Red Hat OpenShift GitOps 1.12

Security

Using security features to configure secure communication and protect the possibly sensitive data in transit

Last Updated: 2024-05-13

Red Hat OpenShift GitOps 1.12 Security

Using security features to configure secure communication and protect the possibly sensitive data in transit

Legal Notice

Abstract

This document provides instructions for using the Transport Layer Security (TLS) encryption with the OpenShift GitOps. It also discusses how to configure secure communication with Redis to protect the possibly sensitive data in transit.

Table of Contents

CHAPTER 1. CONFIGURING SECURE COMMUNICATION WITH REDIS

Using the Transport Layer Security (TLS) encryption with Red Hat OpenShift GitOps, you can secure the communication between the Argo CD components and Redis cache and protect the possibly sensitive data in transit.

You can secure communication with Redis by using one of the following configurations:

- Enable the **autotls** setting to issue an appropriate certificate for TLS encryption.
- Manually configure the TLS encryption by creating the **argocd-operator-redis-tls** secret with a key and certificate pair.

Both configurations are possible with or without the High Availability (HA) enabled.

1.1. PREREQUISITES

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- Red Hat OpenShift GitOps Operator is installed on your cluster.

1.2. CONFIGURING TLS FOR REDIS WITH AUTOTLS ENABLED

You can configure TLS encryption for Redis by enabling the **autotls** setting on a new or already existing Argo CD instance. The configuration automatically provisions the **argocd-operator-redis-tls** secret and does not require further steps. Currently, OpenShift Container Platform is the only supported secret provider.

NOTE

By default, the **autotls** setting is disabled.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create an Argo CD instance with **autotls** enabled:
   a. In the **Administrator** perspective of the web console, use the left navigation panel to go to **Administration → CustomResourceDefinitions**.
   b. Search for **argocds.argoproj.io** and click **ArgoCD** custom resource definition (CRD).
   c. On the **CustomResourceDefinition details** page, click the **Instances** tab, and then click **Create ArgoCD**.
   d. Edit or replace the YAML similar to the following example:

   Example Argo CD CR with autotls enabled

```
apiVersion: argoproj.io/v1beta1
```

```
kind: ArgoCD
metadata:
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  redis:
    autotls: openshift (3)
  ha:
    enabled: true (4)
```

1. The name of the Argo CD instance.
2. The namespace where you want to run the Argo CD instance.
3. The flag that enables the **autotls** setting and creates a TLS certificate for Redis.
4. The flag value that enables the HA feature. If you do not want to enable HA, do not include this line or set the flag value as **false**.

TIP

Alternatively, you can enable the **autotls** setting on an already existing Argo CD instance by running the following command:

```
$ oc patch argocds.argoproj.io <instance-name> --type=merge -p '{"spec":{"redis":
{"autotls":"openshift"}}}'
```

e. Click **Create**.

f. Verify that the Argo CD pods are ready and running:

```
$ oc get pods -n <namespace> (1)
```

1. Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

Example output with HA disabled

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```

NOTE

The HA-enabled TLS configuration requires a cluster with at least three worker nodes. It can take a few minutes for the output to appear if you have enabled the Argo CD instances with HA configuration.

Example output with HA enabled

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h   1/1     Running   0          10m
argocd-redis-ha-server-0            2/2     Running   0          9m9s
argocd-redis-ha-server-1            2/2     Running   0          98s
argocd-redis-ha-server-2            2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh     1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks       1/1     Running   0          10m
```

3. Verify that the **argocd-operator-redis-tls** secret is created:

```
$ oc get secrets argocd-operator-redis-tls -n <namespace>
```

(1) Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

Example output

```
NAME                    TYPE                DATA   AGE
argocd-operator-redis-tls   kubernetes.io/tls   2      30s
```

The secret must be of the **kubernetes.io/tls** type and a size of **2**.

1.3. CONFIGURING TLS FOR REDIS WITH AUTOTLS DISABLED

You can manually configure TLS encryption for Redis by creating the **argocd-operator-redis-tls** secret with a key and certificate pair. In addition, you must annotate the secret to indicate that it belongs to the appropriate Argo CD instance. The steps to create a certificate and secret vary for instances with High Availability (HA) enabled.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create an Argo CD instance:
   a. In the **Administrator** perspective of the web console, use the left navigation panel to go to **Administration → CustomResourceDefinitions**.
   b. Search for **argocds.argoproj.io** and click **ArgoCD** custom resource definition (CRD).
   c. On the **CustomResourceDefinition details** page, click the **Instances** tab, and then click **Create ArgoCD**.
   d. Edit or replace the YAML similar to the following example:

Example ArgoCD CR with autotls disabled

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
```

```
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  ha:
    enabled: true (3)
```

(1) The name of the Argo CD instance.

(2) The namespace where you want to run the Argo CD instance.

(3) The flag value that enables the HA feature. If you do not want to enable HA, do not include this line or set the flag value as **false**.

e. Click **Create**.

f. Verify that the Argo CD pods are ready and running:

```
$ oc get pods -n <namespace> (1)
```

(1) Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

Example output with HA disabled

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```

NOTE

The HA-enabled TLS configuration requires a cluster with at least three worker nodes. It can take a few minutes for the output to appear if you have enabled the Argo CD instances with HA configuration.

Example output with HA enabled

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h  1/1     Running   0          10m
argocd-redis-ha-server-0            2/2     Running   0          9m9s
argocd-redis-ha-server-1            2/2     Running   0          98s
argocd-redis-ha-server-2            2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh 1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks       1/1     Running   0          10m
```

3. Create a self-signed certificate for the Redis server by using one of the following options depending on your HA configuration:

- For the Argo CD instance with HA disabled, run the following command:

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
  -keyout /tmp/redis.key \
  -out /tmp/redis.crt \
  -newkey rsa:4096 \
  -nodes \
  -sha256 \
  -days 10
```

1 Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

Example output

```
Generating a RSA private key
..............++++
.........................++++
writing new private key to '/tmp/redis.key'
```

- For the Argo CD instance with HA enabled, run the following command:

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis-ha-haproxy.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
  -keyout /tmp/redis-ha.key \
  -out /tmp/redis-ha.crt \
  -newkey rsa:4096 \
  -nodes \
  -sha256 \
  -days 10
```

1 Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

Example output

```
Generating a RSA private key
..............++++
.........................++++
writing new private key to '/tmp/redis-ha.key'
```

4. Verify that the generated certificate and key are available in the **/tmp** directory by running the following commands:

```
$ cd /tmp
```

```
$ ls
```

Example output with HA disabled

```
...
redis.crt
redis.key
...
```

Example output with HA enabled

```
...
redis-ha.crt
redis-ha.key
...
```

5. Create the **argocd-operator-redis-tls** secret by using one of the following options depending on your HA configuration:

   - For the Argo CD instance with HA disabled, run the following command:

     ```
     $ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis.key --cert=/tmp/redis.crt
     ```

   - For the Argo CD instance with HA enabled, run the following command:

     ```
     $ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis-ha.key --cert=/tmp/redis-ha.crt
     ```

     #### Example output

     ```
     secret/argocd-operator-redis-tls created
     ```

6. Annotate the secret to indicate that it belongs to the Argo CD CR:

   ```
   $ oc annotate secret argocd-operator-redis-tls argocds.argoproj.io/name=<instance-name> (1)
   ```

   (1) Specify a name of the Argo CD instance, for example **argocd**.

   #### Example output

   ```
   secret/argocd-operator-redis-tls annotated
   ```

7. Verify that the Argo CD pods are ready and running:

   ```
   $ oc get pods -n <namespace> (1)
   ```

   (1) Specify a namespace where the Argo CD instance is running, for example **openshift-gitops**.

   #### Example output with HA disabled

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```

NOTE

It can take a few minutes for the output to appear if you have enabled the Argo CD instances with HA configuration.

Example output with HA enabled

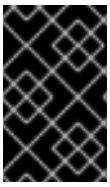
```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h   1/1     Running   0          10m
argocd-redis-ha-server-0            2/2     Running   0          9m9s
argocd-redis-ha-server-1            2/2     Running   0          98s
argocd-redis-ha-server-2            2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh 1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks       1/1     Running   0          10m
```

CHAPTER 2. MANAGING SECRETS SECURELY USING SECRETS STORE CSI DRIVER WITH GITOPS

This guide walks you through the process of integrating the Secrets Store Container Storage Interface (SSCSI) driver with the GitOps Operator in OpenShift Container Platform 4.14 and later.

2.1. OVERVIEW OF MANAGING SECRETS USING SECRETS STORE CSI DRIVER WITH GITOPS

Some applications need sensitive information, such as passwords and usernames which must be concealed as good security practice. If sensitive information is exposed because role-based access control (RBAC) is not configured properly on your cluster, anyone with API or etcd access can retrieve or modify a secret.



IMPORTANT

Anyone who is authorized to create a pod in a namespace can use that RBAC to read any secret in that namespace. With the SSCSI Driver Operator, you can use an external secrets store to store and provide sensitive information to pods securely.

The process of integrating the OpenShift Container Platform SSCSI driver with the GitOps Operator consists of the following procedures:

1. Storing AWS Secrets Manager resources in GitOps repository
2. Configuring SSCSI driver to mount secrets from AWS Secrets Manager
3. Configuring GitOps managed resources to use mounted secrets

2.1.1. Benefits

Integrating the SSCSI driver with the GitOps Operator provides the following benefits:

- Enhance the security and efficiency of your GitOps workflows
- Facilitate the secure attachment of secrets into deployment pods as a volume
- Ensure that sensitive information is accessed securely and efficiently

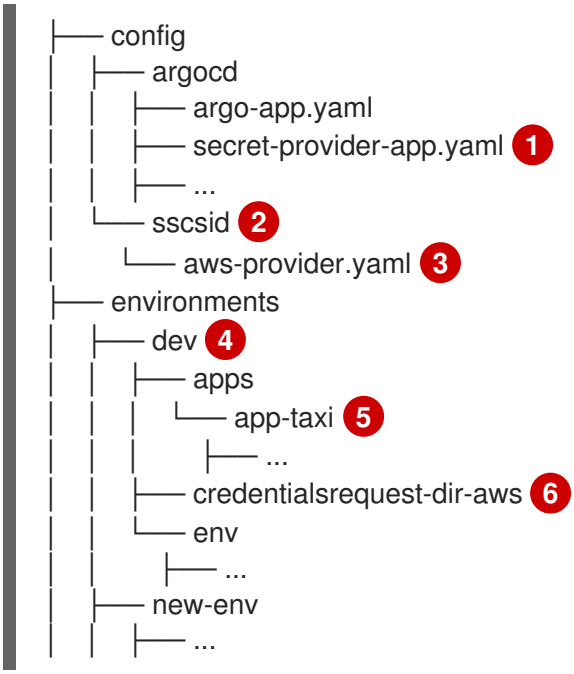
2.1.2. Secrets store providers

The following secrets store providers are available for use with the Secrets Store CSI Driver Operator:

- AWS Secrets Manager
- AWS Systems Manager Parameter Store
- Microsoft Azure Key Vault

As an example, consider that you are using AWS Secrets Manager as your secrets store provider with the SSCSI Driver Operator. The following example shows the directory structure in GitOps repository that is ready to use the secrets from AWS Secrets Manager:

Example directory structure in GitOps repository



2 Directory that stores the **aws-provider.yaml** file.

3 Configuration file that installs the AWS Secrets Manager provider and deploys resources for it.

1 Configuration file that creates an application and deploys resources for AWS Secrets Manager.

4 Directory that stores the deployment pod and credential requests.

5 Directory that stores the **SecretProviderClass** resources to define your secrets store provider.

6 Folder that stores the **credentialsrequest.yaml** file. This file contains the configuration for the credentials request to mount a secret to the deployment pod.

2.2. PREREQUISITES

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have extracted and prepared the **ccoctl** binary.
- You have installed the **jq** CLI tool.
- Your cluster is installed on AWS and uses AWS Security Token Service (STS).
- You have configured AWS Secrets Manager to store the required secrets.
- SSCSI Driver Operator is installed on your cluster .
- Red Hat OpenShift GitOps Operator is installed on your cluster.
- You have a GitOps repository ready to use the secrets.
- You are logged in to the Argo CD instance by using the Argo CD admin account.

2.3. STORING AWS SECRETS MANAGER RESOURCES IN GITOPS REPOSITORY

This guide provides instructions with examples to help you use GitOps workflows with the Secrets Store Container Storage Interface (SSCSI) Driver Operator to mount secrets from AWS Secrets Manager to a CSI volume in OpenShift Container Platform.

IMPORTANT

Using the SSCSI Driver Operator with AWS Secrets Manager is not supported in a hosted control plane cluster.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have extracted and prepared the **ccoctl** binary.
- You have installed the **jq** CLI tool.
- Your cluster is installed on AWS and uses AWS Security Token Service (STS).
- You have configured AWS Secrets Manager to store the required secrets.
- SSCSI Driver Operator is installed on your cluster .
- Red Hat OpenShift GitOps Operator is installed on your cluster.
- You have a GitOps repository ready to use the secrets.
- You are logged in to the Argo CD instance by using the Argo CD admin account.

Procedure

1. Install the AWS Secrets Manager provider and add resources:
   a. In your GitOps repository, create a directory and add **aws-provider.yaml** file in it with the following configuration to deploy resources for the AWS Secrets Manager provider:

IMPORTANT

The AWS Secrets Manager provider for the SSCSI driver is an upstream provider.

This configuration is modified from the configuration provided in the upstream AWS documentation so that it works properly with OpenShift Container Platform. Changes to this configuration might impact functionality.

Example aws-provider.yaml file

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csi-secrets-store-provider-aws-cluster-role
rules:
- apiGroups: [""]
  resources: ["serviceaccounts/token"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: csi-secrets-store-provider-aws-cluster-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: csi-secrets-store-provider-aws-cluster-role
subjects:
- kind: ServiceAccount
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: openshift-cluster-csi-drivers
  name: csi-secrets-store-provider-aws
  labels:
    app: csi-secrets-store-provider-aws
spec:
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: csi-secrets-store-provider-aws
  template:
    metadata:
      labels:
        app: csi-secrets-store-provider-aws
    spec:
      serviceAccountName: csi-secrets-store-provider-aws
      hostNetwork: false
      containers:
        - name: provider-aws-installer
```

```
        image: public.ecr.aws/aws-secrets-manager/secrets-store-csi-driver-provider-
aws:1.0.r2-50-g5b4aca1-2023.06.09.21.19
        imagePullPolicy: Always
        args:
          - --provider-volume=/etc/kubernetes/secrets-store-csi-providers
        resources:
          requests:
            cpu: 50m
            memory: 100Mi
          limits:
            cpu: 50m
            memory: 100Mi
        securityContext:
          privileged: true
        volumeMounts:
          - mountPath: "/etc/kubernetes/secrets-store-csi-providers"
            name: providervol
          - name: mountpoint-dir
            mountPath: /var/lib/kubelet/pods
            mountPropagation: HostToContainer
      tolerations:
      - operator: Exists
      volumes:
        - name: providervol
          hostPath:
            path: "/etc/kubernetes/secrets-store-csi-providers"
        - name: mountpoint-dir
          hostPath:
            path: /var/lib/kubelet/pods
            type: DirectoryOrCreate
      nodeSelector:
        kubernetes.io/os: linux
```

b. Add a **secret-provider-app.yaml** file in your GitOps repository to create an application and deploy resources for AWS Secrets Manager:

Example secret-provider-app.yaml file

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: secret-provider-app
  namespace: openshift-gitops
spec:
  destination:
    namespace: openshift-cluster-csi-drivers
    server: https://kubernetes.default.svc
  project: default
  source:
    path: path/to/aws-provider/resources
    repoURL: https://github.com/<my-domain>/<gitops>.git (1)
  syncPolicy:
    automated:
    prune: true
    selfHeal: true
```

1. Update the value of the **repoURL** field to point to your GitOps repository.

2. Synchronize resources with the default Argo CD instance to deploy them in the cluster:

   a. Add a label to the **openshift-cluster-csi-drivers** namespace your application is deployed in so that the Argo CD instance in the **openshift-gitops** namespace can manage it:

   ```
   $ oc label namespace openshift-cluster-csi-drivers argocd.argoproj.io/managed-by=openshift-gitops
   ```

   b. Apply the resources in your GitOps repository to your cluster, including the **aws-provider.yaml** file you just pushed:

   Example output

   ```
   application.argoproj.io/argo-app created
   application.argoproj.io/secret-provider-app created
   ...
   ```

In the Argo CD UI, you can observe that the **csi-secrets-store-provider-aws** daemonset continues to synchronize resources. To resolve this issue, you must configure the SSCSI driver to mount secrets from the AWS Secrets Manager.

2.4. CONFIGURING SSCSI DRIVER TO MOUNT SECRETS FROM AWS SECRETS MANAGER

To store and manage your secrets securely, use GitOps workflows and configure the Secrets Store Container Storage Interface (SSCSI) Driver Operator to mount secrets from AWS Secrets Manager to a CSI volume in OpenShift Container Platform. For example, consider that you want to mount a secret to a deployment pod under the **dev** namespace which is in the **/environments/dev/** directory.

Prerequisites

- You have the AWS Secrets Manager resources stored in your GitOps repository.

Procedure

1. Grant privileged access to the **csi-secrets-store-provider-aws** service account by running the following command:

   ```
   $ oc adm policy add-scc-to-user privileged -z csi-secrets-store-provider-aws -n openshift-cluster-csi-drivers
   ```

   Example output

   ```
   clusterrole.rbac.authorization.k8s.io/system:openshift:scc:privileged added: "csi-secrets-store-provider-aws"
   ```

2. Grant permission to allow the service account to read the AWS secret object:

   a. Create a **credentialsrequest-dir-aws** folder under a namespace-scoped directory in your GitOps repository because the credentials request is namespace-scoped. For example,

create a **credentialsrequest-dir-aws** folder under the **dev** namespace which is in the **/environments/dev/** directory by running the following command:

```
$ mkdir credentialsrequest-dir-aws
```

b. Create a YAML file with the following configuration for the credentials request in the **/environments/dev/credentialsrequest-dir-aws/** path to mount a secret to the deployment pod in the **dev** namespace:

Example credentialsrequest.yaml file

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: aws-provider-test
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - "secretsmanager:GetSecretValue"
      - "secretsmanager:DescribeSecret"
      effect: Allow
      resource: "<aws_secret_arn>" 1
secretRef:
  name: aws-creds
  namespace: dev 2
serviceAccountNames:
  - default
```

2 The namespace for the secret reference. Update the value of this **namespace** field according to your project deployment setup.

1 The ARN of your secret in the region where your cluster is on. The **<aws_region>** of **<aws_secret_arn>** has to match the cluster region. If it does not match, create a replication of your secret in the region where your cluster is on.

TIP

To find your cluster region, run the command:

```
$ oc get infrastructure cluster -o jsonpath='{.status.platformStatus.aws.region}'
```

Example output

```
us-west-2
```

c. Retrieve the OIDC provider by running the following command:

```
$ oc get --raw=/.well-known/openid-configuration | jq -r '.issuer'
```

-

Example output

```
https://<oidc_provider_name>
```

Copy the OIDC provider name **<oidc_provider_name>** from the output to use in the next step.

d. Use the **ccoctl** tool to process the credentials request by running the following command:

```
$ ccoctl aws create-iam-roles \
    --name my-role --region=<aws_region> \
    --credentials-requests-dir=credentialsrequest-dir-aws \
    --identity-provider-arn arn:aws:iam::<aws_account>:oidc-
provider/<oidc_provider_name> --output-dir=credrequests-ccoctl-output
```

Example output

```
2023/05/15 18:10:34 Role arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-
aws-creds created
2023/05/15 18:10:34 Saved credentials configuration to: credrequests-ccoctl-
output/manifests/my-namespace-aws-creds-credentials.yaml
2023/05/15 18:10:35 Updated Role policy for Role my-role-my-namespace-aws-creds
```

Copy the **<aws_role_arn>** from the output to use in the next step. For example, **arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-aws-creds**.

e. Check the role policy on AWS to confirm the **<aws_region>** of **"Resource"** in the role policy matches the cluster region:

Example role policy

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "secretsmanager:GetSecretValue",
    "secretsmanager:DescribeSecret"
   ],
   "Resource": "arn:aws:secretsmanager:<aws_region>:<aws_account_id>:secret:my-
secret-xxxxxx"
  }
 ]
}
```

f. Bind the service account with the role ARN by running the following command:

```
$ oc annotate -n <namespace> sa/<app_service_account> eks.amazonaws.com/role-
arn="<aws_role_arn>"
```

Example command

-

```
$ oc annotate -n dev sa/default eks.amazonaws.com/role-arn="<aws_role_arn>"
```

Example output

```
serviceaccount/default annotated
```

3. Create a namespace-scoped **SecretProviderClass** resource to define your secrets store provider. For example, you create a **SecretProviderClass** resource in **/environments/dev/apps/app-taxi/services/taxi/base/config** directory of your GitOps repository.

   a. Create a **secret-provider-class-aws.yaml** file in the same directory where the target deployment is located in your GitOps repository:

   Example secret-provider-class-aws.yaml

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-aws-provider (1)
  namespace: dev (2)
spec:
  provider: aws (3)
  parameters: (4)
    objects: |
      - objectName: "testSecret" (5)
        objectType: "secretsmanager"
```

   (1) Name of the secret provider class.

   (2) Namespace for the secret provider class. The namespace must match the namespace of the resource which will use the secret.

   (3) Name of the secret store provider.

   (4) Specifies the provider-specific configuration parameters.

   (5) The secret name you created in AWS.

   b. Verify that after pushing this YAML file to your GitOps repository, the namespace-scoped **SecretProviderClass** resource is populated in the target application page in the Argo CD UI.

NOTE

If the Sync Policy of your application is not set to **Auto**, you can manually sync the **SecretProviderClass** resource by clicking **Sync** in the Argo CD UI.

2.5. CONFIGURING GITOPS MANAGED RESOURCES TO USE MOUNTED SECRETS

You must configure the GitOps managed resources by adding volume mounts configuration to a deployment and configuring the container pod to use the mounted secret.

Prerequisites

- You have the AWS Secrets Manager resources stored in your GitOps repository.
- You have the Secrets Store Container Storage Interface (SSCSI) driver configured to mount secrets from AWS Secrets Manager.

Procedure

1. Configure the GitOps managed resources. For example, consider that you want to add volume mounts configuration to the deployment of **app-taxi** application and the **100-deployment.yaml** file is in the **/environments/dev/apps/app-taxi/services/taxi/base/config/** directory.
   a. Add the volume mounting to the deployment YAML file and configure the container pod to use the secret provider class resources and mounted secret:

Example YAML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: taxi
  namespace: dev (1)
spec:
  replicas: 1
  template:
    metadata:
# ...
    spec:
      containers:
        - image: nginxinc/nginx-unprivileged:latest
          imagePullPolicy: Always
          name: taxi
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: secrets-store-inline
              mountPath: "/mnt/secrets-store" (2)
              readOnly: true
          resources: {}
    serviceAccountName: default
    volumes:
      - name: secrets-store-inline
        csi:
          driver: secrets-store.csi.k8s.io
          readOnly: true
          volumeAttributes:
            secretProviderClass: "my-aws-provider" (3)
    status: {}
# ...
```

1. Namespace for the deployment. This must be the same namespace as the secret provider class.
2. The path to mount secrets in the volume mount.
3. Name of the secret provider class.

b. Push the updated resource YAML file to your GitOps repository.

2. In the Argo CD UI, click **REFRESH** on the target application page to apply the updated deployment manifest.
3. Verify that all the resources are successfully synchronized on the target application page.
4. Verify that you can you can access the secrets from AWS Secrets manager in the pod volume mount:

a. List the secrets in the pod mount:

```
$ oc exec <deployment_name>-<hash> -n <namespace> -- ls /mnt/secrets-store/
```

Example command

```
$ oc exec taxi-5959644f9-t847m -n dev -- ls /mnt/secrets-store/
```

Example output

```
<secret_name>
```

b. View a secret in the pod mount:

```
$ oc exec <deployment_name>-<hash> -n <namespace> -- cat /mnt/secrets-store/<secret_name>
```

Example command

```
$ oc exec taxi-5959644f9-t847m -n dev -- cat /mnt/secrets-store/testSecret
```

Example output

```
<secret_value>
```

2.6. ADDITIONAL RESOURCES

- Obtaining the **ccoctl** tool
- Configuring the Cloud Credential Operator utility
- Configure your AWS cluster to use AWS STS
- Configuring AWS Secrets Manager to store the required secrets

- About the Secrets Store CSI Driver Operator
- Mounting secrets from an external secrets store to a CSI volume