IAM Roles for service accounts(IRSA)

**[What is IRSA and why it is needed?](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa" \l "what-is-irsa-and-why-it-is-needed)**

Before we get into the details of what is IRSA and how does it work, let us first understand what security challenges does it address?

A common challenge architects face when designing a Kubernetes solution on AWS is how to grant containerized workload permissions to access an AWS service or resource. AWS Identity and Access Management (IAM) provides fine-grained access control where you can specify who can access which AWS service or resources, ensuring the principle of least privilege. The challenge when your workload is running in Kubernetes, however, is providing an identity to that Kubernetes workload that IAM can use for authentication.

Let's take an example to illustrate this challenge. Let's deploy a simple pod which needs to list all the AWS S3 buckets.

cat > eks-iam-test1.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: eks-iam-test1

labels:

app: s3-test

spec:

containers:

- name: eks-iam-test1

image: amazon/aws-cli:latest

args: ['s3', 'ls']

restartPolicy: Never

EOF

kubectl apply -f eks-iam-test1.yaml

Check Output

1

pod/eks-iam-test1 created

Run the below command to see the pod status

1

kubectl get pod

The output looks like below

NAME READY STATUS RESTARTS AGE

eks-iam-test1 0/1 Error 0 30s

The pod status shows Error. Let us check logs for the exact reason.

kubectl logs eks-iam-test1

The output looks like below.

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

As you see in the above output, the pod is not able to access the AWS S3 service due to AccessDenied permission error.

The reason for this error is that the Kubernetes Pod is assuming an [IAM Role attached to the Amazon EC2 instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html) and leveraging this role to try and list the S3 buckets. This is because no other AWS credentials were found in the container, so the SDK fell back to the IAM metadata server, as mentioned in the [python boto3 sdk documentation](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html#id1).

As the IAM role within the EC2 Instance Profile does not have necessary permissions to list the buckets, the command received an “Access Denied” error. One way to fix this could be to attach additional permissions to the EC2 instance profile. However, this violates a key security principle, the principle of least privilege. This additional permission would be at the EC2 Node level, not at the Kubernetes Pod level. Therefore, all Pods running on that node would gain access to our S3 buckets. We want to restrict this permission to the Pod level.

This leads us on to the next question: how could we inject AWS credentials into a container so the container does not default to the EC2 instance profile? Injecting AWS credentials via Kubernetes Secrets or environment variables would not be secure, and the user would have to manage the lifecycle of these credentials. We would not recommend either of those approaches.

**[Fine-Grained IAM Roles for service accounts](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa" \l "fine-grained-iam-roles-for-service-accounts)**

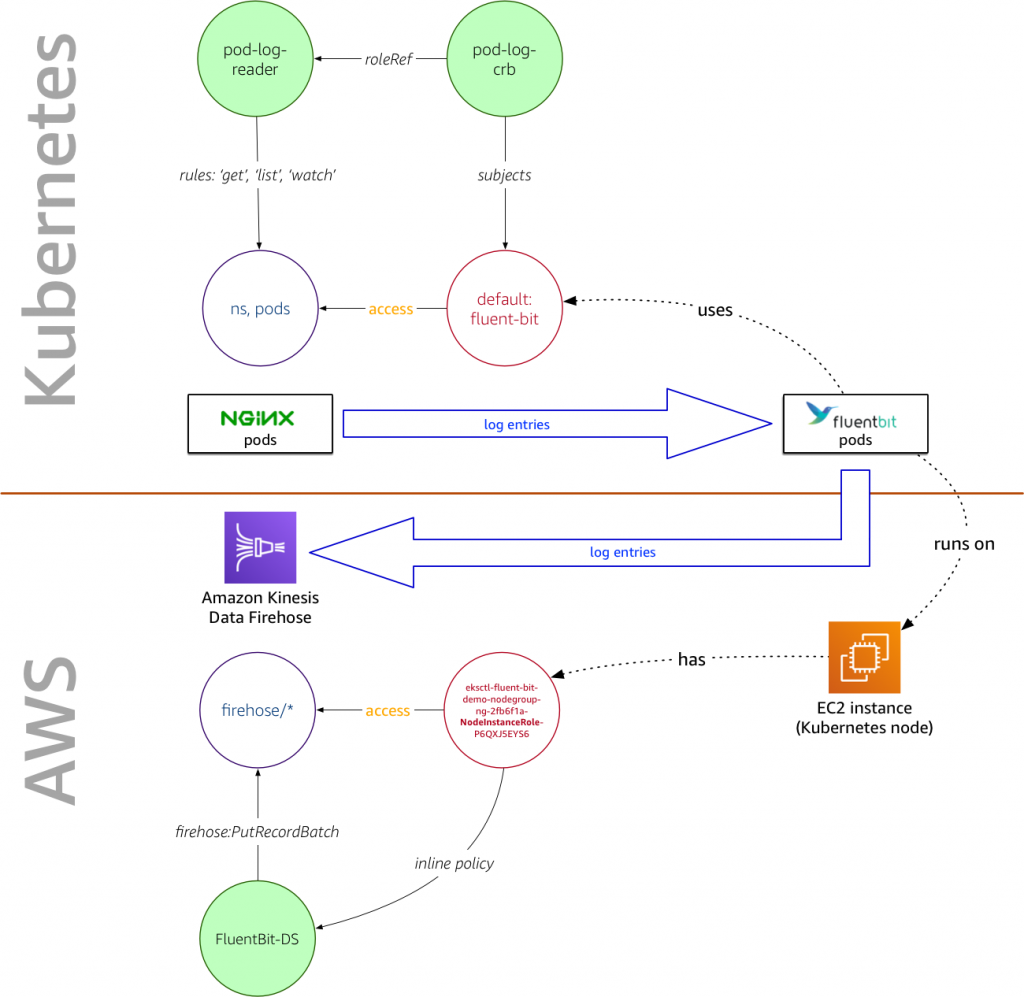
Applications in a Pod’s containers can use an AWS SDK or the AWS CLI to make API requests to AWS services using AWS Identity and Access Management (IAM) permissions. For example, applications may need to upload files to an S3 bucket or query a DynamoDB table. To do so applications must sign their AWS API requests with AWS credentials. [IAM Roles for service accounts (IRSA)](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html) provide the ability to manage credentials for your applications, similar to the way that IAM Instance Profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or relying on the Amazon EC2 Instance Profile for authorization, you associate an IAM Role with a Kubernetes Service Account and configure your Pods to use that Service Account.

You can associate an IAM role with a Kubernetes Service Account. This Service Account can then provide AWS permissions to the containers in any pod that uses that Service Account. With this feature, you no longer need to provide extended permissions to the Amazon EKS node IAM role so that pods on that node can call AWS APIs.

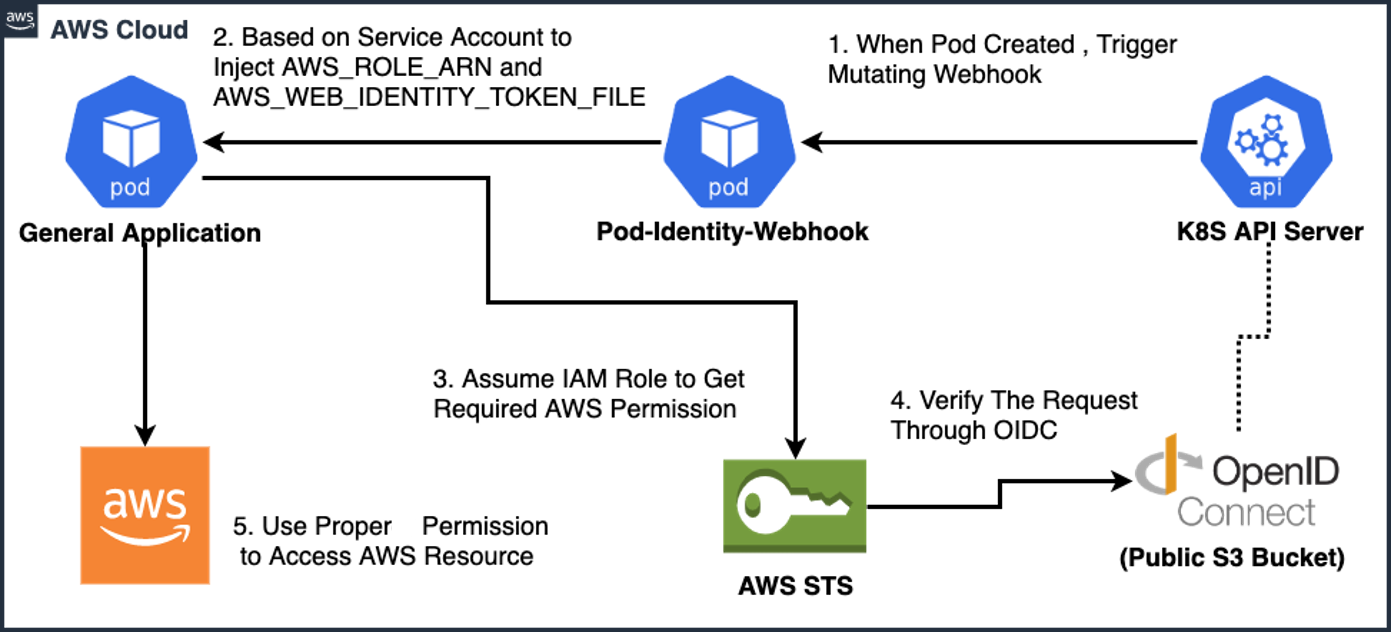
The IAM roles for service accounts feature provides the following benefits:

* **Least privilege** — By using the IAM roles for service accounts feature, you no longer need to provide extended permissions to the node IAM role so that pods on that node can call AWS APIs. You can scope IAM permissions to a service account, and only pods that use that Service Account have access to those permissions.
* **Credential isolation** — A container can only retrieve credentials for the IAM role that is associated with the Service Account to which it belongs. A container never has access to credentials that are intended for another container that belongs to another pod.
* **Auditability** – Access and event logging is available through AWS CloudTrail to help ensure retrospective auditing.

Below diagram from a blog [Introducing fine-grained IAM roles for service accounts](https://aws.amazon.com/blogs/opensource/introducing-fine-grained-iam-roles-service-accounts/) explains how applications running in Amazon EKS access Kubernetes resources using RBAC and AWS Services using IAM Permissions.



IRSA implementation includes various components as shown below.



Enabling IRSA including the following procedures:

1. IAM OIDC provider for Amazon EKS cluster.
2. Configure Kubernetes Service Account to assume IAM role
3. Configure pods to use Kubernetes Service Account

# Projected Service Account Token

Before we get into how IRSA works, let us understand some Basics and underlying Kubernetes which enables IRSA functionality.

### [Kubernetes Service Accounts](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/projected-sa-token" \l "kubernetes-service-accounts)

A service account is a type of non-human account that, in Kubernetes, provides a distinct identity in a Kubernetes cluster. Application Pods, system components, and entities inside and outside the cluster can use a specific ServiceAccount's credentials to identify as that ServiceAccount. This identity is useful in various situations, including authenticating to the API server or implementing identity-based security policies.

There are different ways to manage the credentials for the Service Accounts.

* [TokenRequest AP](https://kubernetes.io/docs/reference/kubernetes-api/authentication-resources/token-request-v1/) (recommended): Request a short-lived service account token from within your own application code. The token expires automatically and can rotate upon expiration. If you have a legacy application that is not aware of Kubernetes, you could use a sidecar container within the same pod to fetch these tokens and make them available to the application workload.
* [Token Volume Projection](https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#serviceaccount-token-volume-projection) (also recommended): In Kubernetes v1.20 and later, use the Pod specification to tell the kubelet to add the service account token to the Pod as a projected volume. Projected tokens expire automatically, and the kubelet rotates the token before it expires.
* [Service Account Token Secrets](https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#manually-create-an-api-token-for-a-serviceaccount) (not recommended): You can mount service account tokens as Kubernetes Secrets in Pods. These tokens don't expire and don't rotate. This method is not recommended, especially at scale, because of the risks associated with static, long-lived credentials. In Kubernetes v1.24 and later, the LegacyServiceAccountTokenNoAutoGeneration feature gate prevents Kubernetes from automatically creating these tokens for ServiceAccounts. LegacyServiceAccountTokenNoAutoGeneration is enabled by default; in other words, Kubernetes does not create these tokens.

#### [Default ServiceAccount credentials from Kubernetes version 1.24 or later](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/projected-sa-token" \l "default-serviceaccount-credentials-from-kubernetes-version-1.24-or-later)

As mentioned above, In Kubernetes v1.24 and later, Kubernetes does not create the service account tokens as Kubernetes Secrets in Pods.

To check this, run below command.

kubectl get sa

The output looks like below

NAME SECRETS AGE

default 0 3h9m

As you see in the above output, there is no secret created for the sevice account.

### [Projected Service Account Token](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/projected-sa-token" \l "projected-service-account-token)

**Projected Volumes**

A [projected volume](https://kubernetes.io/docs/concepts/storage/projected-volumes/) maps several existing volume sources into the same directory. Currently, the following types of volume sources can be projected:

1. secret
2. downwardAPI
3. configMap
4. serviceAccountToken

In Kubernetes 1.12 the [ProjectedServiceAccountToken](https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/" \l "service-account-token-volume-projection" \t "_blank) feature was introduced. This feature allows a fully compliant OIDC JWT token issued by the TokenRequest API of Kubernetes to be mounted into the Pod as a Projected Volume. The relevant Service Account Token Volume Projection flags are enabled by default on an EKS cluster. Therefore, fully compliant OIDC JWT Service Account tokens are being projected into each pod instead of the JWT token mentioned in the previous paragraph.

To inspect this OIDC Token, let us create a new pod that just has a sleep process inside with the following command:

cat > eks-iam-test2.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: eks-iam-test2

spec:

containers:

- name: my-aws-cli

image: amazon/aws-cli:latest

command: ['sleep', '36000']

restartPolicy: Never

EOF

kubectl apply -f eks-iam-test2.yaml

Check Output

Run the below command to see the pod status

kubectl get pod

The output looks like below

NAME READY STATUS RESTARTS AGE

eks-iam-test1 0/1 Error 0 96m

eks-iam-test2 1/1 Running 0 81s

Let us look at the Volumes and volumeMounts in the pod eks-iam-test2 specification.

kubectl get pod eks-iam-test2 -oyaml

The output looks like below. Only relevant fields are shown below.

----

spec:

containers:

- command:

- sleep

- "36000"

image: amazon/aws-cli:latest

volumeMounts:

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

name: kube-api-access-qq6vs

readOnly: true

----

serviceAccountName: default

----

volumes:

- name: kube-api-access-qq6vs

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

----

The volumeMounts section in the above output, the mountPath value **/var/run/secrets/kubernetes.io/serviceaccount** is the base path for all the projected Volume Sources i.e. serviceAccountToken, configMap and downwardAPI.

The generated token is mounted as per serviceAccountToken configuration is **/var/run/secrets/kubernetes.io/serviceaccount/token**

The token can be retrieved and expanded to show a fully compliant OIDC token.

1

kubectl exec -it eks-iam-test2 -- cat /var/run/secrets/kubernetes.io/serviceaccount/token

The Output looks like below.

1

eyJhbGciOiJSUzI1NiIsImtpZCI6ImY2NDU3OGViMmFiMjRlOTIxNWM0NjA4Yjg1NTU5YmNiODgxOTQ1NDQifQ..N-z0JJjHI8yZtDAyPTBfJRz-s7dzhdR1F5QpK5hnuMbeIsPxvuADppanqD82oO8SWy11Nw4WOO85s22ZpRjh3MlSKMuvEIAFkk69iyPooMUrRkqLu7blF3\_EsbZKgACR1plUGEjG2Ge1mGGE5nvem63BKBUfhk7F0Eefg58ZaEvu7Zubk3IEChU6Q2FWmLVQGFJOAjZXreDN571DshOXp53y8ZlO0dbuk5WRXtUwY11DgP8cznGiz8bpoPxf20g-pGj4eGR3r0oFRm78rfa-uC6fI5ccOSzYYrBOMujonRS1vbVXLDF71KFbQkbQxUrnqLoYwmPP4pspt6t3OZsfmw

Decoding this token at <https://jwt.io/> shows below Payload Data.

{

"aud": [

"https://kubernetes.default.svc"

],

"exp": 1709458083,

"iat": 1677922083,

"iss": "https://oidc.eks.us-east-1.amazonaws.com/id/80D562ED8026E91294D52E09BEA261D4",

"kubernetes.io": {

"namespace": "default",

"pod": {

"name": "eks-iam-test2",

"uid": "e5917a3c-108b-4c7b-b045-960b6a59fefb"

},

"serviceaccount": {

"name": "default",

"uid": "75f1899b-9146-4d81-835c-91ce9e9f88a5"

},

"warnafter": 1677925690

},

"nbf": 1677922083,

"sub": "system:serviceaccount:default:default"

}

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-east-1.amazonaws.com/id/80D562ED8026E91294D52E09BEA261D4. This OIDC provider URL will be used during the verification process of the token.

**aud**: It represents the audience of the token which is https://kubernetes.default.svc. This is the address inside a cluster used to reach the Kubernetes API Server. This means the token will be accepted by API Server and will be rejected by any other service.

For security reasons, you may not want to include any token into a Kubernetes Pod if the workload in the Pod is not going to be making calls to the Kubernetes API server. This can be done by passing automountServiceAccountToken: false into the pod Spec when you create a Pod.

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

This compliant OIDC token now gives us a foundation to build upon to find a token that can be used to authenticate to AWS APIs. However, we will need an additional component to inject a second token for use with AWS APIs into our Kubernetes Pods. Kubernetes supports validating and mutating webhooks, and AWS has created an [identity webhook](https://github.com/aws/amazon-eks-pod-identity-webhook/) that comes preinstalled in an Amazon EKS cluster. This webhook listens to create pod API calls and can inject an additional Token into our pods. This webhook can also be installed into self-managed Kubernetes clusters on AWS using [this guide](https://github.com/aws/amazon-eks-pod-identity-webhook/blob/master/SELF_HOSTED_SETUP.md)

This additional token, apart from the service account, enables the IRSA functionality. Let's see it in action in next sections.

# Enable IRSA

### [IAM OIDC provider for your cluster](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/enable-irsa" \l "iam-oidc-provider-for-your-cluster)

The EKS cluster has an OpenID Connect (OIDC) issuer URL associated with it. To use AWS Identity and Access Management (IAM) roles for service accounts, an IAM OIDC provider must exist for your cluster's OIDC issuer URL.

**Create IAM OIDC identity provider for EKS cluster**

Run the below command to retrieve the OpenID Connect issuer URL associated with the Amazon EKS Cluster.

1. Determine whether you have an existing IAM OIDC provider for your cluster. Retrieve your cluster's OIDC provider ID and store it in a variable.

oidc\_id=$(aws eks describe-cluster --name eksworkshop-eksctl --query "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)

echo $oidc\_id

The output will looks like below

80D562ED8026E91294D52E09BEA261D4

1. Determine whether an IAM OIDC provider with your cluster's ID is already in your account.

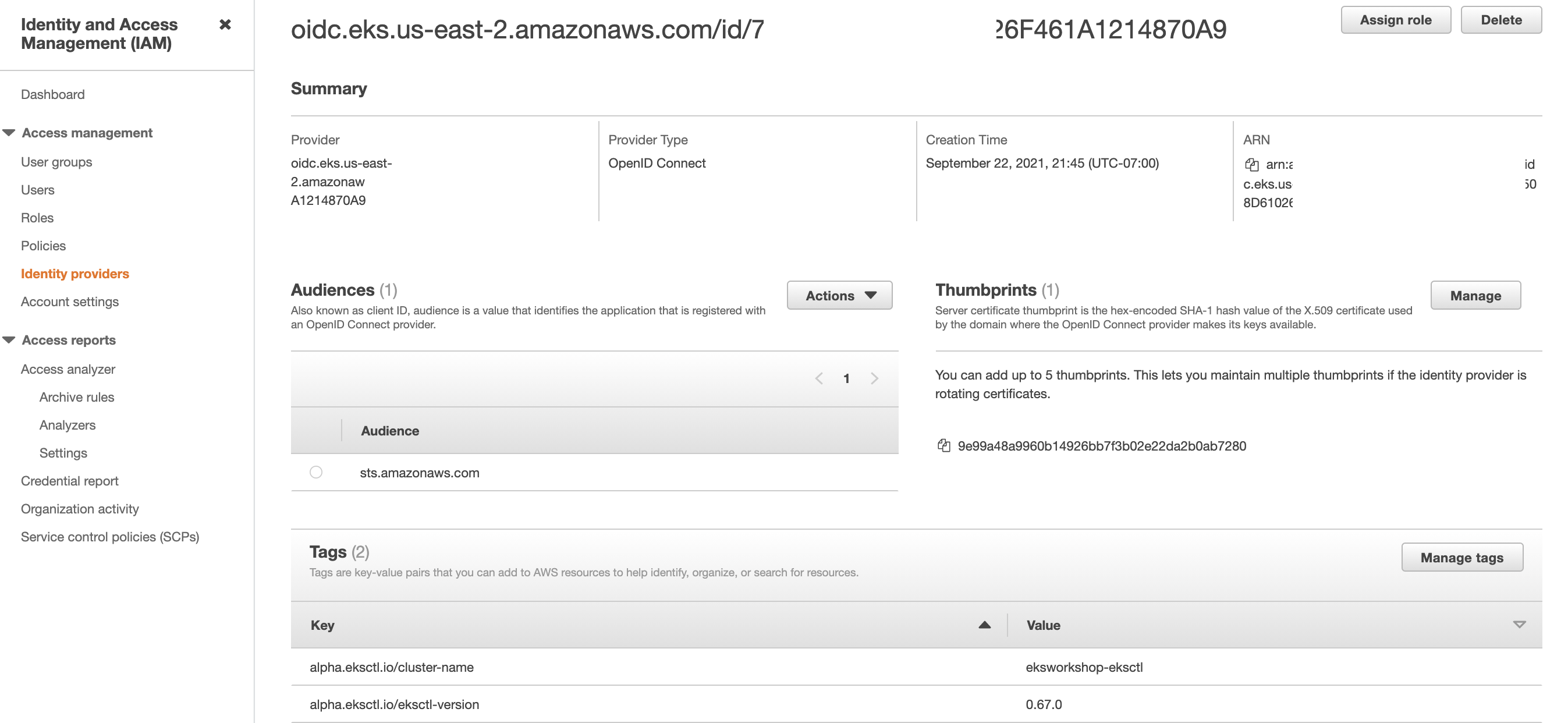
aws iam list-open-id-connect-providers | grep $oidc\_id | cut -d "/" -f4

If output is returned, then you already have an IAM OIDC provider for your cluster and you can skip the next step. If no output is returned, then you must create an IAM OIDC provider for your cluster

1. Create an IAM OIDC identity provider for your cluster with the following command. You only need to do this once for a cluster.

eksctl utils associate-iam-oidc-provider --cluster eksworkshop-eksctl --approve

If you go to the [Identity Providers in IAM Console](https://console.aws.amazon.com/iam/home#/providers), and click on the OIDC provider link, you will see OIDC provider has created for your cluster.



### [Create service account with attaching an IAM role](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/enable-irsa" \l "create-service-account-with-attaching-an-iam-role)

You will create an IAM policy that specifies the permissions that you would like the containers in your pods to have.

In this workshop we will use the AWS managed policy named "**AmazonS3ReadOnlyAccess**" which allow get and list for all your S3 buckets.

Let's start by finding the ARN for the "**AmazonS3ReadOnlyAccess**" policy

aws iam list-policies --query 'Policies[?PolicyName==`AmazonS3ReadOnlyAccess`].Arn'

Check Output

[

"arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"

]

Now you will create a IAM role bound to a service account with read-only access to S3

eksctl create iamserviceaccount \

--name iam-test \

--cluster eksworkshop-eksctl \

--attach-policy-arn arn\:aws\:iam::aws\:policy/AmazonS3ReadOnlyAccess \

--approve \

--override-existing-serviceaccounts

Check Output

2023-03-05 07:36:33 [ℹ] 8 existing iamserviceaccount(s) (amazon-cloudwatch/cloudwatch-agent,amazon-cloudwatch/cwagent-prometheus,amazon-cloudwatch/fluent-bit,default/xray-daemon,karpenter/karpenter,kube-system/aws-node,kube-system/cluster-autoscaler,workshop/iam-test) will be excluded

2023-03-05 07:36:33 [ℹ] 1 iamserviceaccount (default/iam-test) was included (based on the include/exclude rules)

2023-03-05 07:36:33 [!] metadata of serviceaccounts that exist in Kubernetes will be updated, as --override-existing-serviceaccounts was set

2023-03-05 07:36:33 [ℹ] 1 task: {

2 sequential sub-tasks: {

create IAM role for serviceaccount "default/iam-test",

create serviceaccount "default/iam-test",

} }2023-03-05 07:36:33 [ℹ] building iamserviceaccount stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-05 07:36:33 [ℹ] deploying stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-05 07:36:33 [ℹ] waiting for CloudFormation stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-05 07:37:03 [ℹ] waiting for CloudFormation stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-05 07:37:03 [ℹ] created serviceaccount "default/iam-test"

You can see that an IAM role (See the Annotations below) is associated to the service account iam-test in the cluster we just created.

kubectl describe sa iam-test

The output looks like below.

Name: iam-test

Namespace: default

Labels: app.kubernetes.io/managed-by=eksctl

Annotations: eks.amazonaws.com/role-arn: arn:aws:iam::XXXXXXXXXXXX:role/eksctl-eksworkshop-eksctl-addon-iamserviceac-Role1-1CF1FE6ZXXRZF

Image pull secrets: <none>

Mountable secrets: iam-test-token-v8flm

Tokens: iam-test-token-v8flm

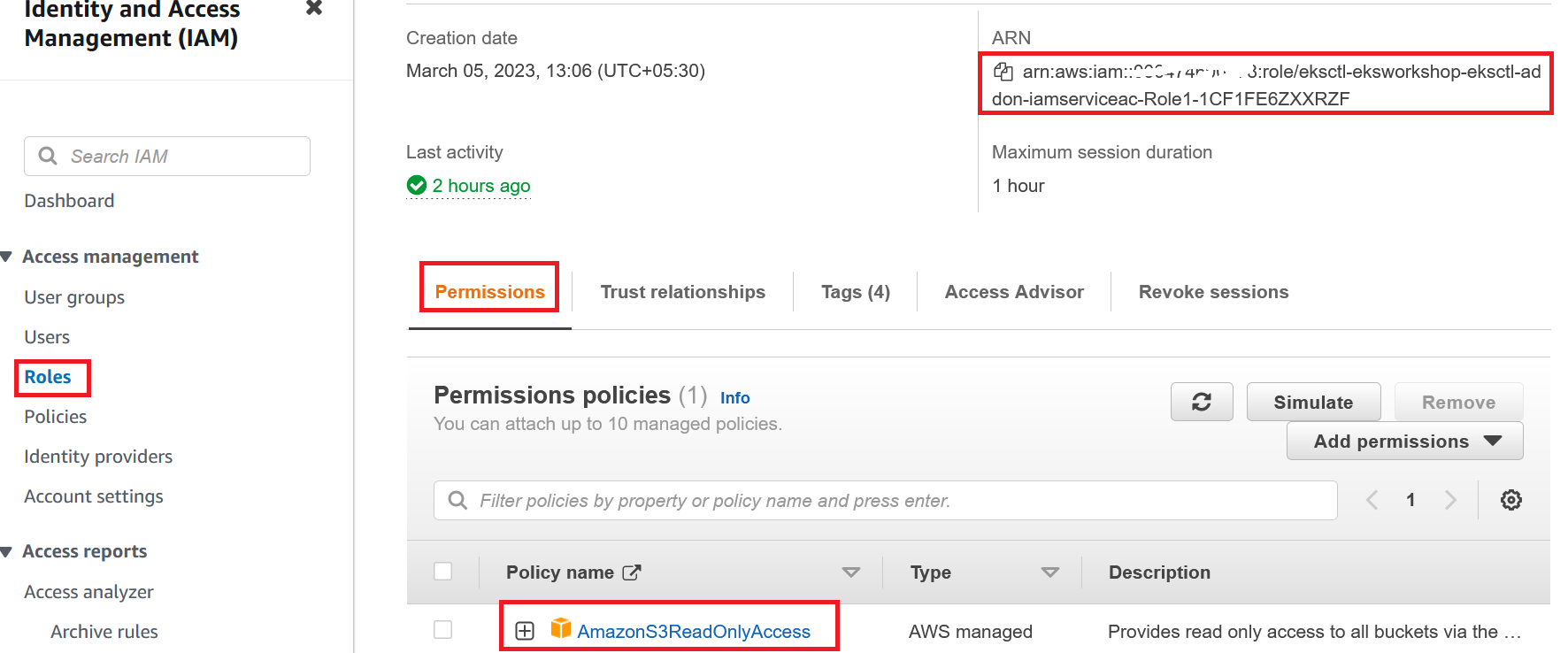
Events: <none>

In the above input, note that the service account annotation contains the IAM Role.

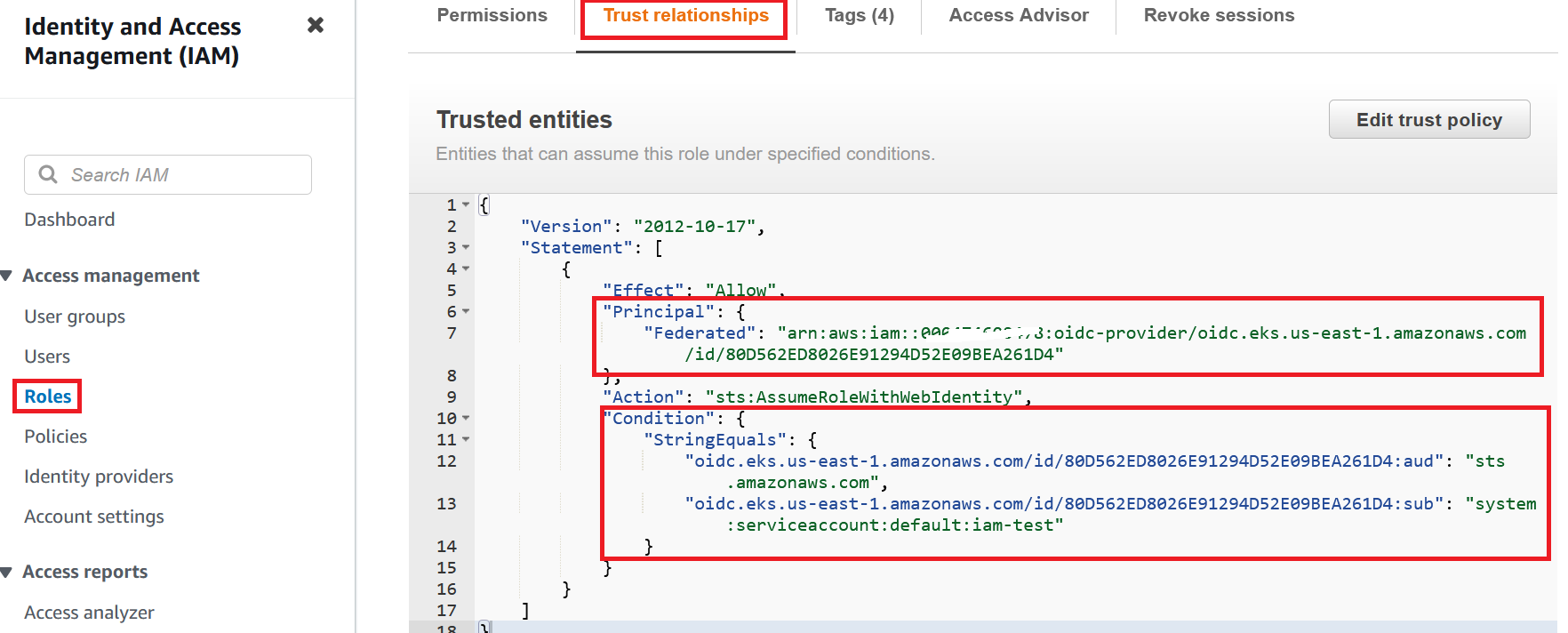
**Note**

If you go to the [AWS CloudFormation in IAM Console](https://console.aws.amazon.com/cloudformation/), you will find that the stack "**eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test**" has created a role for your service account.

Let’s see how this IAM role looks within the AWS Management Console. Navigate to IAM and then IAM Roles and search for the role. You will see the Annotations field when you describe your Service Account.



Select the Trust relationships tab and select Edit trust relationship to view the policy document.



The principal for this policy is arn:aws:iam::XXXXXXXXXX:oidc-provider/oidc.eks.us-west-2.amazonaws.com/id/80D562ED8026E91294D52E09BEA261D41 i.e. the OIDC provider for the Amazon EKS Cluster can only assume this role and allowed action is sts:AssumeRoleWithWebIdentity.

You can also see that there are 2 conditions in this Policy. The first condition contains sub field and ensures that only Kubernetes pod with identity system:serviceaccount:default:iam-test can assume this the IAM Role. The second condition has aud field which says that the audience must be sts.amazonaws.com.

### [Configure Kubernetes Pod with Service Account](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/enable-irsa" \l "configure-kubernetes-pod-with-service-account)

Let's start by testing if the Service Account we created can list the S3 buckets.

First let's create an S3 bucket.

aws s3 mb s3://eksworkshop-$ACCOUNT\_ID-$AWS\_REGION --region $AWS\_REGION

The output looks like below

make\_bucket: eksworkshop-XXXXXXXX-us-west-2

Now let us use the above Service Account with our initial Pod example, which lists S3 objects.

cat > eks-iam-test3.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: eks-iam-test3

labels:

app: s3-test

spec:

serviceAccountName: iam-test

containers:

- name: eks-iam-test3

image: amazon/aws-cli:latest

args: ['s3', 'ls']

#command: ['aws', 's3', 'ls', 'sleep', '36000']

restartPolicy: Never

EOF

kubectl apply -f eks-iam-test3.yaml

Check Output

pod/eks-iam-test3 created

Run the below command to see the pod status.

kubectl get pod

The output looks like below

NAME READY STATUS RESTARTS AGE

NAME READY STATUS RESTARTS AGE

eks-iam-test1 0/1 Error 0 115m

eks-iam-test2 1/1 Running 0 19m

eks-iam-test3 0/1 Completed 0 61s

The pod status shows Completed. Let us check the logs to verify that the command ran successfully this time.

kubectl logs eks-iam-test3

The output should look like below.

2023-03-14 12:32:02 eksworkshop-XXXXXXXXXX-us-west-2

The above output indicates that the container is now able to access the AWS S3 service and list the bucket names successfully.

### [Inspect the Projected Service Account token for IRSA](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/2-identity-and-access-management/2-irsa/enable-irsa" \l "inspect-the-projected-service-account-token-for-irsa)

In the above section, we saw that container is able to get access S3 using fine grained IAM permissions. Now, let us inspect the additional token injected by the pod identity webhook.

cat > eks-iam-test4.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: eks-iam-test4

spec:

serviceAccountName: iam-test

containers:

- name: my-aws-cli

image: amazon/aws-cli:latest

command: ['sleep', '36000']

restartPolicy: Never

EOF

kubectl apply -f eks-iam-test4.yaml

Check Output

pod/eks-iam-test4 created

Run the below command to see the pod status.

kubectl get pod

The output looks like below

NAME READY STATUS RESTARTS AGE

eks-iam-test1 0/1 Error 0 125m

eks-iam-test2 1/1 Running 0 30m

eks-iam-test3 0/1 Completed 0 11m

eks-iam-test4 1/1 Running 0 6m12s

If we inspect the Pod using Kubectl and jq, we can see there are now two volumes mounted into our Pod. The second one has been mounted via that mutating webhook. The aws-iam-token is still being generated by the Kubernetes API Server, but with a new OIDC JWT audience.

Let us look at the Volumes and volumeMounts in the pod specification.

kubectl get pod eks-iam-test4 -oyaml

The output looks like below. Only relevant fields are shown below.

---

spec:

containers:

- command:

- sleep

- "36000"

env:

- name: AWS\_STS\_REGIONAL\_ENDPOINTS

value: regional

- name: AWS\_DEFAULT\_REGION

value: us-west-2

- name: AWS\_REGION

value: us-west-2

- name: AWS\_ROLE\_ARN

value: arn:aws:iam::XXXXXXXXX:role/eksctl-eksworkshop-eksctl-addon-iamserviceac-Role1-1CF1FE6ZXXRZF

- name: AWS\_WEB\_IDENTITY\_TOKEN\_FILE

value: /var/run/secrets/eks.amazonaws.com/serviceaccount/token

image: amazon/aws-cli:latest

volumeMounts:

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

name: kube-api-access-t9g7l

readOnly: true

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

name: aws-iam-token

readOnly: true

serviceAccount: iam-test

volumes:

- name: aws-iam-token

projected:

defaultMode: 420

sources:

- serviceAccountToken:

audience: sts.amazonaws.com

expirationSeconds: 86400

path: token

- name: kube-api-access-t9g7l

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

---

You can see that there are two projected volumes. One of them is for the default Service Account as expected and explained in the previous section. The second one is added by the pod identity webhook for the additional Service Account token used to authenticate with IAM Service to access AWS S3 Service.

The mutating webhook does more than just mount an additional token into the Pod. The mutating webhook also injects environment variables.

Note that there are two additional environment variables added. One of them is AWS\_ROLE\_ARN which contains the ARN of the IAM Role we created earlier for the Pod to assume. Second one is AWS\_WEB\_IDENTITY\_TOKEN\_FILE which contains the path for the additional service token. Note that additional token is mounted at a different path **/var/run/secrets/eks.amazonaws.com/serviceaccount** than the default Service Account token.

kubectl exec -it eks-iam-test4 -- cat /var/run/secrets/eks.amazonaws.com/serviceaccount/token

The Output looks like below.

eyJhbGciOiJSUzI1NiIsImtpZCI6ImY2NDU3OGViMmFiMjRlOTIxNWM0NjA4Yjg1NTU5YmNiODgxOTQ1NDQifQ..T\_s2O-GNW9KrErQ8Vm6WS\_3iiKbWR4Zkiv0F7IJaPVxqiMLtkAGbX7ucYkHOxpumgK-9OphzPrBSWZFpoHFYQkIuvN8YRyWJCvcQEzA2dF9IAJVMH8xzyyaxZdTonnRP2M\_HHBtYIOYJBty4YzLxwsHiUqxA6c\_7Q2Q03eLvaKYU4RM3hibICpcd-ENMEiB9k22B4sgrTe-DzbmJiUghKz4r-Ag-nkS3hiz0Imec2LQ0Xp7BHU74q7015WcbF8J2EdU8kNUwoTqKkuu\_wrksk\_0XpWLasbsNvLptKAnFQ1hOu\_I9wJieL\_0P0GMjz-\_nviUIjdw7OxnWcZPg3qci5w

Decoding this token at <https://jwt.io/> shows below Payload Data.

{

"aud": [

"sts.amazonaws.com"

],

"exp": 1678099678,

"iat": 1678013278,

"iss": "https://oidc.eks.us-west-2.amazonaws.com/id/80D562ED8026E91294D52E09BEA261D4",

"kubernetes.io": {

"namespace": "default",

"pod": {

"name": "eks-iam-test4",

"uid": "0d64402f-5c87-469c-b441-8bd332b1bd07"

},

"serviceaccount": {

"name": "iam-test",

"uid": "8145a6d5-4932-48e9-86b5-7c33eb9d6ad1"

}

},

"nbf": 1678013278,

"sub": "system:serviceaccount:default:iam-test"

}

You can see that the intended audience for this token is now sts.amazonaws.com, the issuer who has created and signed this token is still our OIDC provider, and finally, the expiration of the token is much shorter at 24 hours. We can modify the expiration duration for the Service Account using eks.amazonaws.com/token-expiration annotation in our Pod definition or Service Account definition.

Cleanup

To cleanup, follow these steps.

kubectl delete -f eks-iam-test4.yaml

kubectl delete -f eks-iam-test3.yaml

kubectl delete -f eks-iam-test2.yaml

kubectl delete -f eks-iam-test1.yaml

eksctl delete iamserviceaccount \

--name iam-test \

--cluster eksworkshop-eksctl \

--wait

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

Check Output

pod "eks-iam-test4" deleted

pod "eks-iam-test3" deleted

pod "eks-iam-test2" deleted

pod "eks-iam-test1" deleted

2023-03-14 12:53:47 [ℹ] 1 iamserviceaccount (default/iam-test) was included (based on the include/exclude rules)

2023-03-14 12:53:47 [ℹ] 1 task: {

2 sequential sub-tasks: {

delete IAM role for serviceaccount "default/iam-test",

delete serviceaccount "default/iam-test",

} }2023-03-14 12:53:47 [ℹ] will delete stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-14 12:53:47 [ℹ] waiting for stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test" to get deleted

2023-03-14 12:53:47 [ℹ] waiting for CloudFormation stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-14 12:54:17 [ℹ] waiting for CloudFormation stack "eksctl-eksworkshop-eksctl-addon-iamserviceaccount-default-iam-test"

2023-03-14 12:54:17 [ℹ] deleted serviceaccount "default/iam-test"

remove\_bucket: eksworkshop-XXXXXXXXXX-us-west-2