# EKS Pod Identity and RDS With AWS SecretsManger

Security is a major concern in any Organisation and hence we should change the password or Access Key and Secret Key of the IAM User on a regular basis. For Security reasons it is suggested to store the password in Aws Secrets Manager. The process of changing the password or Access Key and Secrets Key on a periodic basis is known as rotation of secrets or rotation of Keys of IAM User. After rotation of Keys wherever it was used in the code it should be changed on those places. I had used terraform to provision the infrastructure.

For security reasons it is suggestable to use the RBAC (Role Back Access Control) instead of creating the IAM Users Access Key and Secret Key.

In this project I had provisioned the infrastructure using terraform and I had created an AWS Secret with the name of **aws-secrets-dev**. Make sure in your AWS Account a secret with same name should not exist otherwise terraform will give error. If it was existed and of no use the you can delete it first using the command as written below or change the name of AWS Secrets to be created from the terraform script itself.

```
aws secretsmanager delete-secret --secret-id aws-secrets-dev --force-delete-without-recovery --region us-east-2
```
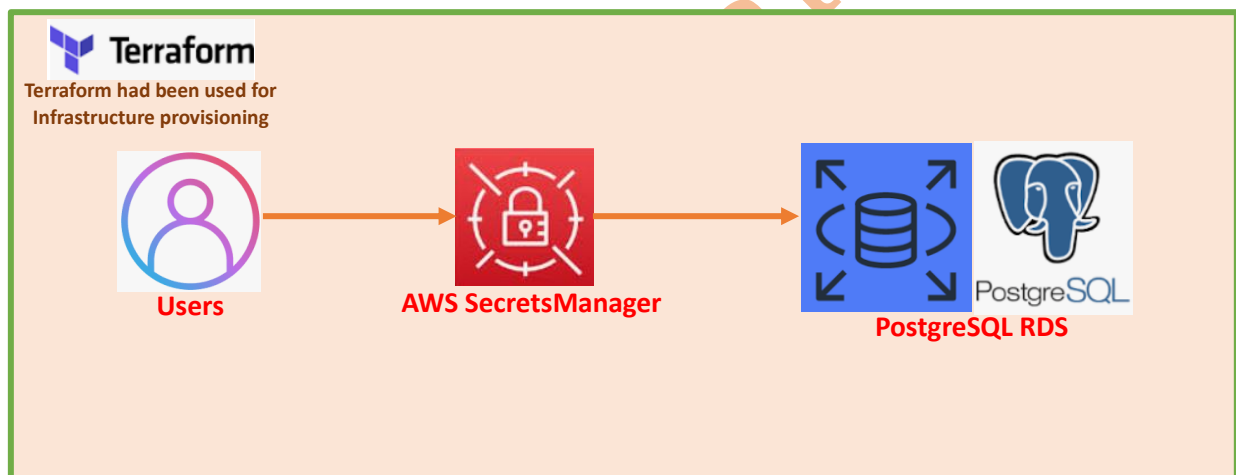

In the first part of this project, I had demonstrated the PostgreSQL RDS and its credentials was stored in AWS Secrets Manager. First, I logged-in with the initial Users Credentials which was stored in the Secrets Manager. Then for demonstration purpose I rotated the Secrets Manager secret (password). However, you should rotate it as per your organisation.

After Rotation the Password for RDS had been changed and user was able to logged-in using the new credentials as shown in the screenshot attached below.

Below screenshot shows the secrets before its rotation.

I was able to logged-in with above shown credential.

```
[root@               ~]# psql -h dbinstance-1.               .us-east-2.rds.amazonaws.com -U postgres --password
Password:
psql (14.13, server 14.9)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=>
```

For demonstration purpose I rotated the secrets from AWS Secrets Manager as shown in the screenshot attached below.

After rotation below is the credentials present in AWS Secrets Manager.

I logged-in with new credentials which was shown in the screenshot attached above.

```
[root@              ~]# psql -h dbinstance-1.              .us-east-2.rds.amazonaws.com -U postgres --password
Password:
psql (14.13, server 14.9)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> \l
                                 List of databases
    Name    |   Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
------------+-----------+----------+-------------+-------------+-----------------------
 demodb     | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres   | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 rdsadmin   | rdsadmin  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | rdsadmin=CTc/rdsadmin
 template0  | rdsadmin  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/rdsadmin          +
            |           |          |             |             | rdsadmin=CTc/rdsadmin
 template1  | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
            |           |          |             |             | postgres=CTc/postgres
(5 rows)

postgres=>
```
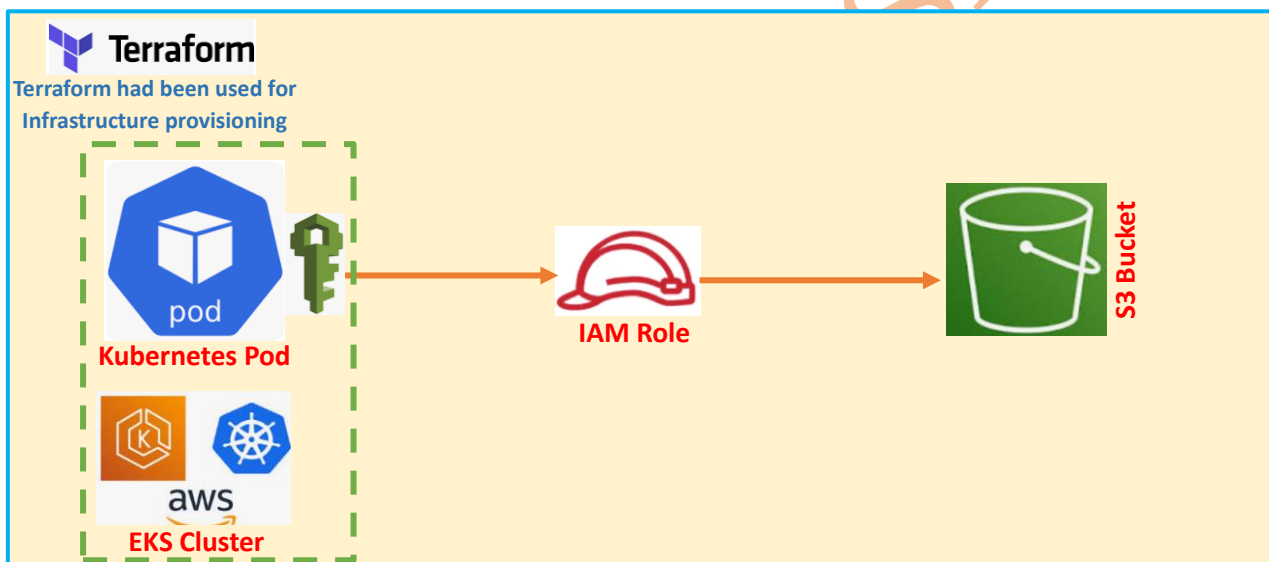

In the second part of this project, I accessed the S3 bucket from the kubernetes Pod present in the EKS Cluster using the Pod Identity. To achieve this using Pod Identity, you should install EKS Pod Identity add on and attach the proper IAM Role with specific namespace and service account in which the pod was existed to the EKS Cluster. For this project I had attached an IAM Role to provide the privilege of Full S3 Bucket and namespace dexter and service account dexter-sa. I had not used the default service account of the namespace (whenever you create a namespace a service account with the name default will also be created).

Below screenshot shows a kubernetes manifest file using which I had created a namespace with the name of dexter, a service account with the name of dexter-sa and a pod with the name of therema. I accessed the S3 Bucket from the Kubernetes Pod as shown in the screenshot attached below.

```
[root@              ~]# kubectl apply -f demo.yaml
namespace/dexter created
serviceaccount/dexter-sa created
pod/therema created
[root@              ~]# kubectl get pods -n dexter
NAME        READY    STATUS            RESTARTS    AGE
therema     0/1      ContainerCreating   0         10s
[root@ip-10-10-4-91 ~]# kubectl get pods -n dexter --watch
NAME        READY    STATUS      RESTARTS    AGE
therema     1/1      Running     0           16s
^C[root@          ~]# kubectl exec -it therema bash -n dexter -- aws s3 ls
2024-12-13 05:47:07 dolo-dempo
[root@          ~]# cat demo.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dexter
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dexter-sa
  namespace: dexter
---
apiVersion: v1
kind: Pod
metadata:
  name: therema
  namespace: dexter
spec:
  serviceAccountName: dexter-sa
  containers:
  - name: demo
    image: amazon/aws-cli
    command: ["/bin/bash", "-c"]
    args: ["aws s3 ls && sleep 7200"]
```

The above pod executed the command **aws s3 ls** if you will check the kubernetes pod logs also you will find it had listed out the s3 buckets which ensures pod had the access of S3 Buckets. You can use below command to check the kubernetes Pod logs **kubectl logs therema -n dexter**.

To see the live logs, you can see the command as written here **kubectl logs -f <pod_name> -n <namespace>**.

```
cat demo.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dexter
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dexter-sa
  namespace: dexter
---
apiVersion: v1
kind: Pod
metadata:
  name: therema
  namespace: dexter
spec:
  serviceAccountName: dexter-sa
  containers:
  - name: demo
    image: amazon/aws-cli
    command: ["/bin/bash", "-c"]
    args: ["aws s3 ls && sleep 7200"]
```

In the third and last part of this project I used Secret Store CSI driver to Access Secrets Manager secret from Kubernetes Pod of the EKS Cluster. To do so first, I installed secret-store-csi-driver using helm chart as shown in the screenshot attached below.

```
[root@_____ ~]# helm repo add secrets-store-csi-driver https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts
"secrets-store-csi-driver" has been added to your repositories
[root@_____ ~]# helm install -n kube-system csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver --set syncSecret.enabled=true --set en
ableSecretRotation=true
NAME: csi-secrets-store
LAST DEPLOYED: _____
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Secrets Store CSI Driver is getting deployed to your cluster.

To verify that Secrets Store CSI Driver has started, run:

  kubectl --namespace=kube-system get pods -l "app=secrets-store-csi-driver"

Now you can follow these steps https://secrets-store-csi-driver.sigs.k8s.io/getting-started/usage.html
to create a SecretProviderClass resource, and a deployment using the SecretProviderClass.
[root@_____ ~]# kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-provider-aws/main/deployment/aws-provider-installer.ya
ml
serviceaccount/csi-secrets-store-provider-aws created
clusterrole.rbac.authorization.k8s.io/csi-secrets-store-provider-aws-cluster-role created
clusterrolebinding.rbac.authorization.k8s.io/csi-secrets-store-provider-aws-cluster-rolebinding created
daemonset.apps/csi-secrets-store-provider-aws created
```

> **helm repo add secrets-store-csi-driver https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts**
>
> **helm install -n kube-system csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver --set syncSecret.enabled=true --set enableSecretRotation=true**
>
> **kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-provider-aws/main/deployment/aws-provider-installer.yaml**

I created a kubernetes manifests file named as mederma.yaml as mentioned below. Using this manifests file I created a namespace named as **mederma,** Secret provider class named as **postgresql-aws-secrets**, service account named as **mederma-sa**, deployment named as **postgresql-asm** and kubernetes service named as **postgresql-svc** in the namespace **mederma**.
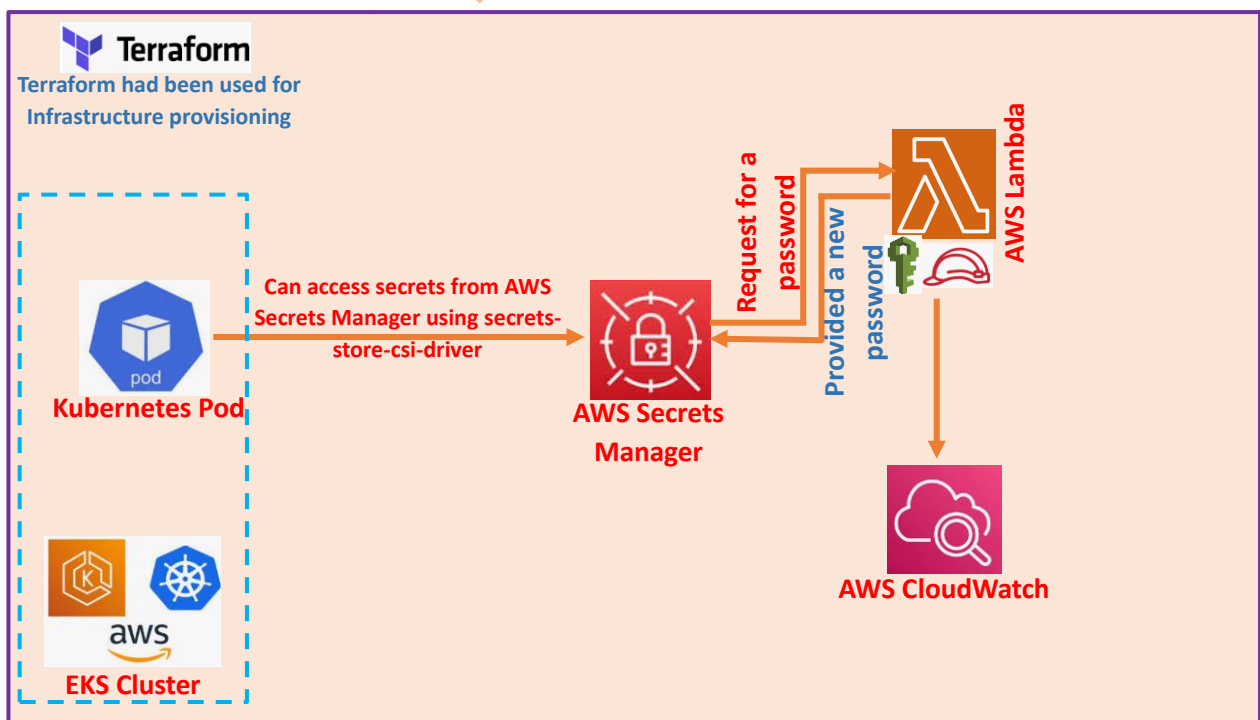
```
[root@            ~]# vim mederma.yaml
[root@            ~]# kubectl apply -f mederma.yaml
namespace/mederma created
secretproviderclass.secrets-store.csi.x-k8s.io/postgresql-aws-secrets created
serviceaccount/mederma-sa created
deployment.apps/postgresql-asm created
service/postgresql-svc created
[root@            ~]# kubectl get pods -n mederma
NAME                          READY   STATUS    RESTARTS   AGE
postgresql-asm-7b465547fc-tqzrq  1/1   Running   0          21s
[root@            ~]# kubectl get svc -n mederma
NAME             TYPE           CLUSTER-IP     EXTERNAL-IP                                          PORT(S)          AGE
postgresql-svc   LoadBalancer   a                                0.us-east-2.elb.amazonaws.com   5432:32699/TCP   44s
[root@            ~]# psql -h a                         0.us-east-2.elb.amazonaws.com -U admin --password
Password:
psql (14.13, server 14.15 (Debian 14.15-1.pgdg120+1))
Type "help" for help.

admin=# \l
                                List of databases
   Name    | Owner | Encoding |  Collate   |   Ctype    | Access privileges
-----------+-------+----------+------------+------------+-------------------
 admin     | admin | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | admin | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin          +
           |       |          |            |            | admin=CTc/admin
 template1 | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin          +
           |       |          |            |            | admin=CTc/admin
(4 rows)
```

I was able to login into the PostgreSQL Kubernetes pod using the Aws secrets Manager secrets credentials as shown below.


```
[root@                    ~]# kubectl get secrets -n mederma
NAME                      TYPE      DATA   AGE
postgresql-k8s-secret     Opaque    2      33m
[root@                    ~]# kubectl get secrets/postgresql-k8s-secret -n mederma -o yaml
apiVersion: v1
data:
  k8s-password: WjZQRjdLNkY0UA==
  k8s-username: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: "                        "
  labels:
    secrets-store.csi.k8s.io/managed: "true"
  name: postgresql-k8s-secret
  namespace: mederma
  ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: postgresql-asm-
    uid:
  resourceVersion: "      "
  uid:
type: Opaque
```

```
[root@                        ~]# echo WjZQRjdLNkY0UA== | base64 -d
Z6PF7K6F4P[root@              ~]#
[root@                        ~]# echo YWRtaW4= | base64 -d
admin[root@                   ~]#
```

```
cat mederma.yaml

---

apiVersion: v1

kind: Namespace

metadata:

  name: mederma

---

apiVersion: secrets-store.csi.x-k8s.io/v1

kind: SecretProviderClass

metadata:

  name: postgresql-aws-secrets

  namespace: mederma

spec:

  provider: aws

  parameters:

    region: us-east-2

    objects: |

      - objectName: "aws-secrets-dev"

        objectType: secretsmanager

        jmesPath:

          - path: username

            objectAlias: medermausername

          - path: password

            objectAlias: medermapassword

  secretObjects:

    - secretName: postgresql-k8s-secret

      type: Opaque

      data:

        - objectName: medermausername

          key: k8s-username

        - objectName: medermapassword
```

```yaml
      key: k8s-password

---

apiVersion: v1

kind: ServiceAccount

metadata:

  name: mederma-sa

  namespace: mederma

  annotations:

    eks.amazonaws.com/role-arn: arn:aws:iam::02XXXXXXXXX6:role/eks-pod-identity-role-
secretmanager-dev

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: postgresql-asm

  namespace: mederma

spec:

  replicas: 1

  selector:

    matchLabels:

      app: postgresql

  template:

    metadata:

      labels:

        app: postgresql

    spec:

      serviceAccountName: mederma-sa

      containers:

        - name: postgresql

          image: postgres:14

          ports:
```

```yaml
        - containerPort: 5432

      volumeMounts:

       - name: secrets

        mountPath: /mnt/secrets

        readOnly: true

      env:

       - name: POSTGRES_USER

        valueFrom:

         secretKeyRef:

          name: postgresql-k8s-secret

          key: k8s-username

       - name: POSTGRES_PASSWORD

        valueFrom:

         secretKeyRef:

          name: postgresql-k8s-secret

          key: k8s-password

    volumes:

     - name: secrets

      csi:

       driver: secrets-store.csi.k8s.io

       readOnly: true

       volumeAttributes:

        secretProviderClass: postgresql-aws-secrets
---
apiVersion: v1

kind: Service

metadata:

 name: postgresql-svc

 namespace: mederma

spec:

 type: LoadBalancer
```
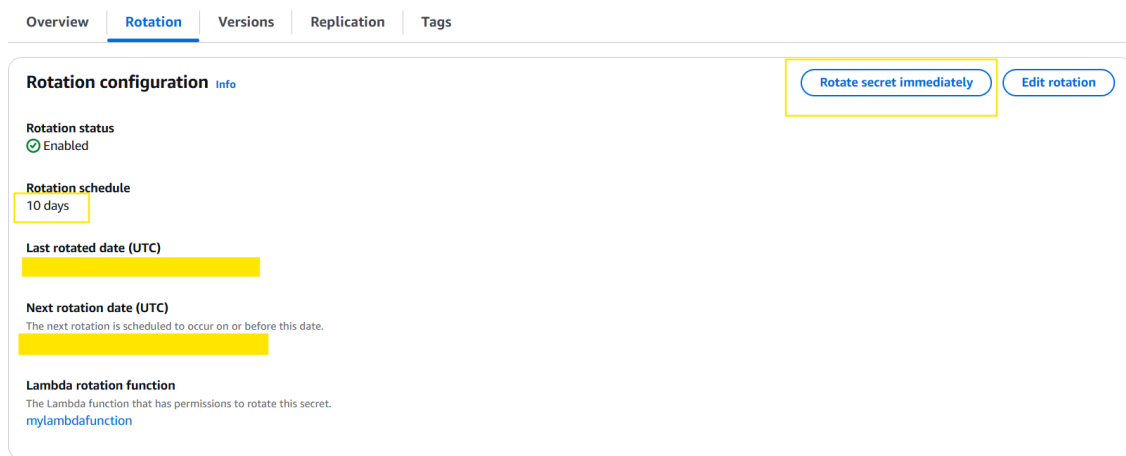
selector:

  app: postgresql

ports:

 - port: 5432

  targetPort: 5432

For demonstration purpose I rotated the password from AWS Secrets Manager. For Password rotation I used Lambda function.
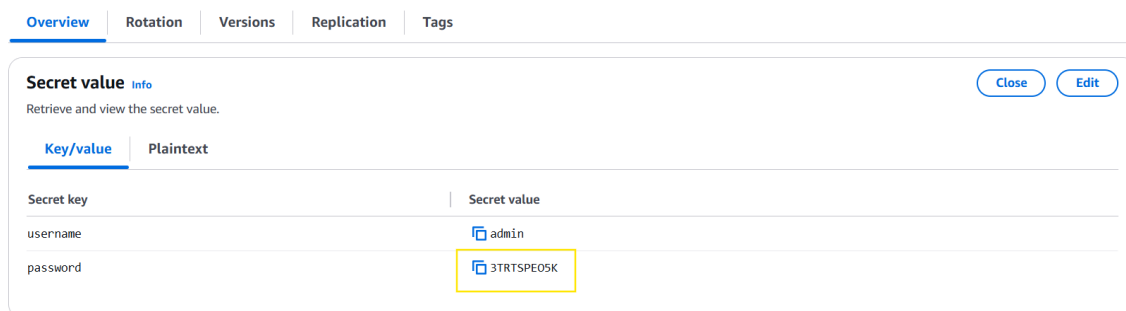

After rotation password had been changed as shown in the screenshot below for AWS Secrets Manager.

I checked the Kubernetes secrets in the namespace **mederma** and found that the new credential had been reflected. Finally, I scaled-down the deployment **postgresql-asm** to replicas as zero and then scaled up the same deployment to replicas as 1. I found the password had been changed and I was able to logged-in into the PostgreSQL pod using the new password as shown in the screenshot attached below.

```
[root@            ~]# kubectl get secrets -n mederma
NAME                     TYPE      DATA    AGE
postgresql-k8s-secret    Opaque    2       40m
[root@            ~]# kubectl get secrets/postgresql-k8s-secret -n mederma -o yaml
apiVersion: v1
data:
  k8s-password: M1RSVFNQRU81Sw==
  k8s-username: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: "                    "
  labels:
    secrets-store.csi.k8s.io/managed: "true"
  name: postgresql-k8s-secret
  namespace: mederma
  ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: postgresql-asm-
    uid:
  resourceVersion: "      "
  uid:
type: Opaque
[root@ip-10-10-4-91 ~]# kubectl scale deployment postgresql-asm --replicas=0 --namespace=mederma
deployment.apps/postgresql-asm scaled
[root@ip-10-10-4-91 ~]# kubectl scale deployment postgresql-asm --replicas=1 --namespace=mederma
deployment.apps/postgresql-asm scaled
```

```
[root@            ~]# echo YWRtaW4= | base64 -d
admin[root@            ~]#
[root@            ~]# echo M1RSVFNQRU81Sw== | base64 -d
3TRTSPEO5K[root@            ~]#
```

```
[root@        ~]# kubectl exec -it postgresql-asm-           bash -n mederma -- cat /mnt/secrets/medermapassword
3TRTSPEO5K[root@        ~]# kubectl exec -it postgresql-asm-           bash -n mederma -- cat /mnt/secrets/medermap
[root@        ~]# kubectl exec -it postgresql-asm-           bash -n mederma -- cat /mnt/secrets/medermausername
admin[root@        ~]#
```

```
[root@        ~]# psql -h a                                    0.us-east-2.elb.amazonaws.com -U admin --password
Password:
psql (14.13, server 14.15 (Debian 14.15-1.pgdg120+1))
Type "help" for help.

admin=# \l
                                List of databases
   Name    | Owner | Encoding | Collate     | Ctype       | Access privileges
-----------+-------+----------+-------------+-------------+-------------------
 admin     | admin | UTF8     | en_US.utf8  | en_US.utf8  |
 postgres  | admin | UTF8     | en_US.utf8  | en_US.utf8  |
 template0 | admin | UTF8     | en_US.utf8  | en_US.utf8  | =c/admin          +
           |       |          |             |             | admin=CTc/admin
 template1 | admin | UTF8     | en_US.utf8  | en_US.utf8  | =c/admin          +
           |       |          |             |             | admin=CTc/admin
(4 rows)
```

OIDC (Open ID Connect) and **Federated Identity** had been used to access the AWS Secrets Manager secret from Kubernetes Pod of EKS Cluster.

OIDC is a protocol using which a User can logged-in into one Application and can access other applications. **Federated Identity** is method by which user can access multiple Applications using single set of credentials.

Reference: - https://github.com/mihirbhatt4687/lambda-secret-rotate.git