Red Hat OpenShift and KeyControl Vault Secrets Integration Guide

2024-11-21

Table of Contents

Chapter 1. Introduction

This guide describes how to integrate a Red Hat OpenShift cluster with Keycontrol Vault Secrets vault.

OpenShift is an application hosting platform by Red Hat. It makes it easy for developers to quickly build, launch, and scale container-based web applications in a public cloud environment.

This integration allows pulling secrets from a secrets vault in Keycontrol Vault and mount them as either environment variables or as volume mounts in containers. It focuses on the way one can pull secrets within OpenShift pods/containers using Entrust's Secrets vault. For other details on the vault, please refer to the Entrust Keycontrol (KCV) documentation.

1.1. Integration architecture

1.1.1. OpenShift cluster

In this integration, a Red Hat OpenShift cluster is deployed on a VMware vSphere instance. Container images are used from a third-party cloud registry.

1.1.2. Container images

Two container images were created for the purpose of this integration to demonstrate how secrets can be pulled into a container from Keycontrol Vault.

Another two images are deployed to support the integration. These images comes from the PASM Vault Kubernetes Agent v1.0. The images are available at: https://github.com/EntrustCorporation/PASM-Vault-Kubernetes-Agent/releases

1.2. Docker Registry

An external docker registry is required. This is where the container images from the PASM Kubernet agents will be stored and reference by the OpenShift containers when created.

1.3. Product configurations

Entrust has successfully tested the integration of KeyControl Secrets Vault with Red Hat OpenShift in the following configurations:

Product	Version
Base OS	Red Hat Enterprise Linux release 9.4 (Plow)
OpenShift	4.14.7
VMware	vSphere 8.0.0.10200
Keycontrol Vault	10.3.1
PASM Vault Kubernetes Agent	1.0

1.4. Requirements

1.4.1. Before starting the integration process

Familiarize yourself with:

- The documentation for the Entrust Keycontrol Vault.
- The documentation and setup process for Red Hat OpenShift.

Chapter 2. Procedures

2.1. RedHat Linux Server

On a RedHat Linux server do the following:

1. Install the OpenShift Client tools.
2. Install docker:

```
% sudo dnf install docker
```

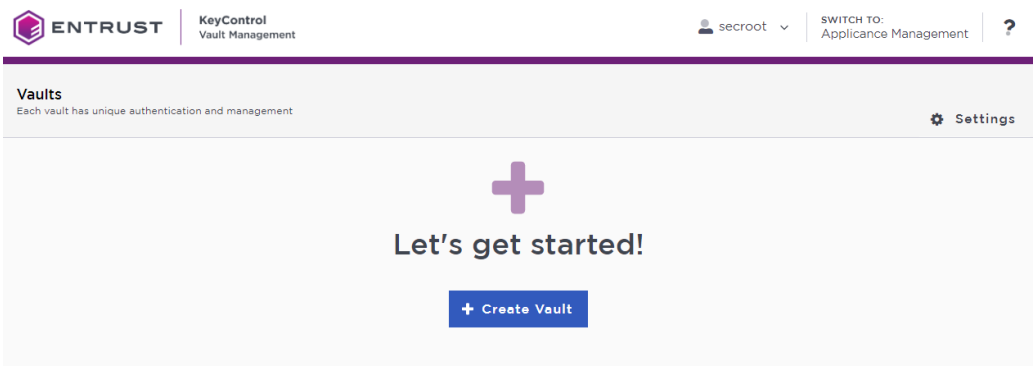
2.2. Deploy a KeyControl Vault cluster

For this integration, KeyControl Vault was deployed as a two-node cluster.

Follow the installation and setup instructions in KeyControl Vault Installation and Upgrade Guide.

2.3. Create a Secrets Vault to be used with Openshift

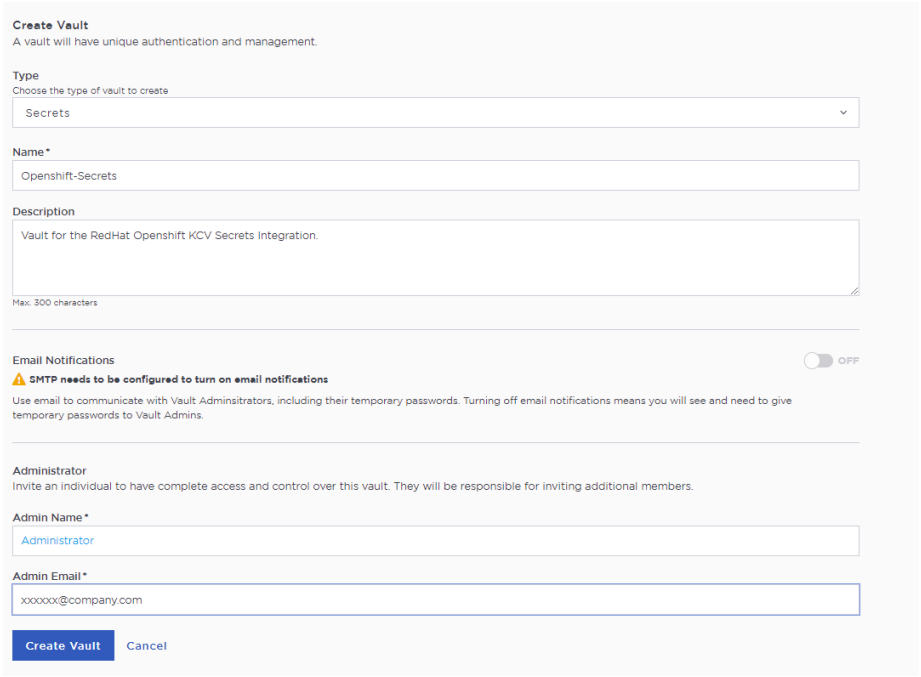
1. Sign in to the KeyControl Vault Manager.
2. In the home page, select **Create Vault**.



   The **Create Vault** dialog appears.

3. In the **Type** drop-down box, select **Secrets**. Enter the required information.
4. Select **Create Vault**.

   For example:



5. When you receive an email with a URL and sign-in credentials to the KeyControl vault, bookmark the URL and save the credentials.

   You can also copy the sign-in credentials when the vault details gets displayed.

6. Sign in to the URL provided.
7. Change the initial password when prompted.

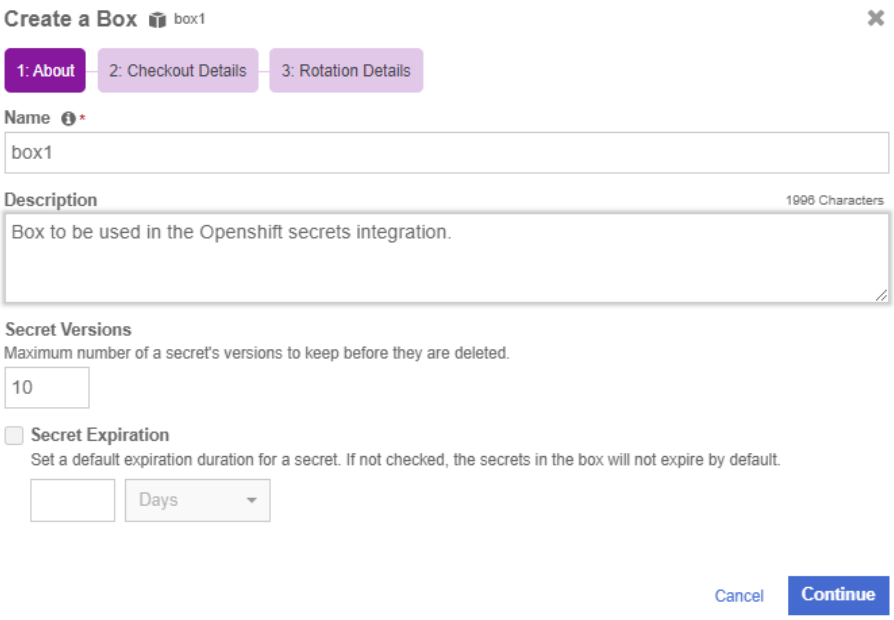
2.4. Create a secret in the Secrets Vault.

After you sign in to the secrets vault, create a box that will contain the secret.

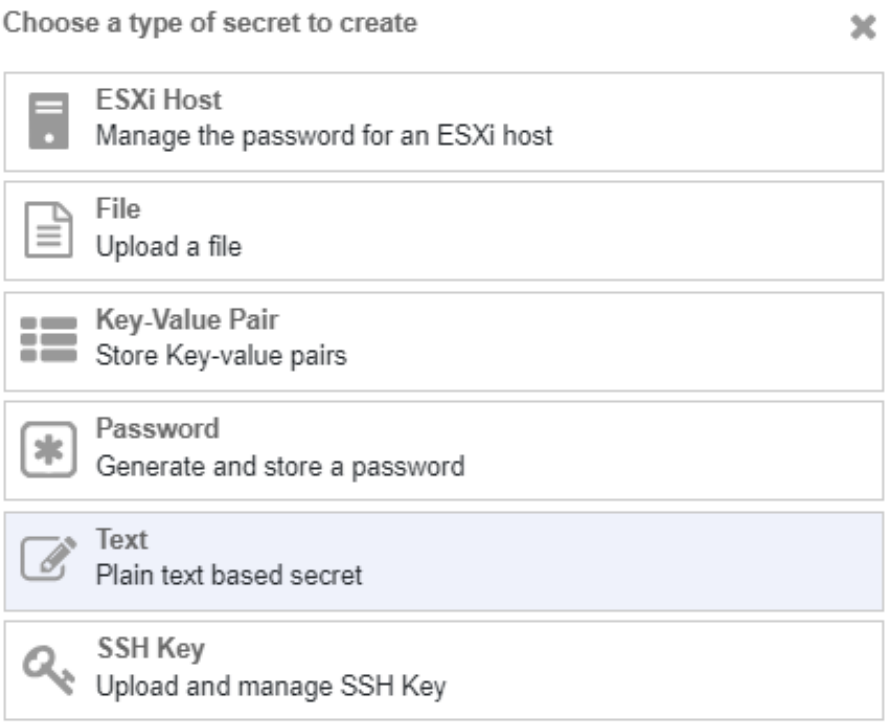
1. Select **Manage** in the Secrets Vault **Home** tab, then select **Manage Boxes**.
2. In the **Manage Boxes** Tab, select **Add a Box Now**.
3. In the **Create a Box** Window, enter the **Name** and **Description**.

   In this integration guide and the configuration file examples it contains, the box will be named **box1**.

4. Select **Continue**.

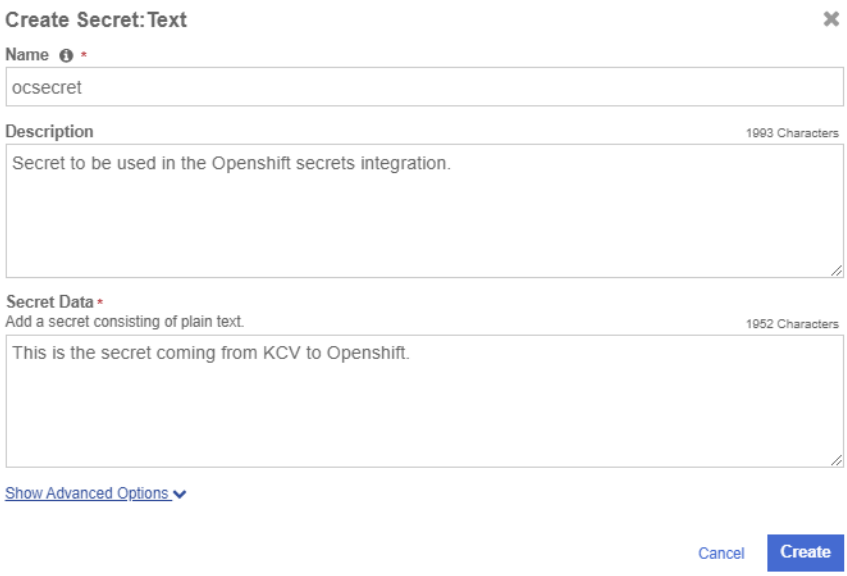


5. In **Checkout Details**, select **Continue**.
6. In **Rotation Details**, select **Create**. The box gets created.
7. Select the new box.
8. In the **Secrets** Pane, select **Add a Secret Now**.
9. From the **Choose a type of secret to create** list, select **Text**.



10. In the **Create Secret: Text** window, enter the following:

a. **Name**: Enter the name of the secret. This will be used later in configuration files in this integration.
b. **Description**: Enter a brief description.
c. **Secret Data**: Enter the value of the secret. This is the value the Openshift containers will try to retrieve during the integration.



11. Select **Create**.

2.5. Setup and Configuration

Configure the environment for the integration. From this point on, all setup and configuration will be done from the linux server that was set up earlier.

2.5.1. Setup KUBECONFIG

This should give you access to the Openshift cluster.

```
% export KUBECONFIG=<path-to-your-kubeconfig-file>
```

Check that you can see the Openshift cluster nodes:

```
% oc get nodes

NAME                           STATUS   ROLES                  AGE    VERSION
ocp4147-r758r-master-0         Ready    control-plane,master   3d1h   v1.27.8+4fab27b
ocp4147-r758r-master-1         Ready    control-plane,master   3d1h   v1.27.8+4fab27b
ocp4147-r758r-master-2         Ready    control-plane,master   3d1h   v1.27.8+4fab27b
ocp4147-r758r-worker-0-ctl28   Ready    worker                 3d     v1.27.8+4fab27b
ocp4147-r758r-worker-0-t7lfl   Ready    worker                 3d     v1.27.8+4fab27b
```

```
ocp4147-r758r-worker-0-vbwrb   Ready    worker                 3d     v1.27.8+4fab27b
```

2.5.2. Set up namespaces

The integration will use two namespaces in Openshift.

1. Set up the mutating webhook namespace.
    a. Create a yaml file containing the following code:

    ```
    apiVersion: v1
    kind: Namespace
    metadata:
      name: mutatingwebhook
    ```

    b. Name the file `mutatingwebhooknamespace.yaml`
    c. Create the namespace:

    ```
    % oc create -f mutatingwebhooknamespace.yaml
    ```

2. Set up the test namespace.
    a. Create a yaml file containing the following code:

    ```
    apiVersion: v1
    kind: Namespace
    metadata:
      name: testnamespace
    ```

    b. Name the file `testnamespace.yaml`.
    c. Create the namespace:

    ```
    % oc create -f testnamespace.yaml
    ```

2.5.3. Docker registry

Register some of the docker container images to a docker registry so they can be used inside the Openshift Cluster.

1. Set up `DOCKER_CONFIG`

    Export `DOCKER_CONFIG` to the directory where your docker configuration will be stored:

```
% export DOCKER_CONFIG=~/.docker
```

2. Log in to the registry:

```
% docker login -u YOURUSERID <registry-url>
```

3. Deploy the init container image.

   The init container image needs to be deployed to the docker registry.

   a. Download the init container image:

```
% wget https://github.com/EntrustCorporation/PASM-Vault-Kubernetes-Agent/releases/download/v1.0/init-container.tar
```

   b. Load the provided image `init-container.tar` into the docker registry:

```
% docker load --input init-container.tar
```

   c. Check the image:

```
% docker images

REPOSITORY                          TAG         IMAGE ID      CREATED        SIZE
localhost/init-container            latest      1c476f57b72c  10 months ago  186 MB
```

   d. Tag the Image:

```
% docker tag localhost/init-container:latest <registry-url>/init-container
```

   e. Push the image into the docker registry that is used within Openshift:

```
% docker push <registry-url>/init-container:latest
```

4. Deploy the mutating webhook image.

   The webhook code is in the form of image and needs to be deployed as container. It needs to be deployed to the docker registry.

   a. Download the webhook container image:

```
% wget https://github.com/EntrustCorporation/PASM-Vault-Kubernetes-
Agent/releases/download/v1.0/mutating-webhook.tar
```

b. Load the provided image `mutating-webhook.tar` into the docker registry:

```
% docker load --input mutating-webhook.tar
```

c. Check the image:

```
% docker images

REPOSITORY                  TAG         IMAGE ID      CREATED        SIZE
localhost/mutating-webhook  latest      805efa734095  10 months ago  222 MB
```

d. Tag the Image:

```
% docker tag localhost/mutating-webhook:latest <registry-url>/mutating-webhook
```

e. Push the image into the docker registry that is used within Openshift:

```
% docker push <registry-url>/mutating-webhook:latest
```

2.5.4. Get the Secrets Vault Authentication token for the vault Admin User.

1. Log in to the Secrets vault as admin using the KeyControl Vault Rest API and get the vault authentication token.

```
https://<VAULT-IP>/vault/1.0/Login/<VAULT-UUID>
```

This URL is visible on vault management UI of the KeyControl. Also note the `VAULT-UUID` from the URL. This will be required in further steps.

Here is an example:

```
https://xx.xxx.xxx.xxx/vault/1.0/Login/7e6e8646-20f0-41a5-8bb6-a17dff64aff2
```

The request body should be in the form:

```
{
 "username":"admin",
 "password":"password"
}
```

The `username` should be one of the administrators of the vault. The examples uses `admin`, but this should be the actual

> admin user name.

2. Get the token using the following curl command:

```
% curl -k -X POST https://xx.xxx.xxx.xxx/vault/1.0/Login/7e6e8646-20f0-41a5-8bb6-a17dff64aff2/ -d '{
"username": "admin", "password":"xxxxxxx" }'
```

The response body for the command above should be similar to this:

```
{
  "is_user": false,
  "box_admin": false,
  "appliance_id": "bc1a085c-70c9-476c-b4f6-abff0c73cce4",
  "is_secondary_approver": false,
  "access_token":
"N2U2ZTg2NDYtMjBmMC00MWE1LThiYjYtYTE3ZGZmNjRhZmYy.eyJkYXRhIjoiU0ZSWFVBRUFldWJCSDNXamtrazRFWkI4ZFA4dzlRMFNtW
npuaEVFNGtnaFRLU1FOVWRMRWlxbVlVUy9aWHR0WUFYTUFwWU9SejZsS1Vhb3NnYitkcGh1N3lRQUFBQ1FBQUFCbWFycFdJbmpGU1NlQk1Z
TjRVZTZHT2llcWVFVVNaRjhhai95bGQ5ZFlxY21xN2FmQ1JSRGgxYTlCSVNBYlREcTJ1dFdUWGI3QnZNODRvakYzZFdPZG54Ym1pUGpQRUJ
xblB5cTFuL2ZpYnpUdmx5SHVWb05aSnh6WXM0cjJXQTNvU1AzeDgwK0ZmMHR5cm4xZHBJSFRUNG4zWHhrL1JRSEdFZ3pSR2pkOUI0WVN0ZG
RwV3hMV1gvSGIyMzBIUUhPYVNvWURzdnJQU3BSb1lyQVZWK2JkMDVkYXV0RzN4OERYRTlJS2VFd0VBbjZwcG9xN3hJSHZTdTJrdDdLaWpRb
1p6Y3pORmNLTzJlSTFaVGszTVRaall5MHpaRGxpTFRRME9UQXRPV1ppWkMxa01UVXpOamd6WkdGaVltTT0iLCJzcGVjIjoxLCJpZCI6IjVl
OTcxNmNjLTNkOWItNDQ5MC05ZmJkLWQxNTM2ODNkYWJiYyJ9",
  "expires_at": "2024-08-27T13:22:02.216046Z",
  "admin": true,
  "user": "admin"
}
```

3. Copy the token received in the response body in the field `access_token`.

This is required when executing APIs described in further steps and is referred to as **Vault Authentication Token**. The `access_token` field will be used in the `curl` headers from now on for the rest of the configuration calls. For example:

```
curl -H "X-Vault-Auth:
N2U2ZTg2NDYtMjBmMC00MWE1LThiYjYtYTE3ZGZmNjRhZmYy.eyJkYXRhIjoiU0ZSWFVBRUFldWJCSDNXamtrazRFWkI4ZFA4dzlRMFNtWn
puaEVFNGtnaFRLU1FOVWRMRWlxbVlVUy9aWHR0WUFYTUFwWU9SejZsS1Vhb3NnYitkcGh1N3lRQUFBQ1FBQUFCbWFycFdJbmpGU1NlQk1ZT
jRVZTZHT2llcWVFVVNaRjhhai95bGQ5ZFlxY21xN2FmQ1JSRGgxYTlCSVNBYlREcTJ1dFdUWGI3QnZNODRvakYzZFdPZG54Ym1pUGpQRUJx
blB5cTFuL2ZpYnpUdmx5SHVWb05aSnh6WXM0cjJXQTNvU1AzeDgwK0ZmMHR5cm4xZHBJSFRUNG4zWHhrL1JRSEdFZ3pSR2pkOUI0WVN0ZGR
wV3hMV1gvSGIyMzBIUUhPYVNvWURzdnJQU3BSb1lyQVZWK2JkMDVkYXV0RzN4OERYRTlJS2VFd0VBbjZwcG9xN3hJSHZTdTJrdDdLaWpRb1
p6Y3pORmNLTzJlSTFaVGszTVRaall5MHpaRGxpTFRRME9UQXRPV1ppWkMxa01UVXpOamd6WkdGaVltTT0iLCJzcGVjIjoxLCJpZCI6IjVlO
TcxNmNjLTNkOWItNDQ5MC05ZmJkLWQxNTM2ODNkYWJiYyJ9" ...
```

2.5.5. Configure Openshift cluster with KeyControl Secrets vault.

To configure an Openshift cluster in the Keycontrol vault, the following API needs to be executed.

```
https://<VAULT-IP>/vault/1.0/SetK8sConfiguration
```

The headers must have the vault authentication token, `X-Vault-Auth*: <VAULT-`

AUTHENTICATION-TOKEN>.

The request body should be in the form:

```
{
 // URL of Openshift API server (or Loadbalancer if one is setup in front of
 // control plane) along with port. This URL must be accessible by PASM vault.
 // If Cluster is behind a service mesh, a proper ingress should be configured
 // so that API server is accessible.
 "k8s_url": "https://<KC-ACCESSIBLE-IP>/<FQDN of K8s API server>:6443",

 // Base64 encoded string of the certificate presented by API server during SSL handshake
 "k8s_ca_string": "<BASE64-ENCODED-CA-STRING>",

 // Indicates whether this configuration should be enabled or not. Valid values
 // are 'enabled' or 'disabled'. If the value is 'disabled', then "k8s_url" and
 // "k8s_ca_string" won't be validated and won't be saved in the configuration.
 "k8s_status": "enabled",
}
```

The certificate string can be obtained from the kubeconfig file or by using following command:

```
% oc config view --raw -o jsonpath='{.clusters[0].cluster.certificate-authority-data}'
```

If Openshift is configured with multiple clusters, replace the number 0 by the appropriate cluster number. The certificate string retrieved is already base64 encoded. You do not need to encode it again.

The final request body:

```
{
 "k8s_url": "https://api.ocp4147.interop.local:6443",
 "k8s_ca_string": "<certificate-string>",
 "k8s_status": "enabled"
}
```

Save the request body in a file named SetK8sConfiguration.body and execute the following command:

```
% curl -H "X-Vault-Auth:
N2U2ZTg2NDYtMjBmMC00MWE1LThiYjYtYTE3ZGZmNjRhZmYy.eyJkYXRhIjoiU0ZSWFVBRUFldWJCSDNXamtrazRFWkI4ZFA4dzlRMFNtWnpuaEVF
NGtnaFRLU1FOVWRMRWlxbVlVUy9aWHR0WUFYTUFwWU9SejZsS1Vhb3NnYitkcGh1N3lRQUFBQ1FBQUFCbWFycFdJbmpGU1NlQk1ZTjRVZTZHT2llc
WVFVVNaRjhhai95bGQ5ZFlxY21xN2FmQ1JSRGgxYTlCSVNBYlREcTJ1dFdUWGI3QnZNODRvakYzZFdPZG54Ym1pUGpQRUJxblB5cTFuL2ZpYnpUdm
x5SHVWb05aSnh6WXM0cjJXQTNvU1AzeDgwK0ZmMHR5cm4xZHBJSFRUNG4zWHhrL1JRSEdFZ3pSR2pkOUI0WVN0ZGRwV3hMV1gvSGIyMzBIUUhPYVN
vWURzdnJQU3BSb1lyQVZWK2JkMDVkYXV0RzN4OERYRTlJS2VFd0VBbjZwcG9xN3hJSHZTdTJrdDdLaWpRb1p6Y3pORmNLTzJlSTFaVGszTVRaall5
MHpaRGxpTFRRME9UQXRPV1ppWkMxa01UVXpOamd6WkdGaVltTT0iLCJzcGVjIjoxLCJpZCI6IjVlOTcxNmNjLTNkOWItNDQ5MC05ZmJkLWQxNTM2O
DNkYWJiYyJ9" -k -X POST https://xx.xxx.xxx.xxx/vault/1.0/SetK8sConfiguration/ --data @./SetK8sConfiguration.body
```

The response should be:

```
{
  "Status": "Success",
  "Message": "Kubernetes configuration updated successfully"
}
```

2.5.6. Set the namespace.

Before proceeding with the rest of the configuration, set the namespace in openshift to the mutatingwebhook namespace created earlier.

```
% oc config set-context --current --namespace=mutatingwebhook
```

2.5.7. Create the registry secrets inside the namespace

The credentials for the external docker registry access need to be created so they can be mentioned in the pod deployment specification.

1. Create the secret in the namespace:

   ```
   % oc create secret generic regcred --from-file=.dockerconfigjson=$HOME/.docker/config.json
   --type=kubernetes.io/dockerconfigjson
   ```

2. Confirm that the secret was created:

   ```
   % oc get secret regcred
   ```

2.5.8. Deploy the webhook container.

Use the following `yaml` specification to deploy the webhook container in Openshift:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mutatingwebhook
  namespace: mutatingwebhook
  labels:
    app: mutatingwebhook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mutatingwebhook
  template:
    metadata:
      labels:
        app: mutatingwebhook
    spec:
      containers:
```

```
      - name: mutatingwebhook
        image: 1.2.3.4:5000/mutating-webhook:latest
        ports:
        - containerPort: 5000
        env:
        # Specify these variables either as a mutatingwebhook container environment variables or
        # add appropriate annotations in the pod specification. Refer documentation further to get
        # the details about annotations. The annotations in pod specifications, if specified, take
        # precedance over these environment variables.
        - name: ENTRUST_VAULT_IPS
          value: <comma-separated-list-of-vault-ips>
        - name: ENTRUST_VAULT_UUID
          value: <uuid-of-vault-from-which-secrets-to-be-fetched>
        - name: ENTRUST_VAULT_INIT_CONTAINER_URL
          value: <complete-url-of-init-container-image-from-image-registry>
        securityContext:                #Required, only if deploying on OpenShift
          allowPrivilegeEscalation: false       #Required, only if deploying on OpenShift
          capabilities:                     #Required, only if deploying on OpenShift
            drop:                   #Required, only if deploying on OpenShift
              - ALL                     #Required, only if deploying on OpenShift
    securityContext:                    #Required, only if deploying on OpenShift
      runAsNonRoot: true                #Required, only if deploying on OpenShift
      runAsUser: 1000700000                 #Required, only if deploying on OpenShift
      seccompProfile:                   #Required, only if deploying on OpenShift
        type: RuntimeDefault                #Required, only if deploying on OpenShift
```

If image registry requires authentication, then make sure to add the credentials as secret in the yaml specification by adding the `imagePullSecrets` tag in the containers section.

Save the yaml in a file called `mutatingwebhook.yaml`.

Example mutatingwebhook file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mutatingwebhook
  namespace: mutatingwebhook
  labels:
    app: mutatingwebhook
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mutatingwebhook
  template:
    metadata:
      labels:
        app: mutatingwebhook
    spec:
      imagePullSecrets:
        - name: regcred                         # external registry secret created earlier
      containers:
      - name: mutatingwebhook
        image: >-
          <registry-url>/mutating-webhook
        ports:
        - containerPort: 5000
        env:
        # Specify these variables either as a mutatingwebhook container environment variables or
        # add appropriate annotations in the pod specification. Refer documentation further to get
```

```
        # the details about annotations. The annotations in pod specifications, if specified, take
        # precedance over these environment variables.
        - name: ENTRUST_VAULT_IPS
          value: 1x.19x.14x.20x,1x.19x.14x.21x
        - name: ENTRUST_VAULT_UUID
          value: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
        - name: ENTRUST_VAULT_INIT_CONTAINER_URL
          value: <registry-url>/init-container
        securityContext:                #Required, only if deploying on OpenShift
          allowPrivilegeEscalation: false       #Required, only if deploying on OpenShift
          capabilities:                     #Required, only if deploying on OpenShift
            drop:                   #Required, only if deploying on OpenShift
              - ALL                     #Required, only if deploying on OpenShift
    securityContext:                    #Required, only if deploying on OpenShift
      runAsNonRoot: true                #Required, only if deploying on OpenShift
      runAsUser: 1000700000                 #Required, only if deploying on OpenShift
      seccompProfile:                   #Required, only if deploying on OpenShift
        type: RuntimeDefault                #Required, only if deploying on OpenShift
```

Pay attention to these sections and adjust according to your environment.

```
      imagePullSecrets:
        - name: regcred                         # external registry secret created earlier

        image: >-
          <registry-url>/mutating-webhook

        - name: ENTRUST_VAULT_IPS
          value: 1x.19x.14x.20x,1x.19x.14x.21x
        - name: ENTRUST_VAULT_UUID
          value: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
        - name: ENTRUST_VAULT_INIT_CONTAINER_URL
          value: <registry-url>/init-container
```

Deploy the file:

```
% oc create -f mutatingwebhook.yaml
```

Check if the multating webhook deployment is running:

```
$ oc get pods
NAME                              READY   STATUS    RESTARTS   AGE
mutatingwebhook-c7958b7f9-dwd9x   1/1     Running   0          5s

$ oc describe pod mutatingwebhook-c7958b7f9-dwd9x
Name:             mutatingwebhook-c7958b7f9-dwd9x
Namespace:        mutatingwebhook
Priority:         0
Service Account:  default
Node:             ocp4147-r758r-worker-0-t7lfl/1x.19x.14x.2xx
Start Time:       Tue, 27 Aug 2024 13:20:12 -0400
Labels:           app=mutatingwebhook
                  pod-template-hash=c7958b7f9
Annotations:      k8s.ovn.org/pod-networks:

{"default":{"ip_addresses":["10.129.2.136/23"],"mac_address":"0a:58:0a:81:02:88","gateway_ips":["10.129.2.1"],"ro
utes":[{"dest":"10.128.0....
                  k8s.v1.cni.cncf.io/network-status:
                    [{
```

```
                        "name": "ovn-kubernetes",
                        "interface": "eth0",
                        "ips": [
                            "10.129.2.136"
                        ],
                        "mac": "0a:58:0a:81:02:88",
                        "default": true,
                        "dns": {}
                    }]
                  openshift.io/scc: restricted-v2
                  seccomp.security.alpha.kubernetes.io/pod: runtime/default
Status:           Running
SeccompProfile:   RuntimeDefault
IP:               10.129.2.136
IPs:
  IP:           10.129.2.136
Controlled By:  ReplicaSet/mutatingwebhook-c7958b7f9
Containers:
  mutatingwebhook:
    Container ID:   cri-o://a05bed83d3c0d5ac39a1d421e9dc992cdbf6cb3550826520dd978eb3da927a02
    Image:          <registry-url>/mutating-webhook
    Image ID:       <registry-url>/mutating-
webhook@sha256:d5379f9b116f725c1e34cda7f10cba2ef7ed369681a768fb985d098cd24f1b1d
    Port:           5000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 27 Aug 2024 13:20:14 -0400
    Ready:          True
    Restart Count:  0
    Environment:
      ENTRUST_VAULT_IPS:                1x.19x.14x.20x,1x.19x.14x.21x
      ENTRUST_VAULT_UUID:               7e6e8646-20f0-41a5-8bb6-a17dff64aff2
      ENTRUST_VAULT_INIT_CONTAINER_URL:  <registry-url>/init-container
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-qpv28 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-qpv28:
    Type:                     Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
    ConfigMapName:           openshift-service-ca.crt
    ConfigMapOptional:       <nil>
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age   From               Message
  ----    ------          ----  ----               -------
  Normal  Scheduled       17s   default-scheduler  Successfully assigned mutatingwebhook/mutatingwebhook-
c7958b7f9-dwd9x to ocp4147-r758r-worker-0-t7lfl
  Normal  AddedInterface  17s   multus             Add eth0 [10.129.2.136/23] from ovn-kubernetes
  Normal  Pulling         17s   kubelet            Pulling image "registry.eselab.net/mutating-webhook"
  Normal  Pulled          16s   kubelet            Successfully pulled image "<registry-url>/mutating-webhook" in
1.14339284s (1.143433867s including waiting)
  Normal  Created         16s   kubelet            Created container mutatingwebhook
  Normal  Started         16s   kubelet            Started container mutatingwebhook
```

2.5.9. Deploy the entrust-pasm service.

1. Create a `yaml` file named `entrustpasmservice.yaml` that contains the following code:

```
apiVersion: v1
kind: Service
metadata:
  name: entrust-pasm        # DO NOT CHANGE THIS
  namespace: mutatingwebhook    # DO NOT CHANGE THIS
spec:
  selector:
    app: mutatingwebhook
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
```

2. Run the following command to create the service:

```
% oc create -f entustpasmservice.yaml
```

3. Check the service:

```
% oc get service
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
entrust-pasm   ClusterIP   172.30.233.96   <none>        5000/TCP   8s
```

2.5.10. Obtain the webhook server certificate.

After the container is deployed, obtain the webhook server certificate. This certificate will be needed to configure the webhook in openshift. Use the following command:

```
% oc exec -it -n mutatingwebhook $(kubectl get pods --no-headers -o custom-columns=":metadata.name" -n
mutatingwebhook) -- wget -q -O- localhost:8080/ca.pem
```

The output should be something like this:

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZTekNDQXpPZ0F3SUJBZ0lVRUFiWFlYazZHVUVJM2lrUGZQbnp4MVRMRjdVd0RRWUpLb1pJa
HZjTkFRRUwKQlFBd1FERXhNQzhHQTFVRUF3d29TSGxVY25WemRDQkxaWGxEYjI1MGNtOXNJRU5sY25ScFptbGpZWFJsSUVGMQpkR2h2Y21sMGVURU
xNQWtHQTFVRUJoTUNWVk13SGhjTk1qUXdPREkzTVRjeU1ERTNXaGNOTXpRd09ESTFNVGN5Ck1ERTNXakJBTVRFd0x3WURWUVFERENoSWVWUnlkWE4
wSUV0bGVVTnZiblJ5YjJ3Z1EyVnlkR2xtYVdOaGRHVWcKUVhWMGFHOXlhWFI1TVFzd0NRWURWUVFHRXdKVlV6Q0NBaUl3RFFZSktvWklodmNOQVFF
QkJRQURnZ0lQQURDQwpBZ29DZ2dJQkFMTlVMQk5KVVVzYU1nSHlrR3hHOWcvdFBtZXdFTi9DODVlSFJjMmREOHZyZndiNkZ5dFg3UG1BCnRhSVliM
mJ4OUdTaldXVFlNNXlPL0g5WkY5L3BjUmJuYkdaNFBMTzBGdG1vM2FicCszRVJjTENWREZCamszQlcKTW9hOWQ1VVBkU2kzYnFBdGFrT2xFZHdOQ2
ZvSWhlMDZNOGF2TDcrVWtZV1FQWGppMzBDODZCUTRRWE01dkFKQwpwaFg1NmhCY3N1RjJDc0w2cGZYUjlJbFVjVTgwZFBKN1BiT3ZubWdNbGpYakp
YUnpHREJOUG9VS25HNmJ5N0tpCjV2TkNBaXYxRmJPQUxXUlZCZWdRUHZEdmpJRmVoMzBZUDRkamlydk5Yc1pNZnRiamFzZFVPWGhIMGFYK1BYN3cK
YTNFRUpZL290TjR4VVlLaTFGRlhteDRRWTZaSTVheWdnTzNHSTZEWDJUcENQUzFqcnpXL2xETzhtd1QzNEhFSApKSno1T05abkRGME40YXJJWnFKa
2dUVTd2Q1RaL3ltaWFsc2lDNkJQM1lUTkl5NmlKY1loczFUYWRQUVpmaElnCmxqeFhPQ2dDTXdLV1poNjdPdEtWTkNmeC83ZzBsbHhyeW1YY0xoY2
5aRy81azRDbmlmZHVTQWl4a1hRRVpXaTcKenRJL0ZIZjJMVGNrVkNnSms5NDlJSlNpSTdVREQwQkFHTkVnTE5hVnFBQ0pRbTNlQmVTS24xZ0c0ODN
HMTRQawp5MkYyMEpEelBreG5ZVlBoVkVpK29scXZ3ZFVPVlFNWFV5UTUyQTZ5RTQwZFZsSjBIZGJoTjdzOUJZQWJSUjB4Ck9xalcvTW0yL2trUE1r
```

```
Z3IyRlNsK2JhcVlzS1hKUktsNFVIemZlQUxNRXNIeCs3L2lMK0hBZ01CQUFHalBUQTcKTUF3R0ExVWRFd0VCL3dRQ01BQXdLd1lEVlIwUkJDUXdJb
0lnWlc1MGNuVnpkQzF3WVhOdExtMTFkR0YwYVc1bgpkMlZpYUc5dmF5NXpkbU13RFFZSktvWklodmNOQVFFTEJRQURnZ0lCQU1aVFpPNmhSS29HVE
s0eHB3b1hqY3p1CnloSzZCbURsejg0akd2ek85Q083dkZJblFRSkVlK1Mrc3lBb1l6N3kwNlpnUDZJYi9lZEx5by9aTFB0UDAralAKSnloSUNoN3l
XVldhNkJyNEE3VExBT1BpSWEvQmtaZHBaR3NMdENrbE5uckF1MFBBbXY0QWpUZUZzZWF2aXREeApBTWlYSUlTYk1TZG55bEhFVTUrMWREdzRMRTBt
N01UR2JlU2E3TnZEelVZdWhCNkxDVlhxTWlUQTQ3MCtpdXF4Cm10dkZ3WVU5RVgvV3ZLdlNibWFZZjRKR24rL3pBMlNpU0d5V0R1NW03RCtjazZoT
XBKenRXWW91YlNROWxQQTUKWmZxcDhBYkdnc1J3a1IyTjF1VzZFK1BJNWEzb1llU2NKMUJJeG5GTTBxNjJuV3ppdHB3SlRQN2VsNTlkSWFEQwpYaE
c3UWVxbDFCSUpWUHR6M2Nra1dCTzhuNlJhOHNtVWxTMVNSNkdUWDZ5RGIrV1YzVzVranhmZ0lrNXJCRnlyClRXeEVPSHlJQkdQc1d2UnAvNFlyTHR
zOW10ekpPUVJqeVpLUVZlaUhmaGI1T1JTSE9MT1Y3MERESHd5K2hIc0wKemtKRkc3Mm9ocDJYeDkweHdoeUVFdlhWYU5mRFpTekdKVXpWSFhTSDRW
dG1WaXFBbkFYQkhLYXdHVlA1OXBRVwphOEtuZWNVZkk3VExlSk9ya09HelZLeU05YUFHaS83MW92WFUwZnZOWXZscTlENU9FajVKME5mQ1hzWXdlc
DE5CnVyYlFpRUJJRFNMbElpekIxK1lUYUsvOTJHQ1IwaU1JeG5ON2NWU3l1UEJ1emRZSkt4QjdNZEhhZXFsM1FERzQKcUpjSm5BNmRhUFZ1dzFqMi
9sZmgKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

This is a base64 encoded certificate. With this value, configure webhook in openshift.

2.5.11. Configure the webhook in Openshift.

In sample YAML spec replace the `caBundle` value with the value obtained in the previous section and configure the webhook with the `oc apply` command.

1. Create a `webhook.yaml` with the following code:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: "entrust-pasm.mutatingwebhook.com"
webhooks:
  - name: "entrust-pasm.mutatingwebhook.com"
    objectSelector:
      matchLabels:
        entrust.vault.inject.secret: enabled
    rules:
    - apiGroups: ["*"]
      apiVersions: ["*"]
      operations: ["CREATE"]
      resources: ["deployments", "jobs", "pods", "statefulsets"]
    clientConfig:
      service:
        namespace: "mutatingwebhook"
        name: "entrust-pasm"
        path: "/mutate"
        port: 5000
        # Replace value below with the value obtained from command
      caBundle:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZTekNDQXpPZ0F3SUJBZ0lVRUFiWFlYazZHVUVJM2lrUGZQbnp4MVRMRjdVd0RRWUp
Lb1pJaHZjTkFRRUwKQlFBd1FERXhNQzhHQTFVRUF3d29TSGxVY25WemRDQkxaWGxEYjI1MGNtOXNJRU5sY25ScFptbGpZWFJsSUVGMQpkR2
h2Y21sMGVURUxNQWtHQTFVRUJoTUNWVk13SGhjTk1qUXdPREkzTVRjeU1ERTNXaGNOTXpRd09ESTFNVGN5Ck1ERTNXakJBTVRFd0x3WURWU
VFERENoSWVWUnlkWE4wSUV0bGVVTnZiblJ5YjJ3Z1EyVnlkR2xtYVdOaGRHVWcKUVhWMGFHOXlhWFI1TVFzd0NRWURWUVFHRXdKVlV6Q0NB
aUl3RFFZSktvWklodmNOQVFFQkJRQURnZ0lQQURDQwpBZ29DZ2dJQkFMTlVMQk5KVVVzYU1nSHlrR3hHOWcvdFBtZXdFTi9DODVlSFJjMmR
EOHZyZndiNkZ5dFg3UG1BCnRhSVliMmJ4OUdTaldXVFlNNXlPL0g5WkY5L3BjUmJuYkdaNFBMTzBGdG1vM2FicCszRVJjTENWREZCamszQl
cKTW9hOWQ1VVBkU2kzYnFBdGFrT2xFZHdOQ2ZvSWhlMDZNOGF2TDcrVWtZV1FQWGppMzBDODZCUTRRWE01dkFKQwpwaFg1NmhCY3N1RjJDc
0w2cGZYUjlJbFVjVTgwZFBKN1BiT3ZubWdNbGpYakpYUnpHREJOUG9VS25HNmJ5N0tpCjV2TkNBaXYxRmJPQUxXUlZCZWdRUHZEdmpJRmVo
MzBZUDRkamlydk5Yc1pNZnRiamFzZFVPWGhIMGFYK1BYN3cKYTNFRUpZL290TjR4VVlLaTFGRlhteDRRWTZaSTVheWdnTzNHSTZEWDJUcEN
QUzFqcnpXL2xETzhtd1QzNEhFSApKSno1T05abkRGME40YXJJWnFKa2dUVTd2Q1RaL3ltaWFsc2lDNkJQM1lUTkl5NmlKY1loczFUYWRQUV
pmaElnCmxqeFhPQ2dDTXdLV1poNjdPdEtWTkNmeC83ZzBsbHhyeW1YY0xoY25aRy81azRDbmlmZHVTQWl4a1hRRVpXaTcKenRJL0ZIZjJMV
GNrVkNnSms5NDlJSlNpSTdVREQwQkFHTkVnTE5hVnFBQ0pRbTNlQmVTS24xZ0c0ODNHMTRQawp5MkYyMEpEelBreG5ZVlBoVkVpK29scXZ3
ZFVPVlFNWFV5UTUyQTZ5RTQwZFZsSjBIZGJoTjdzOUJZQWJSUjB4Ck9xalcvTW0yL2trUE1rZ3IyRlNsK2JhcVlzS1hKUktsNFVIemZlQUx
NRXNIeCs3L2lMK0hBZ01CQUFHalBUQTcKTUF3R0ExVWRFd0VCL3dRQ01BQXdLd1lEVlIwUkJDUXdJb0lnWlc1MGNuVnpkQzF3WVhOdExtMT
FkR0YwYVc1bgpkMlZpYUc5dmF5NXpkbU13RFFZSktvWklodmNOQVFFTEJRQURnZ0lCQU1aVFpPNmhSS29HVEs0eHB3b1hqY3p1CnloSzZCb
...
```

```
URsejg0akd2ek85Q083dkZJblFRSkVlK1Mrc3lBb1l6N3kwNlpnUDZJYi9lZEx5by9aTFB0UDAralAKSnloSUNoN3lXVldhNkJyNEE3VExB
T1BpSWEvQmtaZHBaR3NMdENrbE5uckF1MFBBbXY0QWpUZUZzZWF2aXREeApBTWlYSUlTYk1TZG55bEhFVTUrMWREdzRMRTBtN01UR2JlU2E
3TnZEelVZdWhCNkxDVlhxTWlUQTQ3MCtpdXF4Cm10dkZ3WVU5RVgvV3ZLdlNibWFZZjRKR24rL3pBMlNpU0d5V0R1NW03RCtjazZoTXBKen
RXWW91YlNROWxQQTUKWmZxcDhBYkdnc1J3a1IyTjF1VzZFK1BJNWEzb1llU2NKMUJJeG5GTTBxNjJuV3ppdHB3SlRQN2VsNTlkSWFEQwpYa
Ec3UWVxbDFCSUpWUHR6M2Nra1dCTzhuNlJhOHNtVWxTMVNSNkdUWDZ5RGIrV1YzVzVranhmZ0lrNXJCRnlyClRXeEVPSHlJQkdQc1d2UnAv
NFlyTHRzOW10ekpPUVJqeVpLUVZlaUhmaGI1T1JTSE9MT1Y3MERESHd5K2hIc0wKemtKRkc3Mm9ocDJYeDkweHdoeUVFdlhWYU5mRFpTekd
KVXpWSFhTSDRWdG1WaXFBbkFYQkhLYXdHVlA1OXBRVwphOEtuZWNVZkk3VExlSk9ya09HelZLeU05YUFHaS83MW92WFUwZnZOWXZscTlENU
9FajVKME5mQ1hzWXdlcDE5CnVyYlFpRUJJRFNMbElpekIxK1lUYUsvOTJHQ1IwaU1JeG5ON2NWU3l1UEJ1emRZSkt4QjdNZEhhZXFsM1FER
zQKcUpjSm5BNmRhUFZ1dzFqMi9sZmgKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    admissionReviewVersions: ["v1", "v1beta1"]
    sideEffects: None
    timeoutSeconds: 5
```

2. Run the following command to apply the changes:

```
% oc apply -f webhook.yaml

mutatingwebhookconfiguration.admissionregistration.k8s.io/entrust-pasm.mutatingwebhook.com created
```

2.6. Testing

Now that the openshift cluster and Keycontrol Vault are properly configured and set up let's deploy a couple of pods that will attempt to use the ocsecret created earlier inside the pod containers.

Important points:

- The deployment of pod (in which the secrets need to be fetched) must happen by the service accounts mentioned in the policy created above.
- The service accounts added in policy must have **system:auth-delegator** ClusterRole at the openshift side.

2.6.1. Set the namespace.

We created two namespaces earlier. Currently the integration should be set to the mutatingwebhook namespace. Change the namespace so it points to the test namespace.

```
> oc config set-context --current --namespace=testnamespace
```

2.6.2. Create the registry secrets inside the namespace

The credentials for the external docker registry access need to be created so they can be mentioned in the pod deployment specification.

1. Create the secret in the namespace:

```
% oc create secret generic regcred --from-file=.dockerconfigjson=$HOME/.docker/config.json
--type=kubernetes.io/dockerconfigjson
```

2. Confirm that the secret was created:

```
% oc get secret regcred
```

2.6.3. Create a Policy in Secrets Vault for openshift service accounts

Create a policy in the Secrets Vault that will enable Openshift service accounts to read secrets from the vault. To create such policy, the following API needs to be executed.

URL: https://<VAULT-IP>/vault/1.0/CreatePolicy

The headers must have the vault authentication token.

X-Vault-Auth: <VAULT-AUTHENTICATION-TOKEN>

The request body should be in the form:

```
{
  "name": "vault_user_policy",         // Name of the policy to be created
  "role": "Vault User Role",               // Role of user. DO NOT CHANGE THIS
  "principals": [
     {
       "k8s_user": {
         "k8s_namespace": "default",       // Namespace in which the service account resides
         "k8s_service_account": "default"  // Name of the service account
       }
     }
  ],
  "resources": [
     {
       "box_id": "box1", // Name of the box in which the secret(s) that need access reside. Can be '*' to
indicate all boxes
       "secret_id": [    // List of secrets to which access needs to be granted. Can be '*' to indicate all
secrets.
         "secret1",
         "secret2
       ]
     }
  ]
}
```

Match the namespace `k8s_namespace` to the same namespace the service account resides in and where you will be deploying the pods.

1. Save the request body for the environment to a file named `createpolicy.body`:

```
{
  "name": "openshift_vault_user_policy",
  "role": "Vault User Role",
  "principals": [
     {
       "k8s_user": {
         "k8s_namespace": "testnamespace",
         "k8s_service_account": "default"
       }
     }
  ],
  "resources": [
     {
       "box_id": "box1",
       "secret_id": [
         "*"
       ]
     }
  ]
}
```

In this example, the policy will allow the `default` user in the `testnamespace` to have access to any secret in `box1` in the Keycontrol Secrets vault.

2. Check and adjust the following sections according to your environment:

```
       "k8s_user": {
         "k8s_namespace": "testnamespace",
         "k8s_service_account": "default"

       "box_id": "box1",
       "secret_id": [
         "*"
       ]
```

3. Create the policy:

```
curl -H "X-Vault-Auth: <VAULT-IDENTIFICATION-TOKEN>" -k -X POST
https://10.194.148.206/vault/1.0/CreatePolicy/ --data @./createpolicy.body

{"policy_id": "openshift_vault_user_policy-82756a"}
```

2.6.4. Create the clusterrolebinding.

Create the clusterrolebinding to allow the default user access to the secrets:

```
% oc create clusterrolebinding authdelegator --clusterrole=system:auth-delegator
--serviceaccount=testnamespace:default
```

Test the access by running the following command:

```
% oc auth can-i create tokenreviews --as=system:serviceaccount:testnamespace:default
```

The output should be `yes` if set up correctly.

2.6.5. Deploying Pod with Secrets

Secrets can be added to the pod either as volume mounts or as environment variables.

2.6.5.1. Adding secrets as volume mounts to the pod

The sample pod specification along with labels and annotations required for successfully pulling secrets as volumes mounts:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: testnamespace
  labels:
    app: test
    entrust.vault.inject.secret: enabled        # Must have this label
  annotations:
    entrust.vault.ips: <comma-separated-list-of-vault-ips>
    entrust.vault.uuid: <uuid-of-vault-from-which-secrets-to-be-fetched>
    entrust.vault.init.container.url: <complete-url-of-init-container-image-from-image-registry>
    entrust.vault.secret.file.k8s-box.ca-cert: output/cert.pem  # box and secret name from vault and value
denoting location of secret on the container. The location
spec:
  serviceAccountName: k8suser          # Service account name configured in PASM Vault Policy
  containers:
  - name: ubuntu
    image: 10.254.154.247:5000/ubuntu:latest
    command: ['cat','/output/ans.txt']
    imagePullPolicy: IfNotPresent
    securityContext:               #Required, only if deploying on Openshift
      allowPrivilegeEscalation: false  #Required, only if deploying on Openshift
      capabilities:                #Required, only if deploying on OpenShift
        drop:                          #Required, only if deploying on OpenShift
          - ALL                    #Required, only if deploying on OpenShift
  securityContext:             #Required, only if deploying on OpenShift
    runAsNonRoot: true                 #Required, only if deploying on OpenShift
    runAsUser: 1000700000          #Required, only if deploying on OpenShift
    seccompProfile:                #Required, only if deploying on OpenShift
      type: RuntimeDefault             #Required, only if deploying on OpenShift
```

The pod specification must have the following annotations:

- `entrust.vault.inject.secret: enabled`: This label indicates that it needs secret.
- `entrust.vault.ips`: This value is a comma-separated list of IPs of vaults which

are in the cluster.

This annotation, if specified, will override the `ENTRUST_VAULT_IPS` environment variable from the mutating webhook configuration.

- `entrust.vault.uuid`: This value denotes the UUID of the vault from which we need to fetch the secrets.

  This annotation, if specified, will override the `ENTRUST_VAULT_UUID` environment variable from the mutating webhook configuration.

- `entrust.vault.init.container.url`: This value denotes the complete URL of the init container image pushed into image registry.

  This annotation, if specified, will override the `ENTRUST_VAULT_INIT_CONTAINER_URL` environment variable from the mutating webhook configuration.

- Annotations in the form of `entrust.vault.secret.file.{box-name}.{secret-name}`.

  The `box-name` and `secret-name` placeholders within the annotation should denote the name of the box and the secret name within that box that needs to be pulled.

  For example, if the secret was named `db-secret` and the box was `k8s-box`, then the name of the annotation would be `entrust.vault.secret.file.k8s-box.db-secret`. The value of the annotation should be the path to the file where the secret needs to be stored within the container. The path should be relative to the `/` directory, meaning that if the secret needs to be present in `/output/cert.pem`, then the value of the annotation should be `output/cert.pem`

- In the `spec` section, the `serviceAccountName` value should be the name of the service account that was added in the policy in the Secrets vault.

To add the secrets as volume mounts to the pod:

1. Create a file named `pod1.yaml` with the yaml for the environment. We are using the box and secret we created earlier in the Secrets vault. (`box1` and `ocsecret`). We also had to provide the credentials for the docker registry.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  namespace: testnamespace
  labels:
    app: test
    entrust.vault.inject.secret: enabled               # Must have this label
  annotations:
```

```
    entrust.vault.ips: 1x.19x.14x.20x,1x.19x.14x.21x
    entrust.vault.uuid: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
    entrust.vault.init.container.url: <registry-url>/init-container
    entrust.vault.secret.file.box1.ocsecret: output/ocsecret.txt
spec:
  imagePullSecrets:
    - name: regcred                   # external registry secret created earlier
  serviceAccountName: default         # Service account name configured in PASM Vault Policy
  containers:
  - name: ubuntu
    image: ubuntu
    command: ["sh", "-c"]
    args:
        - echo "Getting secret from KeyControl Secrets Vault";
          cat /output/ocsecret.txt;
          echo;
          echo "DONE" && sleep 3600
    imagePullPolicy: IfNotPresent
    securityContext:                  #Required, only if deploying on OpenShift
      allowPrivilegeEscalation: false  #Required, only if deploying on OpenShift
      capabilities:                   #Required, only if deploying on OpenShift
        drop:                         #Required, only if deploying on OpenShift
          - ALL                       #Required, only if deploying on OpenShift
  securityContext:                    #Required, only if deploying on OpenShift
    runAsNonRoot: true                #Required, only if deploying on OpenShift
    runAsUser: 1000700000             #Required, only if deploying on OpenShift
    seccompProfile:                   #Required, only if deploying on OpenShift
      type: RuntimeDefault            #Required, only if deploying on OpenShift
```

2. Check and adjust the following sections according to your environment:

```
  annotations:
    entrust.vault.ips: 1x.19x.14x.20x,1x.19x.14x.21x
    entrust.vault.uuid: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
    entrust.vault.init.container.url: <registry-url>/init-container
    entrust.vault.secret.file.box1.ocsecret: output/ocsecret.txt

  imagePullSecrets:
    - name: regcred                   # external registry secret created earlier

    command: ["sh", "-c"]
    args:
        - echo "Getting secret from KeyControl Secrets Vault";
          cat /output/ocsecret.txt;
          echo;
          echo "DONE" && sleep 3600
```

3. Deploy the pod:

```
% oc create -f pod1.yaml
```

4. Check the pod output to verify that it was capable of pulling the secret from the KeyControl Secrets vault:

```
% oc logs pod/pod1

Defaulted container "ubuntu" out of: ubuntu, secret-v (init)
Getting secret from KeyControl Secrets Vault
This is the secret coming from KCV to Openshift.
```

```
DONE
```

2.6.5.2. Pulling secret as environment variable within Pod

Openshift supports initiating environment variables for a container either directly or from Openshift secrets. For security purposes, the secrets vault takes the later approach. Also, if secrets are injected using environment variables, a sidecar container will be added which will delete the Openshift secrets after the successful injection of KeyControl Secret Vault secrets as environment variables in the main application container. Since we need to create and delete Openshift secrets for this, the service account must also have the required permissions to create and delete the Openshift secrets.

1. Grant the service account with required permissions: (replace the `namespace` and `serviceaccount` values appropriately)

```
% oc create rolebinding secretrole --namespace testnamespace --clusterrole=edit
--serviceaccount=testnamespace:default
```

2. To check if proper permissions are set up for the service account, use the following commands:

```
% oc auth can-i create secrets -n testnamespace --as=system:serviceaccount:testnamespace:default
% oc auth can-i delete secrets -n testnamespace --as=system:serviceaccount:testnamespace:default
```

   The output of both the above commands should be **yes**.

3. Save the following sample pod specification yaml in a file called `pod2.yaml`. It shows how to deploy a pod with secrets as environment variables.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod2
  namespace: testnamespace
  labels:
    app: test2
    entrust.vault.inject.secret: enabled        # Must have this label
  annotations:
    entrust.vault.ips: 10.19x.14x.20x.,10.19x.14x.21x
    entrust.vault.uuid: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
    entrust.vault.init.container.url: <registry-url>/init-container
    entrust.vault.secret.env.box1.ocsecret: OCSECRET
spec:
  imagePullSecrets:
    - name: regcred                    # external registry secret created earlier
  serviceAccountName: default          # Service account name configured in PASM Vault Policy
  containers:
  - name: ubuntu
    image: ubuntu
```

```
    command: ["sh", "-c"]
    args:
        - echo "Getting secret from KeyControl Secrets Vault";
          printenv OCSECRET;
          echo "DONE" && sleep 3600
    imagePullPolicy: IfNotPresent
    securityContext:              #Required, only if deploying on OpenShift
      allowPrivilegeEscalation: false  #Required, only if deploying on OpenShift
      capabilities:               #Required, only if deploying on OpenShift
        drop:                         #Required, only if deploying on OpenShift
          - ALL                   #Required, only if deploying on OpenShift
  securityContext:             #Required, only if deploying on OpenShift
    runAsNonRoot: true                #Required, only if deploying on OpenShift
    runAsUser: 1000700000         #Required, only if deploying on OpenShift
    seccompProfile:               #Required, only if deploying on OpenShift
      type: RuntimeDefault            #Required, only if deploying on OpenShift
```

4. To pull the secret as an environment variable, add an annotation of the form `entrust.vault.secret.env.{box-name}.{secret-name}`. The value of the annotation should indicate the name of the environment variable in which the secret data is expected to be present.
5. Check and adjust the following sections according to your environment. These annotations are as documented in the previous section.

```
  annotations:
    entrust.vault.ips: 10.19x.14x.20x.,10.19x.14x.21x
    entrust.vault.uuid: 7e6e8646-20f0-41a5-8bb6-a17dff64aff2
    entrust.vault.init.container.url: <registry-url>/init-container
    entrust.vault.secret.env.box1.ocsecret: OCSECRET

  imagePullSecrets:
    - name: regcred                   # external registry secret created earlier

    command: ["sh", "-c"]
    args:
        - echo "Getting secret from KeyControl Secrets Vault";
          printenv OCSECRET;
          echo "DONE" && sleep 3600
```

6. Create and test the pod:

```
% oc create -f pod2.yaml
```

7. Check the pod output to verify that it was capable of pulling the secret from the KeyControl Secrets vault:

```
% oc logs pod/pod2

Defaulted container "ubuntu" out of: ubuntu, pasm-sidecar, secret-v (init)
Getting secret from KeyControl Secrets Vault
This is the secret coming from KCV to Openshift.
DONE
```

Chapter 3. Troubleshooting

The following commands can be used to troubleshoot and look at pods during deployment.

3.1. Look at the logs

You can use `oc logs`:

```
% oc logs pod/podname
```

For example:

```
% oc logs pod/pod1
```

You can also use `oc describe`:

```
% oc describe pod podname
```

For example:

```
% oc describe pod pod1
```

3.2. Looking at the init container

In the guide, the init container is deployed at each pod that gets the secret. To look at the logs for the init container, use the following command:

```
% oc logs pods/pod1 -c secret-v --namespace=testnamespace
```

Chapter 4. Additional resources and related products