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

Table of Contents

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	`chsm-create`
Italic	References and important terms	See *Sample Chapter* in the *CryptoServer - Sample Manual*

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 Kubernetes v1.33.2	8.54.0

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.

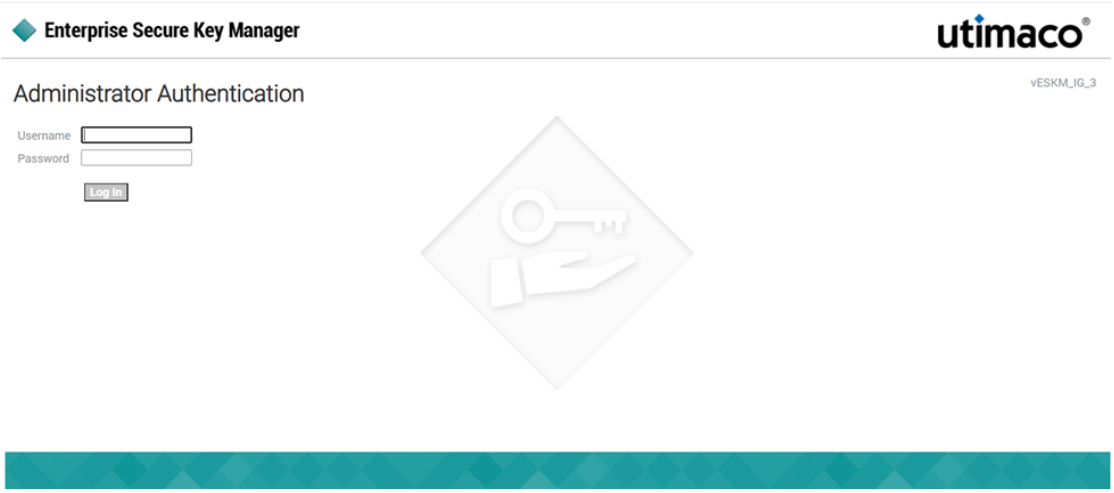

Figure 1 : ESKM Login Page

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`.

Figure 4 : Create Certs Directory

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

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`.

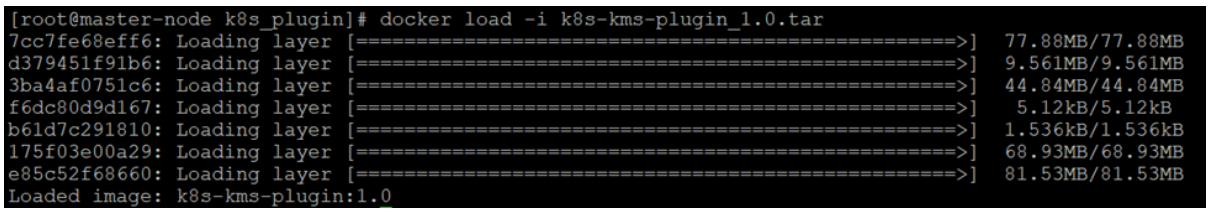

Figure 6 : Load Docker Image

2. Verify the loaded image.

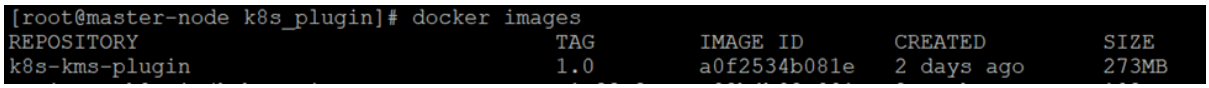

Figure 7 : Verify the Loaded Image

5.1.4 Create a Client Certificate

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

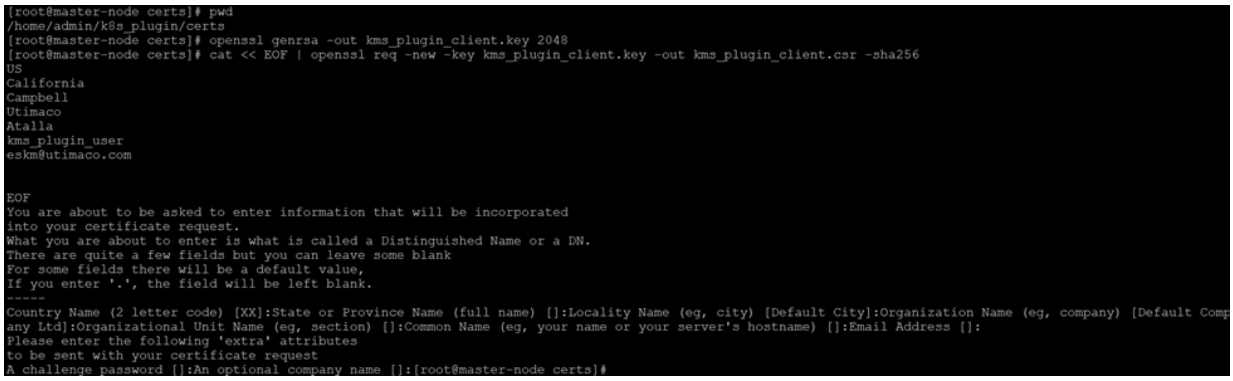

Figure 8 : Generate a CSR

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

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
```

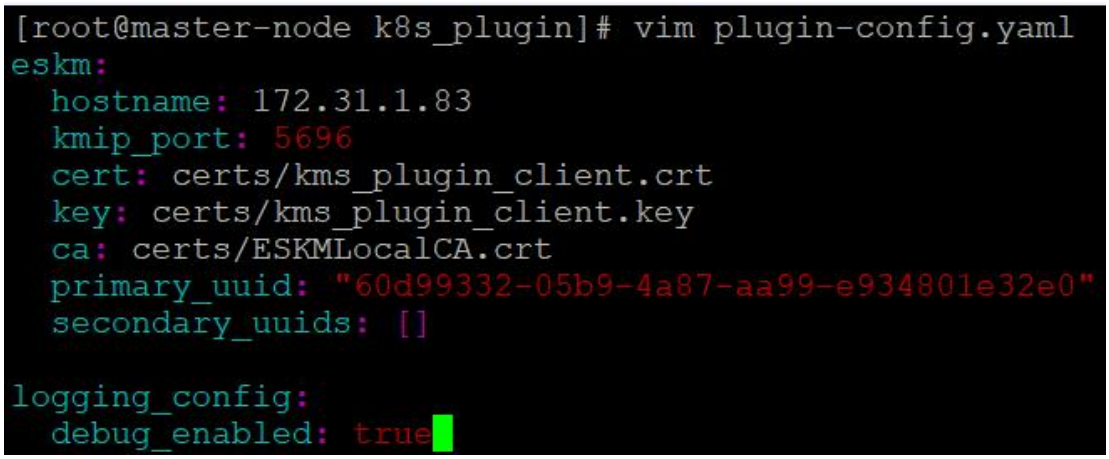

Figure 10 : Configure plugin-config File

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 --fr
om-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=/ho
me/admin/k8s_plugin/certs/kms_plugin_client.key --from-file=/home/admin/k8s_pl
ugin/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
                        AGE
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

5. 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

6. Verify that the `kms-plugin` is loging 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.so
ck
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.so
ck
2025-08-06 10:24:00,612 - __main__ - INFO - KMS Plugin v2 gRPC server starting on socket: /var/lib/kmsplugin/km
splugin-v2.sock
2025-08-06 10:24:00,613 - __main__ - INFO - KMS Plugin v1 gRPC server starting on socket: /var/lib/kmsplugin/km
splugin-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`.

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**.

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**.

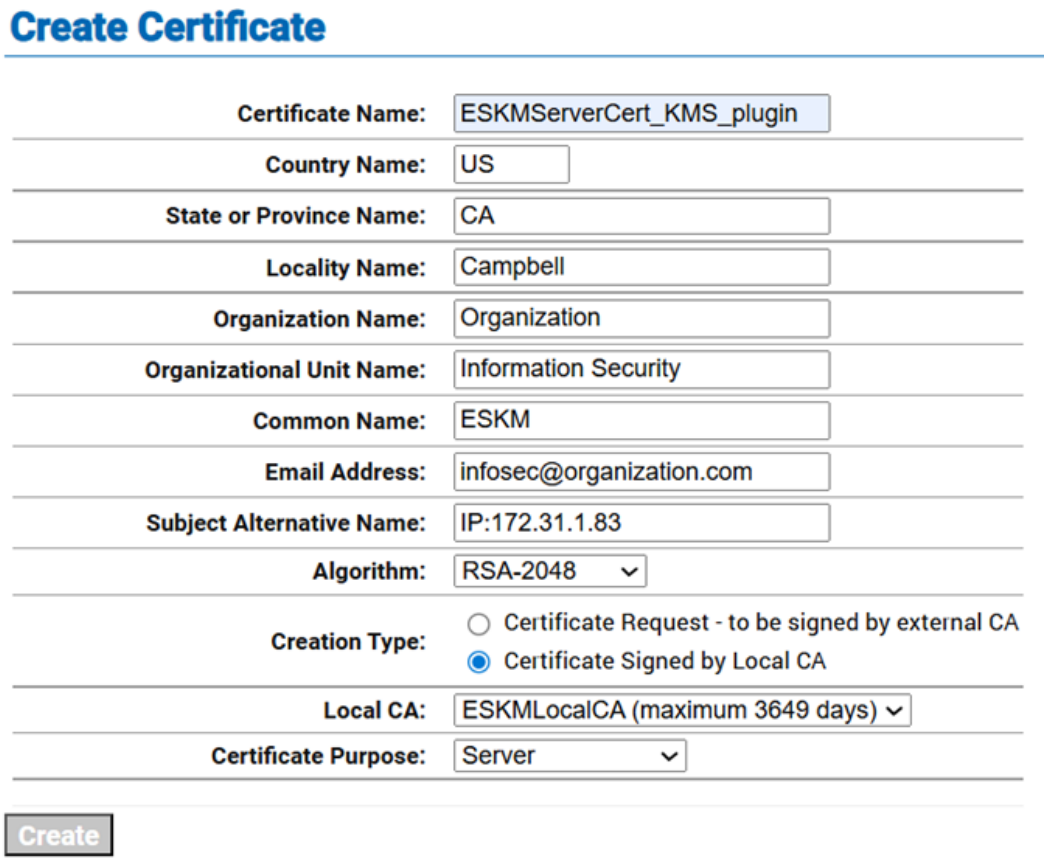

Figure 23 : Create Certificate

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**.

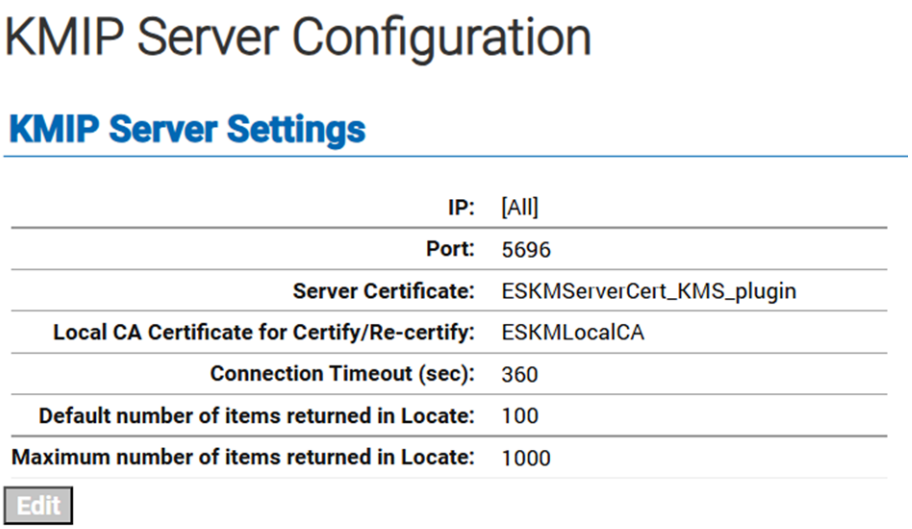

Figure 24 : KMIP Server Configuration

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.

Figure 25 : Generated CSR in k8s_plugin

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

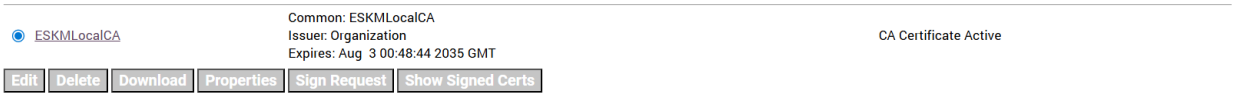

Figure 26 : Local CA

3. Select the created **CA** and click **Sign Request**.

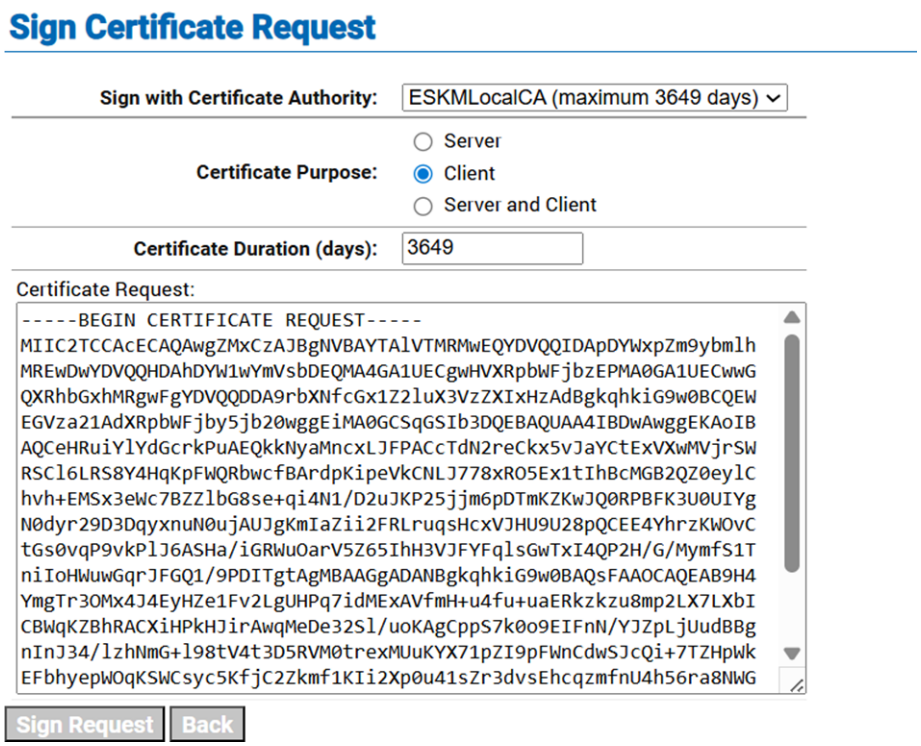

Figure 27 : Sign Certificate Request

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-----
MIIEEjCCAvqgAwIBAgIBHDANBgkqhkiG9w0BAQsFADCBojELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkNBMREwDwYDVQQHEwhDYW1wYmVsbDEVMBMGA1UEChMMT3JnYW5p
emF0aW9uMR0wGwYDVQQLExRJbmZvcm1hdGlvbiBTZWN1cml0eTEUMBIGA1UEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGGluZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDU4MzJaFw0zNTA4MDIwMDU4MzJaMIGTMQswCQYDVQQG
EwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwIQ2FtcGJlbGwxEDAO
BgNVBAoMB1V0aW1hY28xDzANBgNVBAsMBkF0YWxzYTEYMBYGA1UEAwwPa21zX3Bs
dWdpbl91c2VyMR8wHQYJKoZIhvcNAQkBFhBlc2ttQHV0aW1hY28uY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnh0bomJWHRnK5D7gBEJJDcmjJ3MS
yRTwAnE3Tdq3gpMebyWmArRMVV8DFY601kUgpei0UvGOB6iqRVkEW8HHwQK3aSoq
XlZAj3ye+/MUTuRMdbSIQXDBgdkGdHspQob4fhDEsd3lnOwWWZWxvLHvqouDdfw9
riSj9uY45uqQ05im3sCUNETwR3t1NFCGIDdHcq9vQ9w6ssZ7jdLowFCYCpiGmYot
hU367qrB3MV3R1PVNvKUAhBOGIa8yljrwrRxNL6j/b5D5SegEh2v4hkVrjmq1eWe
u3IR91SRWBapbBsE83OED9h/xvzMpn0tU54iKB1rsBqqyRRkNf/TwyE4LQIDAQAB
o2AwXjAJBgNVHRMEAjAAMB0GA1UdDgQWBBSFAfNXm2YP18RGj9ImrlWJDNjo6DAf
BgNVH3MEGDAWgBTMs7TmJGh5EXZreLciXVgokEYJEDARBglghkgBhvhCAQEEBAMC
B4AwDQYJKoZIhvcNAQELBQADggEBAHyjhvYmKq5g5QObqMDXWnXtVMLUCqmehew8
RNjyZZd+zCZsFJ30qZj0m9xq5k1mFtVyBFhy03R0KCEmT3pN16JW6H1r9gzsiZkn
QstbOZ/3amKKg9buU70EeP4q7pfLjiMqQNKqG3fQdpAoBbNlzYvoEMVzMLPFhnET
KjClI02lipBfqGzBDdCNI6iJncozBugPOCA4lkLcOhrMkllcN+kRjpI0FpyaFZce
7I1v8zRlDEmHpyZRVDh56nhNzTyaeV4h4o7iQzdh14AItgkVgyTT9qMa4MyQ3o77
s9LD1GbvC5GnXrM7iBRZxKno02m4qAzKSHXZAbby9IT0JhxAPbk=
-----END CERTIFICATE-----
```


Figure 28 : Signed Certificate Information

```
[root@master-node certs]# vim kms_plugin_client.crt
-----BEGIN CERTIFICATE-----
MIIEEjCCAvqgAwIBAgIBHDANBgkqhkiG9w0BAQsFADCBojELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkNBMREwDwYDVQQHEwhDYW1wYmVsbDEVMBMGA1UEChMMT3JnYW5p
emF0aW9uMR0wGwYDVQQLExRJbmZvcm1hdGlvbiBTZWN1cml0eTEUMBIGA1UEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGGluZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDU4MzJaFw0zNTA4MDIwMDU4MzJaMIGTMQswCQYDVQQG
EwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTERMA8GA1UEBwwIQ2FtcGJlbGwxEDAO
BgNVBAoMB1V0aW1hY28xDzANBgNVBAsMBkF0YWxsYTEYMBYGA1UEAwwPa21zX3Bs
dWdpbl91c2VyMR8wHQYJKoZIhvcNAQkBFhBlc2ttQHV0aW1hY28uY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnh0bomJWHRnK5D7gBEJJDcmjJ3MS
yRTwAnE3Tdq3gpMebyWmArRMVV8DFY60lkUgpei0UvGOB6iqRVkEW8HHwQK3aSoq
XlZAjSye+/MUTuRMdbSIQXDBgdkGdHspQob4fhDEsd3lnOwWWZWxvLHvqouDdfw9
riSj9uY45uqQ05imSsCUNETwRSt1NFCGIDdHcq9vQ9w6ssZ7jdLowFCYCpiGmYot
hUS67qrB3MVSR1PVNvKUAhBOGIa8yljrwrRrNL6j/b5D5SegEh2v4hkVrjmq1eWe
uSIR91SRWBapbBsE8SOED9h/xvzMpn0tU54iKB1rsBqqyRRkNf/TwyE4LQIDAQAB
o2AwXjAJBgNVHRMEAjAAMB0GA1UdDgQWBBSFAfNXm2YP18RGj9ImrlWJDNjo6DAf
BgNVHSMEGDAWgBTMs7TmJGh5EXZreLciXVgokEYJEDARBglghkgBhvhCAQEEBAMC
B4AwDQYJKoZIhvcNAQELBQADggEBAHyjhvYmKq5g5QObqMDXWnXtVMLUCqmehew8
RNjyZZd+zCZsFJS0qZj0m9xq5k1mFtVyBFhy03R0KCEmT3pN16JW6H1r9gzsiZkn
QstbOZ/3amKKg9buU70EeP4q7pfLjiMqQNKqGSfQdpAoBbNlzYvoEMVzMLPFhnET
KjClI02lipBfqGzBDdCNI6iJncozBugPOCA4lkLcOhrMkllcN+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-----
MIIEFDCCAvygAwIBAgIBADANBgkqhkiG9w0BAQsFADCBojELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkNBMREwDwYDVQQHEwhDYW1wYmVsbDEVMBMGA1UEChMMT3JnYW5p
emF0aW9uMR0wGwYDVQQLExRJbmZvcm1hdGlvbiBTZWN1cml0eTEUMBIGA1UEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGGluZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDQ4NDRaFw0zNTA4MDMwMDQ4NDRaMIGiMQswCQYDVQQG
EwJVUzELMAkGA1UECBMCQ0ExETAPBgNVBAcTCENhbXBiZWxsMRUwEwYDVQQKEwxP
cmdhbml6YXRpb24xHTAbBgNVBAsTFEluZm9ybWF0aW9uIFN1Y3VyaXR5MRQwEgYD
VQQDEwtFU0tNTG9jYWxDQTEnMCUGCSqGSIb3DQEJARYYaW5mb3N1Y0Bvcmdhbml6
YXRpb24uY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAySb/YPWh
aQEKFdmXjw1Uk32swOqjk+dBMb9meLOoI2kKQjX3+DzE2uSCX/DDL46dBwe7y30D
IdwmKlBm/xSWgplrmludmmnyJhOj4MiQOSv5PIoymG1aAYNaqB1xb0FFd1xp4wdP
ME4827hiUqZ46zZ72XclDD4ToSU28EW0MaM6vbt128Q/n+PZu2897iOp/cuxwgl1
qufbRAptJXCJxeNLPudhDQlLaXxgKPcZtPaxEV394bVM9MKvOzyztJFgzElAaiDy
9q4xwgswagA8MJmXkHohmUsCnFaQKyesbsObo+sDNWWYDOxsavzE6cxq+pDH5Klk
o3QqN/0Z8Y9PEwIDAQABo1MwUTAdBgNVHQ4EFgQUzLO05iRoeRF2a3i3Il1YKJBG
CRAwHwYDVR0jBBgwFoAUzLO05iRoeRF2a3i3Il1YKJBGCRAwDwYDVR0TAQH/BAUw
AwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAFTOVHxQstxbKQqE6qAy262vFC034tqKP
4UhbezC2r+vD+ZGmxt09z3UvgGnjEBhVkyuZKBk6KkoaaF11YoMbfWZbUoJvG5ee
VikFCZJSfsO2AH5hVRdGWhlPPBZ/0+j0qy10rJOnjbRMo1Pkz+JAmylf0G6cPWed
5EcuWZXUgF4uxz+m2PQWy+4n3KsX7mmDWEWmyWVpJN0pBdQ7PhCAkyS39tWuhrQl
cgnS53HUhpvoz49J1Vd4s2RKflDIUQ3ocV9XbMbM7df2zxbEJ1n+hhggThLL3i64
x83t/00THhjn12UvuSHBtg3PyltH04ZVoHuIRUGIpPOu9MsdtAH/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
CzAJBgNVBAgTAkNBMREwDwYDVQQHEwhDYW1wYmVsbDEVMBMGA1UEChMMT3JnYW5p
emF0aW9uMR0wGwYDVQQLExRJbmZvcmlhdGlvbiBTZWN1cml0eTEUMBIGA1UEAxML
RVNLTUxvY2FsQ0ExJzAlBgkqhkiG9w0BCQEWGGluZm9zZWNAb3JnYW5pemF0aW9u
LmNvbTAeFw0yNTA4MDQwMDQ4NDRaFw0zNTA4MDMwMDQ4NDRaMIGiMQswCQYDVQQG
EwJVUzELMAkGA1UECBMCQ0ExETAPBgNVBAcTCENhbXBiZWxsMRUwEwYDVQQKEwxP
cmdhbml6YXRpb24xHTAbBgNVBAsTFEluZm9ybWF0aW9uIFNlY3VyaXR5MRQwEgYD
VQQDEwtFU0tNTG9jYWxDQTEnMCUGCSqGSIb3DQEJARYYaW5mb3NlY0Bvcmdhbml6
YXRpb24uY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAySb/YPWh
aQEKFdmXjwlUk32swOqjk+dBMb9meLOoI2kKQjX3+DzE2uSCX/DDL46dBwe7y30D
IdwmKlBm/xSWgplrmludmmnyJhOj4MiQOSv5PIoymG1aAYNaqB1xb0FFd1xp4wdP
ME4827hiUqZ46zZ72XclDD4ToSU28EW0MaM6vbt128Q/n+PZu2897iOp/cuxwgl1
qufbRAptJXCJxeNLPudhDQlLaXxgKPcZtPaxEV394bVM9MKvOzyztJFgzElAaiDy
9q4xwgswagA8MJmXkHohmUsCnFaQKyesbsObo+sDNWWYDOxsavzE6cxq+pDH5Klk
o3QqN/0Z8Y9PEwIDAQABo1MwUTAdBgNVHQ4EFgQUzLO05iRoeRF2a3i3Il1YKJBG
CRAwHwYDVR0jBBgwFoAUzLO05iRoeRF2a3i3Il1YKJBGCRAwDwYDVR0TAQH/BAUw
AwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAFTOVHxQstxbKQqE6qAy262vFC034tqKP
4UhbezC2r+vD+ZGmxt09z3UvgGnjEBhVkyuZKBk6KkoaaFl1YoMbfWZbUoJvG5ee
VikFCZJSfsO2AH5hVRdGWhlPPBZ/0+j0qy10rJOnjbRMo1Pkz+JAmylf0G6cPWed
5EcuWZXUgF4uxz+m2PQWy+4n3KsX7mmDWEWmyWVpJN0pBdQ7PhCAkyS39tWuhrQ1
cgnS53HUhpvoz49J1Vd4s2RKflDIUQ3ocV9XbMbM7df2zxbEJ1n+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.

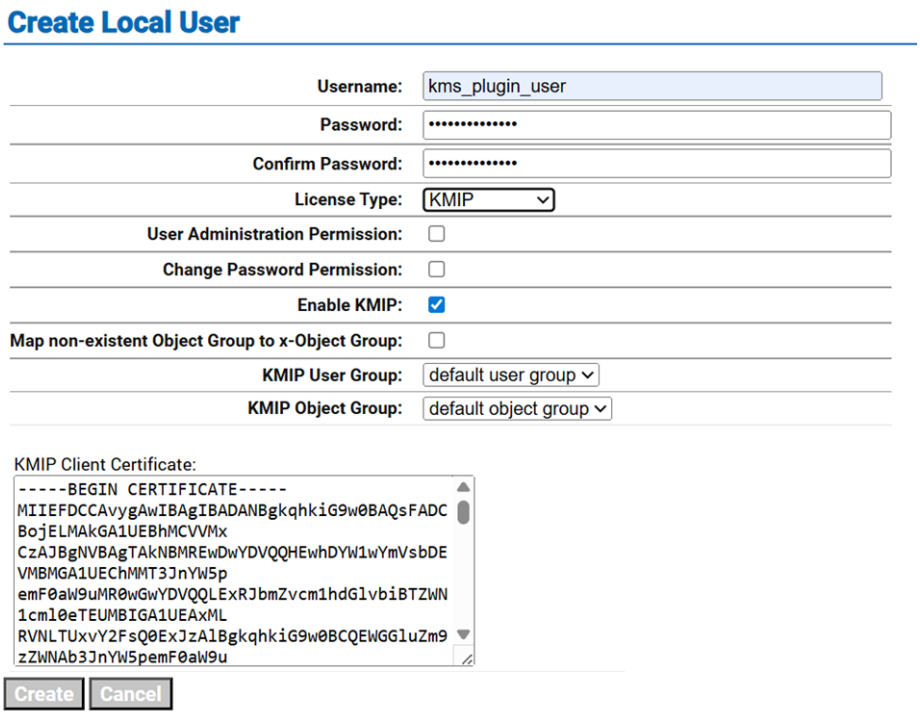

Figure 33 : Create Local User

5.2.6 Create KMIP Object

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

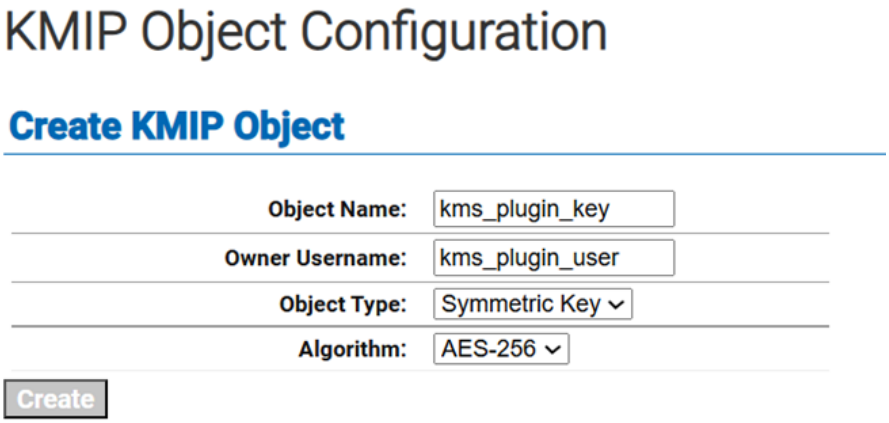

Figure 34 : Create KMIP Object

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.

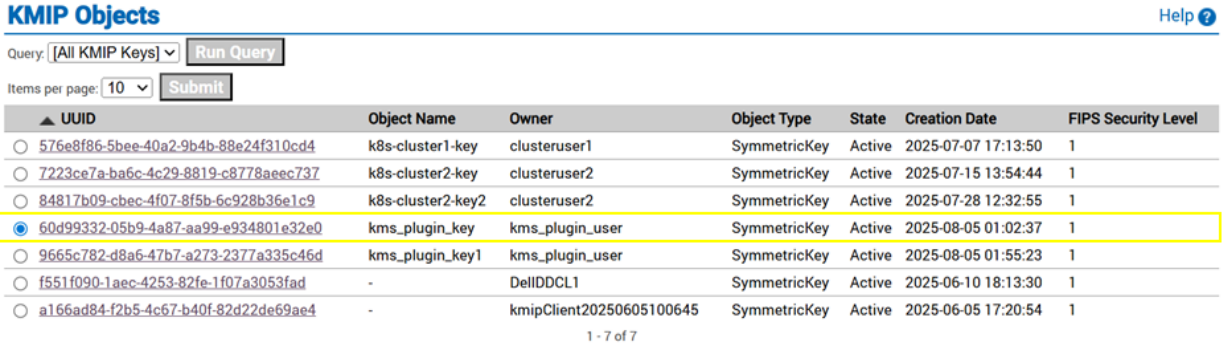

Figure 35 : KMIP Objects

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:

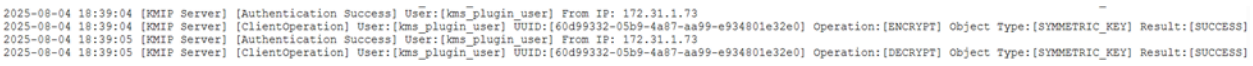

Figure 36 : Verify KMIP Log

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.

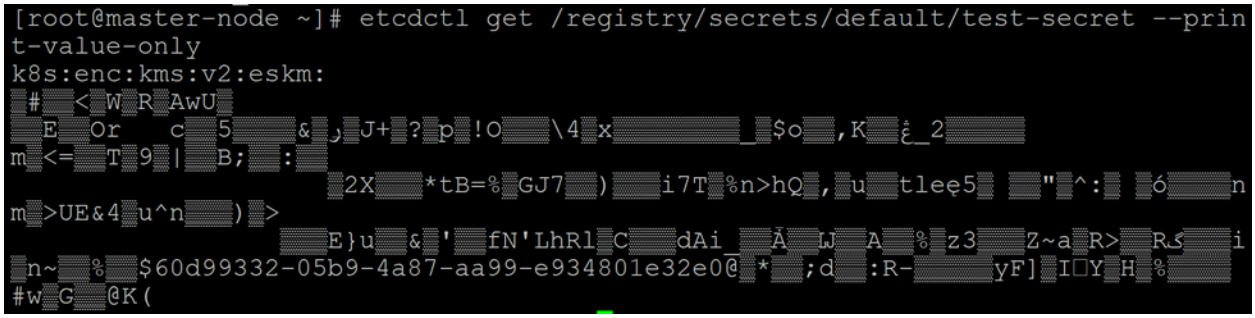

Figure 38 : Verify Encryption

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.

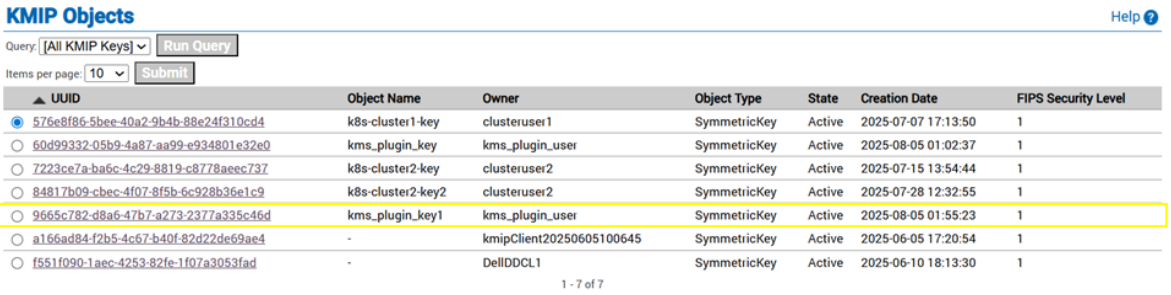

Figure 40 : Create a KMIP Object

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_uuid` 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 k
ube-system
configmap "kms-plugin-config" deleted
[root@master-node k8s_plugin]# kubectl create configmap kms-plugin-config --fr
om-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.

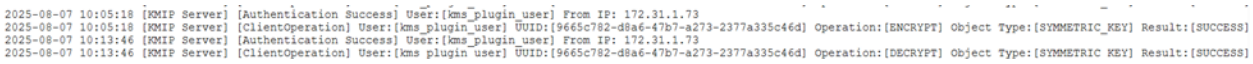

Figure 44 : ESKM KMIP Logs

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.

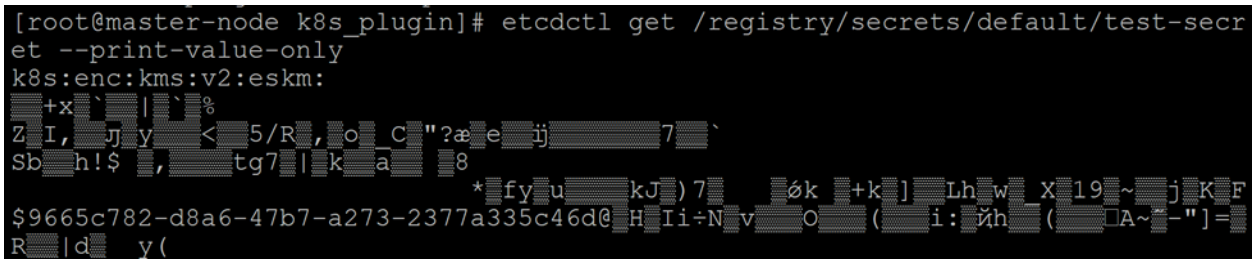

Figure 45 : Verify Key Rotation Success

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.

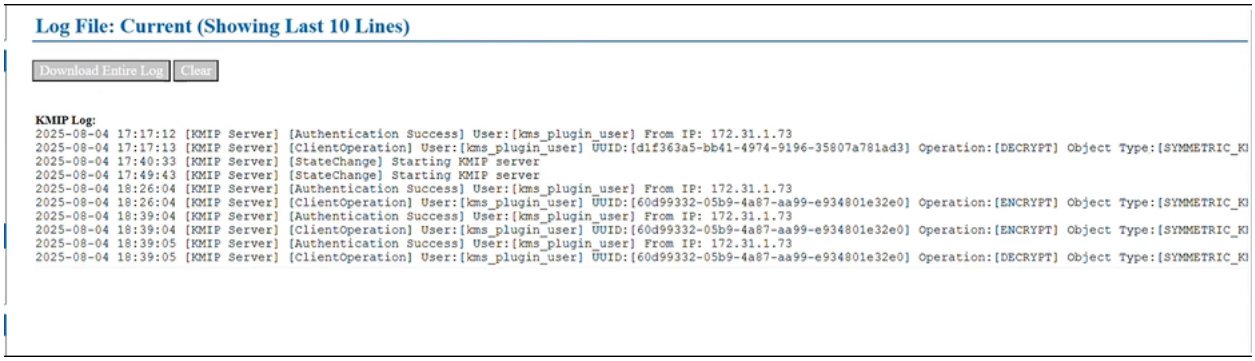

Figure 46 : KMIP Logs

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
`# mkdir /var/lib/kmsplugin`	To create a directory for socket files.
`# mkdir k8s_plugin`	To create a directory for plugin files.
`# mkdir certs`	To create a dircectory for certificates.
`# mkdir logs`	To create a directory for plugin log file.

Command	Purpose
`# docker load -i k8s-kms-plugin_1.0.tar`	To load a Docker image.
`# openssl genrsa -out kms_plugin_client.key 2048`	To create a private key.

`# 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`
	To generate a CSR.

`# cat /home/admin/k8s_plugin/certs/kms_plugin_client.csr`	To view the generated CSR.
`# vim plugin-config.yaml`	To create/edit the plugin-config.yaml file
`# kubectl create configmap kms-plugin-config --from-file=/home/admin/k8s_plugin/plugin-config.yaml`	To create the kube configmap for plugin-config.yaml.

Command	Purpose
`# 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`	To create the kube secret for certificates.
`# vim kms-plugin.yaml`	To create/edit the kms-plugin.yaml daemonset file.
`# kubectl apply -f kms_plugin.yaml`	To apply the kms-plugin.yaml file.
`# kubectl get daemonset -A`	To view the kube system daemonset.
`# kubectl get pods -A`	To view the kube system pods.
`# vim /etc/kubernetes/encryption-config.yaml`	To create/edit the encryption-config.yaml file.
`# vin /etc/kubernetes/manifests/kube-apiserver.yaml`	To edit the kube-apiserver.yaml file.
`# kubectl create secret generic test-secret --from-literal=mykey=mydata`	To create a secret.
`# etcdctl get /registry/secrets/default/test-secret --print-value-only`	To verify encryption in etcd.

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

Table 4: CLI Commands