What is IAM Roles for Service Accounts (IRSA) and Amazon EKS Pod Identity Webhook?

Date: 2023-07-03
modified: 2023-07-12
tags: AWS, EKS, IRSA, IAM, Kubernetes, Webhook, Pod, Amazon
description: IRSA and Amazon EKS Pod Identity Webhook under the hood
category: Container
slug: what-is-aws-iam-roles-for-service-accounts-irsa
Author: Dominik Wombacher
lang: en
transid: what-is-aws-iam-roles-for-service-accounts-irsa
Status: published

This is the first Article of the Series **Integrate Rancher with AWS services**, I did quite a lot with Rancher on Amazon Web Services recently and want to share some of my experiences about the necessary configuration to interact with AWS services for Backup, Logging and Authentication.

I will cover IAM roles for service accounts (Archive: [1], [2]), what it is and how it works under the hood on Amazon Elastic Kubernetes Service (EKS), in this post.

Let's start with a brief overview of the identity and access management challenges that can be solved by using IRSA and some important terminologies.

Update: The recording of my talk Rancher integration with AWS services: possibilities, challenges, outlook (abstract and slide-deck) at openSUSE Conference 23 is online and covers parts of this article as well.

- media.ccc.de (includes options to download video and audio)
- youtube.com

Terminology 1
- AWS IAM Role and Policy 1
- Amazon Elastic Compute Cloud (EC2) Instance IAM Role 2
- AWS IAM access key credentials 2

IAM Roles for Service Accounts (IRSA) 2
- IRSA under the hood 2
- Pod Identity Webhook 3

Conclusion 4

Terminology

AWS IAM Role and Policy

To grant an application access to AWS services, you need an AWS IAM Role and Policy. A IAM Role contains a IAM Permission policy and a IAM Trust relationship. The policy defines *what* can be done with the specified service, for example, uploading files to a Amazon Simple Storage Service (S3) Bucket. The relationship defines *who* can assume the IAM Role to perform the actions based on the attached permission policy.

```
IAM Permission policy >> IAM Role << IAM Trust relationship
```

Amazon Elastic Compute Cloud (EC2) Instance IAM Role

When a IAM Role is attached to an EC2 Instance, every application running on this specific instance has the same level of permissions. If you have one single application running, it's good practice to grant permissions that way. But imagine you have multiple applications running, for example in different containers like in a Kubernetes Environment, then all of them inherit the same set of permissions.

For Example: The IAM Role grants write access to a S3 Bucket, because container #1 is supposed to upload files, by using an EC2 Instance IAM Role, container #2 and container #3 can also write to the same S3 Bucket. Those container run a complete different application, maybe even outside your own control. Worst case, this could cause data corruption, data loss or even a data leak.

```
container 1 --|     IAM Role
              |        |
container 2 --|-- EC2 Instance -- S3 Bucket
              |
container 3 --|
```

AWS IAM access key credentials

Long-term credentials should be avoided whenever possible from a security best practice point of view. AWS access keys are linked to a IAM user and have no expire date. They could be leaked or shared between multiple applications and therefore might have to broad permissions assigned.

I think it's obvious why those type of credentials should not be used, in my opinion not even during development. Start early in the process to replace long-term with short-term credentials!

IAM Roles for Service Accounts (IRSA)

Now that I explained why EC2 Instance IAM Roles and IAM access keys are not a good idea in a Kubernetes environment, what's the alternative? IRSA to the rescue ;) With this feature, IAM Roles are assigned to Kubernetes Service Accounts, which are then linked to specific pods. That way you grant granular permissions on a per service basis inside your Kubernetes cluster.

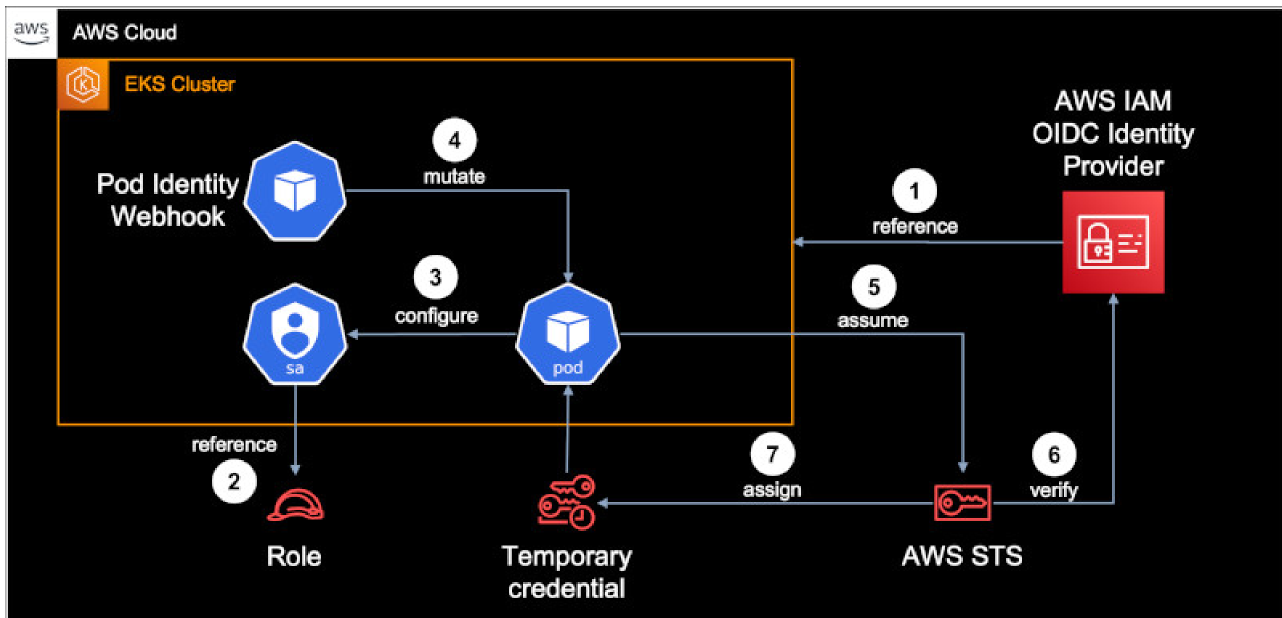
It's an AWS feature and available out-of-the-box on Amazon EKS. the open source solution that makes the dynamic configuration and assignment of temporary credentials possible is Amazon EKS Pod Identity Webhook.

AWS CLI and AWS SDK both fully support IRSA, every application inside a Pod that leverages the CLI or SDK to interact with AWS services, can use IRSA instead of, for example, access keys with none or just minor code changes.

IRSA under the hood

Personally I'm super excited about IRSA and how it works, I think it should be used for every Kubernetes workload on AWS, there is really no reason at all to stick with EC2 Instance IAM Roles or even access key credentials.

That's why I want to dive deeper into how IRSA works and what magic is happening under the hood.



1. A reference between the EKS Cluster and IAM is established via OIDC. This is a one-time setup per cluster and can be done via `eksctl`. Example:

```
eksctl utils associate-iam-oidc-provider --cluster <CUSTER_NAME> --approve
```

2. A reference between a Kubernetes service account and a IAM Role has to be created. This can be done via `eksctl` in two ways, either with an account and the role managed by EKS, or by assigning a role to an existing Kubernetes service account. In case of Rancher, it's option two because Rancher creates and manages Kubernetes service accounts on it's own. Example:

```
eksctl create iamserviceaccount --name <SA_NAME> --namespace <NS_NAME> --cluster <CLUSTER_NAME> \
  --role-name <ROLE_NAME> --attach-policy-arn <IAM_POLICY_ARN> --approve --role-only
```

3. The Kubernetes resource is configured with a appropriate service account annotation, for example as part of a installation via Helm or by adjusting a Manifest.
4. As soon a Pod with a service account annotation comes up, the Pod Identity Webhook will be triggered and reconfigure (mutate) the Pod to use IRSA
5. The Pod assumes the specified IAM Role and connects to the AWS Security Token Service
6. AWS STS verifies the request by contacting AWS IAM
7. If the request could be verified and is valid, AWS STS assigns temporary credentials

AWS CLI and applications leveraging the AWS SDK, can now interact with AWS services based on the permissions of the IAM Role, without the need of EC2 Instance IAM Roles or long-term access key credentials.

As soon the temporary credentials are expired, the process automatically starts over from step #5 to get a new set of temporary credentials, there is no manual interaction required as soon step #1 till #3 are completed.

Further information and examples can be found in the AWS Documentation about IRSA on Amazon EKS.

Pod Identity Webhook

I explained that the Pod Identity Webhook performs a reconfiguration / mutation in step #4 of the IRSA Architecture Diagram. What happens in this step, is that the following Environment variables and Volumes are added to the Pod:

```yaml
env:
- name: AWS_DEFAULT_REGION
  value: us-west-2
- name: AWS_REGION
  value: us-west-2
- name: AWS_ROLE_ARN
  value: "arn:aws:iam::111122223333:role/s3-reader"
- name: AWS_WEB_IDENTITY_TOKEN_FILE
  value: "/var/run/secrets/eks.amazonaws.com/serviceaccount/token"
- name: AWS_STS_REGIONAL_ENDPOINTS
  value: "regional"

volumeMounts:
- mountPath: "/var/run/secrets/eks.amazonaws.com/serviceaccount/"
  name: aws-token

volumes:
- name: aws-token
  projected:
    sources:
    - serviceAccountToken:
        audience: "sts.amazonaws.com"
        expirationSeconds: 86400
        path: token
```

The *region* and *role arn* values are example data and set according to your IAM Role configuration performed in step #2. The Environment variables are used by AWS CLI or SDK to understand that the authentication need to be performed via a token, which is available in the mount `aws-token`. The content of this mount is updated based on the response from AWS STS in step #7.

Conclusion

IRSA (Archive: [1], [2]) is in my opinion a very elegant way to increase security and address some of the most critical IAM challenges. The initial setup on Amazon EKS is normally done within a few minutes, afterwards it's a solution that just works.

Given the fact that the Pod Identity Webhook component is open source, it's also possible to use IRSA on other Kubernetes clusters, which are deployed directly on EC2 and using a CNCF compliant distribution.

With this deep dive into IRSA I wanted to share the benefits and help you to better understand the upcoming articles of this series about Backup and Logging with Rancher on AWS, where it's about how to get it working with IRSA and to avoid long-term credentials.

Article series **Integrate Rancher with AWS services**:

1. **What is IAM Roles for Service Accounts (IRSA) and Amazon EKS Pod Identity Webhook?**
2. Rancher on AWS, Backup to S3 with IRSA for Authentication
3. Rancher on AWS, Logging to CloudWatch with IRSA for Authentication
4. Rancher on AWS, SAML Authentication with AWS IAM Identity Center as SAML IdP (coming soon)
5. Rancher on AWS, GitOps with Fleet and AWS CodeCommit (coming soon)