Containers

Amazon EKS Pod Identity: a new way for applications on EKS to obtain IAM credentials

by George John, Ashok Srirama, and Hemanth AVS | on 28 DEC 2023 | in Amazon Elastic Kubernetes Service, Compute, Containers | Permalink | Share

Introduction

At AWS we are constantly striving to improve customer experience. For instance, we launched IAM Roles for Service Accounts (IRSA) in 2019 that allows customers to configure Kubernetes (k8s) applications running on AWS with fine-grained AWS Identity and Access Management (AWS IAM) permissions to access other AWS resources such as Amazon Simple Storage Service (Amazon S3) buckets, Amazon DynamoDB tables, and more. This enables customers to run multiple applications in the same Amazon Elastic Kubernetes Service (Amazon EKS) cluster, while ensuring each application follows principle of least privilege. IRSA was built to support the various Kubernetes deployment options supported by AWS such as Amazon EKS in the cloud, Amazon EKS Anywhere, Red Hat OpenShift Service on AWS (ROSA), and self-managed Kubernetes clusters on Amazon Elastic Compute Cloud (Amazon EC2) instances. We built IRSA leveraging the foundational constructs offered by AWS IAM such as OpenID Connect (OIDC) identity providers and IAM trust policy for establishing trust between IAM role and Amazon EKS cluster. This ensured IRSA can work across environments without taking any direct dependency on Amazon EKS service or its APIs.

IRSA is widely used by AWS customers, and we plan to continue investing and supporting IRSA. But we have heard from Amazon EKS in the cloud customers that they wish for a more streamlined experience to setup AWS IAM permissions for their applications. Most of these customers, who are typically cluster administrators, usually don't have IAM administrator permissions to create OIDC providers or update IAM trust policy, which are pre-requisites for IRSA. They have to coordinate with IAM administrators in their company to execute IRSA configuration steps, which makes the process lengthy and difficult to automate. Further, cluster administrators have to update the IAM role trust policy each time the role is used in a new cluster during scenarios like blue-green upgrades or failover testing. Additionally, as customers grow their EKS cluster footprint, due to the per cluster OIDC provider requirement in IRSA, customers run into the per account OIDC provider limit. Similarly, as they scale the number of clusters or Kubernetes namespaces in which an IAM role is used, they run into IAM trust policy size limit, which makes them duplicate the IAM roles to overcome the trust policy size limit.

At AWS reInvent 2023, we launched Amazon EKS Pod Identity to address these challenges. Amazon EKS Pod Identity will coexist with IRSA and offers customers an additional solution to obtain IAM permissions. While IRSA works across different Kubernetes deployment options including EKS in the cloud, EKS Anywhere, self-managed Kubernetes clusters on Amazon EC2, and ROSA. EKS Pod Identity is purpose built for EKS in the cloud, designed to offer a simplified experience for obtaining IAM permissions for EKS customers. This simplified experience is made possible by the introduction of a new EKS service principal that can be used to establish trust between IAM roles and EKS service, and the introduction of new APIs on EKS that enables you to setup permissions without the need to execute privileged IAM operations like the setup of an OIDC identity provider. EKS Pod Identity enables us to launch additional features that is not supported in IRSA such as support for IAM role session tags. Role session tags enable IAM administrators to author a single permission policy that can work across roles by allowing access to AWS resources based on matching tags. Role session tags make permissions more intuitive and simplify access management for administrators.

Walkthrough

In the following section, we will look at the high-level flow of using Amazon EKS Pod Identity for configuring an AWS IAM role to grant IAM permissions for applications running in your cluster.

1. Amazon EKS cluster / IAM administrator creates an IAM role that can be assumed by the newly introduced EKS Service principal **pods.eks.amazonaws.com.** The trust policy for the IAM role would look like the policy shown below. Optionally, you can restrict the use of the role to certain EKS clusters by using a conditional string as shown in the following.

JSON
```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringEquals": {
                          "aws:SourceAccount": "my-account-number"
                        },
                       "ArnEquals": {
                           "aws:SourceArn": "arn-of-my-eks-cluster"
                       }
```

2. After the AWS IAM role creation, Amazon EKS cluster administrator creates an association between the IAM role and Kubernetes service account. You can create this association using the new **CreatePodIdentityAssociation** API. We will take a look at all the EKS Pod Identity APIs later in the post.

3. After the AWS IAM role is associated with the service account, any newly created pods using that service account will be intercepted by the EKS Pod Identity webhook. This webhook runs on the Amazon EKS cluster's control plane, and is fully managed by EKS. The webhook mutates the pod spec as shown below with the **bolded** environment variables. All AWS Software Development Kits (SDKs) have a series of places (or sources) that it checks in order to find valid credentials to use, to make a request to an AWS service. After credentials are found, the search is stopped. This systematic search is called the credential provider chain. For EKS Pod Identity, we leverage the HTTP credential provider mechanism that is already built into AWS SDKs and CLI to retrieve

credentials from an HTTP endpoint specified in the environment variable **AWS_CONTAINER_CREDENTIALS_FULL_URI**. This endpoint is served by the EKS Pod Identity Agent running on the worker node, we will talk more about this in the next step. The location to the projected JWT token that is used to exchange for IAM credentials is specified in the environment variable **AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE**.

1. **NOTE:** Your application should use a newer version of AWS SDK that support Amazon EKS Pod Identity. Please see EKS documentation here for the list of SDK/CLI versions that support EKS Pod Identity.

YAML

```yaml
      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-
   ...
    volumeMounts:
    - 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
...
```

4. AWS SDK/CLI calls the Amazon EKS Pod Identity Agent endpoint to retrieve the temporary IAM credentials. EKS Pod Identity Agent runs as a DaemonSet pod on every eligible worker node. This agent is made available to you as an EKS Add-on and is a pre-requisite to use EKS Pod Identity feature.
    1. The EKS Pod Identity agent will do SigV4 signing and make a call to the new EKS Auth API **AssumeRoleForPodIdentity** to exchange the projected token for temporary IAM credentials, which are then made available to the pod.
    2. EKS Auth API (**AssumeRoleForPodIdentity**) decodes the JWT token and validates the role associations with the service account. After successful validation, EKS Auth API will return the temporary AWS credentials. It will also set session tags such as *kubernetes-namespace, kubernetes-service-account, eks-cluster-arn, eks-cluster-name, kubernetes-pod-name, kubernetes-pod-uid*.
5. AWS SDKs will use the vended temporary credentials to access other AWS resources.

NOTE: Amazon EKS Pod Identity agent runs in host network mode and gets its permissions from the arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy managed policy that is attached to the worker nodes.

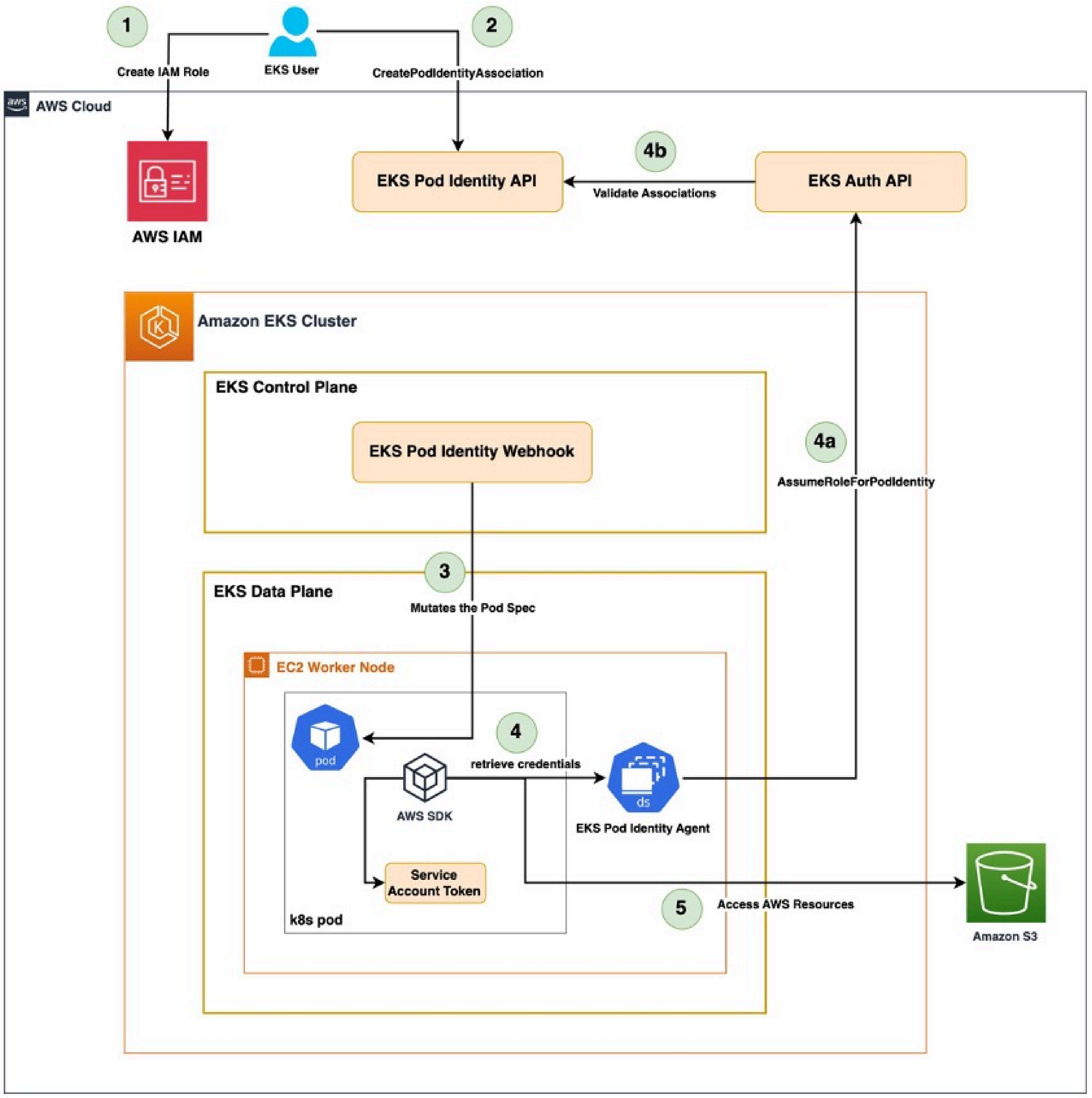


Figure 1: High level Amazon EKS Pod Identity architecture.

Amazon EKS Pod Identity APIs

The following is list of the new APIs introduced:

Management APIs

- **CreatePodIdentityAssociation** – API call to create an Amazon EKS Pod Identity association between a service account in an Amazon EKS cluster and an IAM role with EKS Pod Identity.

- Required parameters
    - name – The name of the cluster to create the association in.
    - namespace – The name of the Kubernetes namespace inside the cluster to create the association in. The Kubernetes service account and the pods that use the service account must be in this namespace.
    - roleArn – The Amazon Resource Name (ARN) of the AWS IAM role to associate with the service account. The Amazon EKS Pod Identity agent manages credentials to assume this role for applications in the containers in the pods that use this service account.
    - serviceAccount – The name of the Kubernetes service account inside the cluster to associate the IAM credentials with.

- **ListPodIdentityAssociations** – API call to list the Amazon EKS Pod Identity associations in a cluster. You can filter the list by the namespace that the association is in or the service account that the association uses.
  - Required parameters
    - name – The name of the cluster that the associations are in.
- **DescribePodIdentityAssociation** – API call to return descriptive information about an EKS Pod Identity association.
  - Required parameters
    - name – The name of the cluster that the association is in.
    - associationId – The ID of the association that you want the description of.
- **DeletePodIdentityAssociation** – API to delete an existing pod identity association.
  - Required parameters
    - name – The name of the cluster that the association is in.
    - associationId – The ID of the association that you want the description of.
- **UpdatePodIdentityAssociation** – API to update an existing pod identity association. Only the IAM role can be changed; an association can't be moved between clusters, namespaces, or service accounts. If you need to edit the namespace or service account, you need to remove the association and then create a new association with your desired settings.
  - name – The name of the cluster that the association is in.
  - associationId – The ID of the association that you want the description of.
  - roleArn – The new IAM role to associate with the service account

DataPlane API

- **AssumeRoleForPodIdentity** – API used by the eks-pod-identity-agent DaemonSet pod to exchange service account token for associated IAM role credentials. The token is a JSON Web Token, which has namespace, pod and service account embedded in it.

How to get started

In this walkthrough, we will demonstrate two use cases:

- A sample python flask application using EKS Pod Identity feature to access Amazon S3 buckets
- How to use IAM Session tags to create fine-grained IAM policies to access AWS Secrets Manager secrets

Prerequisites

- An AWS Account
- Latest version of AWS CLI configured on your device or AWS CloudShell
- eksctl – a simple CLI tool for creating and managing Amazon EKS clusters (v0.165.0 or higher)

Setup

```
export AWS_REGION=us-west-2 #Replace with your AWS Region
export AWS_ACCOUNT=111122223333 #Replace with your AWS Account number
export CLUSTER_NAME=eks-pod-identity-demo #Replace with your EKS cluster name

git clone https://github.com/aws-samples/amazon-eks-pod-identity-demo.git
cd amazon-eks-pod-identity-demo
```

Let's start by creating an Amazon EKS Cluster using eksctl with the new eks-pod-identity-agent addon.

NOTE: Amazon EKS Pod Identity is supported on EKS version 1.24 and higher.

YAML
```yaml
cat << EOF > cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ${CLUSTER_NAME}
  region: ${AWS_REGION}
  version: "1.28"

addons:
  - name: vpc-cni
  - name: coredns
  - name: kube-proxy
  - name: eks-pod-identity-agent

managedNodeGroups:
  - name: ${CLUSTER_NAME}-mng
```

```
      instanceType: m6a.large
      privateNetworking: true
```

You can also use the configuration file or CLI flags to deploy the addon after cluster creation as shown in the following:

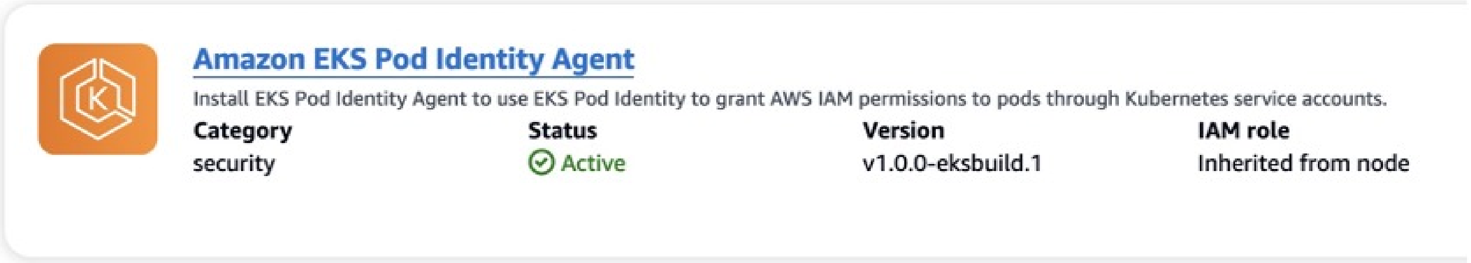
Bash
```
eksctl create addon --config-file=cluster.yaml
```

or

Bash
```
eksctl create addon --name eks-pod-identity-agent --version 1.0.0
```

Wait for cluster creation to be complete and ensure that eks-pod-identity-agent addon is running in the cluster.

Bash
```
eksctl get addon --cluster ${CLUSTER_NAME} --region ${AWS_REGION} --name eks-pod-ider

[
    {
        "Name": "eks-pod-identity-agent",
        "Version": "v1.0.0-eksbuild.1",
        "NewerVersion": "",
        "IAMRole": "",
        "Status": "ACTIVE",
        "ConfigurationValues": "",
        "Issues": null
    }
]
kubectl get pods -n kube-system -l app.kubernetes.io/instance=eks-pod-identity-agent

NAME                                 READY   STATUS    RESTARTS      AGE
eks-pod-identity-agent-tkxb4         1/1     Running   0             79m
eks-pod-identity-agent-zq9k2         1/1     Running   0             79m
```



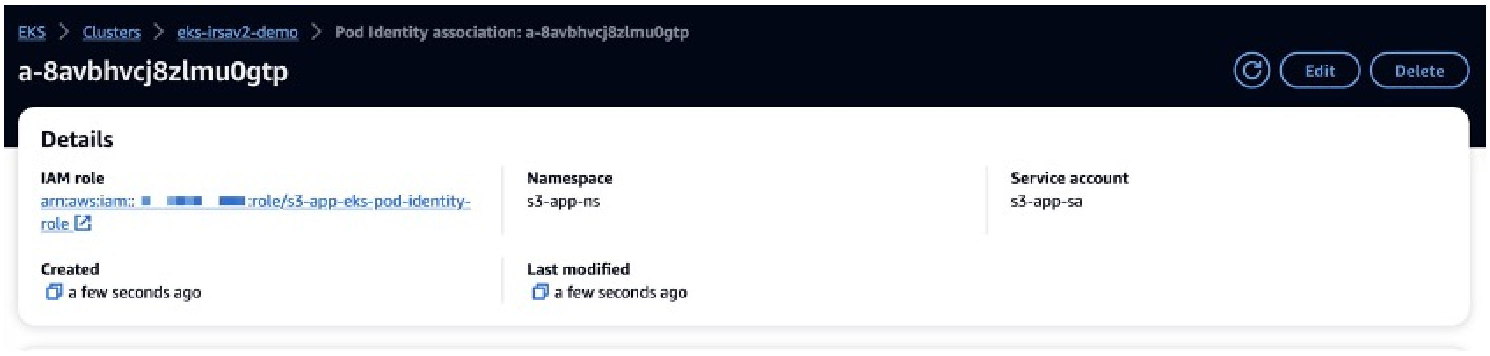
Amazon EKS Pod Identity Agent
Install EKS Pod Identity Agent to use EKS Pod Identity to grant AWS IAM permissions to pods through Kubernetes service accounts.

Category	Status	Version	IAM role
security	✓ Active	v1.0.0-eksbuild.1	Inherited from node

Figure 2: Amazon EKS Console – EKS Pod Identity Agent Addon

Next, we will use Amazon EKS Pod Identity feature to associate an AWS IAM role to the Kubernetes service account that will be used by our deployment. In this example, eksctl is used to simplify the IAM role creation and association process, below command would automatically create the IAM role *s3-app-eks-pod-identity-role*, assigns *AmazonS3ReadOnlyAccess* permission policy to it, and associate it with service account *s3-app-sa* in s3-app-ns namespace.

Bash
```bash
eksctl create podidentityassociation \
--cluster $CLUSTER_NAME \
--namespace s3-app-ns \
--service-account-name s3-app-sa \
--role-name s3-app-eks-pod-identity-role \
--permission-policy-arns arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
--region $AWS_REGION
```


EKS > Clusters > eks-irsav2-demo > Pod Identity association: a-8avbhvcj8zlmu0gtp

a-8avbhvcj8zlmu0gtp

Details

IAM role
arn:aws:iam::[redacted]:role/s3-app-eks-pod-identity-role

Namespace
s3-app-ns

Service account
s3-app-sa

Created
a few seconds ago

Last modified
a few seconds ago

Figure 3: Amazon EKS Console – Pod Identity Association

CreatePodIdentityAssociation API is used to create pod identity association for an IAM role and service account. EKS Pod Identity enables you to reuse same IAM role to associate with multiple service accounts across namespaces within a cluster, or across EKS clusters within an AWS Account.

Now that we have created an Amazon EKS cluster with the necessary pre-requisites for using EKS Pod Identity feature, lets deploy a sample python flask application that utilizes boto3 sdk to interact with Amazon S3 buckets. The python app has been updated to use latest version of boto3 SDK that supports EKS Pod Identity. This is deployed in s3-app-ns namespace and with s3-app-sa service account.

Bash
```bash
kubectl apply -f s3-app.yaml

namespace/s3-app-ns created
serviceaccount/s3-app-sa created
```

```
deployment.apps/s3-app-deployment created
service/s3-app-svc created
```

Wait for the pods and LoadBalancer service to become ready and fetch the LoadBalancer URL using the below command:

Bash
```
kubectl get all -n s3-app-ns

NAME                                      READY    STATUS     RESTARTS   AGE
pod/s3-app-deployment-79c7c5c696-nrfsb    1/1      Running    0          29s
pod/s3-app-deployment-79c7c5c696-rt562    1/1      Running    0          22s

NAME                    TYPE          CLUSTER-IP     EXTERNAL-IP
service/s3-app-svc      LoadBalancer  172.20.10.54   a3c4874577a65486aa23c5508ee3c

NAME                                 READY   UP-TO-DATE    AVAILABLE   AGE
deployment.apps/s3-app-deployment    2/2     2             2           42m

NAME                                           DESIRED    CURRENT    READY    AG
replicaset.apps/s3-app-deployment-79c7c5c696   2          2          2        29

export LB_URL=$(kubectl get svc -n s3-app-ns s3-app-svc -o jsonpath='{.status.loadBala
```

Now curl the below URL to verify if the k8s pod can access the Amazon S3 buckets. If call is successful, you will see list of bucket names.

Bash
```
curl http://$LB_URL/list-buckets
[ LIST OF BUCKETS ]
```

This demonstrates how the Amazon EKS workloads can utilize the new EKS Pod Identity feature to securely access other AWS resources using the temporary AWS IAM credentials.

You can also review the AWS CloudTrail event for the **AssumeRoleForPodIdentity**, eks-pod-identity-agent makes this call when the AWS SDK request for temporary IAM credentials.

JSON
```
{
....
    "eventSource": "eks-auth.amazonaws.com",
    "eventName": "AssumeRoleForPodIdentity",
    "awsRegion": "us-west-2",
    "userAgent": "aws-sdk-go-v2/1.21.2 os/linux lang/go#1.19.13 md/GOOS#linux md/GOARCI
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
.....
}
```

Support for session tags

Session tags are key-value pair attributes that you pass when you assume an IAM role or federate a user in AWS Security Token Service (AWS STS). With Amazon EKS Pod Identity, you have the ability to use session tags to control access to the AWS resources from your k8s pods. IAM role session tags also enable you to create fine-grained IAM permission policies based on tags such as eks-cluster-arn, eks-cluster-name, kubernetes-namespace, kubernetes-service-account, kubernetes-pod-name, and kubernetes-pod-uid.

This allows you to use attribute-based access control (ABAC) in the IAM permission policies with EKS. ABAC is an authorization strategy that defines permissions based on attributes/tags. By adding support for IAM Session Tags, you can enforce tighter security boundaries between clusters, and workloads within clusters, while reusing the IAM roles and IAM policies. The below example allows the **secretsmanager:GetSecretValue** action only if the secret is tagged with the matching *kubernetes-namespace* and *eks-cluster-name* values.

JSON
```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AuthorizetoGetSecretValue",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "secretsmanager:ResourceTag/kubernetes-namespace": "${aws:Princip
                    "secretsmanager:ResourceTag/eks-cluster-name": "${aws:PrincipalTa
                }
            }
        }
    ]
}
```

In this scenario, we will deploy an awscli application in two different namespaces using different service accounts. Both service accounts are associated to same AWS IAM role, which would allow access to AWS Secrets Manager secrets based on their kubernetes-namespace and eks-cluster-name session tags.

Start by creating a custom IAM policy with the above policy document.

Bash
```bash
export SECRETMGR_APP_POLICY_ARN=$(aws iam create-policy --policy-name sm-pod-identity-
--policy-document file://aws-secretmgr-policy.json \
--output text --query 'Policy.Arn')
```

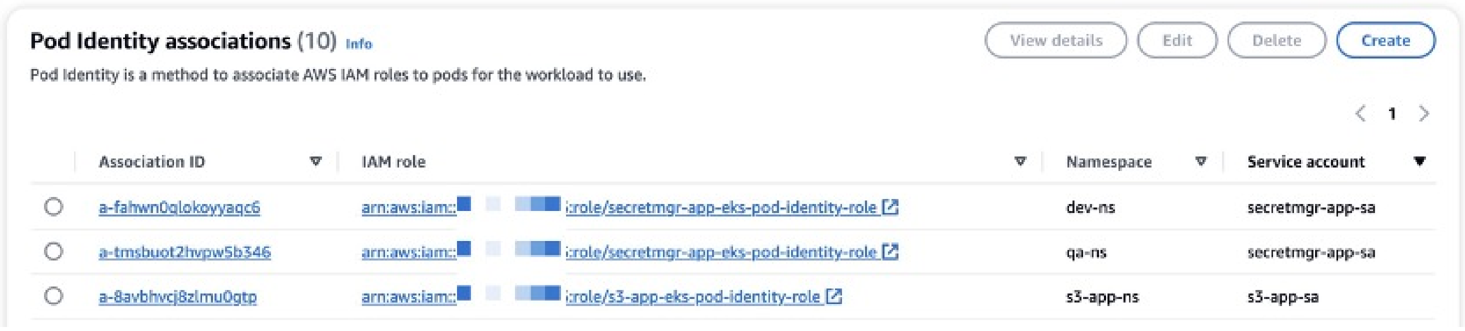
Now, create two AWS Secrets Manager secrets, one tagged with kubernetes-namespace=dev-ns and the other kubernetes-namespace=qa-ns. We are using the pre-defined kubernetes-namespace, eks-cluster-name session tags to restrict access to respective secrets.

Bash
```bash
aws secretsmanager create-secret --region $AWS_REGION --name dev-secret \
--tags Key=kubernetes-namespace,Value=dev-ns Key=eks-cluster-name,Value=$CLUSTER_NAME
--secret-string "This is super secret in dev ns"

aws secretsmanager create-secret --region $AWS_REGION --name qa-secret \
--tags Key=kubernetes-namespace,Value=qa-ns Key=eks-cluster-name,Value=$CLUSTER_NAME \
--secret-string "This is super secret in qa ns"
```

Run the following commands to create the AWS IAM role with the custom IAM policy created above and associate it with two service accounts (secretmgr-app-sa) in dev-ns and qa-ns namespaces.

Bash
```bash
eksctl create podidentityassociation \
--cluster $CLUSTER_NAME \
--namespace dev-ns \
--service-account-name secretmgr-app-sa \
--role-name secretmgr-app-eks-pod-identity-role \
--permission-policy-arns $SECRETMGR_APP_POLICY_ARN \
--region $AWS_REGION

eksctl create podidentityassociation \
--cluster $CLUSTER_NAME \
--namespace qa-ns \
--service-account-name secretmgr-app-sa \
--role-arn "arn:aws:iam::${AWS_ACCOUNT}:role/secretmgr-app-eks-pod-identity-role" \
--region $AWS_REGION
```

You can review the Amazon EKS Pod Identity association in Amazon EKS Console → **Access** tab as shown in the following diagram:



Pod Identity associations (10) Info

Pod Identity is a method to associate AWS IAM roles to pods for the workload to use.

	Association ID	IAM role	Namespace	Service account
○	a-fahwn0qlokoyyaqc6	arn:aws:iam::[redacted]:role/secretmgr-app-eks-pod-identity-role	dev-ns	secretmgr-app-sa
○	a-tmsbuot2hvpw5b346	arn:aws:iam::[redacted]:role/secretmgr-app-eks-pod-identity-role	qa-ns	secretmgr-app-sa
○	a-8avbhvcj8zlmu0gtp	arn:aws:iam::[redacted]:role/s3-app-eks-pod-identity-role	s3-app-ns	s3-app-sa

Figure 4: Amazon EKS Console – Pod Identity Associations

Deploy the sample application in both namespaces using the secretmgr-app-sa service account.

Bash
```bash
kubectl create ns dev-ns
kubectl create ns qa-ns

kubectl apply -f secretmgr-app-pod.yaml -n dev-ns
kubectl apply -f secretmgr-app-pod.yaml -n qa-ns
```

Let's test the secrets manager access using the following commands:

Bash
```bash
kubectl exec -it -n dev-ns secretmgr-app-pod -- aws secretsmanager get-secret-value --
> "This is super secret in dev ns"

kubectl exec -it -n qa-ns secretmgr-app-pod -- aws secretsmanager get-secret-value --s
> "This is super secret in qa ns"

kubectl exec -it -n dev-ns secretmgr-app-pod -- aws secretsmanager get-secret-value --
> An error occurred (AccessDeniedException) when calling the GetSecretValue operation:

kubectl exec -it -n qa-ns secretmgr-app-pod -- aws secretsmanager get-secret-value --s
> An error occurred (AccessDeniedException) when calling the GetSecretValue operation:
```

From the above outputs, you can notice GetSecretValue call is succeeded only when resource tag condition is met based on the kubernetes-namespace and eks-cluster-name tag values. Access is denied if both tag values doesn't match or missing tags on the resources. In summary, pod running in dev-ns is able to access the dev-secret but not the qa-secret and vice-versa.

From the AWS CloudTrail event log, you can also review the session tags being passed in **AssumeRole** API.

JSON
```
 ...
     "requestParameters": {
         "roleArn": "arn:aws:iam::123456789012:role/secretmgr-app-eks-pod-identity-rol
         "roleSessionName": "eks-eks-pod-identity-secretmgr--24690dab-4c88-462f-89cc-3
         "durationSeconds": 21600,
         "tags": [
             {
                 "key": "eks-cluster-arn",
                 "value": "arn:aws:eks:us-west-2:123456789012:cluster/eks-pod-identity
             },
             {
                 "key": "eks-cluster-name",
                 "value": "eks-pod-identity-demo"
             },
             {
                 "key": "kubernetes-namespace",
                 "value": "qa-ns"
             },
```

How to perform cross account access with Amazon EKS Pod Identity

There are scenarios in which you might have your Amazon EKS cluster running in one AWS account and the AWS resources that are being accessed from your application running in another account. You can access cross-account resources using one of the below approaches when using EKS Pod Identity feature.

Approach 1: Resource-based policy access

A resource-based policy can be used to grant cross-account access by specifying principals in other AWS accounts who can access the resource. Both IAM Role and application exists in Account A and appropriate resource policy is attached to resources in Account B to allow access to application IAM role. Refer to IAM documentation on Cross-account access using resource-based policies for details and Services that work with IAM for a list of services that support resource-based policies.

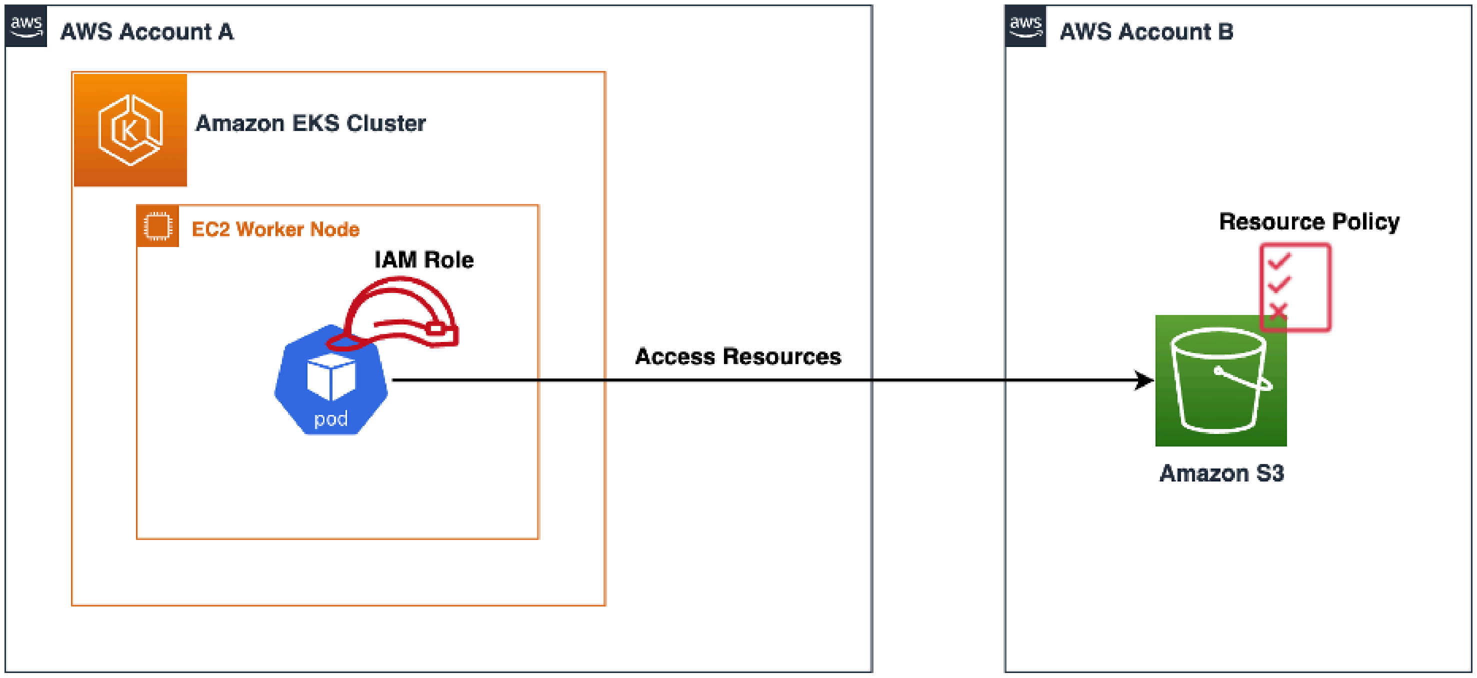


Figure 5: Cross Account Access using resource-based policy

Approach 2: Role chaining in the application

In this approach, Account B creates an IAM role with the appropriate permission policies and creates a trust relationship that allows *AssumeRole* permission to Account A. In Account A, an IAM role is created with AssumeRole permission to assume the IAM Role in Account B, and associated with your k8s pod. When the pod receives temporary IAM credentials from EKS Pod Identity, your application can perform STS *AssumeRole* action to assume the IAM Role in Account B and access the cross-account resources. Read Cross-account access using roles for a detailed walkthrough.

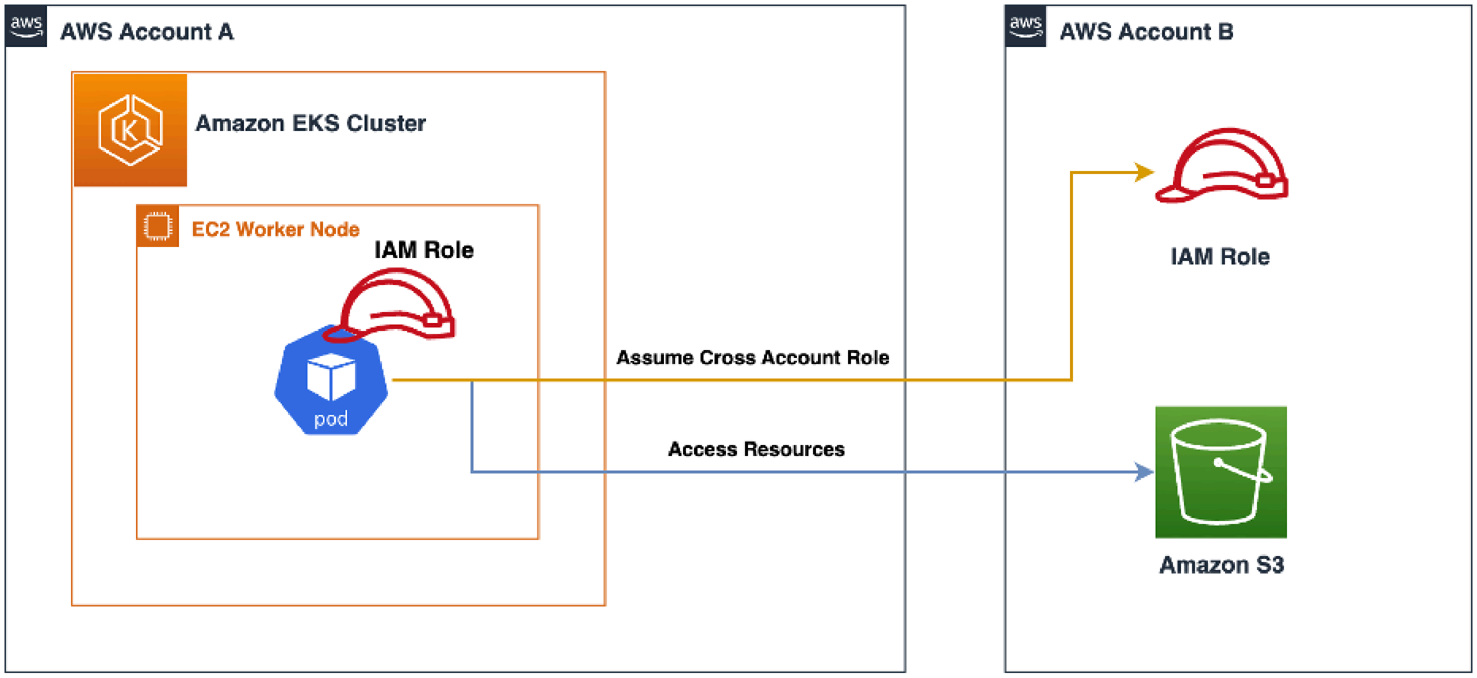


Figure 6: Cross Account Access using IAM Role chaining

Approach 3: Role chaining using AWS Config

This is similar to Approach 2, instead of performing the STS AssumeRole call in the application code, AWS SDK provides a way to define the role chain in ~/.aws/config file. AWS SDK will automatically parse the config file and retrieve the credentials for the destination role. You can create a config file as shown below with both Account A and B roles and specify the AWS_CONFIG_FILE and AWS_PROFILE env vars in your pod spec. Pod identity webhook does not override if the env vars already exists in the pod spec. This approach necessitates the availability of curl, jq utilities within the container image.

```
# Snippet of the PodSpec
containers:
  - name: container-name
    image: container-image:version
    env:
    - name: AWS_CONFIG_FILE
      value: path-to-customer-provided-aws-config-file
    - name: AWS_PROFILE
      value: account_b_role

# Content of the AWS Config file
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
credential_process = /eks-credential-processrole.sh
```

Comparison of Amazon EKS Pod Identity with IRSA

At a high level, both Amazon EKS Pod Identity and IRSA enables you to grant AWS IAM permissions to applications running on EKS clusters. But they are fundamentally different in how you configure them, the limits supported, and features enabled. Below, we compare some of the key facets of both the solutions.

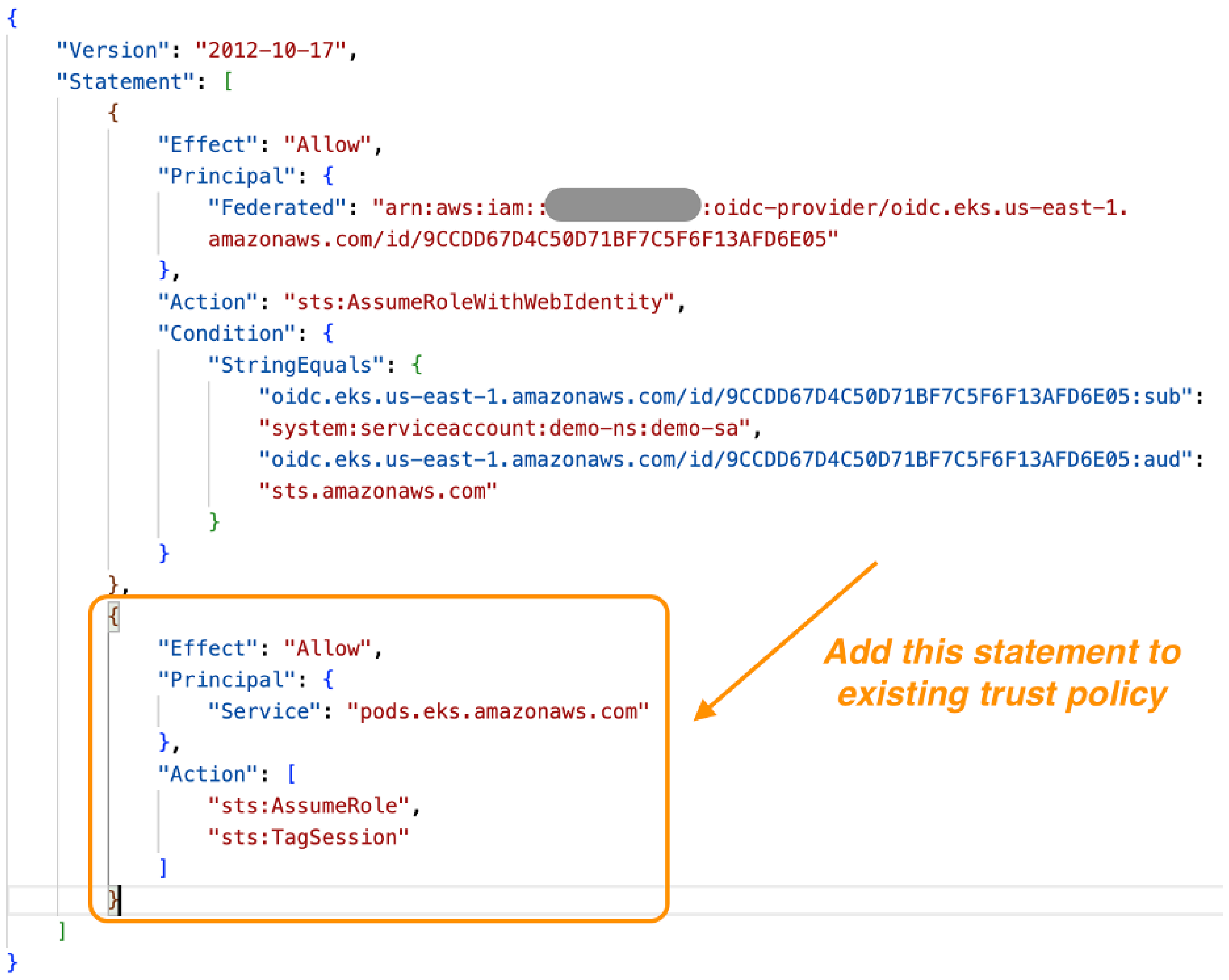
	IRSA	EKS Pod Identity

Role extensibility	You have to update the IAM role's trust policy with the new EKS cluster OIDC provider endpoint each time you want to use the role in a new cluster.	You have to setup the role one time, to establish trust with the newly introduced EKS service principal "pods.eks.amazonaws.com". After this one-time step, you don't need to update the role's trust policy each time it is used in a new cluster.
Account scalability	EKS cluster has an OpenID Connect (OIDC) issuer URL associated with it. To use IRSA, a unique OpenID connect provider needs to be created in IAM for each EKS cluster. IAM OIDC provider has a default global limit of 100 per AWS account. Keep this limit in consideration as you grow the number of EKS clusters per account.	EKS Pod Identity doesn't require users to setup IAM OIDC provider, so this limit doesn't apply.
Role scalability	In IRSA, you define the trust relationship between an IAM role and service account in the role's trust policy. By default, the length of trust policy size is 2048. This means that you can typically define four trust relationships in a single policy. While you can get the trust policy length limit increased, you are typically limited to a maximum of eight trust relationships within a single policy.	EKS Pod Identity doesn't require users to define trust relationship between IAM role and service account in IAM trust policy, so this limit doesn't apply.
Role reusability	IAM role session tags are not supported.	IAM credentials supplied by EKS Pod Identity include support for role session tags. Role session tags enable administrators to author a single IAM role that can be used with multiple service accounts, with different effective permissions, by allowing access to AWS resources based on tags attached to them.
Cluster readiness	IAM roles used in IRSA need to wait for the cluster to be in a "Ready" state, to get the cluster's OpenID Connect Provider URL to complete the IAM role trust policy configuration	IAM roles used in Pod identity can be created ahead of time.

Environments supported	IRSA can be used in EKS, EKS-A, ROSA, self-managed Kubernetes clusters on Amazon EC2	EKS Pod Identity is purpose built for EKS.
Supported EKS versions	All supported EKS versions	EKS version 1.24 and above. See EKS user guide for details.
Cross account access	Cross account here refers to the scenario where your EKS cluster is in one AWS account and the AWS resources that are being accessed by your applications is in another AWS account. In IRSA, you can configure cross account IAM permissions either by creating an IAM identity provider in the account your AWS resources live or by using chained AssumeRole operation. See EKS user guide on IRSA Cross-account IAM permissions for details.	EKS Pod Identity supports cross account access through resource policies and chained AssumeRole operation. See the previous section "How to perform cross account access with EKS Pod Identity" for details.
Mapping inventory	You can find the mapping of IAM roles to service accounts by parsing individual IAM role's trust policy or by inspecting the annotations added to service accounts.	EKS Pod Identity offers a new *ListPodIdentityAssociations* API to centrally see the mapping of roles to service accounts.

How to migrate from IRSA to EKS Pod Identity

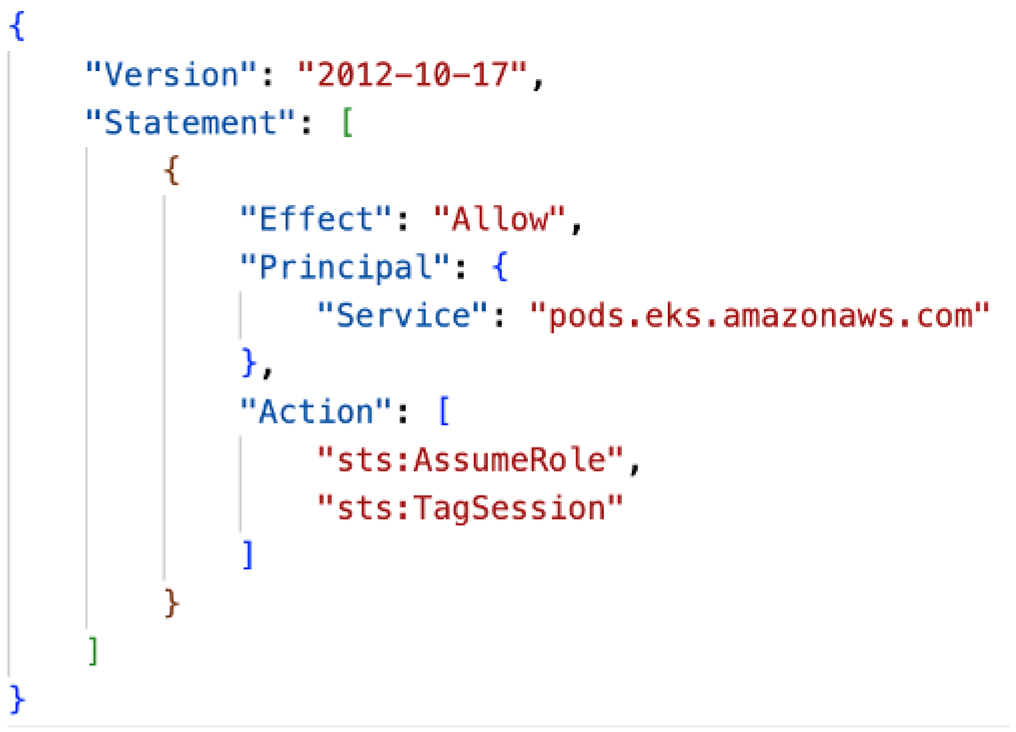
You can migrate your existing IRSA roles to leverage EKS Pod Identity by first meeting the pre-requisites of upgrading your EKS cluster version to version 1.24 and above. Further, you should install the pre-requisite EKS Pod Identity Agent Add-on on your cluster. You also need to update your application to use one of the supported AWS SDK versions called out in EKS documentation here. Once these pre-requisites are met, you can update your IAM role's trust policy with the new EKS service principal *pods.eks.amazonaws.com*. Optionally, to ensure a seamless migration, you can update your existing IAM roles to trust both the new EKS service principal as well as the OIDC provider endpoint supported by IRSA as shown in the following diagram.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::[redacted]:oidc-provider/oidc.eks.us-east-1.
                amazonaws.com/id/9CCDD67D4C50D71BF7C5F6F13AFD6E05"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/9CCDD67D4C50D71BF7C5F6F13AFD6E05:sub":
                    "system:serviceaccount:demo-ns:demo-sa",
                    "oidc.eks.us-east-1.amazonaws.com/id/9CCDD67D4C50D71BF7C5F6F13AFD6E05:aud":
                    "sts.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

Figure 7: Updated IAM Role trust policy to trust both EKS Service Principal and OIDC Provider

The EKS Pod Identity webhook gives preference to EKS Pod Identity over IRSA, when it notices roles setup with both trusted entities. After that, you can create the EKS Pod Identity association by calling the new API **CreatePodIdentityAssociation** to associate the IAM role to the service account. Once the association is in place, all new/restarted pods will have their pod spec mutated with the new environment variables supported by EKS Pod Identity. The AWS SDK used in your application would then use the new environment variable to exchange the projected JWT token for temporary IAM credentials. Once you have verified that everything works as expected, you can remove the old OIDC provider trust association from your IAM role as shown below.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

Figure 8: IAM Role trust policy to trust EKS Service Principal

Considerations for Amazon EKS Pod Identity

- Amazon EKS Pod Identity supports Kubernetes running version 1.24 and above in Amazon EKS. Please refer to the EKS documentation here for the exact Platform versions supported on these Kubernetes versions.
- Amazon EKS Pod Identity feature requires you to upgrade to latest AWS SDKs and install eks-pod-identity-agent EKS Add-on on the cluster. Check out EKS documentation here for the exact list of versions supported.
- EKS Pod Identity is available in all AWS Regions supported by Amazon EKS, except the AWS GovCloud (US) Regions, China (Beijing, operated by Sinnet), and China (Ningxia, operated by NWCD) as of this writing.
- At launch, EKS Pod Identity supports a pre-defined set of IAM role session tags, refer List of session tags added by EKS Pod Identity for the supported tags.
- EKS Add-ons and self-managed Add-ons work with EKS Pod Identity. You must first ensure you are using a version of the add-on that is built using a supported version of AWS SDK. You can call EKS Pod Identity API to create the pod identity association between the IAM role required by the add-on with the appropriate service account. Once the IAM role to service account pod identity association is in place, you can install the add-on to ensure it receives the required IAM permissions. In the future, EKS will add a native integration between EKS add-on and Pod Identity APIs, so Pod Identity APIs are not required to be called separately as pre-requisite step.
- You can associate only one IAM role to a Kubernetes service account using EKS Pod Identity feature. However, you can update the role associations any time.
- EKS Pod Identity is another option for you along with IRSA to grant IAM permissions to applications running on EKS clusters. IRSA works with EKS, EKS-A, ROSA, and self-managed k8s clusters on EC2. EKS Pod Identity is purpose built for EKS, and hence simplifies the experience to obtain IAM permissions for EKS customers. You can use both features simultaneously on a cluster. You can use this approach until all application and Add-ons

running on the cluster are upgraded to use a new AWS SDK that supports EKS Pod Identity. See the previous *How to migrate from IRSA to EKS Pod Identity* section for details.

Cleaning up

To avoid ongoing charges, please make sure to delete Amazon EKS cluster resources created in your AWS account.

Bash
```
# Delete EKS cluster resources
eksctl delete cluster -f cluster.yaml

# Delete AWS Secrets Manager Secrets
aws secretsmanager delete-secret --region $AWS_REGION --secret-id dev-secret
aws secretsmanager delete-secret --region $AWS_REGION --secret-id qa-secret

# Delete IAM Resources
aws iam detach-role-policy --role-name secretmgr-app-eks-pod-identity-role --policy-ar
aws iam delete-role --role-name secretmgr-app-eks-pod-identity-role
aws iam delete-policy --policy-arn $SECRETMGR_APP_POLICY_ARN

aws iam detach-role-policy --role-name s3-app-eks-pod-identity-role --policy-arn arn:a
aws iam delete-role --role-name s3-app-eks-pod-identity-role
```

Conclusion

In this post, we showed you how you can use the Amazon EKS Pod Identity feature to securely grant IAM permissions to Kubernetes applications running in your EKS clusters. We also discussed how to use IAM Session tags to create fine-grained IAM permission policies based on tags, cross-account resource access, and migration steps from IRSA to the new EKS Pod Identity feature. We encourage you to start using this feature and leave us feedback at AWS Containers Roadmap.

TAGS: Amazon EKS, AWS IAM