GUIDE

STORMSHIELD DATA SECURITY FOR KUBERNETES INIT CONTAINER

ADMINISTRATION GUIDE

Version 1.0.1

Document last updated: August 14, 2025

Reference: sds-en-kubernetes-init_container_guide-v1.0.1

Table of contents

1. Getting started

SDS for Kubernetes init container allows Kubernetes administrators to manage Kubernetes secrets in their clusters using state-of-the-art security practices.

Without SDS for Kubernetes init container, a Kubernetes secret is stored unencrypted in the API server's data store (etcd), and can be retrieved by anyone with API access.

With SDS for Kubernetes init container, sensitive data stored in a Kubernetes secret is less exposed as it is only loaded into memory during the pod lifetime.

This solution is based on three technologies:

- **SDS Key Management as a Service (KMaaS)** with the *crypto-api* feature enabled
  Provides an API for encrypting/decrypting sensitive data using Data Encryption Keys (DEKs).
- **Stormshield Software Development Kit** (SDK)
  Provides higher-level and standardized methods to encrypt/decrypt:
  - SDSDK manages DEKs and calls the KMaaS so that the end user does not need to perform these actions,
  - Encrypted sensitive data is stored with metadata using a standardized format.
- **Kubernetes init container**
  An init container is a temporary container that runs before the main application container to perform various preliminary tasks. The SDS for Kubernetes init container solution uses init containers to:
  - Decrypt data using the SDSDK, which calls the KMaaS,
  - Save the decrypted data into an ephemeral Kubernetes volume in RAM, shared with the main application container. This ensures that the decrypted data is no longer accessible when the main application container is terminated.

SDS for Kubernetes init container provides a script, *encryptor.cjs*, to encrypt sensitive data.

NOTE
The use of the solution in any way other than as described in the documentation is not managed. Alternatively, get in touch with Stormshield Support for clarification.

2. Understanding the requirements

2.1 Security requirements

2.1.1 Environment

- The server on which SDS for Kubernetes init container is installed must be healthy. There must be an information system security policy whose requirements are met on the server. This policy shall ensure that the installed software is regularly updated and the system is protected against viruses and spyware or malware (firewall properly configured, antivirus updates, etc.). It is imperative to follow:
  - The operating system security recommendations issued by the ANSSI in their document *ANSSI-BP-028-EN*.
  - The security recommendations for docker container deployments issued by the ANSSI in their document *ANSSI-FT-082* (in French only), and OWASP.
- Access to the administrative functions of the workstation system is restricted only to system administrators.
- The operating system must manage the logs generated by the product in accordance with the security policy of the company. It must for example restrict read access to these logs to only those explicitly permitted. For more information, refer to the section Logging prerequisites.
- You must set up a system upstream of SDS for Kubernetes init container to protect against distributed denial-of-service (DDoS) and brute-force attacks. Please follow the ANSSI recommendations (in French only).
- SDS for Kubernetes init container must be installed on a server whose system and OpenSource contributions are kept up to date.
- The server hosting the solution must be located in a secure physical environment with access control protocols and must be trusted.

2.1.2 Administration

- SDS for Kubernetes init container administrators are considered as trusted. They are responsible for defining SDS for Kubernetes init container security policy in compliance with the state-of-the-art standards.
- The system administrator is also considered as trusted. He/She is responsible for the installation and maintenance of the application and server. He/She applies the security policy defined by SDS for Kubernetes init container administrators.

2.1.3 Kubernetes

The KMaaS API key is a very sensitive security material. Do not store it in clear text, and use it only via environment variables and Kubernetes secrets.

In the pod manifest, you must specify a volume of type *emptyDir: { medium: Memory }* to store the decrypted data so that it can be accessed only during the pod lifetime.

2.1.4 KMaaS

Stormshield recommends having a dedicated tenant for SDS for Kubernetes init container to allow the API key to be revoked without interfering with other applications. We also recommend using only one API key per init container.

2.2 Infrastructure requirements

The SDS for Kubernetes init container requires:

- A Kubernetes cluster version v1.10 or higher, able to support both the init container and the ephemeral volume features,
- An image registry to host the SDS for Kubernetes init container docker images to be used by the Kubernetes cluster,
- An available KMaaS instance version 4.5 or higher. The instance requires:
  - A base KMaaS URL,
  - A tenant identifier with enabled crypto-api feature,
  - An API key for authentication.

  For more information, refer to section Configuring the KMaaS and the KMaaS Guide.
- Node.js 22 or higher to run the preliminary *encryptor.cjs* script that encrypts the sensitive data.
- The following network streams must be open:

Description	Protocol	Source	Base URL	Port	Route
KMaaS for init container	The pod protected by SDS for Kubernetes init container must be able to reach the KMaaS decrypt route.	Kubernetes Pod to be protected	<KMAAS_URL>	<KMAAS_PORT>	api/v1/<tenant_id>/crypto/decrypt
KMaaS for encryptor.cjs	The *encryptor.cjs* script must be able to reach the KMaaS encrypt route.	Encryptor script host	<KMAAS_URL>	<KMAAS_PORT>	api/v1/<tenant_id>/crypto/encrypt

Contact your KMaaS administrator to get the KMaaS URL and port.

3. Preparing the SDS for Kubernetes init container environment

The SDS for Kubernetes init container is provided as a compressed archive containing:

- The Docker image for the init container: *sds-kubernetes-init-container-<version>.tar.gz*
- The *encryptor.cjs* script to encrypt the sensitive data with the SDSDK. It must be run before the pod manifest is applied.
- A Kubernetes pod manifest example (*pod.yaml*) to deploy the "sdskub" pod using the SDS for Kubernetes init container and the test application container.

3.1 Loading the Docker images in Kubernetes

You can either deploy Docker image to a Docker registry or to a local registry.

Load the init container image provided in the *sds-kubernetes-init-container-<version>.tar* archive, and push them into an image registry.

1. To load the image, with Docker, run the following command:
```
$ docker load -i sds-kubernetes-init-container-<version>.tar.gz
```
2. To tag the image and push it in a registry, run the following commands:
```
$ docker tag stormshield/sdskub-init-container:<VERSION> <IMAGE_
REGISTRY_URL>/<IMAGE_TAG>:<VERSION>
$ docker push <IMAGE_REGISTRY_URL>/<IMAGE_TAG>:<VERSION>
```

For more information, refer to the Kubernetes documentation.

3.2 Configuring the KMaaS

The **SDS Key Management as a Service (KMaaS)** provides the API for encrypting/decrypting sensitive data using Data Encryption Keys (DEKs).

You must configure the KMaaS *config.json* file as follows:

- Enable the *crypto-api* feature,
- Specify at least one API key in the authentication method of the *crypto-api*.

For more information, refer to the KMaaS guide.

3.3 Setting the KMaaS parameters in the environment

On the host where you will encrypt the sensitive data, set the KMaaS parameters as environment variables as described in the table below.

The KMAAS_API_KEY value is the following string encoded in base64:

```
_CRYPTO_API_AUTHENTICATION_API_KEY_NAME:_CRYPTO_API_AUTHENTICATION_API_KEY_VALUE
```

You can find it in the KMaaS *config.json* file.

On Unix environments, you can encode the KMaaS API key with this command:

```
$ echo -n "_CRYPTO_API_AUTHENTICATION_API_KEY_NAME:_CRYPTO_API_
AUTHENTICATION_API_KEY_VALUE" | base64
```

Environment variable	Description
KMAAS_URL	URL of the KMaaS.
KMAAS_TENANT_ID	ID of the tenant declared in the *tenant_id* parameter of the KMaaS *config.json* file.
KMAAS_API_KEY	Base64 string of the concanetated values `_CRYPTO_API_AUTHENTICATION_API_KEY_NAME:_CRYPTO_API_AUTHENTICATION_API_KEY_VALUE`of the KMaaS *config.json* file.

```
$ export KMAAS_URL=https://host.kmaas:443
$ export KMAAS_TENANT_ID=025f02fe-bee2-231c-bf76-b5ead30327c0
$ export KMAAS_API_
KEY=NCH2aG9yaXplZEFwaUtleTpvY2dZWENFSzY2UHJUSTYxTnkzSmtBRkxyM0JaL0x4Vw==
```

4. Encrypting and decrypting sensitive data

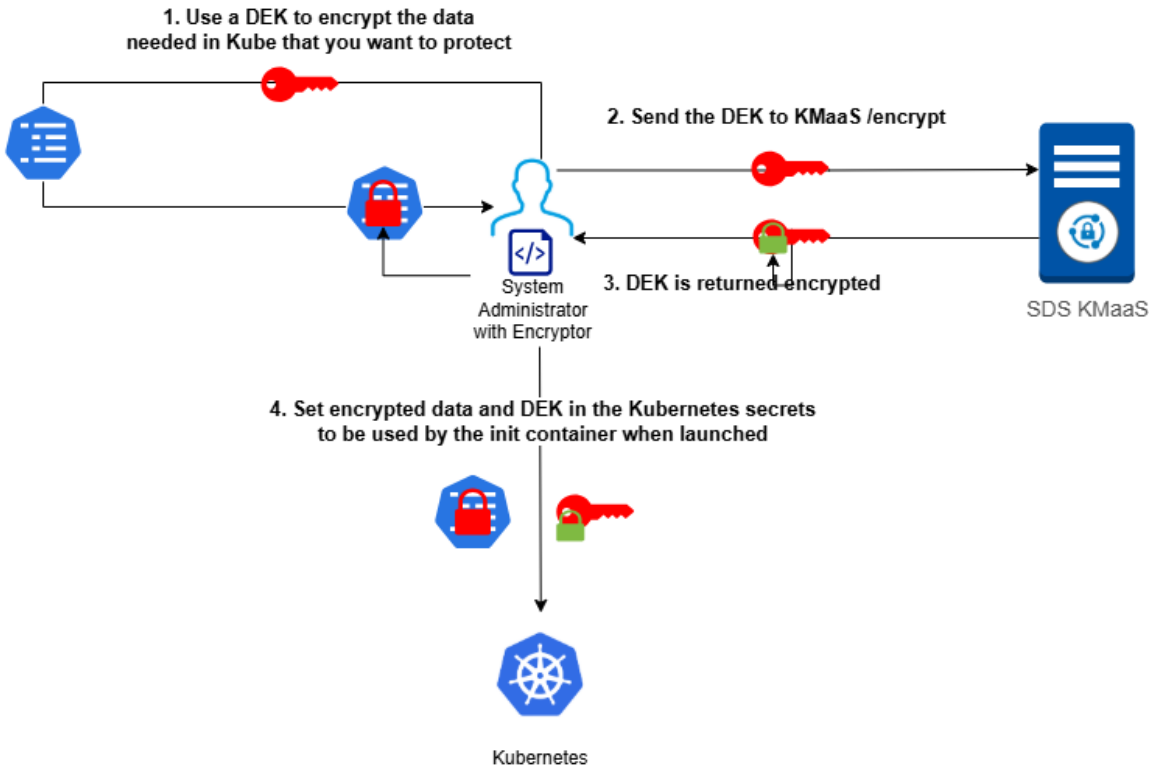
The table below describes the steps of the encrypt and decrypt procedure.

Click on a link to open the corresponding procedure in this guide.

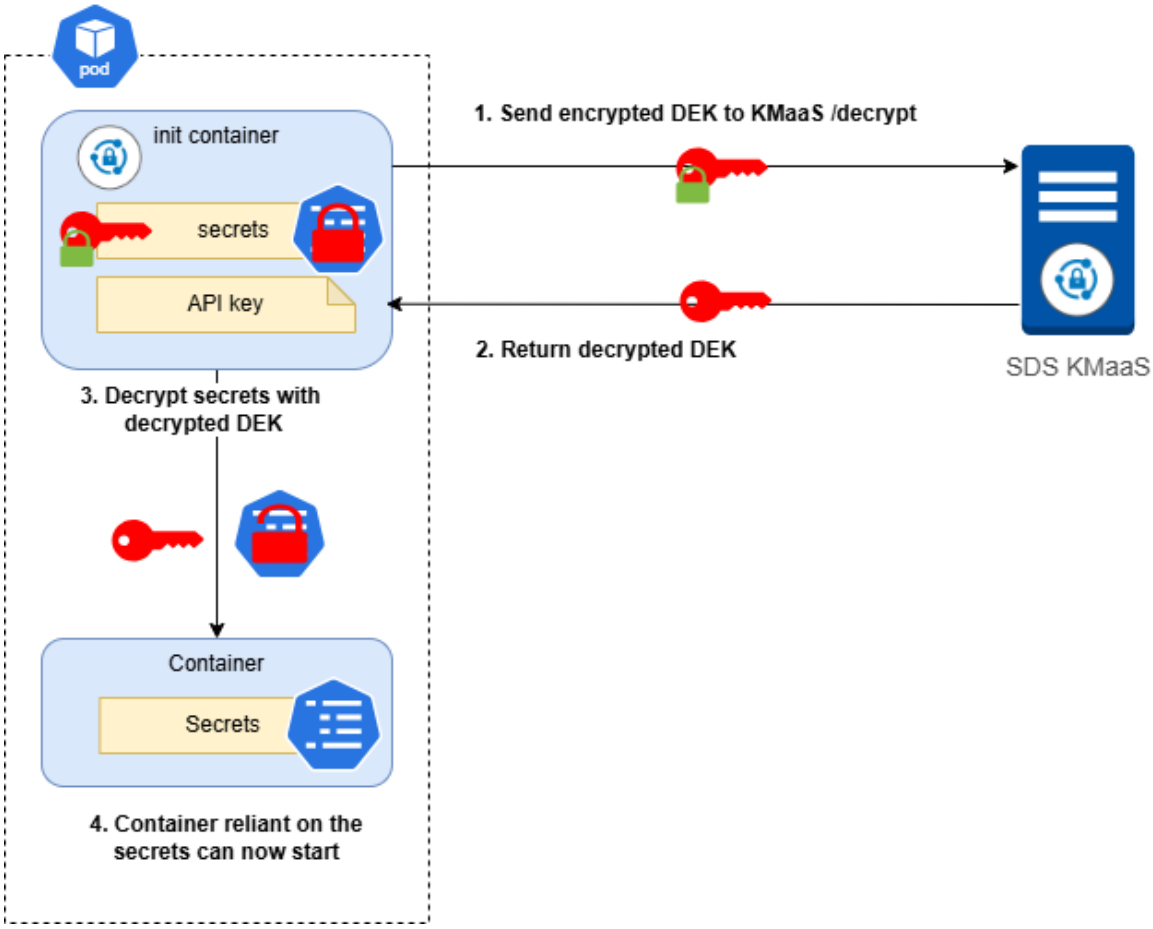
Steps	Description
1	Encrypt the sensitive data using the *encryptor.cjs* script. The result is the *Encrypted sensitive data*, a base64-encoded string containing the encrypted data, the encrypted DEK, and some metadata.
2	Create the Kubernetes secrets for securely storing the KMaaS API key and the *Encrypted sensitive data*.
3	Configure the *pod.yaml* Kubernetes manifest by setting the KMaaS parameters.
4	Decrypt the sensitive data in SDS for Kubernetes init container by deploying the sdskub pod.

4.1 Understanding encrypt and decrypt operations

4.1.1 Encrypt workflow



4.1.2 Decrypt workflow



4.2 Encrypting sensitive data

The *encryptor.cjs* script encrypts your sensitive data before you set it as a Kubernetes secret. We recommend that you perform this operation in a secure and trusted environment, as you will handle data in clear text. In addition, be sure to clean up your command history.

1. Run the *encryptor.cjs* script provided in the *sds-kubernetes-init-container-<version>.tar* archive.
   ```
   $ node encryptor.cjs "sensitive data"
   ```
   The data to be encrypted must not exceed 10 KB.
   The command result is similar to:
   ```
   $ [INFO] {"encrypted_sensitive_data":
   'BKJhY6RhYVBHD4jEAjHoP6FCVF7Ejnifk7prOWqJu88dibUOGuU+8Rai6DEl67Wdc
   zs9KncrRQAYj2qqwjFNvdicASV4='}
   ```
2. Copy the value of the encrypted sensitive data to use it in next procedure.

4.3 Creating the Kubernetes secrets

In Kubernetes, you must create two secrets for the sensitive data that you have encrypted:

- A secret containing the KMaaS API key that will be used to authenticate to the KMaaS during decryption,
- A secret containing the encrypted sensitive data.

1. Create a secret from the base64 API key string set in section Configuring the KMaaS. It allows Kubernetes to connect to the KMaaS to decrypt the sensitive data.
   For example, you can use the following command:

```
$ kubectl create secret generic sdskub-kmaas --from-literal=api-key="NCH2aG9yaXplZEFwaUtleTpvY2dZWENFSzY2UHJUSTYxTnkzSmtBRkxyM0JaL0x4Vw=="
```

2. Create a secret from the encrypted sensitive data obtained in section Encrypting sensitive data.
   For example, you can use the following command:

```
$ kubectl create secret generic sdskub-encrypted-data --from-literal=encrypted-sensitive-data="BKJhY6RhYVBHD4jEAjHoP6FCVF7Ejnifk7prOWqJu88dibUOGuU+8Rai6DEl67Wdczs9KncrRQAYj2qqwjFNvdicASV4="
```

In the above examples, two secrets are created:

- *sdskub-kmaas* is the secret containing the KMaaS API key,
- *sdskub-encrypted-data* is the secret containing the encrypted sensitive data.

4.4 Configuring the Kubernetes manifest

You must configure the pod manifest to indicate that such application will be able to use such and such secrets via the init-container.

1. From the *sds-kubernetes-init-container-<version>.tar* archive file, retrieve the sample *pod.yaml* manifest file.
2. In the file, replace the *name* and *key* placeholders values of the following parameters:

Parameter	Description
KMAAS_API_KEY	Name and key of the secret containing the KMaaS API key (*sdskub-kmaas* and *api-key* in our example).
ENCRYPTED_SENSITIVE_DATA	Name and key of the secret containing the encrypted sensitive data obtained with the *encryptor.cjs* script (*sdskub-encrypted-data* and *encrypted-sensitive-data* in our example).
OUTPUT_SENSITIVE_DATA_PATH	Path used by the init container to write the decrypted sensitive data. It must be the path of a volume shared between the init container and the main container (*/mnt/shared/sensitive-data* in our example).

```
apiVersion: v1
kind: Pod
metadata:
  name: sdskub
spec:
  restartPolicy: Never
  initContainers:
    - name: sdskub-init-container
      image: IMAGE_TAG:VERSION
      env:
           - name: NODE_EXTRA_CA_CERTS
             value: /etc/ssl/certs/kmaas_ca.crt
           - name: KMAAS_API_KEY
             valueFrom:
               secretKeyRef:
                 name: sdskub-kmaas
                 key: api-key
```

```
            - name: ENCRYPTED_SENSITIVE_DATA
              valueFrom:
                secretKeyRef:
                  name: sdskub-encrypted-data
                  key: encrypted-sensitive-data
            - name: OUTPUT_SENSITIVE_DATA_PATH
              value: /mnt/shared/sensitive-data
        volumeMounts:
            - name: sdskub-shared
              mountPath: /mnt/shared
            - name: trusted-ca-store-volume
              mountPath: /etc/ssl/certs/kmaas_ca.crt
              subPath: kmaas_ca.crt
              readOnly: false
  containers: # It is an example container, you have to replace it by your
own image
    - name: sdskub-main-container
      image: busybox:1.28
      command: ["cat", "/mnt/shared/sensitive-data"]
      volumeMounts:
       - name: sdskub-shared
         mountPath: /mnt/shared
  volumes:
    - name: sdskub-shared
      emptyDir: { medium: Memory }
       - name: trusted-ca-store-volume
         secret: {secretName: trusted-ca-store}
```

The environment variable *NODE_EXTRA_CA_CERTS* used in this example contains the path to the KMAAS CA file that is mounted as a volume. This allows a TLS connection to be opened between the sdskub-init-container and the KMAAS.
In our example, the command to create the secret would be:
`kubectl create secret generic trusted-ca-store --from-file=<PATH_TO_KMAAS_CA>`

This is only useful if your KMaas CA certificate is not trusted by default.

4.5 Decrypting sensitive data

To decrypt the sensitive data, you must deploy the *sdskub* pod.

- Run the following command to apply the manifest and deploy the pod:

```
$ kubectl apply -f pod.yaml
```

The sensitive data is decrypted and stored in the shared volume specified by the *OUTPUT_SENSITIVE_DATA_PATH* parameter. If the decryption is successful, the logs are similar to:

```
$ kubectl logs sdskub -c sdskub-init-container

$ { "timestamp": "2025-05-21T08:54:17.262Z", "severity": "info",
"application_version": "1.0.0", "kind": "domain", "category": "secrets",
"action": "decrypt", "secret_path": "/mnt/shared/sensitive-data", "url":
"https://kmaas-host:3000/api/v1/025f02fe-bee2-231c-bf76-
b5ead30327c0/crypto/decrypt"}
```

In the above example, the decrypted sensitive data is stored in */mnt/shared/sensitive-data*.

For more information about logs, refer to section Managing logs

5. Managing logs

SDS for Kubernetes init container generates logs for every operation, making it possible to trace all operations performed and potential issues. It is a technical activity essential to the security of information systems.

5.1 Logging prerequisites

To meet the logging requirements for SDS for Kubernetes init container, you must:

- Follow the security recommendations for logging systems issued by the ANSSI in their document *ANSSI-PA-012* (French only),
- Follow the security recommendations for logging systems issued by the ANSSI in their document *ANSSI-FT-082* (French only).

5.2 Accessing logs

- Run the Kubernetes command to access SDS for Kubernetes init container logs:

```
kubectl logs <POD_NAME> -c <INIT_CONTAINER_NAME>
```

5.3 Generic log fields

The following fields are displayed for all logs generated by SDS for Kubernetes init container in the order shown in the table.

Field	Description	Type	Mandatory/Optional
timestamp	Date and time at which the log was created. In UTC format. Example: "2025-05-21T08:54:17.262"	String in ISO 8601 format	Mandatory
severity	Level of severity of the log. Prescribed values: • *info*: Normal operation information message that requires no action, • *err*: Action failed.	String	Mandatory
application_version	Application version. Example: "1.0.0"		Mandatory
kind	Log family to which the log belongs. Prescribed value: • *domain*: SDS for Kubernetes init container business operation logs.	String	Mandatory
category	Log category. Prescribed values: • *secrets*: Logs related to the sensitive data encrypted by SDS for Kubernetes init container.	String	Mandatory
action	Event that occurred. Prescribed value: • decrypt	String	Mandatory

The fields in the *error* block described below are displayed for all logs generated by the SDS for Kubernetes init container in the event of an error when executing the action:

Field	Description	Type	Mandatory/Optional
error.code	Error identifier.	String	Mandatory
error.message	Error message.	String	Mandatory

5.4 Domain - Business operation logs

The log fields described below relate to business operations performed by SDS for Kubernetes init container. They belong to the *Domain* log family (Kind:domain).

5.4.1 *secrets* category

This category of logs contains all the operations related to the decryption of encrypted sensitive data.

decrypt action

- The *decrypt* action means that a *decrypt* request has been made. This is the case whenever sensitive data is decrypted using SDS for Kubernetes init container.

This action generates an "info" severity log in the event of success, or an "err" severity log in the event of an error.

The log fields for this action are as follows:

Field	Description	Type	Required/Optional
secret_path	Path to the decrypted sensitive data. Example: "/mnt/shared/sensitive-data"	String	Mandatory
url	URL of the KMaaS used for decryption Example: "https://host-kmaas:443"	String	Mandatory

6. Further reading

Additional information and answers to questions you may have about SDS for Kubernetes init container are available on the Documentation website and in the Stormshield knowledge base (authentication required).

documentation@stormshield.eu