

Kubernetes

K8s

v1.33.2

Integration Guide

ESKM

8.54.0

utimaco[®]

Imprint

Copyright 2025	Utimaco IS GmbH Germanusstr. 4 D-52080 Aachen Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226) EMEA +49 800-627-3081 APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	2025-09-23
Status	PUBLISHED
Document No.	IG-2025-0050
All rights reserved	<p>No part of this documentation may be reproduced in any form (printing, photocopy or according to any other process) without the written approval of Utimaco IS GmbH or be processed, reproduced or distributed using electronic systems.</p> <p>Utimaco IS GmbH reserves the right to modify or amend the documentation at any time without prior notice. Utimaco IS GmbH assumes no liability for typographical errors and damages incurred due to them.</p> <p>All trademarks and registered trademarks are the property of their respective owners.</p>

Table of Contents

1	Introduction	5
1.1	About This Guide	5
1.2	Target Audience	5
1.3	Purpose of the Integration	5
1.4	Abbreviations	5
1.5	Document Conventions	6
2	Product Overview.....	8
2.1	Overview of Kubernetes.....	8
2.2	Overview of ESKM	8
2.3	Joint Value Proposition	8
3	Integration Requirements and Prerequisites	9
3.1	Tested Versions.....	9
3.2	Supported Platforms	9
3.3	Software Requirements.....	9
3.4	Prerequisites	10
4	Installation and Configuration.....	11
4.1	Set Up ESKM	11
4.2	Set Up Kubernetes.....	11
5	Integration Steps	12
5.1	Configuration on Kubernetes Control Plane Node.....	12
5.1.1	Control Plane Node Verification	12
5.1.2	Create Directories.....	12
5.1.3	Load Docker Image.....	13
5.1.4	Create a Client Certificate	13
5.1.5	Configure the plugin-config File	14
5.1.6	Create ConfigMap and Secrets.....	15
5.1.7	Deploy the KMS Plugin	16
5.1.8	Configure the Encryption Provider	20
5.1.9	Configure the kube-apiserver	21
5.2	Configure Utimaco ESKM	23
5.2.1	Create a CA.....	23

5.2.2	Create a Server Certificate	24
5.2.3	Configure the KMIP Server	25
5.2.4	Sign the Host Certificate Using ESKM.....	26
5.2.5	Create KMIP User	32
5.2.6	Create KMIP Object	33
6	Verification and Testing	35
6.1	Functional Testing.....	35
6.1.1	Verify KMIP Log.....	35
6.1.2	Create a Secret.....	35
6.1.3	Verify Encryption in etcd	35
6.1.4	Verify Decryption.....	36
6.1.5	Key Rotation.....	37
6.1.5.1	Create a New KMIP Object.....	37
6.1.5.2	Update plugin-config.yaml file	37
6.1.5.3	Re-create ConfigMap.....	38
6.1.5.4	Re-encrypt Existing Secrets	38
6.2	Logs and Validation Steps.....	39
6.2.1	Verify Key Rotation Success	39
7	Troubleshooting	40
7.1	Log Locations and Interpretation	40
8	Contact and Support Information	41
9	Appendices	42
9.1	References	42
9.2	Command Summary.....	42

1 Introduction

1.1 About This Guide

This guide provides step-by-step instructions for integrating the **ESKM** with Kubernetes to enable the **encryption** of sensitive data at rest.

1.2 Target Audience

This guide is intended for Kubernetes and Utimaco ESKM administrators.

1.3 Purpose of the Integration

The purpose of integrating ESKM with Kubernetes is to provide a secured and centralized method for managing encryption keys used by containerized applications and services. Kubernetes supports encryption of sensitive data, such as secrets and configuration information, but relies on an ESKM to securely store and manage encryption keys.

1.4 Abbreviations

Abbreviation	Meaning
HSM	Hardware Security Module
ESKM	Enterprise Security Key Manager
KMIP	Key Management Interoperability Protocol
KMS	Key Management Service
YAML	Yet Another Markup Language
K8s	Kubernetes

Abbreviation	Meaning
API	Application Programming Interface
CSR	Certificate Signing Request

Table 1: Abbreviations

1.5 Document Conventions

The following conventions are used in this guide:

Convention	Use	Example
Bold	Items of the Graphical User Interface (GUI), e.g., menu options	Press OK
Monospaced	Code that is given for explanation or as an example, file paths	<code>chsm-create</code>
<i>Italic</i>	References and important terms	See <i>Sample Chapter</i> in the <i>CryptoServer - Sample Manual</i>

Table 2: Document Conventions

We use special icons to highlight the most important notes and information.



Here you will find important safety information that should be followed.



Here you will find additional notes or supplementary information.



This message marks the result expected after the successful execution of an instruction.

2 Product Overview

2.1 Overview of Kubernetes

Kubernetes is an open-source platform for automating the deployment, scaling, and management of containerized applications. For security, it can encrypt sensitive data, such as Secrets, at rest. This functionality relies on an external **Key Management Service (KMS)** to handle the encryption and decryption. When integrated with the **ESKM**, Kubernetes can leverage a centralized and secure service for these cryptographic operations. This ensures enhanced security and helps meet regulatory compliance requirements for sensitive data within your cluster.

2.2 Overview of ESKM

ESKM is a centralized key management solution that securely stores, distributes, and manages encryption keys throughout their lifecycle. It supports industry standards, including the KMIP, enabling integration with a wide range of enterprise applications and storage systems.

In Kubernetes Integration, ESKM is the KMS provider, allowing the Kubernetes API Server to securely store, retrieve, and rotate encryption keys.

2.3 Joint Value Proposition

The integration of Kubernetes and ESKM provides a complete solution for secure and compliant encryption key management in containerized environments. Kubernetes handles application deployment and management, while ESKM serves as a dedicated, hardened system for securely storing and managing encryption keys.

This joint solution offers several key benefits:

- **Secure Key Management:** Encryption keys are safely stored and managed outside the Kubernetes cluster, protecting them from an etcd database breach.
- **Regulatory Compliance:** It helps meet strict regulatory requirements by providing robust key access control, auditing, and rotation capabilities.
- **Centralized Control:** You gain centralized key management across all your Kubernetes clusters, simplifying security administration.
- **Efficient and Scalable:** The solution is designed to work efficiently for clusters of any size without compromising performance.

3 Integration Requirements and Prerequisites

3.1 Tested Versions

Operating System	Kubernetes Version	Utimaco ESKM Version
Rocky Linux 9.5 (Blue Onyx)	Kubernetes v1.27.16	8.54.0
	Kubernetes v1.33.2	

3.2 Supported Platforms

- Utimaco ESKM hardware appliance.
- Utimaco ESKM virtual/cloud appliance.

3.3 Software Requirements

Software	Software Requirements
Utimaco ESKM	8.54.0.
Kubernetes	v1.33.2 or later.
k8s-kms-plugin	1.0

Kubernetes versions required based on KMS:

Kubernetes Version	KMS v1	KMS v2
v1.10.0 – v1.26	Yes (default)	Not available
v1.27	Yes (default)	Beta

Kubernetes Version	KMS v1	KMS v2
v1.28	Deprecated (default)	In beta
v1.29+	Disabled by default (can enable manually)	Stable (recommended)

3.4 Prerequisites

- Utimaco ESKM version 8.54.0 or later.
- Root access to the Kubernetes cluster.
- Admin access to Utimaco ESKM.
- k8s-kms-plugin Docker image.

4 Installation and Configuration

The following section describes the procedures required to configure ESKM.

4.1 Set Up ESKM

ESKM is configured first before proceeding with the Kubernetes control-plane configuration.

For detailed configuration steps, see the *ESKM_Installation and Replacement_Guide_8.54.0*. After successful installation and configuration, log in to ESKM.

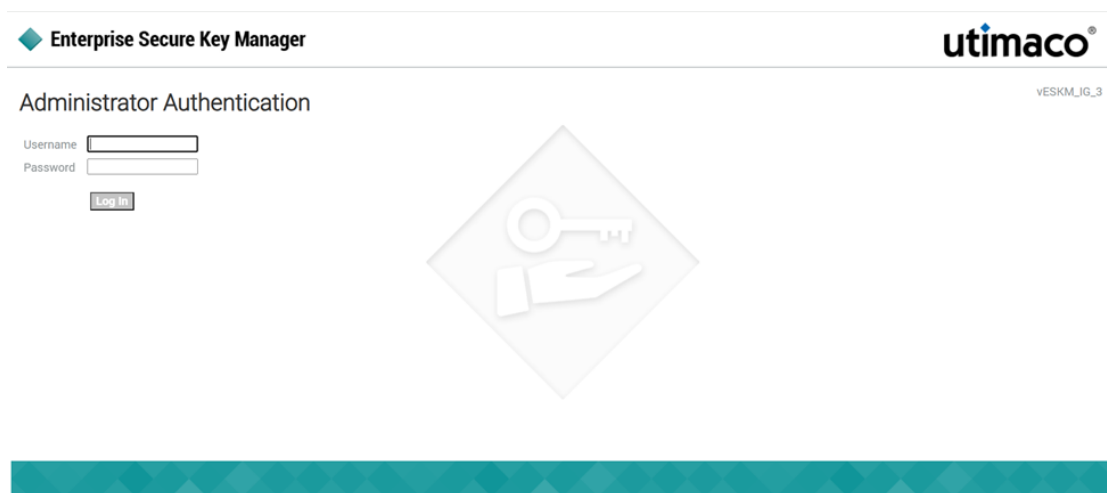


Figure 1 : ESKM Login Page

4.2 Set Up Kubernetes

To set up the Kubernetes cluster, follow the instructions at the following link: [Creating a cluster with kubeadm | Kubernetes](#).

5 Integration Steps

5.1 Configuration on Kubernetes Control Plane Node

5.1.1 Control Plane Node Verification

1. Log in to the control plane node as the root user and verify that all Kubernetes pods are running properly.

```
[root@master-node ~]# kubectl get nodes
NAME          STATUS    ROLES    AGE     VERSION
master-node   Ready    control-plane   6h29m   v1.33.2
[root@master-node ~]# kubectl get pods -A
NAMESPACE     NAME                                                    READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-689744956f-ngd9x             1/1     Running   8 (35m ago) 6h29m
kube-system   calico-node-vtkdp                                     1/1     Running   0           6h29m
kube-system   coredns-674b8bbfcf-d5htv                             1/1     Running   0           6h30m
kube-system   coredns-674b8bbfcf-rjqbz                             1/1     Running   0           6h30m
kube-system   etcd-master-node                                       1/1     Running   0           6h30m
kube-system   kube-apiserver-master-node                             1/1     Running   0           27m
kube-system   kube-controller-manager-master-node                   1/1     Running   7 (27m ago) 6h30m
kube-system   kube-proxy-jg9p6                                       1/1     Running   0           6h30m
kube-system   kube-scheduler-master-node                             1/1     Running   7 (27m ago) 6h30m
```

Figure 2 : Control Plane Node Verification

5.1.2 Create Directories



Create the `kmsplugin` directory under `var/lib`, as it stores the gRPC Unix domain socket files for the KMS plugin implementation. Do not change the directory path. Any modifications may cause the integration to fail or logs to be stored in incorrect locations.

```
[root@master-node ~]# mkdir /var/lib/kmsplugin
```

1. Create the `k8s_plugin` directory under `/home/admin/`.

```
[root@master-node admin]# pwd
/home/admin
[root@master-node admin]#
[root@master-node admin]# mkdir k8s_plugin
[root@master-node admin]#
[root@master-node admin]# cd k8s_plugin/
```

Figure 3 : Create Directory

2. Create the `certs` directory under `/home/admin/k8s_plugin`.

```
[root@master-node k8s_plugin]# mkdir certs
[root@master-node k8s_plugin]# ls
certs  k8s-kms-plugin_1.0.tar
```

Figure 4 : Create Certs Directory

3. Create the `logs` directory under `/home/admin/k8s_plugin`.

```
[root@master-node k8s_plugin]# pwd
/home/admin/k8s_plugin
[root@master-node k8s_plugin]# mkdir logs
```

Figure 5 : Create Logs Directory



If you change the directory path, ensure it is updated in all the places where it is used, including configuration files, deployment specifications, and related settings, to maintain consistency and prevent errors.

5.1.3 Load Docker Image

1. Load the Docker image from `k8s-kms-plugin_1.0.tar` at `/home/admin/k8s_plugin`.

```
[root@master-node k8s_plugin]# docker load -i k8s-kms-plugin_1.0.tar
7cc7fe68eff6: Loading layer [=====>] 77.88MB/77.88MB
d379451f91b6: Loading layer [=====>] 9.561MB/9.561MB
3ba4af0751c6: Loading layer [=====>] 44.84MB/44.84MB
f6dc80d9d167: Loading layer [=====>] 5.12kB/5.12kB
b61d7c291810: Loading layer [=====>] 1.536kB/1.536kB
175f03e00a29: Loading layer [=====>] 68.93MB/68.93MB
e85c52f68660: Loading layer [=====>] 81.53MB/81.53MB
Loaded image: k8s-kms-plugin:1.0
```

Figure 6 : Load Docker Image

2. Verify the loaded image.

```
[root@master-node k8s_plugin]# docker images
REPOSITORY          TAG         IMAGE ID      CREATED      SIZE
k8s-kms-plugin      1.0        a0f2534b081e 2 days ago  273MB
```

Figure 7 : Verify the Loaded Image

5.1.4 Create a Client Certificate

1. Generate a private key and generate a CSR in `/home/admin/k8s_plugin/certs`.

```
[root@master-node certs]# pwd
/home/admin/k8s_plugin/certs
[root@master-node certs]# openssl genrsa -out kms_plugin_client.key 2048
[root@master-node certs]# cat << EOF | openssl req -new -key kms_plugin_client.key -out kms_plugin_client.csr -sha256
US
California
Campbell
Utimaco
Atalla
kms_plugin_user
eskM@utimaco.com
-----
EOF
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:State or Province Name (full name) [:]Locality Name (eg, city) [Default City]:Organization Name (eg, company) [Default Comp
any Ltd]:Organizational Unit Name (eg, section) [:]:Common Name (eg, your name or your server's hostname) [:]:Email Address [:]
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:An optional company name []:[root@master-node certs]#
```

Figure 8 : Generate a CSR

2. Copy the generated CSR and submit it to ESKM for signing by the ESKMLocalCA as a client certificate.

```
[root@master-node ~]# cat /home/admin/k8s_plugin/certs/kms_plugin_client.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIC2TCCAcECAQAwwZMxCzAJBgNVBAYTAlVTMRMwEQYDVQIDApDYWxpZm9ybm1h
MREwDwYDVQQHDAhDYW1wYmVsbDEwMDEwMDEwMDEwMDEwMDEwMDEwMDEwMDEw
QXRhbGxhMRGwFgYDVQQDDA9rbXNfcGx1Z2luX3VzZXIwMDEwMDEwMDEwMDEw
EGVza21AdXRpbWVjby5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQCeHRuiYlYdGcrkPuAEQkkNyaMncxLJFPACcTdN2reCkx5vJaYCTExVXwMVjrSW
RSC16LRS8Y4HqKpFWQRbwcFBArdpKipeVcCNLJ778xRO5Ex1tIhBcMGB2QZ0eylC
hvh+EMSx3eWc7BZZlbG8se+qi4N1/D2uJKP25jjm6pDTmKZKwJQ0RPBFK3U0UIYg
N0dyr29D3DqyxnuN0ujAUJgKmIaZii2FRLruqsHcxVJHU9U28pQCEE4YhrzKWovC
tGs0vqP9vkPlJ6ASHa/iGRWu0arV5Z65IhH3VJFYFqlsGwTxI4QP2H/G/MymfS1T
niIoHWuwGqrJFGQ1/9PDITgtAgMBAAGgADANBgkqhkiG9w0BAQsFAAOCQAQEA9H4
YmgTr30Mx4J4EYHzelFv2LgUHPq7idMExAVfmH+u4fu+uaERkzkzu8mp2LX7LXbI
CBWqKZBhRACXiHPkHJirAwqMeDe32S1/uoKAgCpps7k0o9EIFnN/YJZpLjUudBBg
nInJ34/lzhNmG+198tV4t3D5RVM0trexMUuKX71pZI9pFWnCdwSjcQi+7TZHpWk
EFbhyepWOqKSWCsyc5KfjC2Zkmf1KIi2Xp0u41sZr3dvsEhcqzmfU4h56ra8NWG
SKdFr7zZSgvq2P/Fh3IXsb7BI/9YBAoMb3ZL/s4oJ432roDRT7W6JwNdXznYF26V
KPCS1AUThGwoKj1gQg==
-----END CERTIFICATE REQUEST-----
```

Figure 9 : Generated CSR

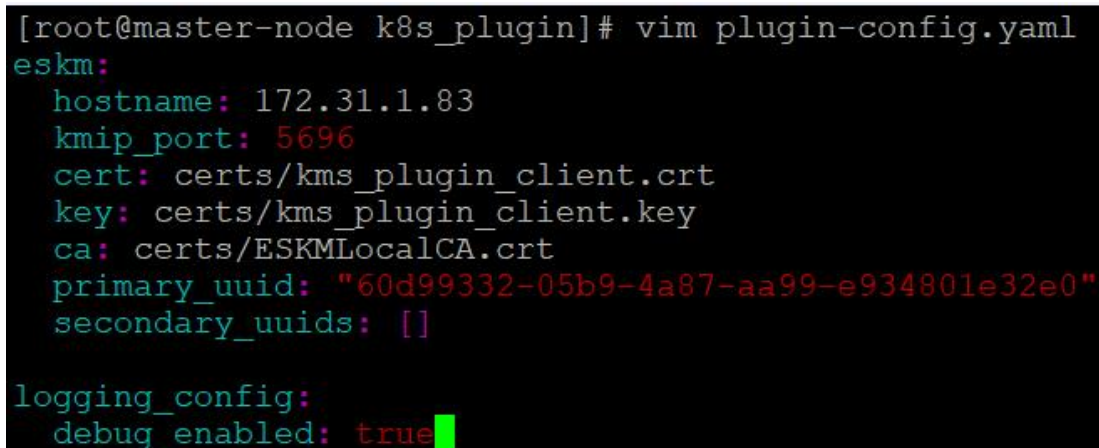
5.1.5 Configure the plugin-config File

1. Create the `plugin-config.yaml` file in the `k8s_plugin` directory.
2. For the `hostname` use the ESKM server host address.
3. Set the `mip_port` to `5696`.
4. For `cert`, specify the path to the client certificate: `certs/kms_plugin_client.crt`.
5. For `key`, specify the path to the private key: `certs/kms_plugin_client.key`.

6. For `ca`, specify the path to the ESKM Local CA: `certs/ESKMLocalCA.crt`.
7. For `primary_uuid`, enter the KMIP object UUID created for the corresponding user, see [Create KMIP Object](#).
8. For `secondary_uuid`, the value can be empty initially.
9. Set `debug_enabled` to `true` to enable debug logs or set it to false to disable them.

```
eskm:
  hostname: 172.31.1.83
  kmip_port: 5696
  cert: certs/kms_plugin_client.crt
  key: certs/kms_plugin_client.key
  ca: certs/ESKMLocalCA.crt
  primary_uuid: ""
  secondary_uuids: []

logging_config:
  debug_enabled: true
```



```
[root@master-node k8s_plugin]# vim plugin-config.yaml
eskm:
  hostname: 172.31.1.83
  kmip_port: 5696
  cert: certs/kms_plugin_client.crt
  key: certs/kms_plugin_client.key
  ca: certs/ESKMLocalCA.crt
  primary_uuid: "60d99332-05b9-4a87-aa99-e934801e32e0"
  secondary_uuids: []

logging_config:
  debug_enabled: true
```

Figure 10 : Configure plugin-config File

5.1.6 Create ConfigMap and Secrets

This step involves creating a Kubernetes **ConfigMap** and a **Secret** to store the configuration and credentials required by the KMS plugin.

The **ConfigMap** (`kms-plugin-config`) holds the plugin's main configuration file (`plugin-config.yaml`). The **Secret** (`kms-plugin-certs`) securely stores the TLS certificates and key used for communication between the plugin and the ESKM server.

```
[root@master-node k8s_plugin]# kubectl create configmap kms-plugin-config --from-file=/home/admin/k8s_plugin/plugin-config.yaml -n kube-system
configmap/kms-plugin-config created
[root@master-node k8s_plugin]# kubectl create secret generic kms-plugin-certs --from-file=/home/admin/k8s_plugin/certs/kms_plugin_client.crt --from-file=/home/admin/k8s_plugin/certs/kms_plugin_client.key --from-file=/home/admin/k8s_plugin/certs/ESKMLocalCA.crt -n kube-system
secret/kms-plugin-certs created
```

Figure 11 : Create ConfigMap kms-plugin-config



Any changes made in the plugin-config or certs corresponding configmap or secret should be deleted and created once, then restart the kms-plugin pod.

5.1.7 Deploy the KMS Plugin

This step involves deploying the `kms-plugin.yaml` file, which creates a Kubernetes DaemonSet to manage the KMS plugin pods across clusters.

1. Create the `kms-plugin.yaml` file in the `k8s_plugin` directory.

```
[root@master-node ~]# vim kms-plugin.yaml
```

Figure 12 : Create kms-plugin.yaml

2. Paste the following into the yaml file and save it.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kms-plugin-ds
  namespace: kube-system
  labels:
    app: kms-plugin
spec:
  selector:
    matchLabels:
      app: kms-plugin
  template:
    metadata:
      labels:
        app: kms-plugin
```

```
spec:
  nodeSelector:
    node-role.kubernetes.io/control-plane: "" # Or "master": "" depending on
your cluster's labels
  tolerations:
    # Tolerate the master/control-plane taint if it exists (common for control
plane nodes)
    - key: node-role.kubernetes.io/control-plane
      operator: Exists
      effect: NoSchedule
    - key: node.kubernetes.io/not-ready # Tolerate not-ready nodes during
startup
      operator: Exists
      effect: NoExecute
      tolerationSeconds: 300
    - key: node.kubernetes.io/unreachable # Tolerate unreachable nodes
      operator: Exists
      effect: NoExecute
      tolerationSeconds: 300

  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet

  containers:
    - name: kms-plugin
      image: k8s-kms-plugin:1.0 #Mention the image name to be run
      securityContext:
        runAsUser: 0
        runAsGroup: 0
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
      env:
        - name: PYTHONUNBUFFERED
          value: "1"
        - name: PYTHONPATH
          value: "/app"
      command: ["python", "-u"]
      args: ["kms_plugin/kms_server.py"]

      livenessProbe:
        exec:
          command:
            - /bin/sh
            - -c
            - "ls /var/lib/kmsplugin/kmsplugin-v1.sock && ls /var/lib/kmsplugin/
kmsplugin-v2.sock"
          initialDelaySeconds: 10
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
      readinessProbe:
        exec:
```

```
      command:
      - /bin/sh
      - -c
      - "ls /var/lib/kmsplugin/kmsplugin-v1.sock && ls /var/lib/kmsplugin/
kmsplugin-v2.sock"
      initialDelaySeconds: 5
      periodSeconds: 5
      timeoutSeconds: 3
      failureThreshold: 1
      volumeMounts:
      - name: config-volume
        mountPath: /app/config/plugin-config.yaml
        subPath: plugin-config.yaml
        readOnly: true
      - name: certs-volume
        mountPath: /app/config/certs
        readOnly: true
      - name: logs-volume
        mountPath: /app/logs
      - name: kms-socket-volume
        mountPath: /var/lib/kmsplugin

      volumes:
      - name: config-volume
        configMap:
          name: kms-plugin-config
      - name: certs-volume
        secret:
          secretName: kms-plugin-certs
      - name: logs-volume
        hostPath:
          path: /home/admin/k8s_plugin/logs
      - name: kms-socket-volume
        hostPath:
          path: /var/lib/kmsplugin
          type: DirectoryOrCreate
```

3. Apply the `kms-plugin.yaml` file.

```
[root@master-node k8s_plugin]# kubectl apply -f kms_plugin.yaml
daemonset.apps/kms-plugin-ds created
```

Figure 13 : Apply kms-plugin.yaml

4. Verify that `daemonset` was applied successfully.

```
[root@master-node k8s_plugin]# kubectl get daemonset -A
NAMESPACE      NAME                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
kube-system    calico-node         1         1         1       1             1           kubernetes.io/os
=linux        47h
kube-system    kms-plugin-ds       1         1         1       1             1           node-role.kubern
etes.io/control-plane=
60s
kube-system    kube-proxy          1         1         1       1             1           kubernetes.io/os
=linux        47h
```

Figure 14 : Verify daemonset

- Verify that the `kms-plugin` is running.

```
[root@master-node k8s_plugin]# kubectl get pods -A
NAMESPACE      NAME                                                         READY   STATUS    RESTARTS   AGE
kube-system    calico-kube-controllers-689744956f-ngd9x                   1/1     Running   13 (40h ago)  47h
kube-system    calico-node-vtkdp                                          1/1     Running   0             47h
kube-system    coredns-674b8bbfcf-d5htv                                   1/1     Running   0             47h
kube-system    coredns-674b8bbfcf-rjqbz                                  1/1     Running   0             47h
kube-system    etcd-master-node                                           1/1     Running   0             47h
kube-system    kms-plugin-ds-hcbqf                                        1/1     Running   0             109s
kube-system    kube-apiserver-master-node                                 1/1     Running   2             40h
kube-system    kube-controller-manager-master-node                       1/1     Running   15 (38h ago)  47h
kube-system    kube-proxy-jg9p6                                           1/1     Running   0             47h
kube-system    kube-scheduler-master-node                                1/1     Running   12 (38h ago)  47h
```

Figure 15 : Verify kms-plugin Pod

- Verify that the `kms-plugin` is logging successfully.

```
[root@master-node k8s_plugin]# kubectl logs kms-plugin-ds-hcbqf -n kube-system
INFO: Loading config from: /app/config/plugin-config.yaml
INFO: Config loaded successfully
INFO: Updated cert path to: /app/config/certs/kms_plugin_client.crt
INFO: Updated key path to: /app/config/certs/kms_plugin_client.key
INFO: Updated ca path to: /app/config/certs/ESKMLocalCA.crt
INFO: Certificate file found: /app/config/certs/kms_plugin_client.crt
INFO: Certificate file found: /app/config/certs/kms_plugin_client.key
INFO: Certificate file found: /app/config/certs/ESKMLocalCA.crt
INFO: All certificate files verified
INFO: Configuration loaded successfully
2025-08-06 10:24:00,581 - __main__ - INFO - Application starting with log level: DEBUG
INFO: Loading config from: /app/config/plugin-config.yaml
INFO: Config loaded successfully
INFO: Updated cert path to: /app/config/certs/kms_plugin_client.crt
INFO: Updated key path to: /app/config/certs/kms_plugin_client.key
INFO: Updated ca path to: /app/config/certs/ESKMLocalCA.crt
INFO: Certificate file found: /app/config/certs/kms_plugin_client.crt
INFO: Certificate file found: /app/config/certs/kms_plugin_client.key
INFO: Certificate file found: /app/config/certs/ESKMLocalCA.crt
INFO: All certificate files verified
2025-08-06 10:24:00,594 - __main__ - INFO - Configuration loaded successfully
2025-08-06 10:24:00,595 - __main__ - INFO - v1 Socket directory created/verified: /var/lib/kmsplugin
2025-08-06 10:24:00,596 - __main__ - INFO - v1 Removed existing socket file: /var/lib/kmsplugin/kmsplugin-v1.sock
2025-08-06 10:24:00,602 - __main__ - INFO - Both v1 and v2 KMS Plugin servers started successfully
2025-08-06 10:24:00,602 - __main__ - INFO - v1 socket: /var/lib/kmsplugin/kmsplugin-v1.sock
2025-08-06 10:24:00,603 - __main__ - INFO - v2 socket: /var/lib/kmsplugin/kmsplugin-v2.sock
2025-08-06 10:24:00,603 - __main__ - INFO - v2 Socket directory created/verified: /var/lib/kmsplugin
2025-08-06 10:24:00,603 - __main__ - INFO - v2 Removed existing socket file: /var/lib/kmsplugin/kmsplugin-v2.sock
2025-08-06 10:24:00,612 - __main__ - INFO - KMS Plugin v2 gRPC server starting on socket: /var/lib/kmsplugin/kmsplugin-v2.sock
2025-08-06 10:24:00,613 - __main__ - INFO - KMS Plugin v1 gRPC server starting on socket: /var/lib/kmsplugin/kmsplugin-v1.sock
2025-08-06 10:24:00,614 - __main__ - INFO - KMS Plugin v2 gRPC server started successfully
2025-08-06 10:24:00,615 - __main__ - INFO - KMS Plugin v1 gRPC server started successfully
```

Figure 16 : Verify kms-pod Logs

5.1.8 Configure the Encryption Provider

1. Create the `encryption-config.yaml` file in `/etc/kubernetes`.

```
[root@master-node k8s_plugin]# vim /etc/kubernetes/encryption-config.yaml
```

Figure 17 : Create encryption-config-yaml

2. Paste the following into the `encryption-config-yaml` file based on your Kubernetes version and save it.

```
kms v1
```

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
- resources:
  - secrets
  providers:
  - kms:
    apiVersion: v1
    name: eskm
    endpoint: unix:///var/lib/kmsplugin/kmsplugin-v1.sock
    timeout: 3s
- identity: {}
```

```
kms v2
```

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
- resources:
  - secrets
  providers:
  - kms:
    apiVersion: v2
    name: eskm
    endpoint: unix:///var/lib/kmsplugin/kmsplugin-v2.sock
    timeout: 3s
  - identity: {}
```

5.1.9 Configure the kube-apiserver

1. Open the `kube-apiserver.yaml` file located in `/etc/kubernetes/manifests`.

```
[root@master-node k8s_plugin]# vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

Figure 18 : kube-apiserver.yaml

2. In the command section, add `encryption-provider-config=/etc/kubernetes/encryption-config.yaml` to enable the use of an external encryption provider.

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.31.1.73
    - --allow-privileged=true
    - --encryption-provider-config=/etc/kubernetes/encryption-config.yaml
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
```

Figure 19 : Add Flag

3. In the `volumeMounts` section, add the following entries:

```
- mountPath: /etc/kubernetes/pki
  name: k8s-certs
  readOnly: true
- mountPath: /etc/kubernetes/encryption-config.yaml
  name: encryption-config
  readOnly: true
- mountPath: /var/lib/kmsplugin
  name: kmsplugin-socket
  readOnly: true
```

Figure 20 : volumeMounts

4. In the `volumes` section, add the corresponding volume definitions.

```
- hostPath:
  path: /etc/kubernetes/encryption-config.yaml
  type: File
  name: encryption-config
- hostPath:
  path: /var/lib/kmsplugin
  type: DirectoryOrCreate
  name: kmsplugin-socket
```

Figure 21 : volumes

5. Save the file and exit the editor.
6. Restart the `kube-apiserver` .

5.2 Configure Utimaco ESKM

Configuring Utimaco ESKM is crucial for secure and efficient key management. This section outlines the essential steps for configuring ESKM to integrate with Kubernetes.

5.2.1 Create a CA

The local CA is used to sign and verify the server certificate and may also be used to sign client certificate requests. To create and install a local CA, perform the following steps.

1. In the ESKM Management console, go to **Security > Certificates & CAs**, and click **Local CAs**.

Create Local Certificate Authority

Certificate Authority Name:	ESKMLocalCA
Country Name:	US
State or Province Name:	CA
Locality Name:	Campbell
Organization Name:	Organization
Organizational Unit Name:	Information Security
Common Name:	ESKMLocalCA
Email Address:	infosec@organization.com
Algorithm:	RSA-2048
Certificate Authority Type:	<input checked="" type="radio"/> Self-signed Root CA CA Certificate Duration (days): 3650 Maximum User Certificate Duration (days): 3650 <input type="radio"/> Intermediate CA Request

Create

Figure 22 : Create Local CA

2. Scroll down to the **Create Certificate** section.
3. Enter a **Certificate Authority Name** and **Common Name**. These may have the same value, for example, ESKMLocalCA.
4. Enter your **Organizational** information.
5. Select the **Algorithm** (e.g., RSA-2048).
6. Select **Self-signed Root CA** and enter the **CA Certification Duration** and **Maximum User Certificate Duration**. These values determine when the certificate must be renewed and should be set in accordance with your company's security policies. The default value for both is 3650 days or 10 years.
7. Click on **Create**.

5.2.2 Create a Server Certificate

To enable secure communication between Kubernetes and ESKM, you must create an ESKM Certificate.

1. In the ESKM Management console, go to **Security > Certificates and CAs**, and click **Certificates**.

Create Certificate

Certificate Name:	ESKMServerCert_KMS_plugin
Country Name:	US
State or Province Name:	CA
Locality Name:	Campbell
Organization Name:	Organization
Organizational Unit Name:	Information Security
Common Name:	ESKM
Email Address:	infosec@organization.com
Subject Alternative Name:	IP:172.31.1.83
Algorithm:	RSA-2048
Creation Type:	<input type="radio"/> Certificate Request - to be signed by external CA <input checked="" type="radio"/> Certificate Signed by Local CA
Local CA:	ESKMLocalCA (maximum 3649 days)
Certificate Purpose:	Server

Create

Figure 23 : Create Certificate

2. Enter **Certificate Name**, **Country Name**, **State or Province Name**, **Locality Name**, **Organization Name**, and **Organizational Unit Name**.
3. From the **Algorithm** dropdown list, Select **RSA-2408**.
4. Select the previously created CA certificate name from the **Local CA** dropdown list.
5. Select **Server** from the **Certificate Purpose** dropdown list.
6. Click **Create**.

5.2.3 Configure the KMIP Server

1. In the ESKM Management console, go to **Device > KMIP Server > KMIP Server**.

KMIP Server Configuration

KMIP Server Settings

IP:	[All]
Port:	5696
Server Certificate:	ESKMServerCert_KMS_plugin
Local CA Certificate for Certify/Re-certify:	ESKMLocalCA
Connection Timeout (sec):	360
Default number of items returned in Locate:	100
Maximum number of items returned in Locate:	1000

Figure 24 : KMIP Server Configuration

2. Select the relevant KMIP Port.
3. Select the created server certificate as **Server Certificate** for the KMIP server.
4. Select the created Local CA from the dropdown list.
5. Click **Save**.

5.2.4 Sign the Host Certificate Using ESKM

Review steps 5.1.1 to 5.1.4 in [Configuration on Kubernetes Control Plane Node](#) and ensure all the steps are completed before proceeding with the steps below.

1. Copy the generated CSR to `/home/admin/k8s_plugin/certs` and submit it to ESKM for signing by the ESKMLocalCA as a client certificate, see [Create a Client Certificate](#).

```
[root@master-node ~]# cat /home/admin/k8s_plugin/certs/kms_plugin_client.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIC2TCCAcECAQAwwZMxCzAJBgNVBAYTAlVTMRMwEQYDVQIDApDYWxpZm9ybm1h
MREwDwYDVQQHDAhDYWlwYmVsbDEQMA4GA1UECgwHVXRpbWFjby5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCeHRuiY1YdGcrkPuAEQkkNyaMncxLJFPACcTdN2reCkx5vJaYCTExVXwMVjrSW
RSCl6LRS8Y4HqKpFWQRbwcFBArdpKipeVkcNLJ778xRO5Ex1tIhBcMGB2QZ0eylC
hvh+EMSx3eWc7BZZ1bG8se+qi4N1/D2uJKP25jjm6pDTmKZKwJQ0RPBFK3U0UIYg
N0dyr29D3DqyxnuN0ujAUJgKmIaZii2FRLruqsHcxVJHU9U28pQCEE4YhrzKWovC
tGs0vqP9vkPlJ6ASHa/iGRWuOarV5Z65IhH3VJFYFqlsGwTxI4QP2H/G/MymfS1T
niIoHWuwGqrJFGQ1/9PDITgtAgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAB9H4
YmgTr3OMx4J4EyHZelFv2LgUHPq7idMExAVfmH+u4fu+uaERkzkzu8mp2LX7LxbI
CBWqKZBhRACXiHPkHJirAwqMeDe32S1/uoKAgCpps7k0o9EIFnN/YJZpLjUudBBg
nInJ34/1zhNmG+198tV4t3D5RVM0trexMUuKYX71pZI9pFWnCdwSjCQi+7TZHpWk
EFbhyepWOqKSWCsyc5KfjC2Zkmf1KIi2Xp0u41sZr3dvsEhcqzmfU4h56ra8NWG
SKdFr7zZSgvq2P/Fh3IXsb7BI/9YBAoMb3ZL/s4oJ432roDRT7W6JwNdXznYF26V
KPCS1AUThGwoKj1gQg==
-----END CERTIFICATE REQUEST-----
```

Figure 25 : Generated CSR in k8s_plugin

2. Go to ESKM Management Console > Security > Certificates & CAs > Local CAs .

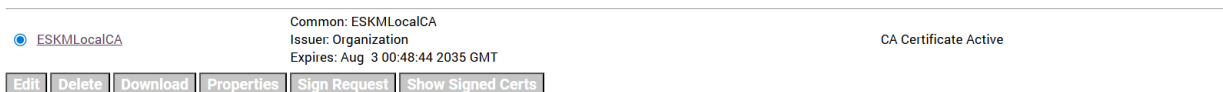


Figure 26 : Local CA

3. Select the created CA and click Sign Request.

Sign Certificate Request

Sign with Certificate Authority:

Certificate Purpose:

Server

Client

Server and Client

Certificate Duration (days):

Certificate Request:

```

-----BEGIN CERTIFICATE REQUEST-----
MIIC2TCCAcECAQAwgZMxCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybm1h
MREwDwYDVQQHDAhDYW1wYmVsbnVsbDEQMA4GA1UECgwHVXRPbWJFjzEPMA0GA1UECwwG
QXRhbGxhMRgwFgYDVQQDDA9rbXNfcGx1Z21uX3VzZXIxH2AdBgkqhkiG9w0BCQEW
EGVza21AdXRpbWJFjzEjY5b20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQCeHRuiYlYdGcrkPuAEQkkNyaMncxLJFPACCtdN2reCkx5vJaYCTExVXwMVjrSW
RSC16LRS8Y4HqkPFWQRbwcFBardpKipeVkcNLJ778xR05Ex1tIhBcMGB2QZ0eylC
hvh+EMsX3eWc7BZZ1bG8se+qi4N1/D2uJKP25jjm6pDTmKZKwJQ0RPBFK3U0UIYg
N0dyr29D3DqyxnuN0ujAUJgKmIaZii2FRLruqsHcxVJHU9U28pQCEE4YhrzKW0vC
tGs0vqP9vkPlJ6ASHa/iGRWuOarV5Z65IhH3VJFYFqlsGwTxI4QP2H/G/MymfS1T
niIoHWuwGqrJFGQ1/9PDI TgtAgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAB9H4
YmgTr30Mx4J4EYHZe1Fv2LgUHPq7idMEXAVfmH+u4fu+uaERkzkzu8mp2LX7LXB
CBWqKZBhRACXiHPKHJirAwqMeDe32S1/uokAgCpps7k0o9EIFnN/YJZpLjUudBBg
nInJ34/lzhNmG+l98tV4t3D5RVM0trexMUuKYX71pZI9pFwncdwsJcQi+7TZHpWk
EFbhypW0kQSWCsyc5KfjC2Zkmf1KIi2Xp0u41sZr3dvsEhcqzmfU4h56ra8NWG
    
```

Figure 27 : Sign Certificate Request

4. Select the previously created CA certificate from the **Sign with Certificate Authority** dropdown list.
5. Select **Client** in the **Certificate Purpose** section.
6. Copy the host certificate content to the **Certificate Request** box.
7. Copy the signed certificate and save it as `kms_plugin_client.crt` in the `certs` directory.

Certificate and CA Configuration

Signed Certificate Information

Serial Number:	0x1C
Key Size:	2048
Start Date:	Aug 4 00:58:32 2025 GMT
Expiration:	Aug 2 00:58:32 2035 GMT
Purpose:	SSL Client
Issuer:	C: US ST: CA L: Campbell O: Organization OU: Information Security CN: ESKMLocalCA emailAddress: infosec@organization.com
Subject:	C: US ST: California L: Campbell O: Utimaco OU: Atalla CN: kms_plugin_user emailAddress: eskm@utimaco.com

```

-----BEGIN CERTIFICATE-----
MIIEEjCCAvqgAwIBAgIBHDANBgkqhkiG9w0BAQsFADCB0jELMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAkNEMREwDwYDVQQHEwhhYmVhYmVhYmVhYmVhYmVhYmVhYmVhYmVh
emF0aW9uMR0wGwYDVQQLEExRJBmZvcmlhdG1vbiBTZW51cm10eTEUMBIGA1UEAAML
RVNLTUxvY2FzQ0ExJzAlBgkqhkiG9w0BCQEWGG1uZm9zZWNAb3JnYW5pemF0aW9u
LnNvbTAEFw0yNTA4MDQwMDU4MzJaFw0zNTA4MDIwMDU4MzJaMIGTMQswCQYDVQQG
EwJVVzETMBEGA1UECAwKQ2FsaWZvcmlhdG1vbiBTZW51cm10eTEUMBIGA1UEBhMC
EgNVBAoMB1V0aW1hY28xZm9zANBgNVBAAsMBkFOYXksYTYEYmVhYmVhYmVhYmVhYmVh
dWdpb191c2V5MR8wHQYJKoZIhvcNAQkBFhBlc2t0QHV0aW1hY28uY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnn0bomJWHRnK5D7gBEJJDcmjJ3MS
yRTwAnE3Tdq3gpMebyWmArRMVU8DFY60lkUgpei0UvGOB6iqRVkEW8HHwQK3a3oq
X1ZAjSye+/MUTuRMdb8IQXDEgdkGdHspQob4fhDEsd3lnOwWWZwXvLHvqouDdfw9
riSj9uY45uqQ05imSsCUNETwRSt1NFCGIDdHc9vQ9w6ss27jdLowFCYCPiGmYot
hUS67qrB3MVSr1PVNvKUAhBOG1a8y1jrwzRrNL6j/b5D53egEh2v4hkVrxmq1eWe
uSIR91SRW8apbB8E8SOED9h/xvaMpn0tU54iKB1rsBqqyRRkNf/TwyE4LQIDAQAB
o2AwXjAUBgNVHRMEEAJAAMB0GA1UdDgQWBBSFAfNXm2YP18RGj9ImrlWJDNjo6DAF
BgNVHSMEDAMgBTMs7TmJGh5EXZreLciXVgokEYJEDARBg1ghkgBhvhCAQEEBAMC
B4AwDQYJKoZIhvcNAQELBQADggEBAHyjhvYmKq5g5Q0bqMDXWnXtVMLUCqmeheW8
RNjy2Zd+zcZsFJS0q2j0m9xq5klmFtVyBFhy03R0KCEmT3pN16JW6H1r9gssi2kn
QstbOZ/3amKKg9buU70EeP4q7pflLjiMqQNKqGSfQdpAoBbN1sYvoEMVzMLPFhnET
KjClI02lipBfgGzBDDcNI6iJncozBugPOCA41kLcOhrMk11cN+kRjpi0FpyaFZce
7I1v8zR1D8mHpyZRVDhS6nhNzTyaev4h4o7iQsdh14AItgkVgyIT9qMa4MyQ3o77
s9LD1GbvC5GnXrM7iBRZxKno02m4qzKSHXZAbby9IT0JhxAPbk=
-----END CERTIFICATE-----

```

Figure 28 : Signed Certificate Information

```
[root@master-node certs]# vim kms_plugin_client.crt
-----BEGIN CERTIFICATE-----
MIIEEjCCAvcqAwIBAgIBHDANBgkqhkiG9w0BAQsFADCBojELMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAkNBMRewDwYDQgHEwhDYWlwYmVsbDEVMBMGAlUEChMMT3JnYW5p
emF0aW9uMR0wGwYDQgLExRJBmZvcmlhdGlvbiBTZW51cm10eTEUMBIGAlUEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGGluZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDU4MzJaFw0zNTA4MDIwMDU4MzJaMIGTMQswCQYDVQQG
EwJVVuZETMBEGAlUECAWkQ2FsaWZvcmlhdGlvbiBTZW51cm10eTEUMBIGAlUEBwwI
Q2FtcGJlbGwxEADAQgBgNVBAOMB1V0aW1hY28xZDZANBgNVBASMBkF0YWxsYTYEYmBYGA1UEAwPa21zX3Bs
dWdpb19lc2VyMR8wHQYJKoZIhvcNAQkBFhBlc2ttQHV0aW1hY28uY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnh0bomJWHRnK5D7gBEJJDcmjJ3MS
yRTwAnE3Tdq3gpMebyWmArRMV8DFY60lkUgpei0UvGOB6iqRVkEW8HHwQK3aSoq
XlZajSye+/MUTuRMdbSIQXDBgdkGdHspQob4fhDEsd3lnOwWWZwXvLHVgouDdfw9
risj9uY45uqQ05imSsCUNETwRst1NFCGIDdHcq9vQ9w6ssZ7jdLowFCYCpiGmYot
hUS67qrB3MVSRLPVNvKUAhBOGla8yljrwrRrNL6j/b5D5SegEh2v4hkVrjmqlWe
uSIR91SRWBapbBsE8SOED9h/xvzmpn0tU54iKB1rsBqqyRRkNf/TwyE4LQIDAQAB
o2AwXjAJBgNVHRMEAjaAMB0GA1UdDgQWBBSFAfNXm2YP18RGj9Imr1WJDNjo6Daf
BgNVHSMEGDAWgBTMs7TmJGh5EXZreLciXVgokEYJEDARBg1ghkgBhvhCAQEEBAMC
B4AwDQYJKoZIhvcNAQELBQADggEBAHyjhvYmKq5g5QObqMDXWnXtVMLUCqmeheW8
RNjyZZd+zCzsFJS0qZj0m9xg5k1mFtVyBFhy03R0KCEmT3pN16JW6H1r9gzsiZkn
QstbOZ/3amKKg9buU70EeP4q7pfLjiMqQNKqGSfQdpAoBbNlzYvoEMVzMLPFhnET
KjClI02lipBfqGzBDdCNI6iJncozBugPOCA41kLcOhrMk1lcN+kRjpI0FpyaFZce
7I1v8zRLDBmHpyZRVDhS6nhNzTyaev4h4o7iQzdh14AItgkVgyTT9qMa4MyQSo77
s9LD1GbvC5GnXrM7iBRZxKno02m4qAzKSHXZAbby9IT0JhxAPbk=
-----END CERTIFICATE-----
```

Figure 29 : kms_plugin_client.crt

8. Copy the ESKMLocalCA certificate and save it as **ESKMLocalCA.crt** in the certs directory.

CA Certificate Information

CA Certificate Name:	ESKMLocalCA
Key Size:	2048
Start Date:	Aug 4 00:48:44 2025 GMT
Expiration:	Aug 3 00:48:44 2035 GMT
Issuer:	C: US ST: CA L: Campbell O: Organization OU: Information Security CN: ESKMLocalCA emailAddress: infosec@organization.com
Subject:	C: US ST: CA L: Campbell O: Organization OU: Information Security CN: ESKMLocalCA emailAddress: infosec@organization.com

```

-----BEGIN CERTIFICATE-----
MIIEFDCCAvygAwIBAgIBADANBgkqhkiG9w0BAQsFADCB0jELMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAkNBMREwDwYDVQQHEwhDYW1wYmVsbDEVMBMGA1UEChMNT3JnYW5p
emF0aW9uMR0wGwYDVQQLEExRJBmZvcmlhdG1vbiBTZWNN1cm10eTEUMBIGI1UEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGG1uZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDQ4NDRaFw0zNTA4MDMwMDQ4NDRaMIGiMQswCQYDVQQG
EwJVUzELMAkGA1UECBMCQ0ExETAPBgNVBACTCENhbXBib2ZwXsMRUwEwYDVQQKEwxF
cmdhbml6YXRpb24xHTAbBgNVBAsTFE1uZm9yYbWF0aW9uIFN1Y3VyaXR5MRQwEgYD
VQQDEwtFU0tNTG9jYWxDQTEhMCUGCSqGSIb3DQEJARYYaW5mb3NlY0Bvcmdhbml6
YXRpb24uY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAySb/YPWh
aQEKFDmXjw1Uk32swOqjk+dBMb9meLOoI2kKQjX3+DzE2uSCX/DD146dBwe7y30D
IdwmK1Bm/xSWgplrm1udmmnyJhOj4MiQOSv5PIoymG1aAYNaqB1xb0FFd1xp4wdP
ME4827hiUqZ46zZ72Xc1DD4ToSU28EW0MaM6vbt128Q/n+PZu2897iOp/cuxwg11
qufbRAptJXCJxENLPudhDQ1LaXxgKPCZtPaxEV394bVM9MKvOzyztJFgzElAaiDy
9q4xwgsWagA8MJmXkHohmUsCnFaQKyeSbsObo+sDNWWYDOxsavzE6cxq+pDH5K1k
o3QqN/0Z8Y9PEwIDAQAB01MwUTAdBgNVHQ4EFgQUzLO05iRoerF2a3i3I11YKJBG
CRAwHwYDVR0jBBgwFoAUzLO05iRoerF2a3i3I11YKJBGcRAwDwYDVR0TAQH/BAUw
AwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAFTOVHxQstxbKQqE6qAy262vFC034tqKP
4UhbezC2r+vD+ZGmxt09z3UvgGnjEBhVkyuZKbk6KkooaF11YoMbFwZbUoJvG5ee
ViKFCZJSfsO2AH5hVRdGWh1PPBZ/0+j0qy10rJOnjbrMo1Pkz+JAmylf0G6cPWed
5EcuWZXUgF4uxz+m2PQWy+4n3KsX7mmDWEWmyWVpJN0pBdQ7PhCAkyS39tWuhrQ1
cgnS53HUhpvoz49J1Vd4s2RKf1DIUQ3ocV9XbMbM7df2zxbEJ1n+hhggThLL3i64
x83t/00THhjn12UvuSHBtg3PyltH04ZVoHuIRUGIpPOu9MsdAH/sg==
-----END CERTIFICATE-----
    
```

Figure 30 : CA Certificate Information

```
[root@master-node certs]# pwd
/home/admin/k8s_plugin/certs
[root@master-node certs]# vim ESKMLocalCA.crt
-----BEGIN CERTIFICATE-----
MIIEFDCCAvygAwIBAgIBADANBgkqhkiG9w0BAQsFADCBojELMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAKNBREwDwYDQgEWhDYWlwYmVsbDEVMBMGAlUEChMMT3JnYW5p
emF0aW9uMR0wGwYDQQLExRjbmZvcmlhdGlvbiBTZW51cml0eTEUMBIGAlUEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGgluzm9zZWNA3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDQ4NDRaFw0zNTA4MDMwMDQ4NDRaMIGiMQswCQYDQgQ
EwJVUzELMAkGA1UECBMCQ0ExETAPBgNVBACTECNhbXBiZWxsMRUwEwYDQgKEwxP
cmdhbm16YXRpb24xHTAbBgNVBAsTFEluZm9ybW90aW9uIFNlY3VyaXR5MRQwEgYD
VQQDEwtFU0tNTG9jYXZlDDE4MjE4MjE4MjE4MjE4MjE4MjE4MjE4MjE4MjE4MjE4
YXRpb24uY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAYsB/YPWh
aQEKfDmXjw1Uk32swOqjk+dBMb9meL0oI2kKQjX3+DzE2uSCX/DD46dBwe7y30D
IdwmK1Bm/xSWgplrmludmmnyJh0j4MiQOSv5PIoymG1aAYNaqBlxb0FFdlxp4wdP
ME4827hiUqZ46zZ72XclDD4ToSU28EW0MaM6vbt128Q/n+PZu2897iOp/cuxwgl1
qufBRaptJXCJxeNLPudhDQ1LaXxgKpcZtPaxEV394bVM9MKvOzyztJFgzElAaiDy
9q4xwgswagA8MJmXkHohmUsCnFaQKyesbsObo+sDNWWYDOxsavzE6cxq+pDH5K1k
o3QqN/0Z8Y9PEwIDAQABo1MwUTAdBgNVHQ4EFgQUzLO05iRoeRF2a3i3I11YKJBG
CRAWHwYDVR0jBBgwFoAUzLO05iRoeRF2a3i3I11YKJBGCRAdwYDVR0TAQH/BAUw
AwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAFTOVHxQstxbKQqE6qAy262vFC034tqKP
4UhbezC2r+vD+ZGmxt09z3UvgGnjEBhVkyuZKBk6KkooaFl1YoMbfWZbUoJvG5ee
VikFCZJSfs02AH5hVRdGWh1PPBZ/0+j0qy10rJOnjBRMolPkz+JAmylf0G6cPWed
5EcuWZXUgF4uxz+m2PQWy+4n3KsX7mmDWEWmyWVpJN0pBdQ7PhCAkyS39tWuhrQ1
cgnS53HUhpvoz49J1Vd4s2RKf1DIUQ3ocV9XBmM7df2zxbEJ1n+hhggThLL3i64
x83t/00THhjn12UvuSHBtg3PyltH04ZVoHuIRUGIpPOu9MsdtAH/sg==
-----END CERTIFICATE-----
```

Figure 31 : ESKMLocalCA.crt

9. Ensure that all three files `ESKMLocalCA.crt`, `kms_plugin_client.crt` and `kms_plugin_client.key` are present in the `certs` directory.

```
[root@master-node certs]# pwd
/home/admin/k8s_plugin/certs
[root@master-node certs]# ls -ll
total 16
-rw-r--r--. 1 root root 1476 Aug  4 11:00 ESKMLocalCA.crt
-rw-r--r--. 1 root root 1472 Aug  4 10:59 kms_plugin_client.crt
-rw-r--r--. 1 root root 1066 Aug  4 10:57 kms_plugin_client.csr
-rw----- . 1 root root 1704 Aug  4 10:56 kms_plugin_client.key
```

Figure 32 : certs Directory

5.2.5 Create KMIP User

1. In the ESKM Management console, go to Security > Users & Groups > Local Users & Groups > Local Users.
2. Click Add.
3. Enter the **Username** in the same way as the common name provided during host certificate creation.

4. Select Enable KMIP.
5. Select KMIP as License Type.
6. In **KMIP Client Certificate** provide the client certificate that is generated by signing the CSR with ESKMLocalCA.

Create Local User

Username:

Password:

Confirm Password:

License Type:

User Administration Permission:

Change Password Permission:

Enable KMIP:

Map non-existent Object Group to x-Object Group:

KMIP User Group:

KMIP Object Group:

KMIP Client Certificate:

```

-----BEGIN CERTIFICATE-----
MIIEFDCCAbygAwIBAgIBADANBgkqhkiG9w0BAQsFADC
BojEELMAkGA1UEBhMCVVMx
CzAJBgNVBAGTAkNBREwDwYDVQQHEwhDYW1wYmVsbnBDE
VMBMGA1UEChMNT3JnYW5p
emF0aW9uMR0wGwYDVQQLEXRJbmZvcmlhdG1vb1BTZW9
1cm10eTEUMBIGA1UEAxML
RVNLTUxvY2FsQ0ExJzA1BgkqhkiG9w0BCQEWGG1uZm9
zZWNA3JnYW5pemF0aW9u
    
```

Figure 33 : Create Local User

5.2.6 Create KMIP Object

1. In the ESKM Management Console go to **Security > KMIP Objects > Create KMIP Objects**.

KMIP Object Configuration

Create KMIP Object

Object Name:

Owner Username:

Object Type:

Algorithm:

Figure 34 : Create KMIP Object

2. Enter kms_plugin_key as Object Name and kms_plugin_user as Owner User Name.
3. Select Symmetric Key from the Object Type dropdown list and AES-256 from the Algorithm dropdown list.

KMIP Objects

[Help ?](#)

Query:

Items per page:

▲ UUID	Object Name	Owner	Object Type	State	Creation Date	FIPS Security Level
<input type="radio"/> 576e8f86-5bee-40a2-9b4b-88e24f310cd4	k8s-cluster1-key	clusteruser1	SymmetricKey	Active	2025-07-07 17:13:50	1
<input type="radio"/> 7223ce7a-ba6c-4c29-8819-c8778aeeec737	k8s-cluster2-key	clusteruser2	SymmetricKey	Active	2025-07-15 13:54:44	1
<input type="radio"/> 84817b09-cbec-4f07-8f5b-6c928b36e1c9	k8s-cluster2-key2	clusteruser2	SymmetricKey	Active	2025-07-28 12:32:55	1
<input checked="" type="radio"/> 60d99332-05b9-4a87-aa99-e934801e32e0	kms_plugin_key	kms_plugin_user	SymmetricKey	Active	2025-08-05 01:02:37	1
<input type="radio"/> 9665c782-d8a6-47b7-a273-2377a335c46d	kms_plugin_key1	kms_plugin_user	SymmetricKey	Active	2025-08-05 01:55:23	1
<input type="radio"/> f551f090-1aec-4253-82fe-1f07a3053fad	-	DelIDDCL1	SymmetricKey	Active	2025-06-10 18:13:30	1
<input type="radio"/> a166ad84-f2b5-4c67-b40f-82d22de69ae4	-	kmipClient20250605100645	SymmetricKey	Active	2025-06-05 17:20:54	1

1 - 7 of 7

Figure 35 : KMIP Objects

6 Verification and Testing

In this chapter, we verify whether the integration of Utimaco ESKM and Kubernetes via the custom KMS plugin works as expected. This includes validating the connectivity between the Kubernetes KMS Plugin and the ESKM server, testing encryption and decryption workflows, and ensuring the Kubernetes control plane successfully interacts with the plugin. By the end of this section, you should be able to confirm that the integration is correctly configured, secure, and fully operational.

6.1 Functional Testing

6.1.1 Verify KMIP Log

After restarting the kube-apiserver, the Utimaco ESKM KMIP server logs show successful communication between Kubernetes and the external KMS plugin. The following log entries confirm successful encryption and decryption operations using the KMIP protocol:

```
2025-08-04 18:39:04 [KMIP Server] [Authentication Success] User:[kms_plugin_user] From IP: 172.31.1.73
2025-08-04 18:39:04 [KMIP Server] [ClientOperation] User:[kms_plugin_user] UUID:[60d99332-05b9-4a87-aa99-e934801e32e0] Operation:[ENCRYPT] Object Type:[SYMMETRIC_KEY] Result:[SUCCESS]
2025-08-04 18:39:05 [KMIP Server] [Authentication Success] User:[kms_plugin_user] From IP: 172.31.1.73
2025-08-04 18:39:05 [KMIP Server] [ClientOperation] User:[kms_plugin_user] UUID:[60d99332-05b9-4a87-aa99-e934801e32e0] Operation:[DECRYPT] Object Type:[SYMMETRIC_KEY] Result:[SUCCESS]
```

Figure 36 : Verify KMIP Log

6.1.2 Create a Secret

First, create a simple secret using `kubectl`. This action will trigger the encryption process via the KMS plugin.

The output `secret/test-secret created` confirms that the secret was successfully submitted to the Kubernetes API.

```
[root@master-node ~]# kubectl create secret generic test-secret --from-literal=mykey=mydata
secret/test-secret created
```

Figure 37 : Create Secret

6.1.3 Verify Encryption in etcd

Use `etcdctl` to directly inspect the secret's value in the etcd database. The output should show that the KMS provider has encrypted the data.

The output will be unreadable and begin with the `k8s:enc:kms:v2:eskm:` header. This header proves that the KMS plugin successfully encrypted the secret data before storing it in etcd.

```
[root@master-node ~]# etcdctl get /registry/secrets/default/test-secret --print-value-only
k8s:enc:kms:v2:eskm:
#<WR_AwU
E Or c 5 & ,J+? p!O \4 x _ $o ,K _ 2
m <= T 9 | B; :
2X *tB=% GJ7 ) i7T %n>hQ , u tlee5 " ^: ó n
m >UE&4 u^n ) >
E}u & ' fN'LhRl C dAi _ A _ J _ A _ % z3 Z~a R> R< i
n~% $60d99332-05b9-4a87-aa99-e934801e32e0@ * ;d :R- yF] I Y H %
#w G @K (
```

Figure 38 : Verify Encryption

6.1.4 Verify Decryption

Retrieve the secret using `kubectl`. This action prompts the kube-apiserver to check its cache for the decrypted data. If the secret is not found in the cache, the kube-apiserver initiates a decryption request to the KMS plugin, which returns the plaintext data.

The output will show the secret's data in its original, unencrypted format (Base64-encoded). The `mykey` field, which was encrypted in etcd, is now readable as `bXlkYXRh` (Base64 for ``mydata``). This confirms that the entire decryption workflow is working as expected.

```
[root@master-node ~]# kubectl get secret test-secret -o yaml
apiVersion: v1
data:
  mykey: bXlkYXRh
kind: Secret
metadata:
  creationTimestamp: "2025-08-07T09:28:21Z"
  name: test-secret
  namespace: default
  resourceVersion: "327991"
  uid: e59486f1-e50f-47a5-9111-4714a8c46a93
type: Opaque
```

Figure 39 : Verify Decryption

6.1.5 Key Rotation

This procedure outlines the steps to safely rotate the encryption keys used by the KMS plugin. Key rotation is a critical security practice that ensures secrets are regularly re-encrypted with a new key.

6.1.5.1 Create a New KMIP Object

Create a new KMIP object in ESKM for the kms_plugin_user, see [Create KMIP Object](#).



UUID	Object Name	Owner	Object Type	State	Creation Date	FIPS Security Level
<input checked="" type="radio"/> 576e8f86-5bee-40a2-9b4b-88e24f310cd4	k8s-cluster1-key	clusteruser1	SymmetricKey	Active	2025-07-07 17:13:50	1
<input type="radio"/> 60d99332-05b9-4a87-aa99-e934801e32e0	kms_plugin_key	kms_plugin_user	SymmetricKey	Active	2025-08-05 01:02:37	1
<input type="radio"/> 7223ce7a-ba6c-4c29-8819-c8778aee737	k8s-cluster2-key	clusteruser2	SymmetricKey	Active	2025-07-15 13:54:44	1
<input type="radio"/> 84817b09-cbec-4f07-8f5b-5c928b36e1c9	k8s-cluster2-key2	clusteruser2	SymmetricKey	Active	2025-07-28 12:32:55	1
<input type="radio"/> 9665c782-d8a6-47b7-a273-2377a335c46d	kms_plugin_key1	kms_plugin_user	SymmetricKey	Active	2025-08-05 01:55:23	1
<input type="radio"/> a166ad84-f2b5-4c67-b40f-82d22de69ae4	-	kmipClient20250605100645	SymmetricKey	Active	2025-06-05 17:20:54	1
<input type="radio"/> f551f090-1aec-4253-82fe-1f07a3053fad	-	DellDDCL1	SymmetricKey	Active	2025-06-10 18:13:30	1

Figure 40 : Create a KMIP Object

6.1.5.2 Update plugin-config.yaml file

1. Set the new key's UUID (9665c782-d8a6...) as the primary_uuid.
2. Move the old primary key's UUID (60d99332-05b9...) to the secondary_uuids list.

```
[root@master-node k8s_plugin]# vim plugin-config.yaml
eskm:
  hostname: 172.31.1.83
  kmip_port: 5696
  cert: certs/kms_plugin_client.crt
  key: certs/kms_plugin_client.key
  ca: certs/ESKMLocalCA.crt
  primary_uuid: "9665c782-d8a6-47b7-a273-2377a335c46d"
  secondary_uuids: ["60d99332-05b9-4a87-aa99-e934801e32e0"]

logging_config:
  debug_enabled: true
```

Figure 41 : plugin-config.yaml file

6.1.5.3 Re-create ConfigMap

1. Delete the existing `kms-plugin-config` ConfigMap and recreate it with the updated `plugin-config.yaml` file, see [Create Configmap and Secrets](#).
2. Restart the `kms-plugin-ds` DaemonSet to ensure the plugin pods are running with the new configuration.

```
[root@master-node k8s_plugin]# kubectl delete configmap kms-plugin-config -n kube-system
configmap "kms-plugin-config" deleted
[root@master-node k8s_plugin]# kubectl create configmap kms-plugin-config --from-file=/home/admin/k8s_plugin/plugin-config.yaml -n kube-system
configmap/kms-plugin-config created
[root@master-node k8s_plugin]# kubectl rollout restart daemonset/kms-plugin-ds -n kube-system
daemonset.apps/kms-plugin-ds restarted
```

Figure 42 : Re-create ConfigMap

6.1.5.4 Re-encrypt Existing Secrets

This step relies on the kube-apiserver's ability to decrypt secrets with the old key and then re-encrypt them with the new one.

The following command retrieves all secrets and forces the kube-apiserver to replace them, triggering re-encryption with the new primary key.

```
[root@master-node k8s_plugin]# kubectl get secrets --all-namespaces -o json | kubectl replace --force -f -
secret "test-secret" deleted
secret "kms-plugin-certs" deleted
secret/test-secret replaced
secret/kms-plugin-certs replaced
```

Figure 43 : Re-encrypt Existing Secrets



This command re-encrypts all secrets, including system secrets. It's recommended that this be performed in a controlled maintenance window.

6.2 Logs and Validation Steps

6.2.1 Verify Key Rotation Success

1. Check the Utimaco ESKM KMIP logs for successful `ENCRYPT` and `DECRYPT` requests associated with the new key's UUID (9665c782-d8a6...). This confirms the kube-apiserver is using the new key for both new and re-encrypted secrets.

```
2025-08-07 10:05:18 [KMIP Server] [Authentication Success] User:[kms_plugin user] From IP: 172.31.1.73
2025-08-07 10:05:18 [KMIP Server] [ClientOperation] User:[kms_plugin user] UUID:[9665c782-d8a6-47b7-a273-2377a335c46d] Operation:[ENCRYPT] Object Type:[SYMMETRIC_KEY] Result:[SUCCESS]
2025-08-07 10:13:46 [KMIP Server] [Authentication Success] User:[kms_plugin user] From IP: 172.31.1.73
2025-08-07 10:13:46 [KMIP Server] [ClientOperation] User:[kms_plugin user] UUID:[9665c782-d8a6-47b7-a273-2377a335c46d] Operation:[DECRYPT] Object Type:[SYMMETRIC_KEY] Result:[SUCCESS]
```

Figure 44 : ESKM KMIP Logs

2. Use `etcdctl` to get a secret's value. The output shows that the secret is now encrypted with the new `UUID (9665c782-d8a6-47b7-a273-2377a335c46d)`. The presence of this UUID confirms a successful key rotation.

```
[root@master-node kms_plugin]# etcdctl get /registry/secrets/default/test-secret --print-value-only
k8s:enc:kms:v2:eskm:
+X`|`%
Z I, J y < 5/R, o C " ?æ e ij 7 `
sb h!$ , tg7 | k a 8
* fy u kJ ) 7 ók +k ] Lh w X 19 ~ j K F
$9665c782-d8a6-47b7-a273-2377a335c46d@H Ii+N v O ( i: ÿh ( A~ -" ] =
R | d y (
```

Figure 45 : Verify Key Rotation Success

7 Troubleshooting

7.1 Log Locations and Interpretation

You can verify the logs from Utimaco ESKM by following the steps below:

1. In the ESKM Management Console, click **Device > Logs & Statistics > Log Viewer > KMIP**.
2. Review logs related to **ENCRYPT** and **DECRYPT** operations performed on the Kubernetes control plane.

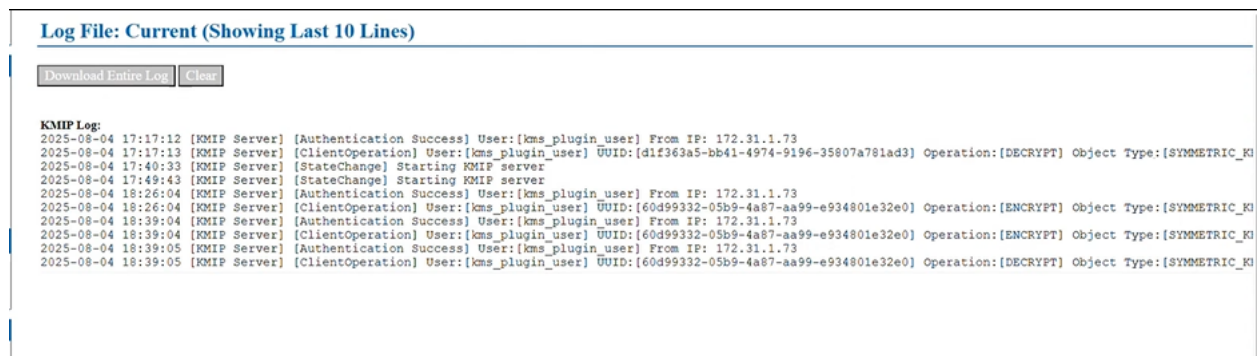


Figure 46 : KMIP Logs

3. For detailed debug logs, see **8s_plugin/logs**.

8 Contact and Support Information

You can reach us from Monday to Friday, 09.00 a.m. to 05.00 p.m., Central European Time (CET).

Utimaco IS GmbH
Germanusstr. 4
52080 Aachen
Germany

RMA Query

If you need to send the device back to Utimaco IS GmbH, please open a new RMA case (Return Merchandise Authorization). We request that you use the following web address. RMA cases cannot be opened by email or phone.

<https://support.hsm.utimaco.com/support/rma/new>

Other Support Queries

- Mail (preferred contact method)
support@utimaco.com
Attach the diagnostic information to your email.
- Web portal
<https://support.hsm.utimaco.com/support/cases/new/>
The diagnostic information will be requested in our response if necessary.
- By phone
AMERICAS +1-844-UTIMACO (+1 844-884-6226)
EMEA +49 800-627-3081
APAC +81 800-919-1301
The diagnostic information will be requested in our response if necessary.

9 Appendices

9.1 References

Title	Description	Document/Link
ESKM Installation Guide	Step-by-step guide for installing and configuring ESKM.	Installation and Replacement Guide
Creating a cluster with kubeadm	Official document for setting up a Kubernetes Cluster.	Creating a cluster with kubeadm Kubernetes
Kubernetes Documentation	Official Kubernetes documentation covering setup, configuration, and management.	Using a KMS provider for data encryption Kubernetes

Table 3: References

9.2 Command Summary

Command	Purpose
<code># mkdir /var/lib/kmsplugin</code>	To create a directory for socket files.
<code># mkdir k8s_plugin</code>	To create a directory for plugin files.
<code># mkdir certs</code>	To create a directory for certificates.
<code># mkdir logs</code>	To create a directory for plugin log file.

Command	Purpose
<pre># docker load -i k8s-kms- plugin_1.0.tar</pre>	To load a Docker image.
<pre># openssl genrsa -out kms_plugin_client.key 2048</pre>	To create a private key.
<pre># cat << EOF openssl req -new -key kms_plugin_client.key -out kms_plugin_client.csr -sha256 US California Campbell Utimaco Atalla kms_plugin_user eskm@utimaco.com EOF</pre>	To generate a CSR.
<pre># cat /home/admin/k8s_plugin/certs/ kms_plugin_client.csr</pre>	To view the generated CSR.
<pre># vim plugin-config.yaml</pre>	To create/edit the plugin-config.yaml file
<pre># kubectl create configmap kms- plugin-config --from-file=/home/admin/ k8s_plugin/plugin-config.yaml</pre>	To create the kube configmap for plugin- config.yaml.

Command	Purpose
<pre># kubectl create secret generic kms- plugin-certs --from-file=/home/admin/ k8s_plugin/certs/kms_plugin_client.crt --from-file=/home/admin/k8s_plugin/ certs/kms_plugin_client.key --from- file=/home/admin/k8s_plugin/certs/ ESKMLocalCA.crt -n kube-system</pre>	To create the kube secret for certificates.
<pre># vim kms-plugin.yaml</pre>	To create/edit the kms-plugin.yaml daemonset file.
<pre># kubectl apply -f kms-plugin.yaml</pre>	To apply the kms-plugin.yaml file.
<pre># kubectl get daemonset -A</pre>	To view the kube system daemonset.
<pre># kubectl get pods -A</pre>	To view the kube system pods.
<pre># vim /etc/kubernetes/encryption- config.yaml</pre>	To create/edit the encryption-config.yaml file.
<pre># vim /etc/kubernetes/manifests/kube- apiserver.yaml</pre>	To edit the kube-apiserver.yaml file.
<pre># kubectl create secret generic test- secret --from-literal=mykey=mydata</pre>	To create a secret.
<pre># etcdctl get /registry/secrets/ default/test-secret --print-value- only</pre>	To verify encryption in etcd.

Command	Purpose
<pre># kubectl get secret test-secret -o yaml</pre>	To verify the description.
<pre># kubectl delete configmap kms- plugin-config -n kube-system.</pre>	To delete the configmap.
<pre># kubectl rollout restart daemonset/ kms-plugin-ds -n kube-system</pre>	To restart the kms-plugin daemonset.
<pre># kubectl get secrets -- all- namespaces -o json kubectl replace -- force -f -</pre>	To re-encrypt existing secrets.

Table 4: CLI Commands