EKS Kubernetes Access to New IAM User

Problem Statement:

As per the requirements of my client (BMW) I've setup AWS EKS cluster. Now they want to enable the access to Dev-IAM User to access this Cluster. We can provide any kind of access to the IAM user, but he supposed to manage the cluster in all aspects (full control).

Solution

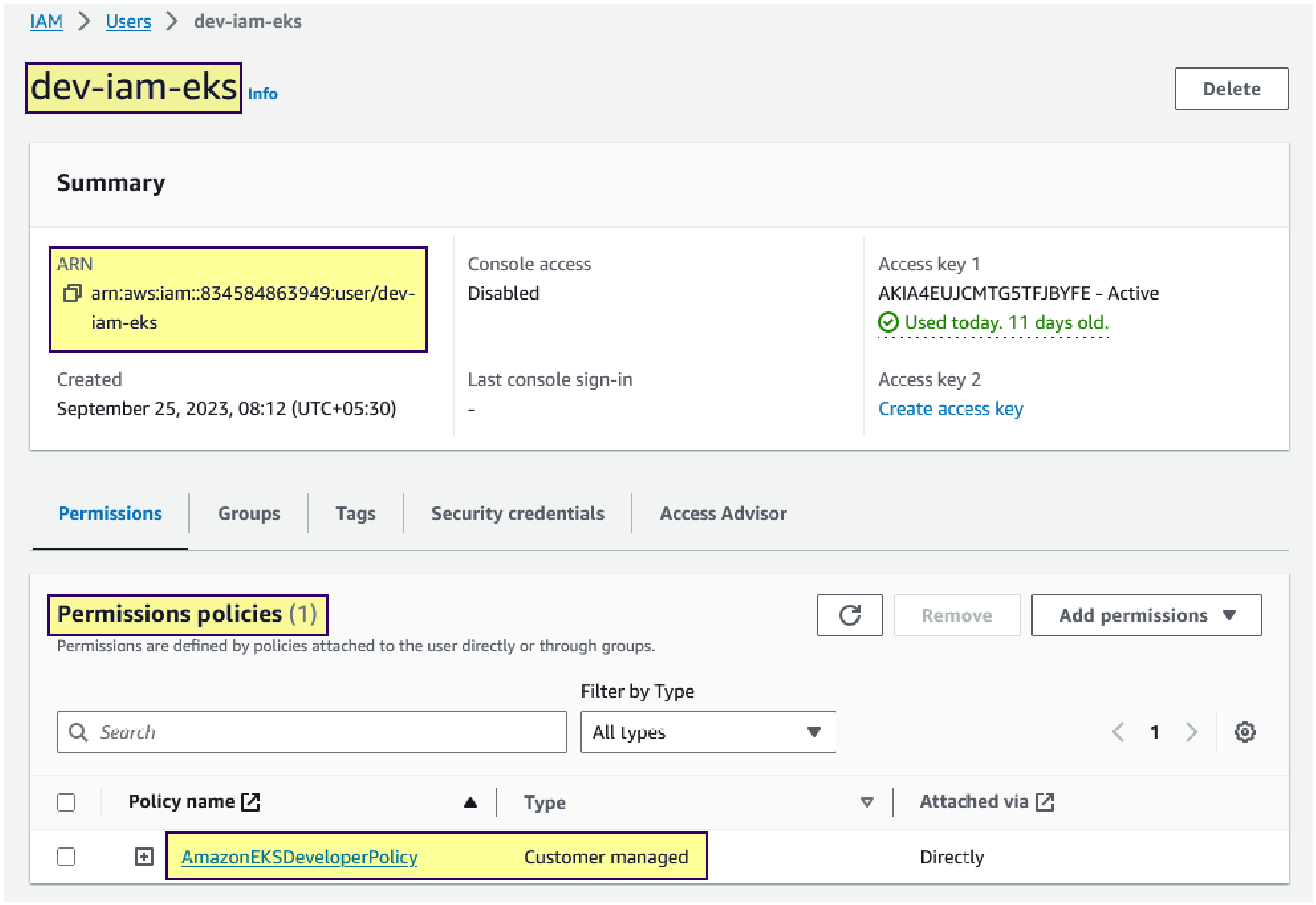
By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. When we create an Amazon EKS cluster, the IAM principal (IAM Users/IAM Groups/AWS Services) that creates the cluster is automatically granted system:masters permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This principal doesn't appear in any visible configuration.

In Kubernetes system:masters is a built-in RBAC group which allows for unrestricted access to the Kubernetes API server without the need for any Roles or RoleBindings. Also there are other built-in RBAC groups that represent different levels of access. To grant additional IAM principals the ability to interact with our cluster, we have to edit the aws-auth ConfigMap in kube-system namespace within our EKS cluster and create a Kubernetes RoleBinding or ClusterRoleBinding with the name of a group that we specify in the aws-auth ConfigMap. For granting specific permission we can also create a custom ClusterRole as well.

In Amazon Elastic Kubernetes Service (EKS), the aws-auth ConfigMap is a Kubernetes configuration resource used to map IAM (Identity and Access Management) roles and users to Kubernetes RBAC (Role-Based Access Control) roles within our EKS cluster. This mapping defines which AWS identities (IAM roles and users) have permissions to perform actions within the EKS cluster. Within the aws-auth ConfigMap, there are two primary sections: mapUsers and mapRoles. Each section specifies the IAM entities and their associated Kubernetes roles.

In the given scenario we have to include the new IAM user entity details in mapUsers section of the aws-auth configmap with system:masters permission in order to provide full access to the target EKS cluster as per client requirement.

So firstly we have to create a new AWS IAM user for our client and provide AWS CLI access. Also we created a customer managed policy named AmazonEKSDeveloperPolicy and attached it to the dev user.



IAM > Users > dev-iam-eks

dev-iam-eks Info

Summary

ARN	Console access	Access key 1
arn:aws:iam::834584863949:user/dev-iam-eks	Disabled	AKIA4EUJCMTG5TFJBYFE - Active Used today. 11 days old.
Created	Last console sign-in	Access key 2
September 25, 2023, 08:12 (UTC+05:30)	-	Create access key

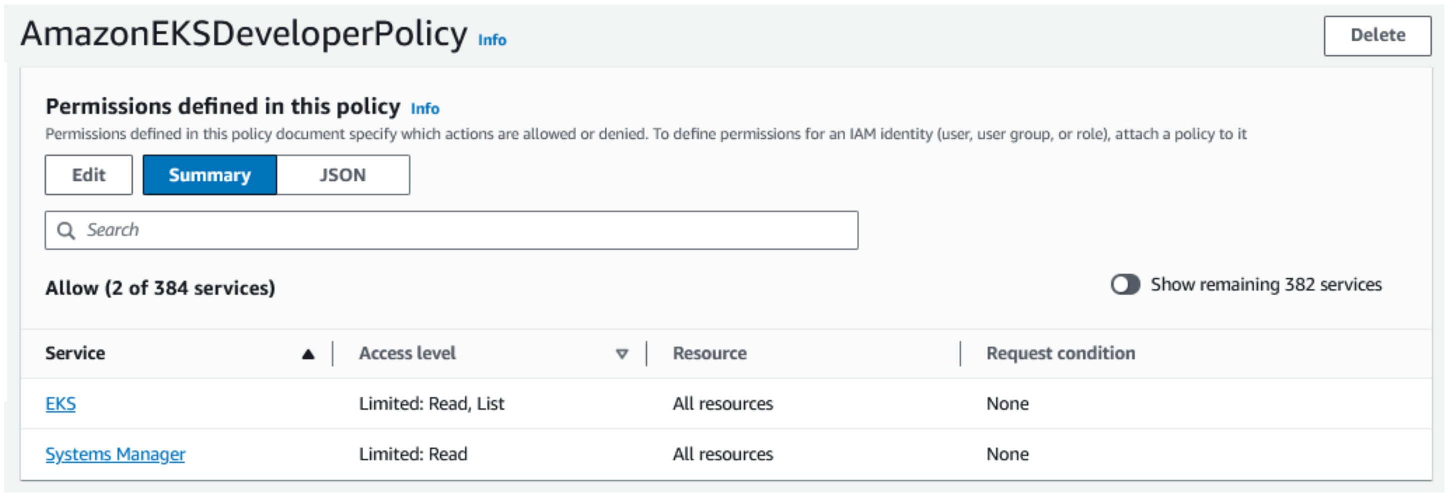
Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

Policy name	Type	Attached via
AmazonEKSDeveloperPolicy	Customer managed	Directly

The customer managed policy named AmazonEKSDeveloperPolicy has read access to Kubernetes API and several other EKS resources. This policy is required to provide visibility to the cluster nodes and workloads for the said user.



AmazonEKSDeveloperPolicy Info

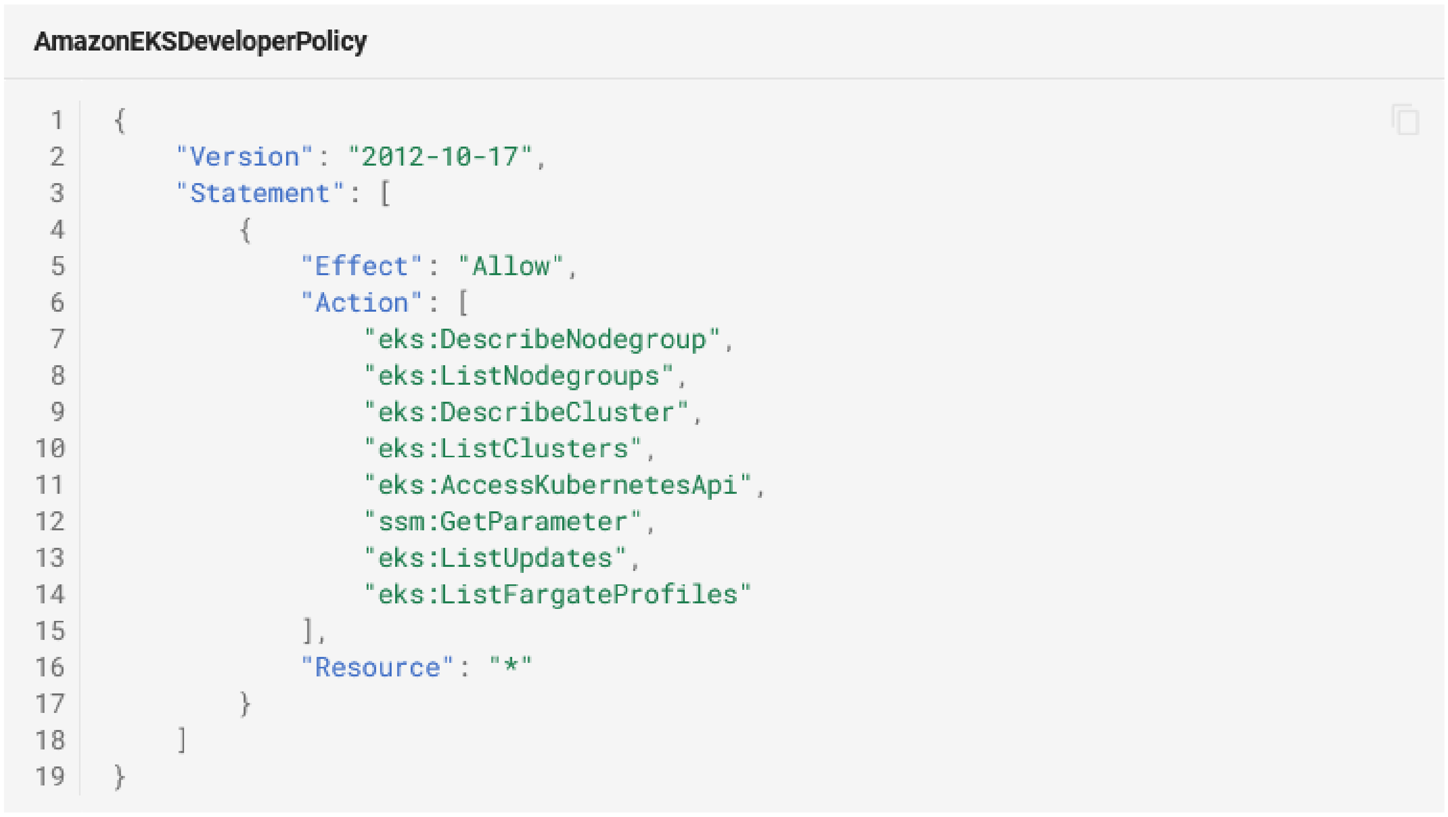
Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (2 of 384 services)

Show remaining 382 services

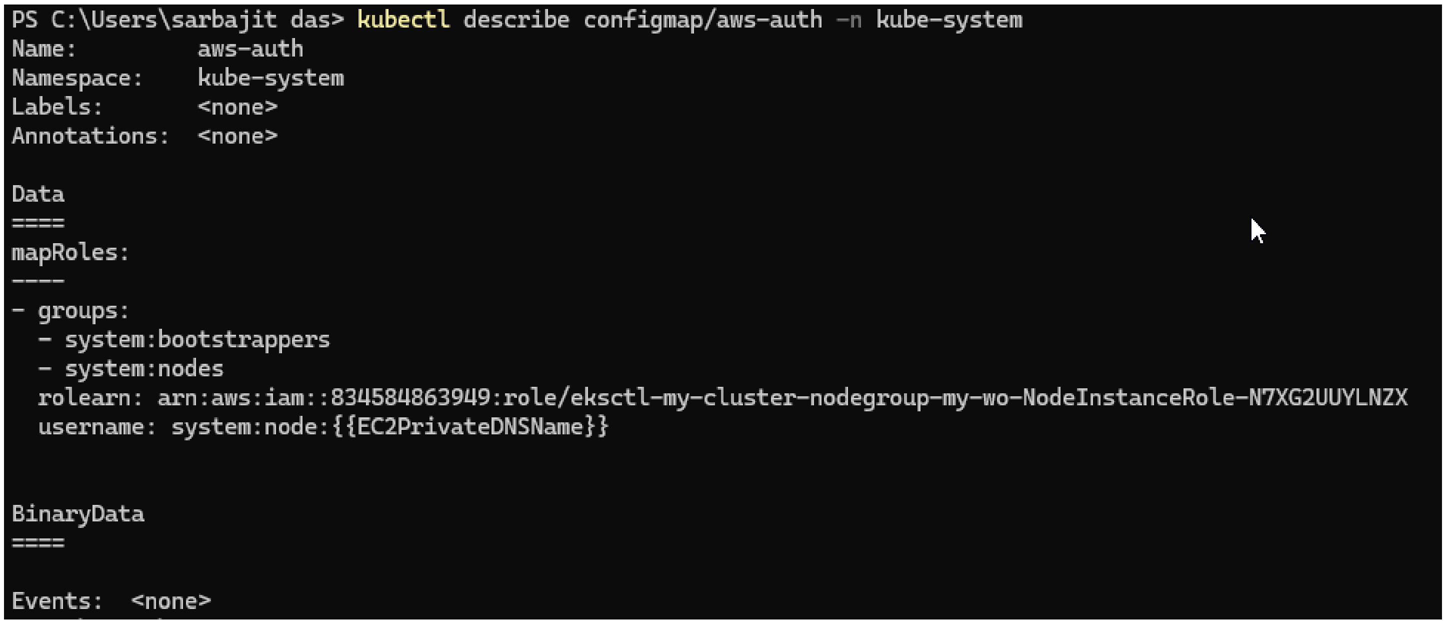
Service	Access level	Resource	Request condition
EKS	Limited: Read, List	All resources	None
Systems Manager	Limited: Read	All resources	None



AmazonEKSDeveloperPolicy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:DescribeCluster",
                "eks:ListClusters",
                "eks:AccessKubernetesApi",
                "ssm:GetParameter",
                "eks:ListUpdates",
                "eks:ListFargateProfiles"
            ],
            "Resource": "*"
        }
    ]
}
```

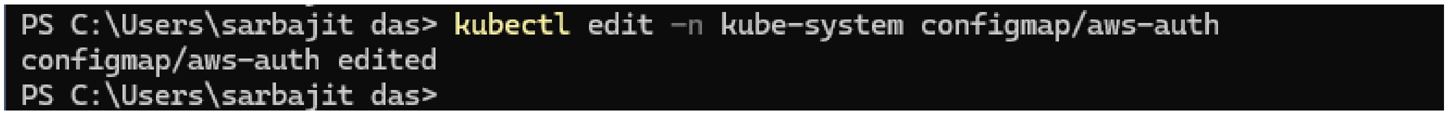
Now we can check the current configuration of the aws-auth configmap before making any changes to it.



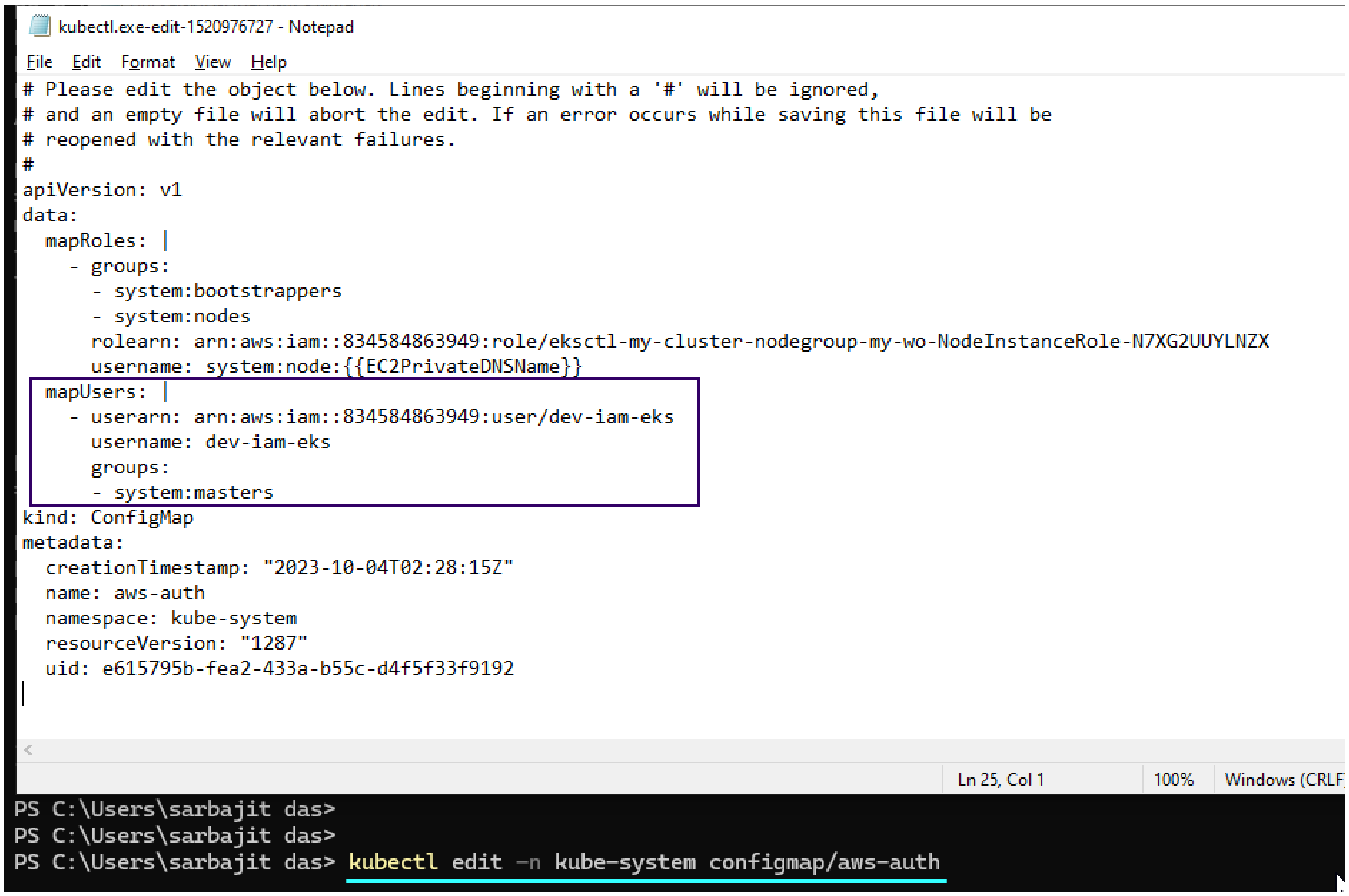
```
PS C:\Users\sarbajit das> kubectl describe configmap/aws-auth -n kube-system
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::834584863949:role/eksctl-my-cluster-nodegroup-my-wo-NodeInstanceRole-N7XG2UUYLNZX
  username: system:node:{{EC2PrivateDNSName}}


BinaryData
====

Events:  <none>
```

At next we have to edit the aws-auth configmap using following command:

kubectl edit -n kube-system configmap/aws-auth

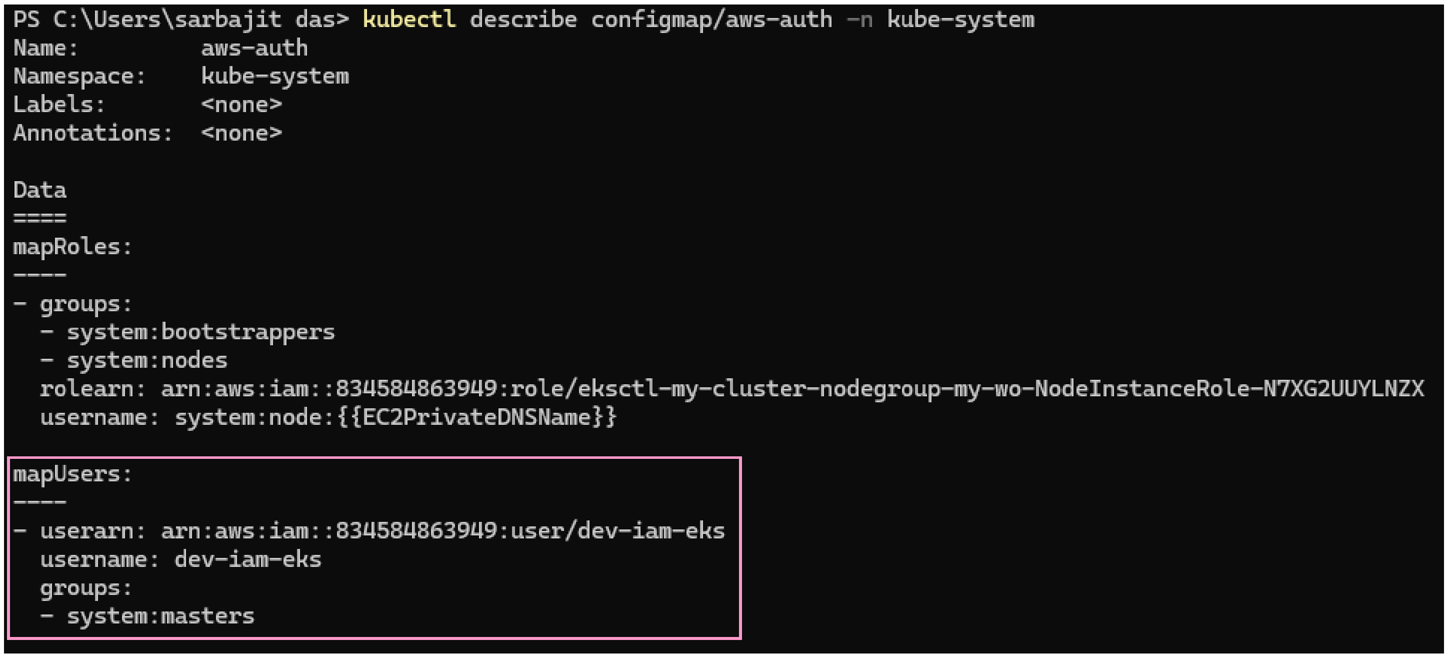


```
PS C:\Users\sarbajit das> kubectl edit -n kube-system configmap/aws-auth
configmap/aws-auth edited
PS C:\Users\sarbajit das>
```


kubectl.exe-edit-1520976727 - Notepad
File Edit Format View Help
Ln 25, Col 1
100%
Windows (CRLF

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::834584863949:role/eksctl-my-cluster-nodegroup-my-wo-NodeInstanceRole-N7XG2UUYLNZX
      username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::834584863949:user/dev-iam-eks
      username: dev-iam-eks
      groups:
      - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "2023-10-04T02:28:15Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "1287"
  uid: e615795b-fea2-433a-b55c-d4f5f33f9192
```

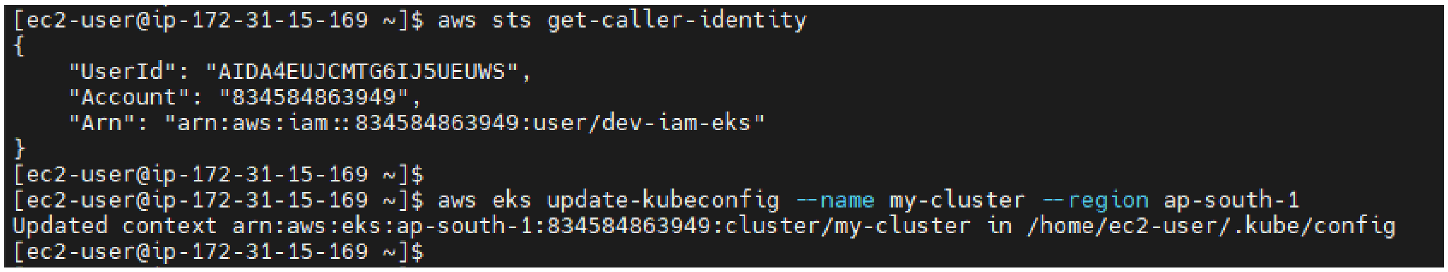
```
PS C:\Users\sarbajit das>
PS C:\Users\sarbajit das>
PS C:\Users\sarbajit das> kubectl edit -n kube-system configmap/aws-auth
```



```
PS C:\Users\sarbajit das> kubectl describe configmap/aws-auth -n kube-system
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::834584863949:role/eksctl-my-cluster-nodegroup-my-wo-NodeInstanceRole-N7XG2UUYLNZX
  username: system:node:{{EC2PrivateDNSName}}

mapUsers:
----
- userarn: arn:aws:iam::834584863949:user/dev-iam-eks
  username: dev-iam-eks
  groups:
  - system:masters
```

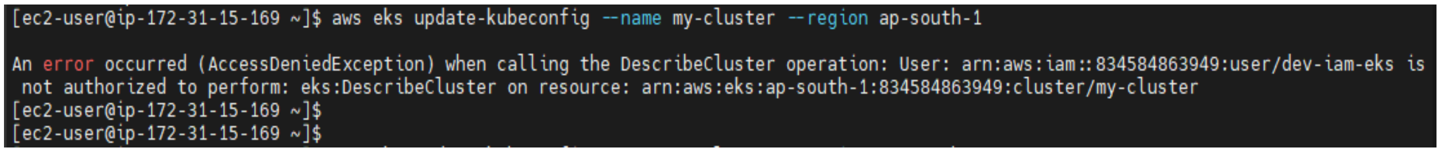
In the above snaps the highlighted segments displays the changes made to aws-auth configmap in our EKS cluster.

Finally we have to update the kubeconfig file in the ec2 instance from where the newly created IAM user is intending to execute all the cluster related commands.



```
[ec2-user@ip-172-31-15-169 ~]$ aws sts get-caller-identity
{
    "UserId": "AIDA4EUJCMTG6IJ5UEUWS",
    "Account": "834584863949",
    "Arn": "arn:aws:iam::834584863949:user/dev-iam-eks"
}
[ec2-user@ip-172-31-15-169 ~]$
[ec2-user@ip-172-31-15-169 ~]$ aws eks update-kubeconfig --name my-cluster --region ap-south-1
Updated context arn:aws:eks:ap-south-1:834584863949:cluster/my-cluster in /home/ec2-user/.kube/config
[ec2-user@ip-172-31-15-169 ~]$
```

Note: If the AmazonEKSDeveloperPolicy was not attached to the IAM user then we would have encountered an AccessDeniedException while trying to update the kubeconfig file.



```
[ec2-user@ip-172-31-15-169 ~]$ aws eks update-kubeconfig --name my-cluster --region ap-south-1

An error occurred (AccessDeniedException) when calling the DescribeCluster operation: User: arn:aws:iam::834584863949:user/dev-iam-eks is
 not authorized to perform: eks:DescribeCluster on resource: arn:aws:eks:ap-south-1:834584863949:cluster/my-cluster
[ec2-user@ip-172-31-15-169 ~]$
[ec2-user@ip-172-31-15-169 ~]$
```

API Verbs

In Kubernetes, an API verb is a term used to describe the action or operation that you want to perform on a resource. API verbs are used when interacting with the Kubernetes API server to create, read, update, or delete resources within the cluster. They define the desired operation on a resource and are part of the HTTP request that is sent to the API server.

There are different API verbs in Kubernetes, for example - GET, POST, PUT, PATCH, and DELETE etc. These API verbs are part of the Kubernetes RESTful API, and they provide a standard way to interact with Kubernetes resources. When using tools like kubectl or making API requests programmatically, you specify the verb along with the resource type and name to perform actions on resources within your Kubernetes cluster.

Checking API Access :: kubectl auth can-i <API_VERB> <RESOURE_TYPE>

Kubectl provides the auth can-i subcommand for quickly querying the API authorization layer. It is a powerful utility in Kubernetes that allows you to check whether a particular user or service account has permission to perform a specific action (api verb) on a specific resource within the Kubernetes cluster. It's a valuable tool for verifying access control and troubleshooting access-related issues in your Kubernetes environment.

The "kubectl auth can-i" command uses the SelfSubjectAccessReview API to determine if the current user can perform a given action, and works regardless of the authorization mode used. SelfSubjectAccessReview is part of the authorization.k8s.io API group, which exposes the API server authorization to external services.

So after providing RBAC access to the new user we can quickly verifying access control and permission using the auth can-i command.

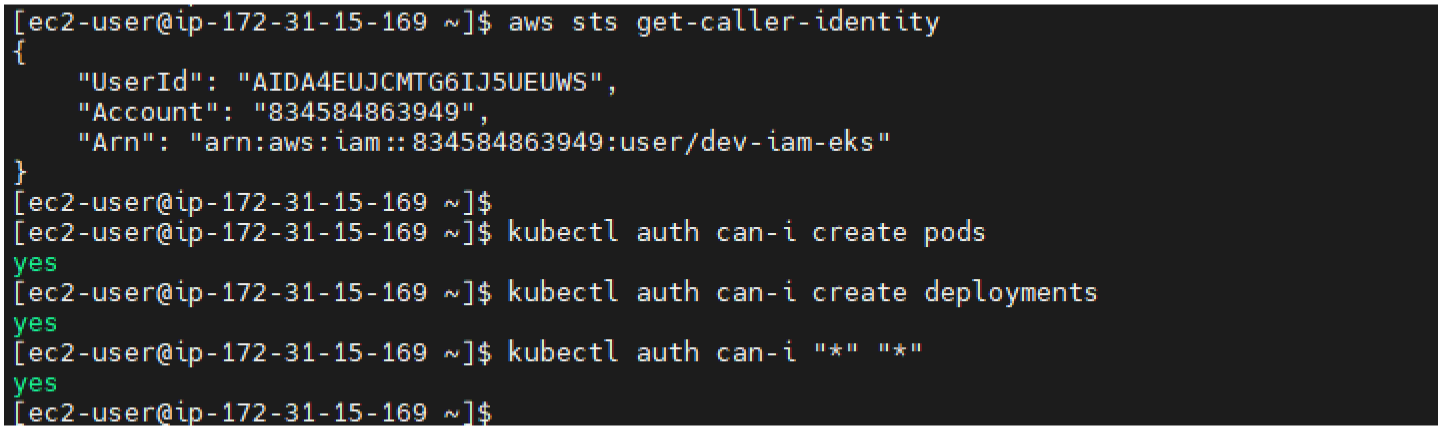
Syntax → **kubectl auth can-i <API_VERB> <RESOURE_TYPE>**

The above mentioned command can return only one of the following results:

Yes → Indicates that the user/service account has permission to perform the specified action on the specified resource.

No → Indicates that the user/service account doesn't have permission to perform the specified action on the specified resource.

Unknown → Indicates that the permission status cannot be determined due to missing or incomplete RBAC (Role-Based Access Control) configuration. This result may also occur if the specified resource does not exist in the cluster.



```
[ec2-user@ip-172-31-15-169 ~]$ aws sts get-caller-identity
{
    "UserId": "AIDA4EUJCMTG6IJ5UEUWS",
    "Account": "834584863949",
    "Arn": "arn:aws:iam::834584863949:user/dev-iam-eks"
}
[ec2-user@ip-172-31-15-169 ~]$
[ec2-user@ip-172-31-15-169 ~]$ kubectl auth can-i create pods
yes
[ec2-user@ip-172-31-15-169 ~]$ kubectl auth can-i create deployments
yes
[ec2-user@ip-172-31-15-169 ~]$ kubectl auth can-i "*" "*"
yes
[ec2-user@ip-172-31-15-169 ~]$
```

As per the above given snapshot the newly created user got positive response from API. We can also execute kubectl auth can-i "*" "*" command which is used to check if the authenticated user or service account has permission to perform any action within the Kubernetes cluster. In this case we also got positive response from API because we have given system:masters permission for the newly created dev user which gives full control of the cluster to the user.