ROLE\_ARN=$IAM\_ROLE\_ARN"

else

echo "IAM Role ${IAM\_ROLE} already exist..."

fi

Check Output

1

IAM Role k8sTeamADev created. IAM\_ROLE\_ARN=arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev

Let us find the Role Id.

export IAM\_ROLE="k8sTeamADev"

export ROLE\_ID=$(aws iam get-role --role-name $IAM\_ROLE --query 'Role.RoleId' --output text)

echo "ROLE\_ID=$ROLE\_ID"

Check Output

1

ROLE\_ID=AROAQAHCJ2QPOIJYQ2RTT

Note that the current Role Id AROAQAHCJ2QPOIJYQ2RTT is different from the earlier one AROAQAHCJ2QPJTOAB3E4V even for same Role ARN.

Let us test access to EKS cluster.

1

kubectl whoami

Check Output

1

Error: Unauthorized

1

kubectl get pod -n team-a

Check Output

1

error: You must be logged in to the server (Unauthorized)

**[Delete and re-create access entry](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/4-eks-cluster-iam-access-management/5-delete-iam-principal-from-ae" \l "delete-and-re-create-access-entry)**

Delete the access entry

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

aws eks delete-access-entry --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN

Re-create the access entry

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

aws eks create-access-entry --cluster-name $EKS\_CLUSTER\_NAME --principal-arn $IAM\_PRINCIPAL\_ARN --type $TYPE

Check Output

{

"accessEntry": {

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev",

"kubernetesGroups": [],

"accessEntryArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:access-entry/eksworkshop-eksctl/role/ACCOUNT\_ID/k8sTeamADev/00c6ad42-1fed-ba20-0c0a-3e40f6d0e5eb",

"createdAt": "2024-01-30T13:17:13.189000+00:00",

"modifiedAt": "2024-01-30T13:17:13.189000+00:00",

"tags": {},

"username": "arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/{{SessionName}}",

"type": "STANDARD"

}

}

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

export ACCESS\_SCOPE="namespace"

export NAMESPACES="team-a"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"

aws eks associate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN \

--access-scope type=$ACCESS\_SCOPE,namespaces=$NAMESPACES

Check Output

{

"clusterName": "eksworkshop-eksctl",

"principalArn": "arn:aws:iam::ACCOUNT\_ID:role/k8sTeamADev",

"associatedAccessPolicy": {

"policyArn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy",

"accessScope": {

"type": "namespace",

"namespaces": [

"team-a"

]

},

"associatedAt": 1703496078.676,

"modifiedAt": 1703496078.676

}

}

\rm -rf ~/.aws/cli/cache

kubectl whoami

Check Output

arn:aws:sts::ACCOUNT\_ID:assumed-role/k8sTeamADev/botocore-session-1703496838

Let's create a pod:

kubectl run nginx-dev2 --image=nginx -n team-a

Check Output

pod/nginx-dev2 created

We can list the pods:

kubectl get pods -n team-a

The output looks like below

NAME READY STATUS RESTARTS AGE

nginx-admin 1/1 Running 0 115m

nginx-dev 1/1 Running 0 36m

nginx-dev2 1/1 Running 0 12s

Cleanup

Once you have completed this chapter, you can cleanup the files and resources you created by issuing the following commands:

unset KUBECONFIG

kubectl delete pod nginx-dev -n team-a

kubectl delete pod nginx-admin -n team-a

kubectl delete namespace team-a

# disassociate the access policies

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamATest"

export ACCESS\_POLICY\_ARN="arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"

aws eks disassociate-access-policy --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN \

--policy-arn $ACCESS\_POLICY\_ARN

# Delete the access entries

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sClusterAdmin"

aws eks delete-access-entry --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamADev"

aws eks delete-access-entry --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN

export IAM\_PRINCIPAL\_ARN="arn:aws:iam::${ACCOUNT\_ID}:role/k8sTeamATest"

aws eks delete-access-entry --cluster-name $EKS\_CLUSTER\_NAME \

--principal-arn $IAM\_PRINCIPAL\_ARN

aws iam remove-user-from-group --group-name k8sClusterAdmin --user-name User1Admin

aws iam remove-user-from-group --group-name k8sTeamADev --user-name User1TeamADev

aws iam remove-user-from-group --group-name k8sTeamATest --user-name User1TeamATest

aws iam delete-group-policy --group-name k8sClusterAdmin --policy-name k8sClusterAdmin-policy

aws iam delete-group-policy --group-name k8sTeamADev --policy-name k8sTeamADev-policy

aws iam delete-group-policy --group-name k8sTeamATest --policy-name k8sTeamATest-policy

aws iam delete-group --group-name k8sClusterAdmin

aws iam delete-group --group-name k8sTeamADev

aws iam delete-group --group-name k8sTeamATest

aws iam delete-access-key --user-name User1Admin --access-key-id=$(jq -r .AccessKey.AccessKeyId /tmp/User1Admin.json)

aws iam delete-access-key --user-name User1TeamADev --access-key-id=$(jq -r .AccessKey.AccessKeyId /tmp/User1TeamADev.json)

aws iam delete-access-key --user-name User1TeamATest --access-key-id=$(jq -r .AccessKey.AccessKeyId /tmp/User1TeamATest.json)

aws iam delete-user --user-name User1Admin

aws iam delete-user --user-name User1TeamADev

aws iam delete-user --user-name User1TeamATest

aws iam delete-role --role-name k8sClusterAdmin

aws iam delete-role --role-name k8sTeamADev

aws iam delete-role --role-name k8sTeamATest

rm /tmp/\*.json

rm /tmp/kubeconfig\*

# reset aws credentials and config files

rm ~/.aws/{config,credentials}

aws configure set default.region ${AWS\_REGION}