Red Hat OpenShift GitOps 1.13

セキュリティー

セキュリティー機能を使用したセキュアな通信の設定と転送中の機密データの保護

Last Updated: 2024-08-12

Red Hat OpenShift GitOps 1.13 セキュリティー

セキュリティー機能を使用したセキュアな通信の設定と転送中の機密データの保護

法律上の通知

概要

このドキュメントでは、OpenShift GitOps で Transport Layer Security (TLS) 暗号化を使用する手順を説明します。また、転送中の機密データを保護するために、Redis とのセキュアな通信を設定する方法についても説明します。

目次

第1章 REDIS とのセキュアな通信の設定

Red Hat OpenShift GitOps で Transport Layer Security (TLS) 暗号化を使用すると、Argo CD コンポーネントと Redis キャッシュ間の通信を保護し、機密情報の可能性がある転送中のデータを保護できます。

次の設定のいずれかを使用して、Redis との通信を保護できます。

- **autotls** 設定を有効にして、TLS 暗号化に適切な証明書を発行します。
- キーと証明書のペアを使用して **argocd-operator-redis-tls** シークレットを作成し、TLS 暗号化を手動で設定します。

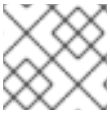
どちらの設定も、高可用性 (HA) が有効になっているかどうかに関係なく可能です。

1.1. 前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。

1.2. AUTOTLS を有効にして REDIS の TLS を設定する

新規または既存の Argo CD インスタンスで **autotls** 設定を有効にすることで、Redis の TLS 暗号化を設定できます。この設定では、**argocd-operator-redis-tls** シークレットが自動的にプロビジョニングされるため、それ以上の手順は必要ありません。現時点で、OpenShift Container Platform は唯一サポートされているシークレットプロバイダーです。



注記

デフォルトでは、**autotls** 設定は無効になっています。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **autotls** を有効にして Argo CD インスタンスを作成します。
   a. Web コンソールの **Administrator** パースペクティブで、左側のナビゲーションパネルを使用して、**Administration → CustomResourceDefinitions** に移動します。
   b. **argocds.argoproj.io** を検索し、**ArgoCD** カスタムリソース定義 (CRD) をクリックします。
   c. **CustomResourceDefinition の詳細** ページで、**Instances** タブをクリックし、**Create ArgoCD** をクリックします。
   d. 次の例のように YAML を編集または置換します。

   autotls を有効にした Argo CD CR の例

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

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

1. Argo CD インスタンスの名前。
2. Argo CD インスタンスを実行する namespace。
3. **autotls** 設定を有効にし、Redis の TLS 証明書を作成するフラグ。
4. HA 機能を有効にするフラグの値。HA を有効にしたくない場合は、この行を含めないか、フラグ値を **false** に設定します。

ヒント

あるいは、次のコマンドを実行して、既存の Argo CD インスタンスで **autotls** に設定を有効にすることもできます。

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

e. **Create** をクリックします。

f. Argo CD Pod が準備ができており、実行中であることを確認します。

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

1. Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

HA を無効にした場合の出力例

```
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
```

注記

HA 対応の TLS 設定には、少なくとも 3 つのワーカーノードを備えたクラスターが必要です。HA 設定で Argo CD インスタンスを有効にしている場合は、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

```
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. **argocd-operator-redis-tls** シークレットが作成されていることを確認します。

```
$ oc get secrets argocd-operator-redis-tls -n <namespace>
```
(1)

(1) Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

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

シークレットは **kubernetes.io/tls** タイプで、サイズが **2** である必要があります。

1.3. AUTOTLS を無効にして REDIS の TLS を設定する

キーと値のペアを使用して **argocd-operator-redis-tls** シークレットを作成して、Redis の TLS 暗号化を手動で設定できます。さらに、シークレットにアノテーションを付けて、それが適切な Argo CD インスタンスに属していることを示す必要があります。証明書とシークレットを作成する手順は、高可用性 (HA) が有効になっているインスタンスによって異なります。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. Argo CD インスタンスを作成します。
   a. Web コンソールの **Administrator** パースペクティブで、左側のナビゲーションパネルを使用して、**Administration → CustomResourceDefinitions** に移動します。
   b. **argocds.argoproj.io** を検索し、**ArgoCD** カスタムリソース定義 (CRD) をクリックします。
   c. **CustomResourceDefinition の詳細** ページで、**Instances** タブをクリックし、**Create ArgoCD** をクリックします。
   d. 次の例のように YAML を編集または置換します。

autotls を無効にした ArgoCD CR の例

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
```
(1)

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

(1) Argo CD インスタンスの名前。

(2) Argo CD インスタンスを実行する namespace。

(3) HA 機能を有効にするフラグの値。HA を有効にしたくない場合は、この行を含めないか、フラグ値を **false** に設定します。

e. **Create** をクリックします。

f. Argo CD Pod が準備ができており、実行中であることを確認します。

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

(1) Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

HA を無効にした場合の出力例

```
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
```

注記

HA 対応の TLS 設定には、少なくとも 3 つのワーカーノードを備えたクラスターが必要です。HA 設定で Argo CD インスタンスを有効にしている場合は、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

```
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. HA 設定に応じて、次のいずれかのオプションを使用して、Redis サーバーの自己署名証明書を作成します。

- HA が無効になっている Argo CD インスタンスの場合は、次のコマンドを実行します。

```
$ 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 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

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

- HA が有効になっている Argo CD インスタンスの場合は、以下のコマンドを実行します。

```
$ 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 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

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

4. 次のコマンドを実行して、生成された証明書とキーが **/tmp** ディレクトリーで利用できることを確認します。

```
$ cd /tmp
```

```
$ ls
```

HA を無効にした場合の出力例

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

HA を有効にした場合の出力例

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

5. HA 設定に応じて、次のいずれかのオプションを使用して、**argocd-operator-redis-tls** シークレットを作成します。
   - HA が無効になっている Argo CD インスタンスの場合は、次のコマンドを実行します。

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

   - HA が有効になっている Argo CD インスタンスの場合は、以下のコマンドを実行します。

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

     出力例

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

6. シークレットにアノテーションを付けて、それが Argo CD CR に属していることを示します。

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

   1 Argo CD インスタンスの名前を指定します (例: **argocd**)。

   出力例

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

7. Argo CD Pod が準備ができており、実行中であることを確認します。

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

   1 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

   HA を無効にした場合の出力例

```
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
```

注記

HA 設定で Argo CD インスタンスを有効にしている場合は、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

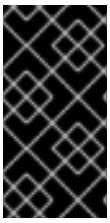
```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h   1/1     Running   0          10m
argocd-redis-ha-server-0            2/2     Running   0          9m9s
argocd-redis-ha-server-1            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
```

第2章 GITOPS で SECRETS STORE CSI ドライバーを使用したシークレットのセキュアな管理

このガイドでは、OpenShift Container Platform 4.14 以降で Secrets Store Container Storage Interface (SSCSI) ドライバーを GitOps Operator と統合するプロセスを説明します。

2.1. GITOPS で SECRETS STORE CSI ドライバーを使用したシークレット管理の概要

アプリケーションには、パスワードやユーザー名などの機密情報が必要なものがありますが、適切なセキュリティー対策として、これらを非表示にする必要があります。ロールベースのアクセス制御 (RBAC) がクラスターで適切に設定されていないために機密情報が公開された場合、API または etcd アクセス権を持つユーザーは誰でもシークレットを取得または変更できます。



重要

namespace で Pod を作成する権限を持つユーザーであれば、その RBAC を使用して対象の namespace 内のシークレットを読み取ることができます。SSCSI Driver Operator を使用すると、外部シークレットストアを使用して機密情報を保存し、Pod に安全に提供できます。

OpenShift Container Platform SSCSI ドライバーを GitOps Operator と統合するプロセスは、以下の手順で構成されます。

1. GitOps リポジトリーへの AWS Secrets Manager リソースの保存
2. AWS Secrets Manager からシークレットをマウントする SSCSI ドライバーの設定
3. マウントされたシークレットを使用するための GitOps 管理リソースの設定

2.1.1. 利点

SSCSI ドライバーを GitOps Operator と統合すると、次の利点があります。

- GitOps ワークフローのセキュリティーと効率を向上する
- ボリュームとしてのデプロイメント Pod へのシークレットのセキュアな接続を容易にする
- 機密情報に安全かつ効率的にアクセスできるようにする

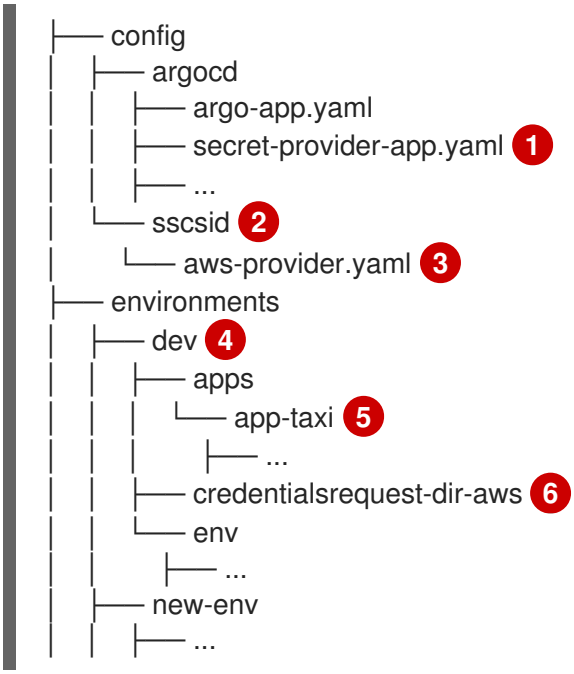
2.1.2. シークレットストアプロバイダー

次のシークレットストアプロバイダーは、Secrets Store CSI Driver Operator で使用できます。

- AWS Secrets Manager
- AWS Systems Manager パラメーターストア
- Microsoft Azure Key Vault

たとえば、SSCSI Driver Operator でシークレットストアプロバイダーとして AWS Secrets Manager を使用していることを考えてみましょう。次の例は、AWS Secrets Manager からのシークレットを使用する準備ができている GitOps リポジトリーのディレクトリー構造を示しています。

GitOps リポジトリーのディレクトリー構造の例



2 **aws-provider.yaml** ファイルを保存するディレクトリー。

3 AWS Secrets Manager プロバイダーをインストールし、そのリソースをデプロイする設定ファイル。

1 アプリケーションを作成し、AWS Secrets Manager のリソースをデプロイする設定ファイル。

4 デプロイメント Pod と認証情報リクエストを保存するディレクトリー。

5 **SecretProviderClass** リソースを格納するディレクトリー。シークレットストアプロバイダーを定義します。

6 **credentialsrequest.yaml** ファイルを保存するフォルダー。このファイルには、シークレットをデプロイメント Pod にマウントするための認証情報要求の設定が含まれます。

2.2. 前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- **ccoctl** バイナリーを抽出して準備している。
- **jq** CLI ツールがインストールされている。
- クラスターが AWS にインストールされ、AWS Security Token Service (STS) を使用している。
- 必要なシークレットを保存するように AWS Secrets Manager を設定している。
- SSCSI Driver Operator がクラスターにインストールされている。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。
- GitOps リポジトリーでシークレットを使用する準備ができている。

- Argo CD 管理アカウントを使用して Argo CD インスタンスにログインしている。

2.3. GITOPS リポジトリーへの AWS SECRETS MANAGER リソースの保存

このガイドでは、Secrets Store Container Storage Interface (SSCSI) Driver Operator で GitOps ワークフローを使用して、AWS Secrets Manager から OpenShift Container Platform の CSI ボリュームにシークレットをマウントする方法を例とともに説明します。

重要

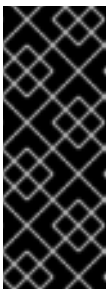
AWS Secrets Manager での SSCSI Driver Operator の使用は、ホストされたコントロールプレーンクラスターではサポートされていません。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- **ccoctl** バイナリーを抽出して準備している。
- **jq** CLI ツールがインストールされている。
- クラスターが AWS にインストールされ、AWS Security Token Service (STS) を使用している。
- 必要なシークレットを保存するように AWS Secrets Manager を設定している。
- SSCSI Driver Operator がクラスターにインストールされている。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。
- GitOps リポジトリーでシークレットを使用する準備ができている。
- Argo CD 管理アカウントを使用して Argo CD インスタンスにログインしている。

手順

1. AWS Secrets Manager プロバイダーをインストールし、リソースを追加します。
   a. GitOps リポジトリーでディレクトリーを作成し、**aws-provider.yaml** ファイルを次の設定に追加して、AWS Secrets Manager プロバイダーのリソースをデプロイします。



重要

SSCSI ドライバー用の AWS Secrets Manager プロバイダーは、アップストリームのプロバイダーです。

この設定は、OpenShift Container Platform で適切に動作するように、アップストリームの AWS ドキュメント で提供されている設定から変更されています。この設定を変更すると、機能に影響が出る場合があります。

aws-provider.yaml ファイルの例

```
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. GitOps リポジトリーに **secret-provider-app.yaml** ファイルを追加して、アプリケーションを作成し、AWS Secrets Manager のリソースをデプロイします。

secret-provider-app.yaml ファイルの例

```
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. **repoURL** フィールドの値を更新して、GitOps リポジトリーを指すようにします。

2. リソースをデフォルトの Argo CD インスタンスと同期して、それらをクラスターにデプロイします。

   a. アプリケーションがデプロイされている **openshift-cluster-csi-drivers** namespace にラベルを追加し、**openshift-gitops** namespace の Argo CD インスタンスが管理できるようにします。

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

   b. プッシュした **aws-provider.yaml** ファイルを含む、GitOps リポジトリー内のリソースをクラスターに適用します。

   出力例

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

Argo CD UI では、**csi-secrets-store-provider-aws** daemonset がリソースの同期を継続していることがわかります。この問題を解決するには、AWS Secrets Manager からシークレットをマウントするように SSCSI ドライバーを設定する必要があります。

2.4. AWS SECRETS MANAGER からシークレットをマウントする SSCSI ドライバーの設定

シークレットを安全に保存および管理するには、GitOps ワークフローを使用し、シークレットを AWS Secrets Manager から OpenShift Container Platform の CSI ボリュームにマウントするように Secrets Store Container Storage Interface (SSCSI) Driver Operator を設定します。たとえば、**/environments/dev/** ディレクトリーにある **dev** namespace のデプロイメント Pod にシークレットをマウントする場合などです。

前提条件

- GitOps リポジトリーに AWS Secrets Manager リソースが保存されている。

手順

1. 次のコマンドを実行して、**csi-secrets-store-provider-aws** サービスアカウントへの特権アクセスを付与します。

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

   出力例

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

2. サービスアカウントに AWS シークレットオブジェクトの読み取りを許可するアクセス許可を付与します。

   a. 認証情報要求は namespace スコープであるため、GitOps リポジトリーの namespace スコープディレクトリーに **credentialsrequest-dir-aws** フォルダーを作成します。たとえば、次のコマンドを実行して、**/environments/dev/** ディレクトリーにある **dev** namespace に **credentialsrequest-dir-aws** フォルダーを作成します。

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

   b. **/environments/dev/credentialsrequest-dir-aws/** パスに認証情報リクエスト用の次の設定を含む YAML ファイルを作成し、**dev** namespace のデプロイメント Pod にシークレットをマウントします。

   credentialsrequest.yaml ファイルの例

   ```
   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 シークレット参照の namespace。プロジェクトのデプロイメント設定に従って、この **namespace** フィールドの値を更新します。

   1 クラスターが置かれているリージョンのシークレットの ARN。**<aws_secret_arn>** の **<aws_region>** は、クラスターのリージョンと一致する必要があります。一致しない場合は、クラスターが置かれているリージョンにシークレットのレプリケーションを作成します。

ヒント

クラスターリージョンを見つけるには、次のコマンドを実行します。

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

出力例

```
us-west-2
```

c. 次のコマンドを実行して、OIDC プロバイダーを取得します。

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

出力例

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

次のステップで使用するために、出力から OIDC プロバイダー名 **<oidc_provider_name>** をコピーします。

d. **ccoctl** ツールを使用して、次のコマンドを実行して認証情報リクエストを処理します。

```
$ 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
```

出力例

```
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
```

次のステップで使用するために、出力から **<aws_role_arn>** をコピーします。たとえば、**arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-aws-creds** です。

e. AWS のロールポリシーをチェックして、ロールポリシー内の **"Resource"** の **<aws_region>** がクラスターリージョンと一致していることを確認します。

ロールポリシーの例

```
{
 "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. 次のコマンドを実行して、ロール ARN を持つサービスアカウントをバインドします。

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

コマンドの例

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

出力例

```
serviceaccount/default annotated
```

3. namespace スコープの **SecretProviderClass** リソースを作成し、シークレットストアプロバイダーを定義します。たとえば、GitOps リポジトリーの **/environments/dev/apps/app-taxi/services/taxi/base/config** ディレクトリーに **SecretProviderClass** リソースを作成します。

   a. ターゲットのデプロイメントが GitOps リポジトリーにあるのと同じディレクトリーに、**secret-provider-class-aws.yaml** ファイルを作成します。

   secret-provider-class-aws.yaml の例

```
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) シークレットプロバイダークラスの名前。

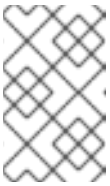
(2) シークレットプロバイダークラスの namespace。namespace は、シークレットを使用するリソースの namespace と一致する必要があります。

(3) シークレットストアプロバイダーの名前。

(4) プロバイダー固有の設定パラメーターを指定します。

(5) AWS で作成したシークレット名。

b. この YAML ファイルを GitOps リポジトリーにプッシュした後、namespace スコープの **SecretProviderClass** リソースが Argo CD UI のターゲットアプリケーションページに設定されていることを確認します。



注記

アプリケーションの Sync Policy が **Auto** に設定されていない場合は、Argo CD UI で **Sync** をクリックして、**SecretProviderClass** リソースを手動で同期できます。

2.5. マウントされたシークレットを使用するための GITOPS 管理リソースの設定

ボリュームマウント設定をデプロイメントに追加し、マウントされたシークレットを使用するようにコンテナー Pod を設定して、GitOps 管理リソースを設定する必要があります。

前提条件

- GitOps リポジトリーに AWS Secrets Manager リソースが保存されている。
- AWS Secrets Manager からシークレットをマウントするように Secrets Store Container Storage Interface (SSCSI) ドライバーが設定されている。

手順

1. GitOps 管理リソースを設定します。たとえば、**app-taxi** アプリケーションのデプロイメントにボリュームのマウント設定を追加し、**100-deployment.yaml** ファイルが **/environments/dev/apps/app-taxi/services/taxi/base/config/** ディレクトリーにあるとします。

   a. デプロイメント YAML ファイルにボリュームマウントを追加し、シークレットプロバイダークラスリソースおよびマウントされたシークレットを使用するようにコンテナー Pod を設定します。

   サンプル YAML ファイル

```
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. デプロイメントの namespace。これは、シークレットプロバイダークラスと同じ namespace である必要があります。
2. ボリュームマウントにシークレットをマウントするパス。
3. シークレットプロバイダークラスの名前。

b. 更新されたリソース YAML ファイルを GitOps リポジトリーにプッシュします。

2. Argo CD UI で、ターゲットアプリケーションページで **REFRESH** をクリックし、更新されたデプロイメントマニフェストを適用します。
3. ターゲットアプリケーションページですべてのリソースが正常に同期されていることを確認します。
4. Pod ボリュームマウントの AWS Secrets Manager からシークレットにアクセスできることを確認します。

a. Pod マウント内のシークレットをリスト表示します。

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

コマンドの例

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

出力例

```
<secret_name>
```

b. Pod マウントのシークレットを表示します。

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

コマンドの例

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

出力例

```
<secret_value>
```

2.6. 関連情報

- **ccoctl** ツールの取得
- Cloud Credential Operator について
- Cloud Credential Operator モードの決定
- AWS STS を使用する AWS クラスターの設定
- 必要なシークレットを保存するための AWS Secrets Manager の設定
- Secrets Store CSI Driver Operator について
- 外部シークレットストアから CSI ボリュームへのシークレットのマウント