# Amazon EKS Pod Identity

## [Granting AWS IAM permissions to workloads on Amazon EKS clusters](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity" \l "granting-aws-iam-permissions-to-workloads-on-amazon-eks-clusters)

Amazon EKS provides 2 ways to grant AWS IAM permissions to workloads that run in Amazon EKS clusters.

**1. IAM roles for service accounts**

IAM roles for service accounts (IRSA) configures Kubernetes applications running on AWS with fine-grained IAM permissions to access various AWS resources. IRSA was build to support various Kubernetes deployment options supported by AWS such as Amazon EKS, Amazon EKS Anywhere, Red Hat OpenShift Service on AWS, and self managed Kubernetes clusters on Amazon EC2 instances. Thus, IRSA was build using foundational AWS service like IAM, and did not take any direct dependency on the Amazon EKS service and the EKS API. For more information, see [IAM roles for service accounts](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html).

**2. EKS Pod Identities**

EKS Pod Identity offers cluster administrators a simplified workflow for authenticating applications to access various AWS resources. EKS Pod Identity is for EKS only, and as a result, it simplifies how cluster administrators can configure Kubernetes applications to obtain IAM permissions. For more information, see [EKS Pod Identities](https://docs.aws.amazon.com/eks/latest/userguide/pod-identities.html).

## [Introduction to Amazon EKS Pod Identity](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity" \l "introduction-to-amazon-eks-pod-identity)

Amazon EKS supports EKS Pod Identity, a new feature that simplifies how cluster administrators can configure Kubernetes applications to obtain AWS [IAM](https://aws.amazon.com/iam/) permissions. These permissions can now be easily configured with fewer steps directly through EKS console, APIs, and CLI.

EKS Pod Identity supports following features.

* Reuse IAM role across multiple EKS clusters
* Reuse of permission policies across IAM roles
* Allow access to AWS resources based on matching tags

EKS Pod Identity offers cluster administrators a simplified workflow for authenticating applications to all AWS resources such as Amazon S3 buckets, Amazon DynamoDB tables, and more. As a result, cluster administrators need not switch between the EKS and IAM services, or execute privileged IAM operations to configure permissions required by your applications. IAM roles can now be used across multiple clusters without the need to update the role trust policy when creating new clusters. IAM credentials supplied by EKS Pod Identity include support for [role session tags](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_tags.html), with support for attributes such as cluster name, namespace, service account name. Role session tags enable administrators to author a single role that can work across service accounts by allowing access to AWS resources based on matching tags.

## [How does Amazon EKS Pod Identity work?](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity" \l "how-does-amazon-eks-pod-identity-work)

EKS Pod Identity makes it easier to configure and automate granting AWS permissions to Kubernetes identities. As the cluster administrator, you no longer need to switch between Amazon EKS and IAM services to authenticate your applications to all AWS resources. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you associate an IAM role with a Kubernetes service account and configure your Pods to use the service account.

The overall workflow to use Amazon EKS Pod Identity consiste of below steps.

1. Create IAM role with required permissions and specify pods.eks.amazonaws.com as the service principal in trust policy.
2. Install Amazon EKS Pod Identity Agent add-on
3. Map the role to a service account directly.

Now, any new pods that use that service account will automatically be configured to receive IAM credentials. Each EKS Pod Identity association maps a role to a service account in a namespace in the specified cluster. If you have the same application in multiple clusters, you can make identical associations in each cluster without modifying the trust policy of the role. If a pod uses a service account that has an association, Amazon EKS sets environment variables in the containers of the pod. The environment variables configure the AWS SDKs, including the AWS CLI, to use the EKS Pod Identity credentials.

## [Comparing EKS Pod Identity and IRSA](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity" \l "comparing-eks-pod-identity-and-irsa)

As part of this workshop we will cover the following modules:

|  | **EKS Pod Identity** | **IRSA** |
| --- | --- | --- |
| **Role extensibility** | No need to update the role's trust policy for each new cluster. | Need to update role's trust policy with new EKS cluster OIDC provider endpoint |
| **Cluster scalability** | No need to setup IAM OIDC provider | Need to setup IAM OIDC provider. default global limit of 100 OIDC providers for AWS account applies |
| **Role scalability** | No need to define trust relationship between IAM role and service account in the trust policy | Need to define trust relationship between IAM role and service account in the trust policy. max of 8 trust relationships within a single trust policy applies due to limit on trust policy size |
| **Role reusability** | AWS STS temporary credentials supplied by EKS Pod Identity include role session tags, such as cluster name, namespace, service account name. | AWS STS session tags are not supported. You can reuse a role between clusters but every pod receives all of the permissions of the role |
| **Environments supported** | Only available on Amazon EKS | IRSA can be used such as Amazon EKS, Amazon EKS Anywhere, Red Hat OpenShift Service on AWS, and self managed Kubernetes clusters on Amazon EC2 instances. |
| **EKS versions supported** | EKS Kubernetes versions 1.24 or later. | All of the supported EKS cluster versions. |

Enable EKS Pod Identity

**[Deploy a Sample App without EKS Pod Identity](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "deploy-a-sample-app-without-eks-pod-identity)**

Before we configure the EKS Pod Identity feature, let us first deploy a Sample App and test if can access list of S3 buckets or not.

**[Create a App template file](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "create-a-app-template-file)**

Run below command to generate a App template file, which we will be using throughout this workshop module.

cat > ~/environment/app-template.yaml <<EOF

apiVersion: v1

kind: Namespace

metadata:

name: \$NS

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: \$SA

namespace: \$NS

---

apiVersion: v1

kind: Pod

metadata:

name: \$APP

namespace: \$NS

labels:

app: \$APP

spec:

serviceAccountName: \$SA

containers:

- name: \$APP

image: amazon/aws-cli:latest

command: ['sleep', '36000']

restartPolicy: Never

EOF

**[Deploy a Sample App](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "deploy-a-sample-app)**

Run below command to deploy a Sample App app1 with a Kubernetes Service account sa1 in Namspace ns-a.

export APP=app1

export NS=ns-a

export SA=sa1

envsubst < ~/environment/app-template.yaml > ~/environment/$APP.yaml

kubectl apply -f ~/environment/$APP.yaml

Check Output

namespace/ns-a created

serviceaccount/sa1 created

pod/app1 created

Check if the Pod is running fine.

kubectl -n $NS get pod

Check Output

NAME READY STATUS RESTARTS AGE

app1 1/1 Running 0 2m29s

Check if the Pod can access any S3 Buckets.

kubectl -n $NS exec -it $APP -- aws s3 ls

Check Output

An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied

command terminated with exit code 254

The AccessDenied error is expected since the Pod is not confugured with any IAM permissions to list S3 Buckets.

**[Configure Amazon EKS Pod Identity](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "configure-amazon-eks-pod-identity)**

**[Step1: Create an IAM Role](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "step1:-create-an-iam-role)**

Create a trust policy and configure the principal to pods.eks.amazonaws.com

export IAM\_ROLE="eks-pod-s3-read-access-role"

export IAM\_ROLE\_TRUST\_POLICY="eks-pod-s3-read-access-trust-policy"

export IAM\_POLICY="eks-pod-s3-read-access-policy"

export ROLE\_DESCRIPTION="To allow Kubernetes Pods to allow readonly acces to S3"

cat > ~/environment/$IAM\_ROLE\_TRUST\_POLICY.json << EOF

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "pods.eks.amazonaws.com"

},

"Action": [

"sts:AssumeRole",

"sts:TagSession"

]

}

]

}

EOF

Using above trust policy, create the IAM 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 file://~/environment/$IAM\_ROLE\_TRUST\_POLICY.json \

--output text \

--query 'Role.Arn')

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

else

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

fi

Check Output

An error occurred (NoSuchEntity) when calling the GetRole operation: The role with name eks-pod-s3-read-access-role cannot be found.

IAM Role eks-pod-s3-read-access-role created. IAM\_ROLE\_ARN=arn:aws:iam::ACCOUNT\_ID:role/eks-pod-s3-read-access-role

Let us create a custom IAM Policy for S3 to list buckets and get Objects.

cat > ~/environment/$IAM\_POLICY.json <<EOF

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:ListAllMyBuckets"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:GetObjectTagging"

],

"Resource": "\*"

}

]

}

EOF

Create the IAM Policy.

policyArn=$(aws iam create-policy --policy-name $IAM\_POLICY --policy-document file://~/environment/$IAM\_POLICY.json --output text --query Policy.Arn)

echo "policyArn=$policyArn"

Check Output

policyArn=arn:aws:iam::ACCOUNT\_ID:policy/eks-pod-s3-read-access-policy

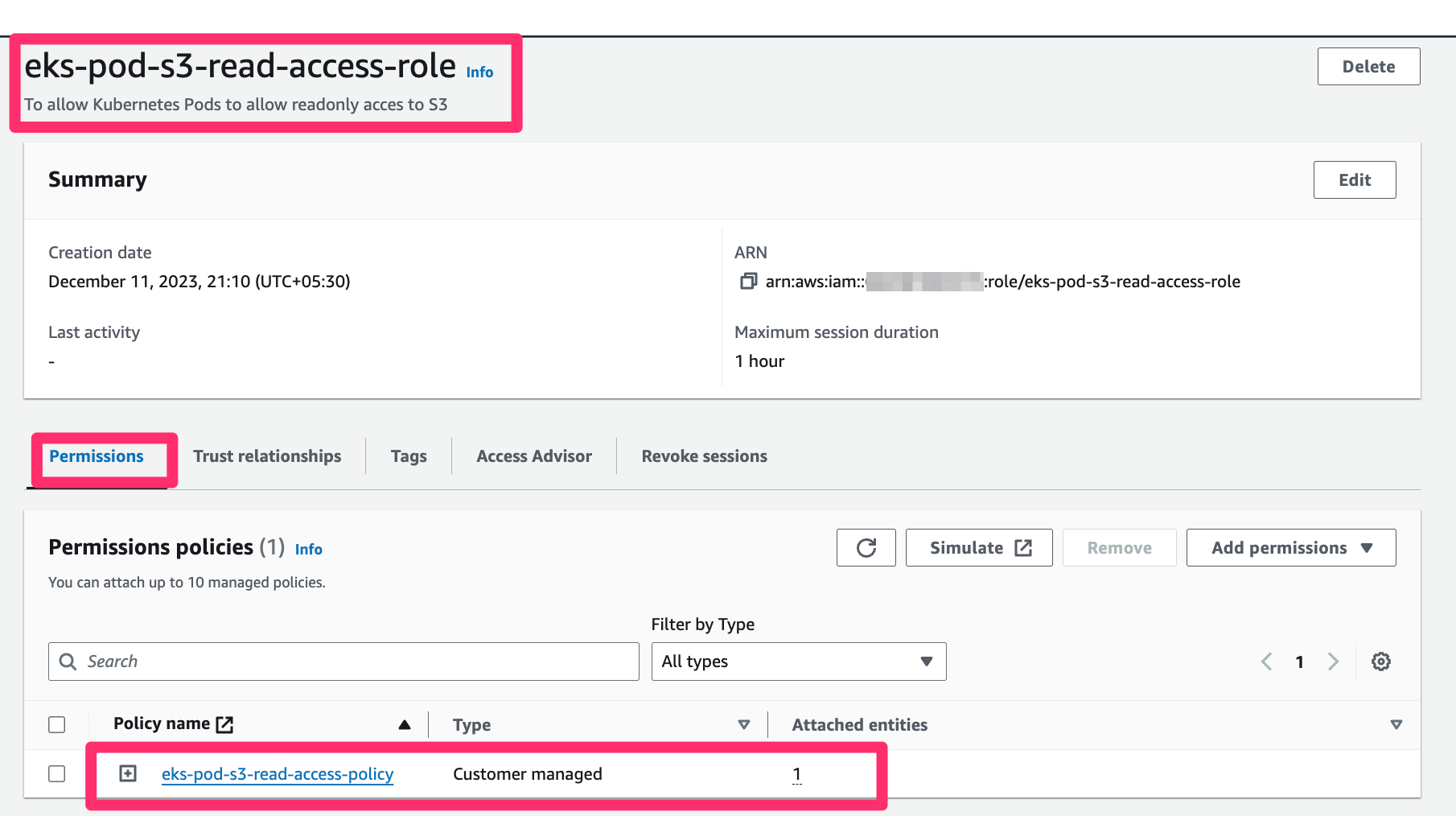
Attache the policy to the IAM Role.

aws iam attach-role-policy \

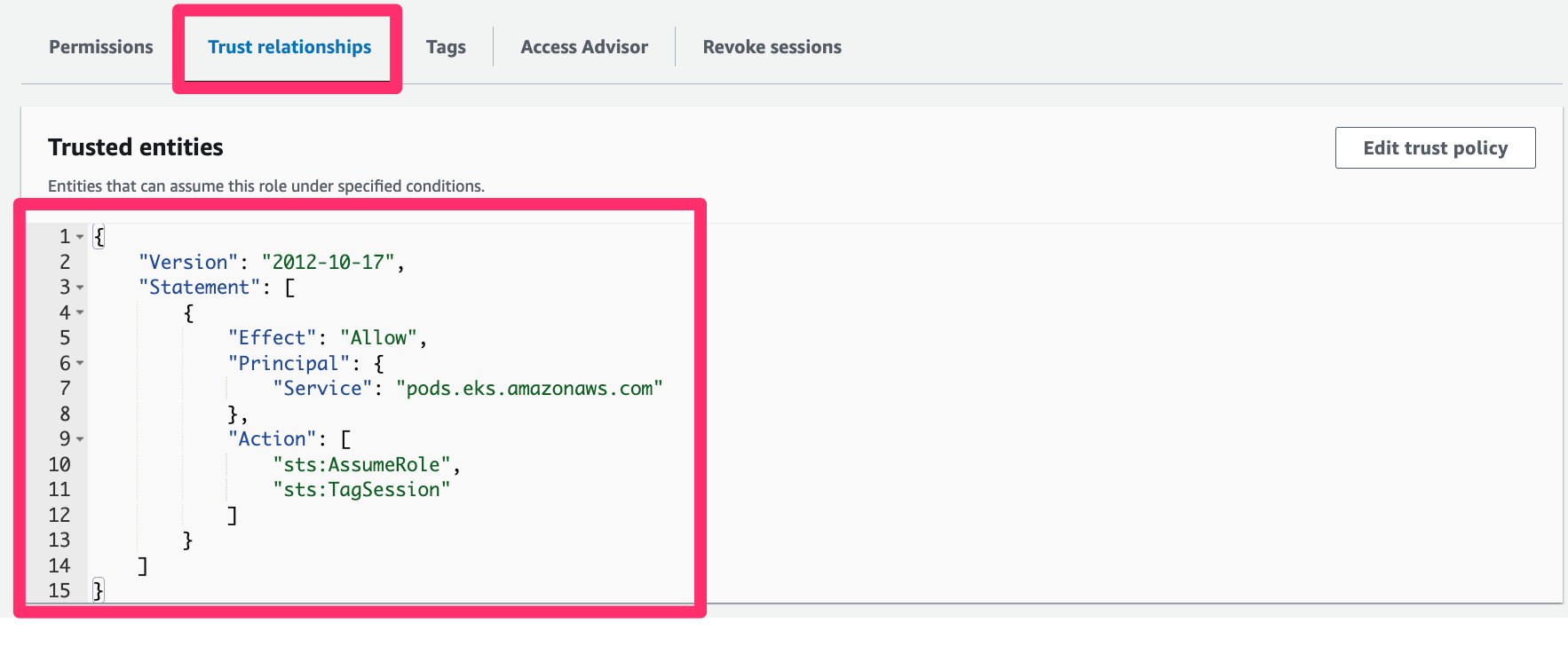
--role-name $IAM\_ROLE \

--policy-arn $policyArn

Go to IAM Console and view the IAM Role.



Look at the trust policy.



**[Step2: Add Amazon EKS Pod Identity Agent add-on](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "step2:-add-amazon-eks-pod-identity-agent-add-on)**

1

2

export EKS\_POD\_IDENTITY\_ADDON\_NAME="eks-pod-identity-agent"

aws eks create-addon --cluster-name $EKS\_CLUSTER\_NAME --addon-name $EKS\_POD\_IDENTITY\_ADDON\_NAME

Check Output

{

"addon": {

"addonName": "eks-pod-identity-agent",

"clusterName": "eksworkshop-eksctl",

"status": "CREATING",

"addonVersion": "v1.0.0-eksbuild.1",

"health": {

"issues": []

},

"addonArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:addon/eksworkshop-eksctl/eks-pod-identity-agent/3cc62ccc-4901-453e-3659-e23c441f1d1b",

"createdAt": "2023-12-11T15:57:00.387000+00:00",

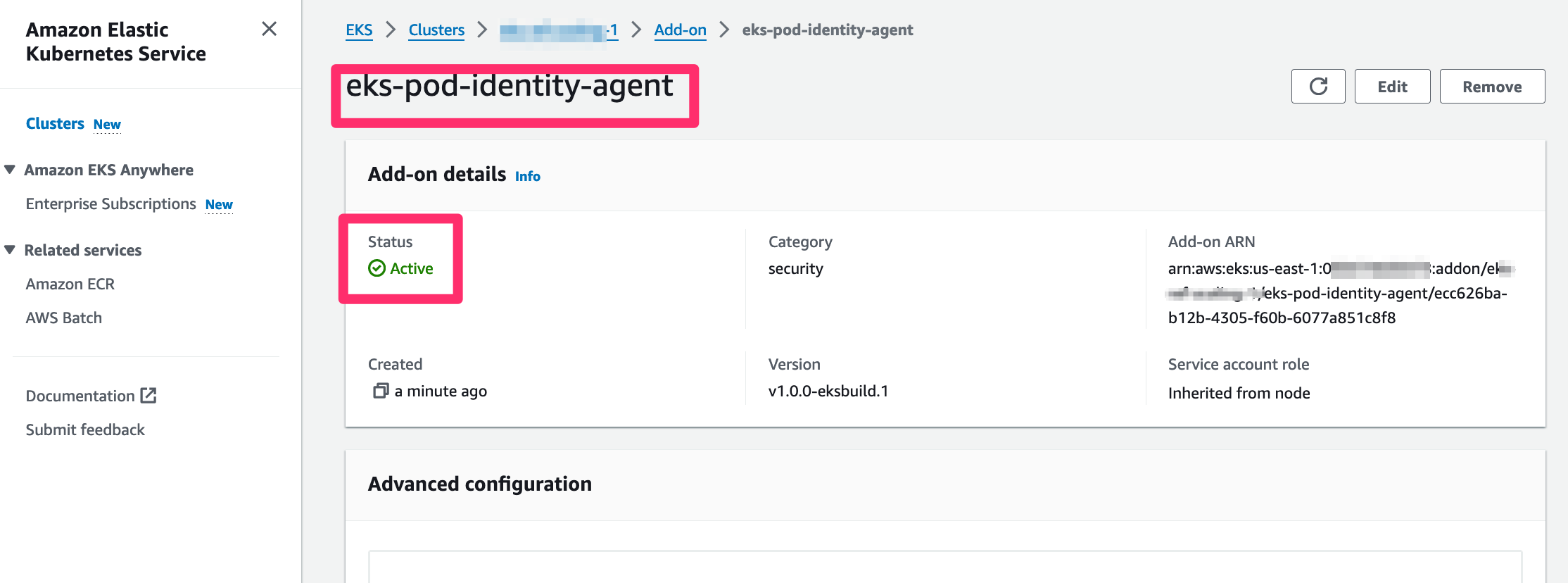
"modifiedAt": "2023-12-11T15:57:00.402000+00:00",

"tags": {}

}

}

Go to EKS Console and view the eks-pod-identity-agent under the **Add-on** tab.



**[Step3: Create Pod Identity association](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/1-eks-pod-identity-config" \l "step3:-create-pod-identity-association)**

Create the EKS Pod Identity association for the Service account sa1 in Namespace ns-a for the IAM Role eks-pod-s3-read-access-role.

aws eks create-pod-identity-association \

--cluster-name $EKS\_CLUSTER\_NAME \

--namespace $NS \

--service-account $SA \

--role-arn $IAM\_ROLE\_ARN

Check Output

{

"association": {

"clusterName": "eksworkshop-eksctl",

"namespace": "ns-a",

"serviceAccount": "sa1",

"roleArn": "arn:aws:iam::ACCOUNT\_ID:role/eks-pod-s3-read-access-role",

"associationArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:podidentityassociation/eksworkshop-eksctl/a-fyvwurwka5tyvgnmd",

"associationId": "a-fyvwurwka5tyvgnmd",

"tags": {},

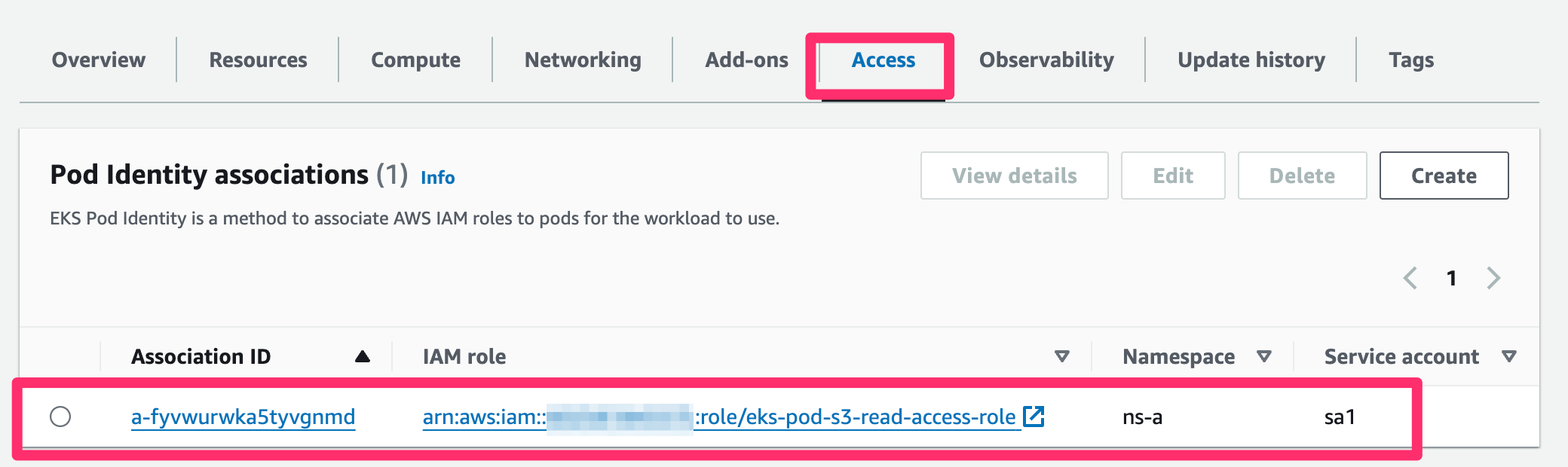
"createdAt": "2023-12-11T17:34:29.617000+00:00",

"modifiedAt": "2023-12-11T17:34:29.617000+00:00"

}

}

Go to EKS Console and view the Pod Identity associations under the **Access** tab.



We can get the list of current EKS Pod Identity associations using below API.

aws eks list-pod-identity-associations --cluster-name $EKS\_CLUSTER\_NAME

Check Output

{

"associations": [

{

"clusterName": "eksworkshop-eksctl",

"namespace": "ns-a",

"serviceAccount": "sa1",

"associationArn": "arn:aws:eks:us-east-1:ACCOUNT\_ID:podidentityassociation/eksworkshop-eksctl/a-1gdi6ws8nzackqyvg",

"associationId": "a-1gdi6ws8nzackqyvg"

}

]

}

EKS also provides few more [commands](https://docs.aws.amazon.com/cli/latest/reference/eks/#cli-aws-eks) to manage the Pod Identity association such delete-pod-identity-association, describe-pod-identity-association and update-pod-identity-association. You can use the update-pod-identity-association to update the IAM Role anytime.

The namespace and service account cannot be edited. To change them, delete the association and create a new association.

# Test EKS Pod Identity

In this section, we test EKS Pod Identity feature to grant access to a Pod to list S3 buckets.

## [Test Amazon EKS Pod Identity](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/2-eks-pod-identity-test" \l "test-amazon-eks-pod-identity)

### [Create S3 bucket](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/2-eks-pod-identity-test" \l "create-s3-bucket)

First let's create a Amazon S3 bucket

1

2

export S3\_BUCKET="ekspodidentity-$ACCOUNT\_ID-$AWS\_REGION"

aws s3 mb s3://$S3\_BUCKET --region $AWS\_REGION

Check Output

1

make\_bucket: ekspodidentity-ACCOUNT\_ID-us-west-2

### [Test S3 read access](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/2-eks-pod-identity-test" \l "test-s3-read-access)

Test the S3 access now.

1

kubectl -n $NS exec -it $APP -- aws s3 ls

Check Output

1

2

An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied

command terminated with exit code 254

**Why is the Pod still getting**AccessDenied**error?**

This is because the Pod Identity association was created after the Pod was created. So let's re-deploy the Pod again.

Delete the Pod and re-create it.

1

2

kubectl -n $NS delete pod $APP --force --grace-period=0

kubectl apply -f ~/environment/$APP.yaml

Check Output

namespace/ns-a unchanged

serviceaccount/sa1 unchanged

pod/app1 created

Test the S3 access again.

1

kubectl -n $NS exec -it $APP -- aws s3 ls

Check Output

1

2023-12-12 05:28:12 ekspodidentity-ACCOUNT\_ID-us-west-2

The App is now able to access the list of S3 Buckets.

# Deep Dive into EKS Pod Identity

## [Deep Dive into EKS Pod Identity](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "deep-dive-into-eks-pod-identity)

In this section, let's deep dive into EKS Pod Identity and understand what is happening under the hood.

### [Stage 1: During the Pod Creation](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "stage-1:-during-the-pod-creation)

In the previous section, we created an IAM Role eks-pod-s3-read-access-role and called an API create-pod-identity-association to create an association between the IAM role and kubernetes service account sa1 in the Namespace ns-a.

When Amazon EKS starts a new pod that uses a service account with an EKS Pod Identity association, the [EKS Pod Identity webhook](https://github.com/aws/amazon-eks-pod-identity-webhookhttps:/github.com/aws/amazon-eks-pod-identity-webhook) mutates the pod spec by adding two environment variables AWS\_CONTAINER\_CREDENTIALS\_FULL\_URI and AWS\_CONTAINER\_AUTHORIZATION\_TOKEN\_FILE.

This is because EKS Pod Identities have been added to the [Container credential provider](https://docs.aws.amazon.com/sdkref/latest/guide/feature-container-credentials.html) which is searched by AWS SDKs in a step in the default credential chain.

Let us see the Pod spec and look for these variables.

1

kubectl -n $NS get pod $APP -oyaml

Check Output

---

- name: AWS\_CONTAINER\_CREDENTIALS\_FULL\_URI

value: http://169.254.170.23/v1/credentials

- name: AWS\_CONTAINER\_AUTHORIZATION\_TOKEN\_FILE

value: /var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token

image: amazon/aws-cli:latest

volumeMounts:

- mountPath: /var/run/secrets/kubernetes.io/serviceaccount

name: kube-api-access-mcz7j

readOnly: **true**

- mountPath: /var/run/secrets/pods.eks.amazonaws.com/serviceaccount

name: eks-pod-identity-token

readOnly: **true**

---

volumes:

- name: eks-pod-identity-token

projected:

defaultMode: 420

sources:

- serviceAccountToken:

audience: pods.eks.amazonaws.com

expirationSeconds: 86400

path: eks-pod-identity-token

- name: kube-api-access-mcz7j

projected:

defaultMode: 420

sources:

- serviceAccountToken:

expirationSeconds: 3607

path: token

- configMap:

items:

- key: ca.crt

path: ca.crt

name: kube-root-ca.crt

- downwardAPI:

items:

- fieldRef:

apiVersion: v1

fieldPath: metadata.namespace

path: namespace

---

Notice there are two Projected Service Account Tokens in the output. One of them is kube-api-access-mcz7j which is the default Service token created and injected by the API Server. The second one is eks-pod-identity-token is created and injected by the EKS Pod Identity webhook as explained above.

Let us exec into the Pod and see what does it contains.

kubectl -n $NS exec -it $APP -- bash

Run below commands from inside the Pod to view the EKS Pod Identity Token.

export EKS\_POD\_IDENTITY\_TOKEN=$(cat $AWS\_CONTAINER\_AUTHORIZATION\_TOKEN\_FILE)

echo $EKS\_POD\_IDENTITY\_TOKEN

Decoding this token at <https://jwt.io/>

The Header in the Token looks like below.

{

"alg": "RS256",

"kid": "63a691b676600ed2406d731ee9beedfb62bdd3b1"

}

The Payload in the Token looks like below.

{

"aud": [

"pods.eks.amazonaws.com"

],

"exp": 1702445428,

"iat": 1702359028,

"iss": "https://oidc.eks.us-west-2.amazonaws.com/id/8BA4A70AA33A68D27898EB4903D8A6E7",

"kubernetes.io": {

"namespace": "ns-a",

"pod": {

"name": "app1",

"uid": "fd6df8ef-d820-4a7a-92c5-97f91dafb80a"

},

"serviceaccount": {

"name": "sa1",

"uid": "aad7d4eb-6302-40f6-9f89-92f8293bca5e"

}

},

"nbf": 1702359028,

"sub": "system:serviceaccount:ns-a:sa1"

}

Let us understand few important fields in the above output.

**iss** : It represents the issuer of the token which is an OIDC Provider https://oidc.eks.us-west-2.amazonaws.com/id/8BA4A70AA33A68D27898EB4903D8A6E7. This OIDC provider URL will be used during the verification process of the token.

**aud**: It represents the audience of the token which is pods.eks.amazonaws.com. This is the EKS Pod Identity Service i.e. EKS Auth. This means the token will be accepted only by the EKS Auth Service and will be rejected by any other service.

**exp** and **iat** : These represents the expiry time for the token which basically enables the time bound tokens.

**kubernetes.io**: It represents that this token is bound to a very specific pod app1 with a Service account sa1 in the Namespace ns-a. This means, this token cannot be used by any other pod even with the same Service token and same Namespace.

### [Stage 2: During the call to S3 API to list Buckets](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "stage-2:-during-the-call-to-s3-api-to-list-buckets)

To list the S3 buckets, we used this command aws s3 ls in the previous section.

The AWS CLI searches for the IAM credentials in the default [Credential provider chain](https://docs.aws.amazon.com/sdkref/latest/guide/standardized-credentials.html). In the search, it finds the [Container credential provider](https://docs.aws.amazon.com/sdkref/latest/guide/feature-container-credentials.html) in the list. That means, it will make a call to HTTP URL endpoint mentioned in the container environment variable AWS\_CONTAINER\_CREDENTIALS\_FULL\_URI by passing EKS Pod Identity token in the Authorization header.

Let us make this call ourselves from inside the pod.

1

curl 169.254.170.23/v1/credentials -H "Authorization: $EKS\_POD\_IDENTITY\_TOKEN"

The output will like look below.

{

"AccessKeyId": "ASIAQAHCJ2QPOKXPLCQ4",

"SecretAccessKey": "UYEyaLMLoa0y6lx1FvpSzSwHJzZml7b9qiSRU2ry",

"Token": "",

"AccountId": "ACCOUNT\_ID",

"Expiration": "2023-12-12T12:50:44Z"

}

Run the command exit to exit from the pod

### [Stage 3: At the Amazon EKS Pod Identity Agent](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "stage-3:-at-the-amazon-eks-pod-identity-agent)

The EKS Pod Identity Agent runs as a Kubernetes DaemonSet and only provides credentials to pods on the node that it runs on. The EKS Pod Identity Agent uses the hostNetwork and uses port 80 and port 2703 on a link-local address on the node. This address is 169.254.170.23 for IPv4 and [fd00:ec2::23] for IPv6 clusters.

#### [EKS Pod Identity agent](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "eks-pod-identity-agent)

Let us see the EKS Pod Identity Agent pods running in the cluster.

kubectl -n kube-system get ds -lapp.kubernetes.io/name=eks-pod-identity-agent

Check Output

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE

eks-pod-identity-agent 3 3 3 3 3 <none> 3d1h

Let us also see the EKS Pod Identity Agent specification. Notice the hostNetwork: true in the specification.

kubectl -n kube-system get ds -lapp.kubernetes.io/name=eks-pod-identity-agent -oyaml

Check Output

apiVersion: v1

items:

- apiVersion: apps/v1

kind: DaemonSet

metadata:

annotations:

deprecated.daemonset.template.generation: "1"

creationTimestamp: "2023-12-09T07:23:09Z"

generation: 1

labels:

app.kubernetes.io/instance: eks-pod-identity-agent

app.kubernetes.io/managed-by: Helm

app.kubernetes.io/name: eks-pod-identity-agent

app.kubernetes.io/version: 0.0.25

helm.sh/chart: eks-pod-identity-agent-1.0.0

name: eks-pod-identity-agent

namespace: kube-system

resourceVersion: "6080325"

uid: f8dcd0a2-baf3-477b-bc47-fcf42bbd6d9e

spec:

revisionHistoryLimit: 10

selector:

matchLabels:

app.kubernetes.io/instance: eks-pod-identity-agent

app.kubernetes.io/name: eks-pod-identity-agent

template:

metadata:

creationTimestamp: **null**

labels:

app.kubernetes.io/instance: eks-pod-identity-agent

app.kubernetes.io/name: eks-pod-identity-agent

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/os

operator: In

values:

- linux

- key: kubernetes.io/arch

operator: In

values:

- amd64

- arm64

- key: eks.amazonaws.com/compute-type

operator: NotIn

values:

- fargate

containers:

- args:

- --port

- "80"

- --cluster-name

- eksworkshop-eksctl

- --probe-port

- "2703"

command:

- /go-runner

- /eks-pod-identity-agent

- server

env:

- name: AWS\_REGION

value: us-west-2

image: 602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/eks-pod-identity-agent:0.0.25

imagePullPolicy: Always

livenessProbe:

failureThreshold: 3

httpGet:

host: localhost

path: /healthz

port: probes-port

scheme: HTTP

initialDelaySeconds: 30

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 10

name: eks-pod-identity-agent

ports:

- containerPort: 80

name: proxy

protocol: TCP

- containerPort: 2703

name: probes-port

protocol: TCP

readinessProbe:

failureThreshold: 30

httpGet:

host: localhost

path: /readyz

port: probes-port

scheme: HTTP

initialDelaySeconds: 1

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 10

resources: {}

securityContext:

capabilities:

add:

- CAP\_NET\_BIND\_SERVICE

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

dnsPolicy: ClusterFirst

hostNetwork: **true**

initContainers:

- command:

- /go-runner

- /eks-pod-identity-agent

- initialize

image: 602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/eks-pod-identity-agent:0.0.25

imagePullPolicy: Always

name: eks-pod-identity-agent-init

resources: {}

securityContext:

privileged: **true**

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

priorityClassName: system-node-critical

restartPolicy: Always

schedulerName: default-scheduler

securityContext: {}

terminationGracePeriodSeconds: 30

updateStrategy:

rollingUpdate:

maxSurge: 0

maxUnavailable: 10%

type: RollingUpdate

status:

currentNumberScheduled: 3

desiredNumberScheduled: 3

numberAvailable: 3

numberMisscheduled: 0

numberReady: 3

observedGeneration: 1

updatedNumberScheduled: 3

kind: List

metadata:

resourceVersion: ""

#### [EKS Pod Identity calling EKS Auth API](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "eks-pod-identity-calling-eks-auth-api)

When the application Pod make a HTTP call to 169.254.170.23/v1/credentials, the EKS Pod Identity agent running on that node, receives this call. The EKS Pod Identity Agent further call below API to the EKS Auth API Service.

(

TOKEN=$(kubectl -n $NS exec -it $APP -- /bin/bash -c 'cat $AWS\_CONTAINER\_AUTHORIZATION\_TOKEN\_FILE')

echo TOKEN=$TOKEN

aws eks-auth assume-role-for-pod-identity --cluster-name $EKS\_CLUSTER --token $TOKEN

)

The EKS worker node role need to have IAM permissions for the Pod Identity Agent to call AssumeRoleForPodIdentity action in EKS Auth API. You can use the AWS managed policy: AmazonEKSWorkerNodePolicy which is updated to include this permission. Alternatively you can also add custom policy like below.

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"eks-auth:AssumeRoleForPodIdentity"

],

"Resource": "\*"

}

]

}

An example output from the above call aws eks-auth assume-role-for-pod-identity is mentioned below just for reference. We can see the namespace and service account, the associationArn, and the AssumeRoleArn, with the IAM credentials for this session.

{

"subject": {

"namespace": "ns-a",

"serviceAccount": "app1"

},

"audience": "pods.eks.amazonaws.com",

"podIdentityAssociation": {

"associationArn": "arn:aws:eks:us-west-2:ACCOUNT\_ID:podidentityassociation/eksworkshop-eksctl/a-lfw52xh1ihr1szuwp",

"associationId": "a-lfw52xh1ihr1szuwp"

},

"assumedRoleUser": {

"arn": "arn:aws:sts::ACCOUNT\_ID:assumed-role/eks-pod-s3-readonly-access/eks-eks-ref-sc-app2-17903385-a093-4b17-9f47-97e9137a30fc",

"assumeRoleId": "AROAQAHCJ2QPFDE3D2IDP:eks-eks-ref-sc-app2-17903385-a093-4b17-9f47-97e9137a30fc"

},

"credentials": {

"sessionToken": "",

"secretAccessKey": "ftZLqkr+iQTZsGV/l3HZPCQVCQZUwgVwgYaaIiPY",

"accessKeyId": "ASIAQAHCJ2QPPKHZNJFH",

"expiration": "2023-12-10T11:10:32+00:00"

}

}

You can also lookup for the CloudTrail event for the above call.

EVENT\_NAME="AssumeRoleForPodIdentity"

aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=$EVENT\_NAME --max-items=1

Check Output

{

"Events": [

{

"EventId": "e05b6d2e-4be6-45dd-8ad5-ba1e076002a5",

"EventName": "AssumeRoleForPodIdentity",

"ReadOnly": "true",

"AccessKeyId": "ASIAQAHCJ2QPOFDF6NOW",

"EventTime": "2023-12-13T01:14:20+00:00",

"EventSource": "eks-auth.amazonaws.com",

"Username": "i-064d652934ecf0bfc",

"Resources": [

{

"ResourceType": "AWS::EKS::Cluster",

"ResourceName": "eksworkshop-eksctl"

}

],

"CloudTrailEvent": "{\"eventVersion\":\"1.09\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AROAQAHCJ2QPM6KW4RJQK:i-064d652934ecf0bfc\",\"arn\":\"arn:aws:sts::ACCOUNT\_ID:assumed-role/platform-eks-node-group-20231128085853957000000011/i-064d652934ecf0bfc\",\"accountId\":\"ACCOUNT\_ID\",\"accessKeyId\":\"ASIAQAHCJ2QPOFDF6NOW\",\"sessionContext\":{\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AROAQAHCJ2QPM6KW4RJQK\",\"arn\":\"arn:aws:iam::ACCOUNT\_ID:role/platform-eks-node-group-20231128085853957000000011\",\"accountId\":\"ACCOUNT\_ID\",\"userName\":\"platform-eks-node-group-20231128085853957000000011\"},\"attributes\":{\"creationDate\":\"2023-12-13T01:04:37Z\",\"mfaAuthenticated\":\"false\"},\"ec2RoleDelivery\":\"2.0\"}},\"eventTime\":\"2023-12-13T01:14:20Z\",\"eventSource\":\"eks-auth.amazonaws.com\",\"eventName\":\"AssumeRoleForPodIdentity\",\"awsRegion\":\"us-west-2\",\"sourceIPAddress\":\"44.219.101.145\",\"userAgent\":\"aws-sdk-go-v2/1.21.2 os/linux lang/go#1.19.13 md/GOOS#linux md/GOARCH#amd64 api/eksauth#1.0.0-zeta.e49712bf27d5\",\"requestParameters\":{\"clusterName\":\"eksworkshop-eksctl\",\"token\":\"HIDDEN\_DUE\_TO\_SECURITY\_REASONS\"},\"responseElements\":null,\"requestID\":\"7f8d9b80-3603-40eb-827b-e0afafc49507\",\"eventID\":\"e05b6d2e-4be6-45dd-8ad5-ba1e076002a5\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",\"managementEvent\":true,\"recipientAccountId\":\"ACCOUNT\_ID\",\"eventCategory\":\"Management\",\"tlsDetails\":{\"tlsVersion\":\"TLSv1.3\",\"cipherSuite\":\"TLS\_AES\_128\_GCM\_SHA256\",\"clientProvidedHostHeader\":\"eks-auth.us-west-2.api.aws\"}}"

}

],

"NextToken": "eyJOZXh0VG9rZW4iOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ=="

}

```

Few things to observe from above output:

**Resource Type** for this API is AWS::EKS::Cluster.

**Resource name** corresponding to the **Resource Type** is our EKS Cluster name i.e. eksworkshop-eksctl.

**AWS principal** making the above call is the assumeRole of the EKS worker node i.e. arn:aws:sts::ACCOUNT\_ID:assumed-role/platform-eks-node-group-20231128085853957000000011/i-064d652934ecf0bfc

**eventSource** at which this event occurred is EKS Auth API Service i.e. eks-auth.amazonaws.com

In the **CloudTrailEvent**, there are other informations, and you can tweak the previous command to have a prettier print of this part:

1

aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=$EVENT\_NAME --max-items=1 | jq '.Events[] | (.CloudTrailEvent | fromjson)'

Check Output

{

"eventVersion": "1.09",

"userIdentity": {

"type": "AssumedRole",

"principalId": "AROA37QFXJP5E2DPF472Q:i-07574d724fb843fda",

"arn": "arn:aws:sts::823571991546:assumed-role/eks-security-workshop/i-07574d724fb843fda",

"accountId": "823571991546",

"accessKeyId": "ASIA37QFXJP5LMALW5PH",

"sessionContext": {

"sessionIssuer": {

"type": "Role",

"principalId": "AROA37QFXJP5E2DPF472Q",

"arn": "arn:aws:iam::823571991546:role/eks-security-workshop",

"accountId": "823571991546",

"userName": "eks-security-workshop"

},

"attributes": {

"creationDate": "2024-04-16T08:23:33Z",

"mfaAuthenticated": "false"

},

"ec2RoleDelivery": "2.0"

}

},

"eventTime": "2024-04-16T08:29:33Z",

"eventSource": "eks-auth.amazonaws.com",

"eventName": "AssumeRoleForPodIdentity",

"awsRegion": "us-west-2",

"sourceIPAddress": "34.222.6.96",

"userAgent": "aws-cli/2.15.38 Python/3.11.8 Linux/5.10.213-201.855.amzn2.x86\_64 exe/x86\_64.amzn.2 prompt/off command/eks-auth.assume-role-for-pod-identity",

"requestParameters": {

"clusterName": "eksworkshop-eksctl",

"token": "HIDDEN\_DUE\_TO\_SECURITY\_REASONS"

},

"responseElements": null,

"requestID": "1f3aa1ae-d897-4310-bab4-3e89f8d42f5d",

"eventID": "eaef7c6d-f6d0-4a9a-8380-4547a9ac326e",

"readOnly": true,

"eventType": "AwsApiCall",

"managementEvent": true,

"recipientAccountId": "823571991546",

"eventCategory": "Management",

"tlsDetails": {

"tlsVersion": "TLSv1.3",

"cipherSuite": "TLS\_AES\_128\_GCM\_SHA256",

"clientProvidedHostHeader": "eks-auth.us-west-2.api.aws"

}

}

the **requestParameters** in the above call includes **clusterName** i.e. eksworkshop-eksctl and EKS Pod Identity Service Account **token** which is HIDDEN\_DUE\_TO\_SECURITY\_REASONS

#### [EKS Pod Identity validation](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "eks-pod-identity-validation)

The EKS Auth API Service extracts the Service Account and Namespace details from the token and validates them against the EKS Pod Identity association we created earlier.

#### [EKS Pod Identity call to AWS STS service](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/3-eks-pod-identity-deep-dive" \l "eks-pod-identity-call-to-aws-sts-service)

Once verified, the EKS Auth API also extracts the IAM Role eks-pod-s3-read-access-role mapped in EKS Pod Identity association and calls AWS STS Service to get the temporary credentials. It then sends these credentials to EKS Pod Identity agent, which again sends it back to the Application pod.

You can also lookup for the CloudTrail event for the call to AWS STS Service to get the temporary credentials. We filter on the CloudTrailEvent parameter of the API call containing a tag with name 'eks-cluster-arn' so that it is one from the EKS Pod Identity:

events=$(aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventSource,AttributeValue=sts.amazonaws.com --max-items 100)

echo $events | jq '.Events[] | (.CloudTrailEvent | fromjson | select(.requestParameters.tags[]?.key=="eks-cluster-arn"))'

*If there is no result, that means that the call was not in the last 100 events, you can ask the pod-identity daemonset to restart, so that it will make this call again*

*kubectl rollout restart daemonset eks-pod-identity-agent -n kube-system*

AWS STS event

{

"eventVersion": "1.08",

"userIdentity": {

"type": "AWSService",

"invokedBy": "pods.eks.amazonaws.com"

},

"eventTime": "2024-04-16T09:20:04Z",

"eventSource": "sts.amazonaws.com",

"eventName": "AssumeRole",

"awsRegion": "us-west-2",

"sourceIPAddress": "pods.eks.amazonaws.com",

"userAgent": "pods.eks.amazonaws.com",

"requestParameters": {

"roleArn": "arn:aws:iam::823571991546:role/eks-pod-s3-read-access-role",

"roleSessionName": "eks-eksworksho-app1-498124c2-a555-40bf-9059-ca9a26821eb0",

"durationSeconds": 21600,

"tags": [

{

"key": "eks-cluster-arn",

"value": "arn:aws:eks:us-west-2:823571991546:cluster/eksworkshop-eksctl"

},

{

"key": "eks-cluster-name",

"value": "eksworkshop-eksctl"

},

{

"key": "kubernetes-namespace",

"value": "ns-a"

},

{

"key": "kubernetes-service-account",

"value": "sa1"

},

{

"key": "kubernetes-pod-name",

"value": "app1"

},

{

"key": "kubernetes-pod-uid",

"value": "0cda7f85-c1f9-4dee-b6e9-a0b50c8f3012"

}

],

"transitiveTagKeys": [

"eks-cluster-arn",

"eks-cluster-name",

"kubernetes-namespace",

"kubernetes-service-account",

"kubernetes-pod-name",

"kubernetes-pod-uid"

]

},

"responseElements": {

"credentials": {

"accessKeyId": "ASIA37QFXJP5MQOWRZ6Q",

"sessionToken": "",

"expiration": "Apr 16, 2024, 3:20:04 PM"

},

"assumedRoleUser": {

"assumedRoleId": "AROA37QFXJP5LOO5CGXEC:eks-eksworksho-app1-498124c2-a555-40bf-9059-ca9a26821eb0",

"arn": "arn:aws:sts::823571991546:assumed-role/eks-pod-s3-read-access-role/eks-eksworksho-app1-498124c2-a555-40bf-9059-ca9a26821eb0"

},

"packedPolicySize": 56

},

"requestID": "3821a55e-cecc-4d68-9c9b-fc8b6d6994f4",

"eventID": "7b7e4636-4713-3d04-b86a-cd6774cdbae9",

"readOnly": true,

"resources": [

{

"accountId": "823571991546",

"type": "AWS::IAM::Role",

"ARN": "arn:aws:iam::823571991546:role/eks-pod-s3-read-access-role"

}

],

"eventType": "AwsApiCall",

"managementEvent": true,

"recipientAccountId": "823571991546",

"sharedEventID": "c30ac43f-e051-426b-94bc-ab73b05e34e9",

"eventCategory": "Management"

}

We can see from the output, the source of this call if from AWSService pods.eks.amazonaws.com. That means that EKS Pod Identity retrieve the temporary credentials (l53) for our IAM Role (l71) and that it has attached some Session Tags (l17) that can be use to filter access to AWS resources.

This means, we can further configure our S3 read access IAM Role for fine grained IAM permissions to restrict the access to this Role for any specifc EKS Cluster, Namespace, Service Account, Pod Name or Pod UID. We will explore on how this works in the next module.

# Access control to IAM Role using EKS Pod Identity Usecases

In this section, we explore how we can control access to an IAM Role using EKS Pod Identity across EKS Clusters, Namespaces, Service accounts etc.

## [Control access to IAM Role to specific EKS Clusters.](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "control-access-to-iam-role-to-specific-eks-clusters.)

### [Update IAM Role Trust Policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "update-iam-role-trust-policy)

By default the IAM Role arn:aws:iam::ACCOUNT\_ID:role/eks-pod-s3-read-access-role can be used across EKS Clusters in this AWS Account without any changes anywhere.

We can restrict the access to the IAM Role to specific EKS Clusters in this AWS Account by modifying the IAM Role Trust Policy.

Let us say we want to restrict access to the IAM Role to only two EKS Clusters which do not exist, say eks-cluster1-does-not-exist and eks-cluster2-does-not-exist

Let us update the IAM Role Trust policy document as below.

export IAM\_ROLE="eks-pod-s3-read-access-role"

export IAM\_ROLE\_TRUST\_POLICY\_CLUSTER="eks-pod-s3-read-access-trust-policy-cluster"

cat > ~/environment/$IAM\_ROLE\_TRUST\_POLICY\_CLUSTER.json << EOF

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "pods.eks.amazonaws.com"

},

"Action": [

"sts:AssumeRole",

"sts:TagSession"

],

"Condition": {

"StringEquals": {

"aws:RequestTag/eks-cluster-name": [

"eks-cluster1-does-not-exist",

"eks-cluster2-does-not-exist"

]

}

}

}

]

}

EOF

Let us update the IAM Role with the new Trust policy.

aws iam update-assume-role-policy \

--role-name $IAM\_ROLE \

--policy-document file://~/environment/$IAM\_ROLE\_TRUST\_POLICY\_CLUSTER.json

### [Test Access to IAM Role from clusters](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "test-access-to-iam-role-from-clusters)

Now, let us test access to S3 from the Pod. Since the Pod may already have cached the temporary credentials, we will delete and re-create it.

Delete the Pod and re-create it.

kubectl -n $NS delete pod $APP

kubectl apply -f $APP.yaml

Check Output

namespace/ns-a unchanged

serviceaccount/sa1 unchanged

pod/app1 created

Test the S3 access again.

kubectl -n $NS exec -it $APP -- aws s3 ls

Check Output

Error when retrieving credentials from container-role: Error retrieving metadata: Received non 200 response 400 from container metadata: [b69b1279-d140-4eee-b7ea-b124286f544c]: (AccessDeniedException): Unauthorized Exception! EKS does not have permissions to assume the associated role., fault: client

command terminated with exit code 255

The error indicates that the our EKS cluster eksworkshop-eksctl is not allowed to assume the IAM Role.

You can also lookup for the CloudTrail event for the above call.

EVENT\_NAME="AssumeRoleForPodIdentity"

aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=$EVENT\_NAME --max-items=1

Check Output

{

"Events": [

{

"EventId": "d647586b-b713-46af-b7f9-512c7f5145dd",

"EventName": "AssumeRoleForPodIdentity",

"ReadOnly": "true",

"AccessKeyId": "ASIA26YVAA7XWZTLILHG",

"EventTime": "2024-01-30T07:52:08+00:00",

"EventSource": "eks-auth.amazonaws.com",

"Username": "i-02930ce60274d87a7",

"Resources": [

{

"ResourceType": "AWS::EKS::Cluster",

"ResourceName": "eksworkshop-eksctl"

}

],

"CloudTrailEvent": "{\"eventVersion\":\"1.09\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AROA26YVAA7X3XEV3BAHM:i-02930ce60274d87a7\",\"arn\":\"arn:aws:sts::753273931759:assumed-role/eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa/i-02930ce60274d87a7\",\"accountId\":\"753273931759\",\"accessKeyId\":\"ASIA26YVAA7XWZTLILHG\",\"sessionContext\":{\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AROA26YVAA7X3XEV3BAHM\",\"arn\":\"arn:aws:iam::753273931759:role/eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa\",\"accountId\":\"753273931759\",\"userName\":\"eks-bootstrap-template-ws-EKSNodegroupRole-E1potkq4Auqa\"},\"attributes\":{\"creationDate\":\"2024-01-30T06:57:57Z\",\"mfaAuthenticated\":\"false\"},\"ec2RoleDelivery\":\"2.0\"}},\"eventTime\":\"2024-01-30T07:52:08Z\",\"eventSource\":\"eks-auth.amazonaws.com\",\"eventName\":\"AssumeRoleForPodIdentity\",\"awsRegion\":\"us-west-2\",\"sourceIPAddress\":\"100.20.39.202\",\"userAgent\":\"aws-sdk-go-v2/1.21.2 os/linux lang/go#1.19.13 md/GOOS#linux md/GOARCH#amd64 api/eksauth#1.0.0-zeta.e49712bf27d5\",\"errorCode\":\"AccessDenied\",\"errorMessage\":\"Unauthorized Exception! EKS does not have permissions to assume the associated role.\",\"requestParameters\":{\"clusterName\":\"eksworkshop-eksctl\",\"token\":\"HIDDEN\_DUE\_TO\_SECURITY\_REASONS\"},\"responseElements\":null,\"requestID\":\"b69b1279-d140-4eee-b7ea-b124286f544c\",\"eventID\":\"d647586b-b713-46af-b7f9-512c7f5145dd\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",\"managementEvent\":true,\"recipientAccountId\":\"753273931759\",\"eventCategory\":\"Management\",\"tlsDetails\":{\"tlsVersion\":\"TLSv1.3\",\"cipherSuite\":\"TLS\_AES\_128\_GCM\_SHA256\",\"clientProvidedHostHeader\":\"eks-auth.us-west-2.api.aws\"}}"

}

],

"NextToken": "eyJOZXh0VG9rZW4iOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ=="

}

## [Control access to IAM Role to specific Namespaces in a cluster.](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "control-access-to-iam-role-to-specific-namespaces-in-a-cluster.)

In this section, we will see how we can restrict access to IAM Role to specific namespaces in a EKS cluster.

### [Update IAM Role Trust Policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "update-iam-role-trust-policy)

Let us update the IAM Role Trust policy document to restrict access to only two namespaces ns-a, ns-b in our EKS cluster eksworkshop-eksctl

export IAM\_ROLE="eks-pod-s3-read-access-role"

export IAM\_ROLE\_TRUST\_POLICY\_NAMESPACE="eks-pod-s3-read-access-trust-policy-namespace"

export NS1="ns-a"

export NS2="ns-b"

cat > ~/environment/$IAM\_ROLE\_TRUST\_POLICY\_NAMESPACE.json << EOF

{

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

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "pods.eks.amazonaws.com"

},

"Action": [

"sts:AssumeRole",

"sts:TagSession"

],

"Condition": {

"StringEquals": {

"aws:RequestTag/eks-cluster-name": "$EKS\_CLUSTER\_NAME",

"aws:RequestTag/kubernetes-namespace": [

"$NS1",

"$NS2"

]

}

}

}

]

}

EOF

Let us update the IAM Role with the new Trust policy.

aws iam update-assume-role-policy \

--role-name $IAM\_ROLE \

--policy-document file://~/environment/$IAM\_ROLE\_TRUST\_POLICY\_NAMESPACE.json

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

### [Test Access to IAM Role across Namespaces](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "test-access-to-iam-role-across-namespaces)

#### [Create EKS Pod Identity Association for Second App](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "create-eks-pod-identity-association-for-second-app)

export APP="app2"

export NS="ns-b"

export SA="sa2"

aws eks create-pod-identity-association \

--cluster-name $EKS\_CLUSTER\_NAME \

--namespace $NS \

--service-account $SA \

--role-arn $IAM\_ROLE\_ARN

#### [Deploy a Sample App](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "deploy-a-sample-app-app2-in-namespace-ns-b)**[app2](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "deploy-a-sample-app-app2-in-namespace-ns-b)**[in Namespace](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "deploy-a-sample-app-app2-in-namespace-ns-b)**[ns-b](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "deploy-a-sample-app-app2-in-namespace-ns-b)**

Run below command to deploy a Sample App app2 with a Kubernetes Service account sa2 in Namspace ns-b.

export APP=app2

export NS=ns-b

export SA=sa2

envsubst < app-template.yaml > $APP.yaml

kubectl apply -f $APP.yaml

Check Output

namespace/ns-b created

serviceaccount/sa2 created

pod/app2 created

Check if the Pod is running fine.

1

kubectl -n $NS get pod

Check Output

1

2

NAME READY STATUS RESTARTS AGE

app2 1/1 Running 0 2m29s

Check if the Pod can access any S3 Buckets.

1

kubectl -n $NS exec -it $APP -- aws s3 ls

Check Output

1

2023-12-12 05:28:12 ekspodidentity-ACCOUNT\_ID-us-west-2

As you can see, the App app2 in Namespace ns-b can list S3 buckets.

Similarly, you can also restrict access to the IAM Role to a specific Service account and a Pod.

## [Control access to AWS Service (S3 Bucket) to specific Cluster/Namespaces/Service account in a cluster.](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "control-access-to-aws-service-(s3-bucket)-to-specific-clusternamespacesservice-account-in-a-cluster.)

In this section, we will explore how to control access to S3 Bucket and Objects for a specific EKS cluster, namespace using AWS Resource Tags.

### [Update IAM Policy for fine grained access control](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "update-iam-policy-for-fine-grained-access-control)

Earlier we created one S3 Bucket ekspodidentity-ACCOUNT\_ID-us-west-2

We will create two S3 objects say customer1.txt and customer2.txt in this S3 bucket. We want to provide access for the S3 object customer1.txt for only a specific Service account say sa1 in Namespace ns-a based on S3 Objects tags. Similarly, we want to provide access for the S3 object customer2.txt for only a specific Service account say sa2 in Namespace ns-b based on S3 Objects tags.

Let us create a custom IAM Policy for S3 read access role to configure the granular IAM permissions.

export IAM\_POLICY="eks-pod-s3-read-access-policy"

export IAM\_POLICY\_S3="eks-pod-s3-read-access-policy-s3"

cat > ~/environment/$IAM\_POLICY\_S3.json <<EOF

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:ListAllMyBuckets"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:GetObjectTagging"

],

"Resource": "\*",

"Condition": {

"StringEquals": {

"s3:ExistingObjectTag/my-namespace": "\${aws:PrincipalTag/kubernetes-namespace}",

"s3:ExistingObjectTag/my-service-account": "\${aws:PrincipalTag/kubernetes-service-account}"

}

}

}

]

}

EOF

1

2

s3policyArn=$(aws iam create-policy --policy-name $IAM\_POLICY\_S3 --policy-document file://~/environment/$IAM\_POLICY\_S3.json --output text --query Policy.Arn)

echo "s3policyArn=$s3policyArn"

Check Output

1

s3policyArn=arn:aws:iam::ACCOUNT\_ID:policy/eks-pod-s3-read-access-policy-s3

Let us remove the earlier policy and attach the updated IAM policy to the IAM role.

1

2

aws iam detach-role-policy --role-name $IAM\_ROLE --policy-arn $policyArn

aws iam attach-role-policy --role-name $IAM\_ROLE --policy-arn $s3policyArn

### [Create S3 Objects in the S3 Bucket](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "create-s3-objects-in-the-s3-bucket)

Let us create 3 S3 objects named customer1.txt, customer2.txt and common.txtand upload them to the S3 bucket.

Let us create a simple text file customer1.txt and upload it to the S3 bucket by tagging custom Resource Tags with key/value pairs such as my-namespace=ns-a and my-service-account=sa1 to reflect that this Object needs to be accessed only by that specific Service account and Namespace.

export NS="ns-a"

export SA="sa1"

export S3\_OBJECT="customer1.txt"

cat > $S3\_OBJECT <<EOF

This needs to be accessed only by Service account $SA in Namespace $NS

EOF

aws s3api put-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" --body "${S3\_OBJECT}" --tagging "my-namespace=$NS&my-service-account=$SA"

Check Output

{

"ETag": "\"052d1a31b3ae1ac27c4b0d1b23199d65\"",

"ServerSideEncryption": "AES256"

}

Let us create another simple text file customer2.txt and upload it to the S3 bucket by tagging custom Resource Tags with key/value pairs such as my-namespace=ns-b and my-service-account=sa2 to reflect that this Object needs to be accessed only by that specific Service account and Namespace.

export NS="ns-b"

export SA="sa2"

export S3\_OBJECT="customer2.txt"

cat > $S3\_OBJECT <<EOF

This needs to be accessed only by Service account $SA in Namespace $NS

EOF

aws s3api put-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" --body "${S3\_OBJECT}" --tagging "my-namespace=$NS&my-service-account=$SA"

Check Output

{

"ETag": "\"a3bffd3ac901fb7510179c440b22a1e9\"",

"ServerSideEncryption": "AES256"

}

### [Test access to S3 Objects](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/3-eks-pod-identity/4-eks-pod-identity-usecases" \l "test-access-to-s3-objects)

Let us try to access the S3 Objects customer1.txt and customer2.txt from Service account sa1 in Namespace ns-a

export APP="app1"

export NS="ns-a"

export SA="sa1"

export S3\_BUCKET="ekspodidentity-$ACCOUNT\_ID-$AWS\_REGION"

export S3\_OBJECT="customer1.txt"

kubectl -n $NS exec -it $APP -- aws s3api get-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" "${S3\_OBJECT}"

Check Output

{

"AcceptRanges": "bytes",

"LastModified": "2023-12-13T11:44:49+00:00",

"ContentLength": 72,

"ETag": "\"052d1a31b3ae1ac27c4b0d1b23199d65\"",

"ContentType": "binary/octet-stream",

"ServerSideEncryption": "AES256",

"Metadata": {},

"TagCount": 2

}

As expected Service account sa1 in Namespace ns-a can access S3 Object customer1.txt since tags are matching.

Let us try accessing S3 Object customer2.txt

export S3\_OBJECT="customer2.txt"

kubectl -n $NS exec -it $APP -- aws s3api get-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" "${S3\_OBJECT}"

Check Output

1

An error occurred (AccessDenied) when calling the GetObject operation: Access Denied

As expected, we see AccessDenied error.

Let us now try to access the S3 Objects customer1.txt and customer2.txt from Service account sa2 in Namespace ns-b

export APP="app2"

export NS="ns-b"

export SA="sa2"

export S3\_BUCKET="ekspodidentity-$ACCOUNT\_ID-$AWS\_REGION"

export S3\_OBJECT="customer1.txt"

kubectl -n $NS exec -it $APP -- aws s3api get-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" "${S3\_OBJECT}"

Check Output

1

An error occurred (AccessDenied) when calling the GetObject operation: Access Denied

As expected, we see AccessDenied error.

Let us try accessing S3 Object customer2.txt

export S3\_OBJECT="customer2.txt"

kubectl -n $NS exec -it $APP -- aws s3api get-object --bucket "${S3\_BUCKET}" --key "${S3\_OBJECT}" "${S3\_OBJECT}"

Check Output

{

"AcceptRanges": "bytes",

"LastModified": "2023-12-13T11:50:19+00:00",

"ContentLength": 72,

"ETag": "\"a3bffd3ac901fb7510179c440b22a1e9\"",

"ContentType": "binary/octet-stream",

"ServerSideEncryption": "AES256",

"Metadata": {},

"TagCount": 2

}

As expected Service account sa2 in Namespace ns-b can access S3 Object customer2.txt since tags are matching.

# Cleanup

To cleanup, follow these steps.

kubectl delete -f app1.txt

kubectl delete -f app2.txt

aws s3api delete-object --bucket $S3\_BUCKET --key customer1.txt

aws s3api delete-object --bucket $S3\_BUCKET --key customer2.txt

aws s3 rb s3://ekspodidentity-$ACCOUNT\_ID-$AWS\_REGION --region $AWS\_REGION --force