KeySafe 5

Keysafe 5 v1.2 Installation and Upgrade Guide

8 April 2024

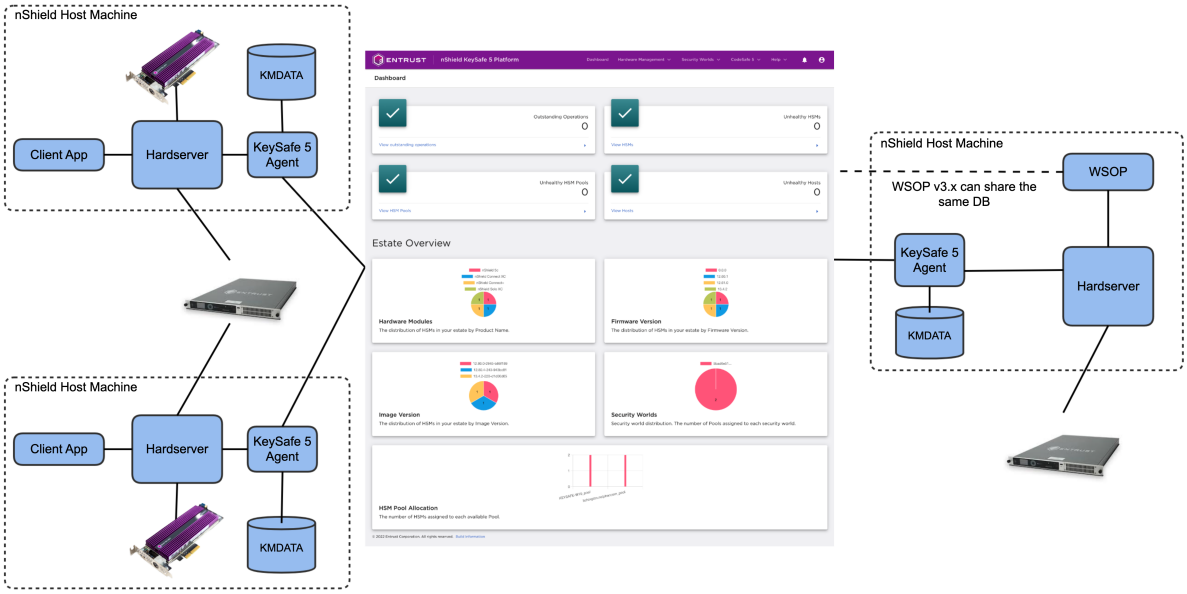
Table of Contents

1. Introduction

KeySafe 5 provides a centralized means to securely manage a distributed nShield HSM estate, including the creation and management of Security Worlds and associated resources (Softcards & Card Sets).

KeySafe 5 provides this capability in two forms: HTTP REST APIs for HSM Management and Security World management, and a graphical user interface. Only authenticated clients are permitted access to the service, providing assurance that your HSM and Security World data remain usable only by clients that are permitted access.

Typical KeySafe 5 deployment:



The main central management platform of KeySafe 5 is deployed as either a Kubernetes application or as a cluster of Virtual Machines (VMs).

For each nShield client machine that you want to manage using this platform, you must install a KeySafe 5 agent binary alongside the existing nShield hardserver. A KeySafe 5 agent is installed on the nShield Connect for nShield Connect images released with Security World v13.4 and later software.

2. Release Package

The release package is provided in `.tar.gz` format and has the following contents.

2.1. OpenAPI specifications

The API specification documents for the RESTful web services follow v3.0 of the OpenAPI specification.

- `api/codesafe-mgmt.yml` defines the CodeSafe Management API
- `api/hsm-mgmt.yml` defines the HSM Management API
- `api/sw-mgmt.yml` defines the Security World Management API

2.2. Helm charts

The KeySafe 5 Kubernetes-based deployment consists of 3 Helm charts:

- `helm-charts/nshield-keysafe5-backend-1.2.0.tgz`

  This installs the backend API services (HSM Management and Security World Management).

- `helm-charts/nshield-keysafe5-ui-1.2.0.tgz`

  This installs the graphical user interface for KeySafe 5.

- `helm-charts/nshield-keysafe5-istio-1.2.0.tgz`

  This configures an existing Istio Ingress Gateway to allow external access (routing and authentication) to the services deployed by the other two Helm charts.

This split enables you to deploy the backend services only, if you do not need the UI, or the UI only, if you want to point it at some existing backend services already running elsewhere.

You can also use a different Kubernetes Ingress other than Istio if desired.

For more information on configuring and installing the Helm chart, see Helm Chart Installation .

2.3. Docker images

The Docker images are provided as tar archives. You can load them into a local Docker

image registry using the `docker load` command, then push to a private container registry.

For example:

```
$ docker load < docker-images/hsm-mgmt.tar
Loaded image: hsm-mgmt:1.2.0
$ docker tag hsm-mgmt:1.2.0 private.registry.local/keysafe5/hsm-mgmt:1.2.0
$ docker login private.registry.local
$ docker push private.registry.local/keysafe5/hsm-mgmt:1.2.0
```

The Docker images provided are:

- `docker-images/codesafe-mgmt.tar` is the CodeSafe Management service
- `docker-images/hsm-mgmt.tar` is the HSM Management service
- `docker-images/sw-mgmt.tar` is the Security World Management service
- `docker-images/ui.tar` is the KeySafe 5 user interface

These Docker images are intended to be deployed via the provided Helm charts. See the Helm chart configuration for details of how to configure and run each image.

2.4. KeySafe 5 agent installers

You can use the Linux and Windows installers provided to install the KeySafe 5 agent on nShield client machines. See KeySafe 5 Agent Installation for details on configuring and installing the agent.

3. Prerequisites

The following table contains the required software version that this release of KeySafe 5 has been tested on and any minimum version requirements.

Software	Minimum version	Tested version
Kubernetes	1.22	1.27
MongoDB	4.4	5.0.19
RabbitMQ	3.0	3.11.19

In addition to the set of required software, this release of KeySafe 5 requires:

- A location for storing large objects (for example CodeSafe machines)
- An external identity provider that supports OIDC and OAuth2 for user and machine authentication.

3.1. Optional Software

KeySafe 5 is shipped with a Helm chart that configures an Istio Ingress Gateway to provide external access to the KeySafe 5 application running in Kubernetes. Other Ingress Gateways can be used, see Configure a custom ingress provider.

Software	Minimum version	Tested version
Istio	1.13	1.18

3.2. Hardware Requirements

Entrust recommends the following hardware specification to ensure smooth running of KeySafe 5.

- CPU: 2 minimum (4 recommended)
- RAM: 8GB minimum
- Disk Storage: 64GB minimum
    - For optimal performance Entrust recommends use of an SSD.

Requirements will vary depending on the size of the nShield estate being managed and if services, such as MongoDB and RabbitMQ, are being hosted on the same machine as the Kubernetes cluster or exter-

> nally.

3.3. Kubernetes cluster

KeySafe 5 has been tested on Kubernetes version 1.27.

3.3.1. Using namespaces

When deploying an application to a Kubernetes cluster that is shared with many users spread across multiple teams, Entrust recommends using Namespaces to isolate groups of resources.

To create a namespace for the KeySafe 5 application:

```
$ kubectl create namespace nshieldkeysafe5
namespace/nshieldkeysafe5 created
```

To set the namespace for a current request, use the `--namespace` flag. For example:

```
$ helm install --namespace=nshieldkeysafe5 my-release helm-keysafe5-backend/
$ kubectl --namespace=nshieldkeysafe5 get pods
```

If you are using Istio in your Kubernetes cluster, beyond acting as an API Gateway for KeySafe 5, you might also want to configure Istio injection for this Kubernetes namespace to take advantage of other Istio features. This step is not required for KeySafe 5 to function.

```
$ kubectl label namespace nshieldkeysafe5 istio-injection=enabled
namespace/nshieldkeysafe5 labeled
```

3.4. MongoDB

MongoDB is the persistent data store for the KeySafe 5 application data. Any sensitive Security World data stored in the database is stored in standard nShield encrypted blobs. You should restrict access to this database in the same way that you would normally restrict access to the `Key Management Data` directory on an nShield client machine.

> The MongoDB used must be a Replica Set.

If you have an existing MongoDB database you can configure the application to use this, otherwise you must securely deploy a MongoDB instance.

Entrust recommend that you configure MongoDB with authentication enabled and TLS enabled and with RBAC configured. The database user for KeySafe 5 should be given the minimum capabilities required, see Database: User Roles.

3.4.1. Bitnami MongoDB Helm chart

To install MongoDB in the same Kubernetes cluster as the KeySafe 5 application, you can install the Bitnami MongoDB Helm chart.

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm install mongo-chart \
 --set image.tag=5.0.19-debian-11-r3 \
 --set architecture=replicaset \
 --set auth.enabled=true \
 --set tls.enabled=true \
 --set tls.existingSecret=my-mongo-tls-secret \
bitnami/mongodb --version 12.1.31
```

KeySafe 5 has been tested with version 12.1.31 of the Bitnami MongoDB Helm Chart.

MongoDB 5.0 requires use of the AVX instruction set on processors. For more information, see MongoDB Production Notes

Monitor MongoDB for security vulnerabilities and regularly update the version of MongoDB installed to apply any required updates.

3.5. RabbitMQ

You need a RabbitMQ message bus for the interprocess communication between the KeySafe 5 backend services and KeySafe 5 agent instances running on nShield client machines.

If you have an existing RabbitMQ instance you can configure the application to use this, oth erwise you must securely deploy a RabbitMQ instance.

The RabbitMQ instance should have a sufficient maximum channel count to support the number of resources managed by KeySafe 5. Use the following table to estimate the num- ber of channels required for each resource:

Resource	Required channels
Replica count in KeySafe 5 backend Helm Chart (default value 3)	8

Resource	Required channels
KeySafe 5 Agent	6
Managed HSM	2

Entrust recommend that you configure RabbitMQ with authentication enabled and TLS enabled. If the RabbitMQ instance being used is shared with other applications, then a dedicated virtual host should be created for the KeySafe 5 application. RabbitMQ users created for KeySafe 5 should only have access to that specific virtual host.

3.5.1. Bitnami RabbitMQ Helm chart

To install RabbitMQ in the same Kubernetes cluster as the KeySafe 5 application, you can install the Bitnami RabbitMQ Helm chart.

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm install rabbit-chart \
 --set image.tag=3.11.19-debian-11-r18 \
 --set auth.username=user \
 --set auth.existingPasswordSecret=my-password-secret \
 --set auth.tls.enabled=true \
 --set auth.tls.existingSecret=my-rabbitmq-tls-secret \
 --set extraConfiguration='
   channel_max = 4095
   listeners.ssl.default = 5671
   ssl_options.versions.1 = tlsv1.3
   ssl_options.depth=0
   ssl_options.verify = verify_peer
   ssl_options.fail_if_no_peer_cert = true
   auth_mechanisms.1 = EXTERNAL
   ssl_cert_login_from = subject_alternative_name
   ssl_cert_login_san_type = dns
   ssl_cert_login_san_index = 0' \
 --set plugins="" \
 --set extraPlugins="rabbitmq_auth_mechanism_ssl" \
bitnami/rabbitmq --version 11.16.2
```

KeySafe 5 has been tested with version 11.16.2 of the Bitnami RabbitMQ Helm Chart.

Monitor RabbitMQ for security vulnerabilities and regularly update the version of RabbitMQ installed to apply any required updates.

3.5.1.1. RabbitMQ TLS authentication

If using TLS authentication between the KeySafe 5 agent and RabbitMQ, the RabbitMQ server will extract the username for incoming connections from the client's TLS certificate. For use with KeySafe 5, RabbitMQ must be configured to extract the username from the

TLS certificate using one of the following options:

- The first DNS field in the SAN (`ssl_cert_login_from=subject_alternative_name`, `san_type=dns` and `san_index=0`)
- Common Name (`ssl_cert_login_from=common_name`)
- Distinguished Name (`ssl_cert_login_from=distinguished_name`)

3.6. Large Object Storage

A location for storing objects that are too large for a traditional database, such as CodeSafe machines, is required. This storage can be located either within the Kubernetes cluster, or externally.

To configure this storage you must specify a Persistent Volume Claim (PVC) for the `helm-keysafe5-backend` Helm Chart to use via the `objectStore.pvc` Chart parameter. If a Kubernetes namespace has been created for the KeySafe 5 application, then the PVC must be in the same namespace as the application. The PVC may use any type of storage supported by Persistent Volumes in your Kubernetes Cluster (for example, NFS). See Persistent Volumes for supported storage types.

To set the user and group IDs used by the KeySafe 5 application when accessing the object storage, configure the `podSecurityContext.runAsUser`, `podSecurityContext.runAsGroup` and `podSecurityContext.fsGroup` Chart parameters.

You should ensure that the size of the configured storage is sufficient to meet the application's needs. For this release of KeySafe 5 that should include:

- All CodeSafe 5 machine images that will be managed by KeySafe 5.

3.6.1. NFS Object Storage Configuration

To use an NFS for object storage you must know the NFS server address and the path of the directory being exported from the NFS server.

Create a Persistent Volume containing the configuration of your NFS.

```
$ cat << EOF | kubectl -n nshieldkeysafe5 apply -f -
  apiVersion: v1
  kind: PersistentVolume
  metadata:
    name: nfs-pv
    labels:
      application: keysafe5
  spec:
    capacity:
```

```
      storage: 2Gi
    volumeMode: Filesystem
    accessModes:
      - ReadWriteMany
    persistentVolumeReclaimPolicy: Recycle
    storageClassName: nfs
    nfs:
      path: ${NFS_PATH}
      server: ${NFS_IP}
EOF
```

Create a Persistent Volume Claim to use that Persistent Volume.

```
$ cat << EOF | kubectl -n nshieldkeysafe5 apply -f -
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: data-nshield-keysafe5
  spec:
    storageClassName: nfs
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 2Gi
    selector:
      matchLabels:
        application: keysafe5
EOF
```

3.7. External identity provider (IdP)

You need an external identity provider that supports OIDC and OAuth2 to provide user and machine authentication to KeySafe 5. This is required to gain access to the UI and authenticate commands sent to the backend services.

At the IdP, Entrust typically expects the following to be configured:

- A single OIDC public client application

  This provides user identity information and an `id_token` for use by the UI.

- Multiple OAuth2 private client application

  This provides machine-to-machine credentials. Typically you would want an instance of this per application identity required to limit the sharing of the `client_secret` value.

For more information, refer to Client Types.

3.7.1. OIDC public client

The OIDC public client application provides user identity information and an `id_token` for use by the UI in its calls to the backend services. It is a public client due to the UI being a client side application, and as such cannot be trusted with the `client_secret` like a server side application would be.

Entrust recommends the following settings:

Setting	Value
Grant Types	Authorization Code with PKCE
	Refresh Token
Authorization Code PKCE Code Challenge Method	S256
Scopes	openid

For more information, refer to Authorization Code flow with PKCE.

3.7.2. OAuth2 private client

The OAUTH2 private client provides client credentialing for machine-to-machine authentication by applications that do not hold user identification. This requires the use of the `client_secret` value, which must be securely held.

You would typically want a separate private client instance for each application for which you provide access, resulting in a separate `client_id` and `client_secret` for each application. This eases management of the `client_secret` by reducing the number of applications that have knowledge of it. It also provides easy identification of which application is doing what at the KeySafe 5 end.

Entrust recommend the following settings:

Setting	Value
Grant Type	Client Credentials

For more information, refer to Client Credentials.

4. Helm Chart Installation

To install the KeySafe 5 REST APIs you must install the helm-keysafe5-backend chart.

To install the KeySafe 5 graphical user interface you must also install the helm-keysafe5-ui chart.

To expose these services externally to the Kubernetes cluster, see Configure external access to KeySafe 5.

If you are upgrading an existing KeySafe 5 install, see Helm Chart Upgrade.

4.1. Helm

Helm is a package manager for Kubernetes.

As with `kubectl` you can apply a namespace to the Kubernetes resources created by a Helm chart by specifying `--namespace my-namespace` when using `helm`.

To install the chart with the release name `my-release`:

```
$ helm install my-release helm-keysafe5-backend/
```

List all releases using `helm list`.

To upgrade or modify the existing `my-release` deployment, configure the chart parameters and then run:

```
$ helm upgrade --install my-release helm-keysafe5-backend/
```

To uninstall or delete the `my-release` deployment:

```
$ helm delete my-release
```

4.2. helm-keysafe5-backend

The keysafe5-backend Helm chart deploys Kubernetes Services for the KeySafe 5 REST APIs.

- codesafe-mgmt (nShield CodeSafe Management Service API)

- hsm-mgmt (nShield HSM Management Service API)
- sw-mgmt (nShield Security World Management Service API)

This Helm chart installs the KeySafe 5 services into a Kubernetes cluster but does not configure external access to the services. See Configure external access to KeySafe 5.

4.2.1. Kubernetes Secrets

The helm-keysafe5-backend Helm chart expects to be provided with pre-existing Kubernetes Secrets for the Database and AMQP connections.

For more information, refer to Kubernetes Secrets.

Purpose	Description	Secret Type
MongoDB SCRAM Auth	The username and password pairing used to authenticate with the MongoDB server	kubernetes.io/basic-auth
MongoDB TLS Certificates	TLS certificates used to connect to the MongoDB server. Can also be used as authentication if X509 auth is enabled	Opaque
AMQP SCRAM Auth	The username and password pairing used to authenticate with the AMQP server	kubernetes.io/basic-auth
AMQP TLS Certificates	TLS certificates used to connect to the AMQP server	Opaque

Because TLS secrets have the `Opaque` type, the filenames they are created with are critical. Entrust expects the following:

- CA certificate to be named `ca.crt`.
- TLS certificate to be named `tls.crt`.
- TLS key file to be named `tls.key`.

Your certificates will need to adhere to X.509 v3, sometimes known as a multiple-domain certificates, or SAN certificates. The X.509 extension Subject Alternative Name (SAN) allows specifying multiple hostnames, and has replaced Common Name as the source of the hostname.

4.2.2. Configuration

To deploy the application, configure the chart with:

- The Docker images to use for `codesafe-mgmt`, `hsm-mgmt` and `sw-mgmt`.
- Database connection configuration.
- AMQP connection configuration.
- A reference to the Persistent Volume Claim, in the same Kubernetes namespace, to use for large object storage.

For further details on the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart. For example, you may wish to:

- Increase or decrease the verbosity of the logging in the backend services.
- Change the period of time before a resource liveness health check is marked as failing. For example, if you have decreased the rate at which KeySafe 5 agents report to the central platform, you will want to increase the `health.livenessFailurePeriod` value.

An example install:

```
$ helm install my-release \
  --create-namespace --namespace nshieldkeysafe5 \
  --set database.type=mongo \
  --set database.mongo.hosts="mongo-chart-mongodb-0.mongo-chart-mongodb-
headless.mongons.svc.cluster.local:27017,mongo-chart-mongodb-1.mongo-chart-mongodb-
headless.mongons.svc.cluster.local:27017" \
  --set database.mongo.replicaSet=rs0 \
  --set database.mongo.auth.type=pwd \
  --set database.mongo.auth.existingSecret=my-mongo-credentials-secret \
  --set database.mongo.tls.enabled=true \
  --set database.mongo.tls.existingSecret=my-mongo-tls-secret \
  --set amqp.URL="rabbit-chart-rabbitmq-0.rabbit-chart-rabbitmq-headless.rabbitns.svc.cluster.local:5671" \
  --set amqp.auth.type=pwd \
  --set amqp.auth.existingSecret=my-amqp-credentials-secret \
  --set amqp.tls.enabled=true \
  --set amqp.tls.existingSecret=my-amqp-tls-secret \
  --set objectStore.pvc=data-nshield-keysafe5 \
  --set logging.level=info \
  --set logging.format=json \
 helm-keysafe5-backend/
```

4.3. helm-keysafe5-ui

The keysafe5-ui Helm chart deploys a Kubernetes Service for the KeySafe 5 Graphical User Interface.

The chart deploys the web front-end of KeySafe 5. For the UI to be usable it must point to a deployed KeySafe 5 back-end services endpoint (as installed by helm-keysafe5-backend).

This Helm chart installs the KeySafe 5 UI into a Kubernetes cluster, but does not configure external access to the service. (See Configure external access to KeySafe 5.)

4.3.1. Configuration

To deploy the application, configure the chart with:

- The Docker image to use for the `ui` container.
- OIDC Identity Provider config, if Authentication is enabled.
- The location of the KeySafe 5 back-end services API that this UI displays information for.

For further details on the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart.

An example install:

```
# Untar the chart and copy your OIDC provider config file into the config directory
$ tar -xf helm-charts/nshield-keysafe5-ui-1.2.0.tgz -C helm-charts
$ cp my-oidc-provider-config.json helm-charts/nshield-keysafe5-ui/config/OIDCProviders.json

# Install the chart
$ helm install keysafe5-ui \
  --create-namespace --namespace nshieldkeysafe5 \
  --set svcEndpoint="https://XXX.XXX.XXX.XXX" \
  --set authMethod=oidc \
  --wait --timeout 10m \
helm-charts/nshield-keysafe5-ui
```

Because the OIDC Provider configuration is volume mapped into the Kubernetes application, you must untar the packaged Helm chart so that you can copy in the `OIDCProviders.json` file to the correct location before installing the chart.

4.3.2. Configure UI authentication

If `authMethod` is set to `oidc` then you must provide an OIDC configuration file detailing the accepted Identity Providers.

Copy the OIDC configuration file to `config/OIDCProviders.json` in the root directory of the Helm chart before installing the Helm chart. This file is a JSON document. You can find an example file at `config/OIDCProviders.json.example`.

The configuration document is a JSON list of individual provider configurations.

An example of provider configuration:

```
{
    "name":"Example Provider",
    "authority": "https://example-auth-provider.com/auth/auth",
    "client_id":"8acc1449-7275-4524-b25f-4a60ddddfe8d",
```

```
    "redirect_uri": "https://example-keysafe5.com/callback",
    "response_type": "code",
    "scope": "openid",
    "post_logout_redirect_uri": "https://example-keysafe5.com/",
    "issuer": "https://example-auth-provider.com/auth",
    "authorization_endpoint": "https://example-auth-provider.com/auth/callback/authorize",
    "token_endpoint": "https://example-auth-provider.com/auth/token",
    "jwks_uri": "https://example-auth-provider.com/auth/keys",
    "end_session_endpoint": "https://example-auth-provider.com/auth/logout",
    "userinfo_endpoint": "https://example-auth-provider.com/auth/userinfo",
    "iconSVG" : "login.svg"
}
```

To display a custom icon for the provider in the user interface, copy an SVG-format image to the `config` directory and reference the file name under the key `iconSVG` in the provider configuration.

4.4. Configure external access to KeySafe 5

To enable external access to the services installed by `helm-keysafe5-backend` and `helm-keysafe5-ui`, you need to configure a Kubernetes Ingress Gateway to route requests to the appropriate Kubernetes Services.

If you use Istio, you can use the `helm-keysafe5-istio` Helm chart to configure an existing Istio Ingress Gateway. Alternatively, you can configure your own Ingress Gateway. (See Configure a custom ingress provider.)

4.4.1. helm-keysafe5-istio

The keysafe5-istio Helm chart creates an Istio Gateway and VirtualService for the KeySafe 5 back-end services and user interface to be accessible externally from the Kubernetes clus ter.

The chart:

- Routes HTTP requests to the KeySafe 5 application running in the same Kubernetes cluster.
- Authenticates requests, if authentication is enabled (default) and an Identity Provider (IdP) is configured.
- Applies CORS policy to requests to limit Cross-Origin Resource Sharing.
- Applies security related HTTP Headers to responses including automatically applying the Content-Security-Policy header to help reduce Cross Site Scripting (XSS) risks.
- Limits TLS protocol used to TLS v1.2 and higher.
- Limits ciphers supported by the gateway.

4.4.2. Configure helm-keysafe5-istio

For further details of the configurable values of the Helm chart, see the `README.md` in the root directory of the Helm chart.

An example install:

```
$ helm install keysafe5-istio \
  --create-namespace --namespace=nshieldkeysafe5 \
  --set requireAuthn=true \
  --set httpsEnabled=true \
  --set portNumber=443 \
  --set issuer[0].authIssuer="https://foobar.auth0.com" \
  --set issuer[0].authJWKsURI="https://www.googleapis.com/oauth2/v1/certs" \
  --set issuer[0].authAudiences[0]="https://keysafe5.location" \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.2.0.tgz
```

4.4.3. helm-keysafe5-istio port number

If the port number changes to something other than 443 or 80, you need to open a port on the Istio Ingress Gateway. You can do this using your own istioctl install manifest. For more information, refer to https://istio.io/latest/docs/setup/install/istioctl/.

4.4.4. helm-keysafe5-istio authentication

Authentication can be provided by any OpenID Connect Provider, such as Entrust Identity As A Service.

If `requireAuthn` is `true`, then at least one `authIssuer` must be configured.

An example configuration:

```
issuers:
  - authIssuer: 'https://foobar.auth0.com'
    authJWKsURI: 'https://www.googleapis.com/oauth2/v1/certs'
    authJWKs: ''
    authAudiences:
    - 'https://keysafe5.location'
```

For each `authIssuer`, `authAudiences` and one of `authJWKsURI` or `authJWKs` must be specified.

Key	Description
`authIssuer`	Identifies the issuer that issued the JWT. A JWT with different iss claim will be rejected.

Key	Description
`authAudiences`	Identifies the list of JWT audiences that are allowed access. A JWT containing any of these audiences will be accepted.
`authJWKsURI`	URL of the provider's public key set to validate signature of the JWT. For more information, refer to https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata.
`authJWKs`	JSON Web Key Set of public keys to validate signature of the JWT. For more information, refer to https://auth0.com/docs/jwks.

For additional information, refer to https://istio.io/latest/docs/reference/config/security/jwt/#JWTRule.

Once authenticated, a user has full access to view and manage the HSM and Security World resources accessible from the platform. No authorization policy is applied to requests.

4.4.5. Enable HTTPS for helm-keysafe5-istio

Configuring a TLS certificate for the Istio Gateway requires creating a Kubernetes secret.

To create a Kubernetes secret from an existing TLS private key and certificate:

You must create the Kubernetes Secret in the same namespace as the Istio Ingress Gateway.

```
$ kubectl --namespace istio-system create secret tls keysafe5-server-credential \
  --cert=path/to/cert/file \
  --key=path/to/key/file
```

4.5. Configure a custom ingress provider

If you do not want to use Istio, you can configure your own Kubernetes Ingress.

If you configure your own Ingress to the application, then it is your responsibility to configure routing to the services and any authentication or authorization to access the services.

Each part of the KeySafe 5 application is exposed as a Kubernetes Service.

`helm-keysafe5-backend` exposes the following Kubernetes Services for serving requests to the RESTful APIs:

ClusterIP Port	API endpoints
18080	/mgmt/v1/hsms /mgmt/v1/hosts /mgmt/v1/pools /mgmt/v1/feature-certificates
18081	/mgmt/v1/worlds
18082	/codesafe/v1

`helm-keysafe5-ui` exposes a Kubernetes Service called `keysafe5-ui-svc` on ClusterIP port `8080` for accessing the graphical user interface of KeySafe 5.

5. KeySafe 5 Agent Installation

The KeySafe 5 agent runs alongside the existing hardserver and enables the central manage ment platform to manage all HSMs and Security Worlds visible to the hardserver.

The KeySafe 5 agent is a privileged client of the hardserver. For more information on privileged clients, see the nShield Security World Software documentation.

The connection between the agent and the central monitoring platform is via the RabbitMQ message bus using the Advanced Message Queueing Protocol (AMQP). It is configured in the KeySafe 5 agent configuration file.

Ensure the system clock of the KeySafe 5 agent is synchronized with the central platform.

The KeySafe 5 agent ensures that all key management data, with the exception of keys, is synchronized between the nShield client machine and a central (MongoDB) database.

This means that when resources, such as Card Sets or Softcards, appear in the `kmdata/local` directory on a client machine, they are automatically stored in the central database. It also means that when a Card Set or Softcard is created via the new management tools, the resource also appears in `kmdata/local` on any host machine that is in the right Security World.

The Card Set or Softcard can then be used with the traditional nShield tools on each nShield client machine.

If a resource is deleted via the KeySafe 5 application then it will be removed from `kmdata/local` for all client machines, and Connects, running a KeySafe 5 agent. If the resource is deleted locally on a nShield client machine then that deletion is not synchronized to other client machines in the same Security World.

The KeySafe 5 agent will also report on the status of CodeSafe 5 machines/certificates visible to the agent, and allow these resources to be managed via KeySafe 5. The amount of time taken for the agent to publish a CodeSafe 5 update message will increase by several seconds per CodeSafe 5 resource (machine or certificate) in the system. This means that in systems with many CodeSafe 5 machines/certificates present, KeySafe 5 will be slower to reflect local changes in the state of these resources.

If you are upgrading an existing KeySafe 5 Agent install, see Agent Upgrade.

5.1. Install on Linux

1. Untar the KeySafe 5 agent install package to the root directory of the machine. The agent install package can be found in `keysafe5-agent` directory of the KeySafe 5 release package.

   This unpacks the agent and associated scripts into the `/opt/nfast/` directory.

   ```
   $ cd /
   $ sudo tar -xf /path/to/keysafe5-1.2.0-Linux-keysafe5-agent.tar.gz
   ```

2. Configure this KeySafe 5 agent instance as described in Agent Configuration and AMQP authentication.
3. Run the appropriate install script depending on the state of the hardserver:
   - If the hardserver is running, use the KeySafe 5 specific install script so the hardserver is not restarted.

     ```
     sudo /opt/nfast/keysafe5/sbin/install
     ```

   - When the hardserver is not running, use the nShield install script to install both the KeySafe 5 agent and the hardserver.

     ```
     sudo /opt/nfast/sbin/install
     ```

The agent must point to a working RabbitMQ otherwise it will fail to start.

The installer creates the following items, as required:

- Either a SysV-style init script or systemd script for automatically starting and stopping the service.
- The `keysafe5d` user.

  This user is dedicated to running the `keysafe5-agent` service, and is a member of the `nfast` and `nfastadmin` groups.

The KeySafe 5 agent is not affected by the standard nShield `/opt/nfast/sbin/init.d-ncipher` script. To stop, start, or restart the KeySafe 5 agent you may either:

- Use `/opt/nfast/scripts/init.d/keysafe5-agent`, or
- Use your standard init system scripts, addressing the `nc_keysafe5-agent` service.

5.2. Install on Windows

The KeySafe 5 Agent requires the hardserver TCP ports be enabled. To do this, either:

- Run `config-serverstartup.exe --port 9000 --privport 9001`, or
- Edit the file (located at `%NFAST_KMDATA%\config\config`) and set `nonpriv_port=9000` and `priv_port=9001`.

After enabling the hardserver TCP ports, you must restart the hardserver service.

If those ports are not available and different ports are set, then the environment variables `NFAST_SERVER_PORT` and `NFAST_SERVER_PRIVPORT` must also be set appropriately as described in the nShield documentation. They may be set globally in System Environment Variables, or only for this service by adding a `Multi-String Value` named `Environment` under `Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nShield KeySafe 5 Agent`, and to *Value data* adding the lines `NFAST_SERVER_PORT=port-number` and `NFAST_SERVER_PRIVPORT=port-number`. You may need to restart the computer after adding the System Environment Variables.

1. Launch the `Keysafe5-agent.msi` installer. The installer is in the `keysafe5-agent` directory of the KeySafe 5 release package.

   This msi creates the nShield KeySafe 5 agent service but does not start it.

2. Populate the KeySafe 5 agent configuration file as described in Agent Configuration and AMQP authentication.

   The nShield KeySafe 5 agent service will not start if the certificates are not installed.

3. Populate the `amqp` authentication files.
4. Start the nShield KeySafe 5 agent service using Windows Service Manager.

5.3. Agent Configuration

The KeySafe 5 agent configuration file is located at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml`.

The install contains an example configuration file at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml.example`. Make a copy of it at the same location and rename the copy to `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml`.

Unless configured otherwise, `%NFAST_DATA_HOME%` is located at `/opt/nfast` on Linux and `%ProgramData%\nCipher` on Windows.

Configuration Key	Description	Example Value
override_hostname	Set the hostname for this agent. This appears as the Host resource name in KeySafe 5. The hostname is set by the operating system if not overridden here. This is not related to the hostname used by TLS authentication.	hostname
logging.level	Minimum severity level of log statements to output. Valid values: trace, debug, info, warning, error. The default is to output at info level and above.	info
logging.format	Format of the log statements. Valid values: json, logfmt. The default is to output in json format.	json
logging.file.enabled	To enable log output to file, set to true. The default is to output to file (true).	true
logging.file.path	The absolute path of the file that logs should be written to. The default is /opt/nfast/log/keysafe5-agent.log on Linux and %ProgramData%\nCipher\Log Files\KeySafe5-agent.log on Windows.	/opt/nfast/log/keysafe5-agent.log
amqp.url	The URL points to the AMQP service with its IP address or DNS name, including the port number. For an OVA install, this is the IP address or DNS name of the VM. IPv6 addresses must be in the form [host]:port. If required, a virtual host may be specified at the end of the address. This parameter is required, there is no default.	127.0.0.1:5671/nshieldvhost
amqp.auth_type	Authentication method for the AMQP connection. Valid values: none, pwd, tls. The default is to use TLS authentication.	tls
amqp.tls_username_location	For auth_type 'tls', the AMQP username is encoded in a field of the X.509 certificate. When a RabbitMQ server is configured to allow X.509 authentication, it is specified which field to extract the username from within the certificate. This agent configuration item identifies which field of the certificate contains the username and should match the setting on the RabbitMQ server. Valid values: DistinguishedName, CommonName, SAN-DNS-Field0 (the first DNS field in the Subject Alternative Name of the certificate).	SAN-DNS-Field0
amqp.disable_tls	To disable Mutual TLS for the AMQP connection, set to true. The default is to use Mutual TLS (false). Entrust recommends that this is always set to false.	false

Configuration Key	Description	Example Value
amqp.min_protocol_version	The minimum TLS protocol version that is used by the AMQP connection of the keysafe5 agent. The default is TLSV1_2.	TLSV1_2
amqp.cipherSuites	The available ciphersuites for the AMQP connection of the keysafe5 agent. The defaults are ECDHE-ECDSA-AES128-GCM-SHA256, ECDHE-RSA-AES128-GCM-SHA256, ECDHE-ECDSA-AES256-GCM-SHA384, ECDHE-RSA-AES256-GCM-SHA384, ECDHE-ECDSA-CHACHA20-POLY1305, ECDHE-RSA-CHACHA20-POLY1305	ECDHE-ECDSA-AES128-GCM-SHA256
kmdata_poll_interval	The rate at which the agent polls the kmdata directory to look for changes. The format is as used for update_interval. The default is once a second.	1s
update_interval	The period of time between publishing data updates. The interval string is a sequence of decimal numbers, each with optional fraction and a unit suffix, such as "300ms", "1.5h" or "2h45m". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h". The default is once a minute.	1m
max_update_message_response_time	The maximum amount of time to allow the central plat form to pull update messages sent by this agent. Update messages published by this agent will expire after the lower of this time, or the configured update_interval. This setting impacts the freshness of the data processed by the central platform. The default is 1 minute.	1m
health_interval	The period of time between checking the underlying service health and attempting recovery if necessary. The format is as used for update_interval. The default is once a minute.	1m
codesafe_update_interval	The period of time between publishing CodeSafe 5 data updates. The format is as used for update_interval. The default is once every 5 minutes.	3m
codesafe_cache_period	The codesafe_cache_period specifies how often the CodeSafe 5 certificate cache will expire. The caching is performed to negate performance impacts of running unnecessary CodeSafe 5 certificate commands. Cache expiry may be performed earlier if approaching a time where a certificates validity status may change, that is, when it is approaching NotBefore or NotAfter. The cache is invalidated if there is a change in CodeSafe 5 certificates. The format is as used for update_interval. The default is 60 minutes.	60m

Configure the KeySafe 5 agent's AMQP connection to use the same RabbitMQ instance used by the central management platform that you want to connect to. In an OVA deployment the address of the RabbitMQ instance is the IP address or the DNS name of a node in the cluster.

5.4. AMQP authentication

You can configure the authentication method for the AMQP connection as one of the following options:

- `none` No authentication.
- `pwd` Username and password authentication.
- `tls` X.509 certificate authentication.

Entrust recommends restricting access to files containing sensitive authentication details. On Linux, the KeySafe 5 agent installer will create the required files with appropriate permissions.

The username must contain the KeySafe 5 agent's hostname (set either by `override_hostname` in the agent configuration file, or defaults to the machine's hostname). If the username does not contain the KeySafe 5 agent's hostname then the agent will not start.

5.4.1. TLS

You must create the `%NFAST_DATA_HOME%/keysafe5/conf/amqp/tls` directory manually so that you can store the TLS key and certificates for the agent's connection to AMQP there in the following files:

ca.crt The CA certificate.

tls.key The agent's private key.

tls.crt A valid certificate of the key signed by the Certificate Authority.

Your certificates will need to adhere to X.509 v3, sometimes known as a multiple-domain certificates, or SAN certificates. The X.509 extension Subject Alternative Name (SAN) allows specifying multiple hostnames, and has replaced Common Name as the source of the hostname.

The username extracted from the TLS client certificate (`tls.crt`) is defined by the agent configuration item `amqp.tls_username_location`. By default, this is set to `SAN-DNS-Field0`

(the first DNS field in the Subject Alternative Name of the certificate) but can optionally be set to `Distinguished Name` or `CommonName` to match the TLS authentication settings on the RabbitMQ server.

The extracted certificate username must contain the KeySafe 5 agent's hostname (set either by `override_hostname` in the agent configuration file, or defaults to the machine's hostname). If the username does not contain the KeySafe 5 agent's hostname then the agent will not start.

5.4.1.1. Generating a KeySafe 5 agent private key and TLS certificate

To generate a private key and certificate signing request (CSR) for a specific KeySafe 5 agent, use `%NFAST_HOME%/keysafe5/bin/amqptls`.

1. Generate the agent's private key

   On Linux:

   ```
   $ /opt/nfast/keysafe5/bin/amqptls -keypath=/opt/nfast/keysafe5/conf/amqp/tls/tls.key -keygen
   Private key has been generated and saved to /opt/nfast/keysafe5/conf/amqp/tls/tls.key

   When configuring AMQP TLS for this KeySafe 5 agent, the key should be saved to
   /opt/nfast/keysafe5/conf/amqp/tls/tls.key with file permissions and ownership as documented in the KeySafe
   5 Installation Guide
   ```

   On Windows the command to write to `%NFAST_DATA_HOME%\keysafe5\conf\amqp\tls\tls.key` is:

   ```
   %NFAST_HOME%\bin\amqptls.exe -keypath=C:\ProgramData\nCipher\keysafe5\conf\amqp\tls\tls.key -keygen
   ```

   This will generate an ECDSA P-521 private key and save it to the file pointed to by the `keypath` option. If `keypath` is not specified the file `tls.key` is saved to the current directory.

2. Generate the CSR

   On Linux:

   ```
   $ /opt/nfast/keysafe5/bin/amqptls -keypath=/opt/nfast/keysafe5/conf/amqp/tls/tls.key -csrgen
   CSR has been generated and saved to ks5agent_demohost.csr
   ```

   On Windows the command is:

   ```
   %NFAST_HOME%\bin\amqptls.exe -keypath=C:\ProgramData\nCipher\keysafe5\conf\amqp\tls\tls.key -csrgen
   ```

This will generate a certificate signing request and save it to `ks5agent_<agent_hostname>.csr` (where `<agent_hostname>` is the value of `override_hostname` in the KeySafe 5 agent configuration file, if set, or the host machines host name). Alternatively, the CSR can be printed to the console, rather than saved to a file, by specifying `-csr-stdout`.

The generated CSR requests a certificate that contains the agent hostname as the CommonName and as a DNS SubjectAlternativeName (SAN).

3. Create a TLS Certificate for this KeySafe 5 agent

   The CSR should be provided to a KeySafe 5 administrator who, in a secure location/environment, creates a RabbitMQ service client TLS certificate using the CA trusted by the RabbitMQ server.

   If using a RabbitMQ instance installed by the `deploy.sh` script then the agent certificate can be generated on the central platform on which the `deploy.sh` script was run. Run the `agentcert.sh` script that is shipped alongside the deployment scripts from the preserved directory in which the script was run.

```
$ ./agentcert.sh ks5agent_demohost.csr 365
Certificate generated to ks5agent_demohost.crt. Valid for 365 days
CA Certificate available at /path/to/internalCA/cacert.pem

RabbitMQ will need to be configured to allow access for the user 'ks5agent_demohost':
   export RUN_RABBIT="kubectl -n rabbitns exec rabbit-chart-rabbitmq-0 -c rabbitmq -- "
   ${RUN_RABBIT} rabbitmqctl add_user ks5agent_demohost ephemeralpw
   ${RUN_RABBIT} rabbitmqctl set_permissions -p nshieldvhost ks5agent_demohost '.*' '.*' '.*'
   ${RUN_RABBIT} rabbitmqctl clear_password ks5agent_demohost
```

If you get a message about a lack of permissions opening `/etc/rancher/k3s/k3s.yaml` you can set up `kubectl` access by running:

```
$ mkdir -p $\{HOME}/.kube
$ sudo /usr/local/bin/k3s kubectl config view --raw > $\{HOME}/.kube/config
$ chmod 600 $\{HOME}/.kube/config
$ export KUBECONFIG=$\{HOME}/.kube/config
```

You many append the `export KUBECONFIG=${HOME}/.kube/config` to your shell's configuration file.

4. The KeySafe 5 administrator should then configure the RabbitMQ server to allow the user specified in this TLS certificate to access the appropriate virtual host in RabbitMQ.

```
$ rabbitmqctl add_user ks5agent_demohost ephemeralpw
$ rabbitmqctl set_permissions -p nshieldvhost ks5agent_demohost '.*' '.*' '.*'
$ rabbitmqctl clear_password ks5agent_demohost
```

5. Configure the KeySafe 5 agent

   The resulting TLS certificate and accompanying CA certificate, along with the agent's private key should be stored within `%NFAST_DATA_HOME%/keysafe5/conf/amqp/tls` in the following files:

   - `ca.crt` - The CA certificate.
   - `tls.key` - The agent's private key.
   - `tls.crt` - A valid certificate of the key signed by the Certificate Authority.

5.5. KeySafe 5 agent on network-attached HSMs

A KeySafe 5 agent is installed on the nShield Connect for nShield Connect images released with Security World v13.4 and later software. This agent allows an nShield Connect to be monitored and managed without, or in addition to, the Connect being enrolled to a nShield host machine (a machine with nShield Security World software installed) which also has a KeySafe 5 agent installed.

By default, the KeySafe 5 agent on the nShield Connect is disabled. It must be configured to communicate with the central KeySafe 5 platform, and enabled. The agent can only be configured via the nShield Connect serial console.

5.5.1. ks5agent command (only on serial console network-attached HSMs)

The KeySafe 5 agent is configured and managed on the Connect using the `ks5agent` Serial Console command.

```
(cli)help ks5agent

        Manage the KeySafe 5 agent

        USAGE
          ks5agent
          ks5agent enable
          ks5agent disable
          ks5agent version
          ks5agent logs [tail [linecount]]
          ks5agent cfg [amqp.url=x.x.x.x:5671]
          ks5agent resetcfg
          ks5agent amqpcsr
          ks5agent amqptls [ca.crt|tls.crt] [data]
          ks5agent amqpuser
          ks5agent amqppwd


        OPTIONS
          enable      Start the KeySafe 5 agent (setting will persist on reboot)
          disable     Stop the KeySafe 5 agent
```

```
          version     Show version information for the KeySafe 5 agent
          logs        Display the KeySafe 5 agent log file
          cfg         Configure the KeySafe 5 agent
          resetcfg    Restore the KeySafe 5 agent configuration file back to the default values for this Connect
          amqpcsr     Generate a Certificate Signing request for creation of KeySafe 5 agent TLS certificate
          amqptls     Show/set the TLS certificates for the KeySafe 5 Agent AMQP connection
          amqpuser    Show the agent's username for AMQP password or TLS based authentication
          amqppwd     Configure the password for the agent to use for AMQP password-based authentication

        If no action is specified, the current status of the KeySafe 5 agent will be
        displayed.
```

5.5.1.1. ks5agent cfg

The agent configuration can be displayed and set using the `ks5agent cfg` command.

```
(cli)ks5agent cfg
override_hostname: nshield_module_AAAA-AAAA-AAAA
logging:
  level: info
  format: json
  file:
    enabled: false
    path: /opt/nfast/log/keysafe5-agent.log
amqp:
  url: 127.0.0.1:5671
  auth_type: tls
  tls_username_location: SAN-DNS-Field0
  disable_tls: false
  minProtocolVersion: TLSV1_2
  cipherSuites:
  - ECDHE-ECDSA-AES128-GCM-SHA256
  - ECDHE-RSA-AES128-GCM-SHA256
  - ECDHE-ECDSA-AES256-GCM-SHA384
  - ECDHE-RSA-AES256-GCM-SHA384
  - ECDHE-ECDSA-CHACHA20-POLY1305
  - ECDHE-RSA-CHACHA20-POLY1305
kmdata_poll_interval: 1s
update_interval: 1m
max_update_message_response_time: 1m
health_interval: 1m
codesafe_update_interval: 3m
codesafe_cache_period: 60m

(cli)ks5agent cfg amqp.url=<IPADDRESS>:5671/nshieldvhost update_interval=2m
override_hostname: nshield_module_AAAA-AAAA-AAAA
logging:
  level: info
  format: json
  file:
    enabled: false
    path: /opt/nfast/log/keysafe5-agent.log
amqp:
  url: <IPADDRESS>:5671/nshieldvhost
  auth_type: tls
  tls_username_location: SAN-DNS-Field0
  disable_tls: false
  minProtocolVersion: TLSV1_2
  cipherSuites:
  - ECDHE-ECDSA-AES128-GCM-SHA256
  - ECDHE-RSA-AES128-GCM-SHA256
  - ECDHE-ECDSA-AES256-GCM-SHA384
  - ECDHE-RSA-AES256-GCM-SHA384
```

```
  - ECDHE-ECDSA-CHACHA20-POLY1305
  - ECDHE-RSA-CHACHA20-POLY1305
kmdata_poll_interval: 1s
update_interval: 2m
max_update_message_response_time: 1m
health_interval: 1m
codesafe_update_interval: 3m
codesafe_cache_period: 60m
```

The agent configuration values for `override_hostname` and `logging.file` are fixed and can not be set on nShield Connect. The value for `override_hostname` will be set to `nshield_module_{esn}`.

Multiple configuration items may be set with a single command.

```
ks5agent cfg amqp.url=0.0.0.0:5671/nshieldvhost update_interval=5m
```

If no configuration update is provided, the contents of the KeySafe 5 agent config file are displayed.

To update a configuration item, use the format `key=value` using a `.` character for nested configuration items. Examples:

```
ks5agent cfg update_interval=5m
ks5agent cfg logging.level=debug
ks5agent cfg amqp.url=0.0.0.0:5672
```

If the agent is currently running, it will be restarted to pick up the change in configuration.

By default, you can only set values for keys that already exist in the configuration file. To force setting a key that does not currently exist in the configuration file, specify `--force`.

```
ks5agent cfg newkey=value --force
```

Running the `ks5agent resetcfg` command will reset the agent configuration to the default configuration for this agent on the nShield Connect.

5.5.1.2. AMQP authentication for ks5agent

The AMQP authentication method is configured using the `ks5agent cfg` command and setting the `amqp.auth_type` configuration item.

```
ks5agent cfg amqp.auth_type=tls
ks5agent cfg amqp.auth_type=pwd
ks5agent cfg amqp.auth_type=none
```

The username for password or TLS based authentication can be displayed using the `ks5agent amqpuser` command.

```
(cli)ks5agent amqpuser
ks5agent_nshield_module_AAAA-AAAA-AAAA
```

5.5.1.2.1. Password authentication

To configure password based authentication, use the `ks5agent amqppwd` command. This command will display the username that will be used for AMQP authentication, and then prompt for the password to be input.

```
(cli)ks5agent amqppwd
Configuring password to use for KeySafe 5 agent AQMP user 'ks5agent_nshield_module_AAAA-AAAA-AAAA'
This should match the passphrase configured for the user on the RabbitMQ server
New passphrase:
Confirm passphrase:
passphrase set
```

5.5.1.2.2. TLS

TLS certificate authentication is configured with the following workflow:

1. Generate a CSR for this agent using the `ks5agent amqpcsr` Serial Console command on the nShield Connect.
2. If using a RabbitMQ instance installed by the `deploy.sh` script, then the agent certificate can be generated using the `agentcert.sh` script that is shipped alongside the deployment scripts. See AMQP authentication for ks5agent.
3. Store the TLS certificate for this agent and the CA certificate using the `ks5agent amqptls` Serial Console command on the nShield Connect. These certificates must be entered in base64 encoded format. To create suitable input on a Unix system you can run `base64 --wrap=0 tls.crt`.

```
(cli)ks5agent amqpcsr
<output will contain the CSR>

# Obtain a TLS certificate for the above CSR

(cli)ks5agent amqptls ca.crt <base64encoded_data>
Saved ca.crt

(cli)ks5agent amqptls tls.crt <base64encoded_data>
Saved tls.crt
```

5.5.1.3. Status

To show the current status of the KeySafe 5 agent on the nShield Connect, run the `ks5a-gent` Serial Console command with no arguments.

```
(cli)ks5agent
KeySafe 5 agent is disabled
```

The agent can be enabled and disabled using the `ks5agent enable` and `ks5agent disable` commands. This setting will persist over reboots.

To identify the version of agent installed on the Connect, use the `ks5agent version` command.

```
(cli)ks5agent version
1.2.0-2752f5de
```

5.5.2. Logging

The agent logs of the KeySafe 5 agent running on the nShield Connect may be obtained by using the `ks5agent logs` Serial Console command.

By default, this will print the entire contents of the agent log file to the console. To display just the last 10 lines of the log file, use `ks5agent logs tail`. To display the last `n` lines of the log file, use `ks5agent logs tail <n>` where `<n>` is the number of lines to display.

```
(cli)ks5agent logs
(cli)ks5agent logs tail
(cli)ks5agent logs tail 20
```

If the nShield Connect is configured to append logs to the RFS, or configured to send logs to a Remote Syslog server, then the KeySafe 5 agent logs will be sent with these logs. For more information about configuring logging on the Connect, see the *nShield Connect User Guide*.

6. Upgrade

This chapter details how to update an existing KeySafe 5 install to the latest version.

When upgrading KeySafe 5 it is recommended to first update the Helm charts installed in the central platform and then update all KeySafe 5 agent installs on host machines being managed by KeySafe 5.

Ensure all pods are healthy prior to performing an upgrade, unhealthy pods can prevent helm from fully completing an upgrade.

6.1. Helm Chart Upgrade

To upgrade the release of a Helm Chart we do a `helm upgrade` command, see Helm Upgrade

List all installed releases using `helm list -A`.

```
$ helm list -A
NAME                NAMESPACE       REVISION    UPDATED                                 STATUS      CHART
APP VERSION
keysafe5-backend    nshieldkeysafe5 1           2023-07-03 12:23:50.419332325 +0100 BST deployed    nshield-
keysafe5-backend-1.1.0  1.1.0
keysafe5-istio      nshieldkeysafe5 1           2023-07-03 12:24:29.852443337 +0100 BST deployed    nshield-
keysafe5-istio-1.1.0    1.1.0
mongo-chart         mongons         1           2023-07-03 12:21:42.118470777 +0100 BST deployed    mongodb-
12.1.31                 5.0.19
rabbit-chart        rabbitns        1           2023-07-03 12:21:44.58324842 +0100 BST  deployed    rabbitmq-
11.1.2                  3.11.19
```

To upgrade or modify the existing `my-release` deployment:

1. Obtain the current configuration of the currently installed helm chart release.
2. Run a `helm upgrade` command, overriding any existing configuration values using the `--set` directive.

The example shown below is for the helm-keysafe5-backend Chart but the same process applies also to helm-keysafe5-ui and helm-keysafe5-istio.

Documentation for each configurable value in the KeySafe 5 Helm charts can be found by untarring the chart.tgz and viewing the contents of either README.md or the default values.yaml file.

```
$ helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-backend > keysafe5-backend-values.yaml

$ helm upgrade --install keysafe5-backend \
  --namespace=nshieldkeysafe5 \
```

```
  --values keysafe5-backend-values.yaml \
  --set example.ConfigKey=updatedValue \
  helm-keysafe5-backend/
```

List all installed releases using `helm list` to see the upgraded Helm chart.

```
$ helm list -A
NAME                NAMESPACE       REVISION    UPDATED                                 STATUS      CHART
APP VERSION
keysafe5-backend    nshieldkeysafe5 2           2023-07-03 14:17:49.456099223 +0100 BST deployed    nshield-
keysafe5-backend-1.2.0  1.2.0
keysafe5-istio      nshieldkeysafe5 1           2023-07-03 12:24:29.852443337 +0100 BST deployed    nshield-
keysafe5-istio-1.1.0    1.1.0
mongo-chart         mongons         1           2023-07-03 12:21:42.118470777 +0100 BST deployed    mongodb-
12.1.31                 5.0.19
rabbit-chart        rabbitns        1           2023-07-03 12:21:44.58324842 +0100 BST  deployed    rabbitmq-
11.1.2                  3.11.19
```

To view the release history for a Helm Chart using the `helm history` command.

```
$ helm history -n nshieldkeysafe5 keysafe5-backend
REVISION    UPDATED                     STATUS      CHART                           APP VERSION DESCRIPTION
1           Mon Jul  3 12:23:50 2023    superseded  nshield-keysafe5-backend-1.1.0  1.1.0       Install complete
2           Mon Jul  3 14:17:49 2023    deployed    nshield-keysafe5-backend-1.2.0  1.2.0       Upgrade complete
```

To revert to a previous installed release, use the `helm rollback` command.

```
$ helm -n nshieldkeysafe5 rollback keysafe5-backend 1

$ helm history -n nshieldkeysafe5 keysafe5-backend
REVISION    UPDATED                     STATUS      CHART                           APP VERSION DESCRIPTION
1           Mon Jul  3 12:23:50 2023    superseded  nshield-keysafe5-backend-1.1.0  1.1.0       Install complete
2           Mon Jul  3 14:17:49 2023    superseded  nshield-keysafe5-backend-1.2.0  1.2.0       Upgrade complete
3           Mon Jul  3 14:22:03 2023    deployed    nshield-keysafe5-backend-1.1.0  1.1.0       Rollback to 1
```

6.1.1. Upgrade from KeySafe 5 1.1

To upgrade from KeySafe 5 1.1, see the following example commands.

KeySafe 5 1.2 requires the addition of a new database and additional database user configuration. It also adds a requirement for a persistent object storage. For more details, see MongoDB User Configuration and Object Storage.

The paths to Docker images should be updated to point to where you have stored the shipped Docker images. See Docker Images. The Helm chart directories pointed to when upgrading should be the directory for the Helm chart version that you are upgrading to.

```
helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-backend > keysafe5-backend-values.yaml
helm upgrade --install keysafe5-backend \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-backend-values.yaml \
  --set hsm_mgmt.image=private.registry.local/keysafe5/hsm-mgmt:1.2.0 \
  --set hsm_mgmt.pullPolicy=Always \
  --set sw_mgmt.image=private.registry.local/keysafe5/sw-mgmt:1.2.0 \
  --set sw_mgmt.pullPolicy=Always \
  --set codesafe_mgmt.image=private.registry.local/keysafe5/codesafe-mgmt:1.2.0 \
  --set codesafe_mgmt.pullPolicy=Always \
  --set objectStore.pvc=data-nshield-keysafe5 \
  helm-keysafe5-backend/

helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-ui > keysafe5-ui-values.yaml
helm upgrade --install keysafe5-ui \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-ui-values.yaml \
  --set ui.image=private.registry.local/keysafe5/mgmt-ui:1.2.0 \
  --set ui.pullPolicy=Always \
  helm-keysafe5-ui/

helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-istio > keysafe5-istio-values.yaml
helm upgrade --install keysafe5-istio \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-istio-values.yaml \
  helm-keysafe5-istio/
```

6.2. Agent Upgrade

To update the KeySafe 5 agent installed on a machine:

- Take a backup of the agent config directory located at `%NFAST_DATA_HOME%/keysafe5/conf`.
- Uninstall the existing KeySafe 5 agent as detailed in the KeySafe 5 Installation Guide for the currently installed version of the product.
- Install the new KeySafe 5 agent as detailed in chapter *KeySafe 5 Agent Installation*.
- Restore existing agent configuration and restart the agent.

6.3. Upgrading supporting software

6.3.1. Upgrade from KeySafe 5 1.1 recommended versions

KeySafe 5 1.1 recommended RabbitMQ 3.11.3 and MongoDB 5.0.10. This section details how to upgrade the software from these versions to the latest recommended compatible versions.

6.3.2. MongoDB 5.0.10 to 5.0.19

To update a non-Kubernetes existing Mongo install to a Mongo 5.0.19 install, see the official documentation at Upgrade to the Latest Revision of MongoDB.

To update a Mongo 5.0.10 install deployed via Bitnami Helm Charts:

```
# Obtain details of currently deployed helm charts
# Substitute chart and namespace values in the commands below as required
helm list -A

# Fetch current MongoDB helm chart deployed values
helm -n mongons get values --all --output yaml mongo-chart > mongo-chart-values.yaml

# Obtain the names of the existing mongo secrets
# Substitute secret names in the commands below as required
kubectl get secrets -n mongons

# Make copies of the existing secrets, this is required as the existing ones will be removed during the upgrade
process
kubectl get secret mongodb-server-certificates -n=mongons -o yaml \
  | sed 's/mongodb-server-certificates/mongodb-server-certificates-upgrade/' \
  | kubectl apply -f -
kubectl get secret mongo-chart-mongodb -n=mongons -o yaml \
  | sed 's/mongo-chart-mongodb/mongo-chart-mongodb-upgrade/' \
  | kubectl apply -f -

# Upgrade helm chart based on existing deployed values
helm upgrade --install mongo-chart \
    --namespace=mongons \
    --values mongo-chart-values.yaml \
    --set image.tag=5.0.19-debian-11-r3 \
    --set auth.existingSecret=mongo-chart-mongodb-upgrade \
    --set tls.autoGenerated=false \
    --set tls.existingSecret=mongodb-server-certificates-upgrade\
    --wait --timeout 5m \
    bitnami/mongodb --version 12.1.31

# Obtain details of newly deployed helm charts
helm list -A
```

The helm listing for mongo-chart may fail to update app version from 5.0.10 to 5.0.19. This bug is only visual and the image will have updated successfully. This can be confirmed by using kubectl.

6.3.2.1. MongoDB User Configuration

To configure the MongoDB database correctly, access the MongoDB shell to create a user with the minimal permissions required for using the `codesafe-mgmt-db`, `hsm-mgmt-db` and `sw-mgmt-db` databases. See Database: User Roles.

Note that the username needs to match the subject of the client certificate, as found by the following command.

```
$ kubectl get secret --namespace nshieldkeysafe5 mongodb-client-certificates \
  -o jsonpath="{.data.tls\.crt}" | base64 --decode | \
  openssl x509 -out mongo-client-cert.pem
```

```
$ openssl x509 -in mongo-client-cert.pem -subject | head -n 1
```

In this example we will use ks5-mongo-user

We then run the mongo client container.

```
$ export MONGO1=mongo-chart-mongodb-0.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGO2=mongo-chart-mongodb-1.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGODB=${MONGO1},${MONGO2}
$ export MONGO_RUN="kubectl -n mongons exec mongo-chart-mongodb-0 0 -- "
$ export TLS_PRIVKEY="$(${MONGO_RUN} bash -c 'cat /certs/mongodb.pem')"
$ export TLS_CERT="$(${MONGO_RUN} bash -c 'cat /certs/mongodb-ca-cert')"
$ export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace mongons \
  mongo-chart-mongodb-upgrade -o jsonpath="{.data.mongodb-root-password}" \
  | base64 --decode)
$ kubectl run --namespace mongons mongo-chart-mongodb-client \
  --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" \
  --env="TLS_PRIVKEY=$TLS_PRIVKEY" --env="TLS_CERT=$TLS_CERT" --env="MONGODB=$MONGODB" \
  --image bitnami/mongodb:5.0.19-debian-11-r3 --command -- bash
```

Once inside the mongo client container, we need to set up a connection to the server before we can start mongo admin and create the user. After we finish creating the user, we need to exit mongo admin, and then the mongo-client container.

```
$ echo "$TLS_CERT" > /tmp/tls.crt
$ echo "$TLS_PRIVKEY" > /tmp/tls.key
$ mongo admin --tls --tlsCAFile /tmp/tls.crt --tlsCertificateKeyFile /tmp/tls.key \
  --host $MONGODB --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

> use admin
> db.createRole(
  {
    role: "hsm-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "hsm-mgmt-db", "collection": ""},
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> db.createRole(
  {
    role: "sw-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "sw-mgmt-db", "collection": ""},
          "actions": ["createIndex", "dropCollection", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> db.createRole(
  {
    role: "codesafe-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "codesafe-mgmt-db", "collection": ""},
```

```
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> use $external
> x509_user = {
   "roles" : [
     {"role": "codesafe-mgmt-db-user", "db": "admin" },
     {"role": "hsm-mgmt-db-user", "db": "admin" },
     {"role": "sw-mgmt-db-user", "db": "admin" },
   ]
 }
> db.updateUser("CN=ks5-mongo-user", x509_user)
> exit
$ exit
```

6.3.3. RabbitMQ 3.11.3 to 3.11.19

To update a non-Kubernetes existing RabbitMQ install, see the official documentation at Upgrading RabbitMQ.

To update a RabbitMQ 3.11.3 install deployed via Bitnami Helm Charts:

```
# Obtain details of currently deployed helm charts
# Substitute chart and namespace values in the commands below as required
helm list -A

# Fetch current rabbitmq helm chart deployed values
helm -n rabbitns get values --all --output yaml rabbit-chart > rabbit-chart-values.yaml

# Upgrade helm chart based on existing deployed values
helm upgrade --install rabbit-chart \
  --namespace=rabbitns \
  --values rabbit-chart-values.yaml \
  --set image.tag=3.11.19-debian-11-r18 \
  --set service.type=LoadBalancer \
  --set extraPlugins="rabbitmq_auth_mechanism_ssl" \
  --wait --timeout 10m \
  bitnami/rabbitmq --version 11.16.2

# Obtain details of newly deployed helm charts
helm list -A
```

6.3.4. Istio

The version of Istio installed aligns with the software version of `istioctl`.

For details of how to upgrade Istio, see Upgrade Istio.

6.3.5. Object Storage

Upgrading to KeySafe 5 1.2 requires a Persistent Volume Claim in the nshieldkeysafe5 Kubernetes namespace.

For details on how to create a PVC, see Object Storage.

7. Uninstall

7.1. Central platform

To fully remove the KeySafe 5 application from your Kubernetes cluster, use `helm uninstall`. This uninstalls all KeySafe 5 Helm charts.

```
$ helm list --short --all-namespaces
keysafe5-backend
keysafe5-istio
keysafe5-ui

$ helm uninstall keysafe5-backend --namespace="nshieldkeysafe5"
release "keysafe5-backend" uninstalled

$ helm uninstall keysafe5-ui --namespace="nshieldkeysafe5"
release "keysafe5-ui" uninstalled

$ helm uninstall keysafe5-istio --namespace="nshieldkeysafe5"
release "keysafe5-istio" uninstalled
```

KeySafe 5 application data remains in your MongoDB database after uninstalling the application. To clear this data from the database, remove the databases defined by `hsm_mgmt.dbName` and `sw_mgmt.dbName` in the `helm-keysafe5-backend` chart.

7.2. KeySafe 5 agent

Before uninstalling the nShield KeySafe 5 agent, Entrust recommends that you back up any configuration files and certificates from the install.

7.2.1. Linux

To remove the KeySafe 5 agent from a Linux host run the KeySafe 5 uninstaller:

```
/opt/nfast/keysafe5/sbin/install -u
```

Then proceed to remove the following files and directories:

- `/opt/nfast/keysafe5`
- `/opt/nfast/sbin/keysafe5-agent`
- `/opt/nfast/lib/versions/keysafe5-agent-atv.txt`
- `/opt/nfast/scripts/install.d/12keysafe5-agent`

- `/opt/nfast/log/keysafe5-agent.log`

The agent log file will be located in a different location if you have changed the default value of `logging.file.path` in the agent configuration file.

If required, you can also remove the `keysafe5d` user that was created as part of the installation.

7.2.2. Windows

To remove the KeySafe 5 agent from a Windows host:

1. Stop the KeySafe 5 agent service using **Windows Service Manager**.
2. Open the **Control Panel** and select **Programs and Features**.
3. Select the **nShield KeySafe 5 Agent** package.
4. Select **Uninstall** and follow the on-screen instructions.

To remove any configuration files, delete the `%NFAST_DATA_HOME%\keysafe5` directory and remove the log file located at `C:\ProgramData\nCipher\Log Files\KeySafe5-agent.log`

The agent log file will be located in a different location if you have changed the default value of `logging.file.path` in the agent configuration file.

8. Security Guidance

Your nShield HSM protects the confidentiality and integrity of your Security World keys. KeySafe 5 allows an authorized client to remotely configure and manage an estate of nShield HSMs. All network traffic between KeySafe 5 and clients using the UI, the REST API, or both, passes through a secure channel. This TLS based secure channel is set up using token-based client authentication. The administrator of the KeySafe 5 system must remain diligent concerning the entities who are given access to the system and the secure configu ration of the system.

Entrust recommends the following security-related actions for KeySafe 5 deployments:

- Ensure that log levels are set appropriately for your environment.

  More verbose log levels might expose information that is useful for auditing users of KeySafe 5, but the log information also reveals which REST API operations were performed. While this log information might be useful for diagnostics, it could also be considered sensitive and should be suitably protected when stored.

- Rotate the logs regularly. The log files could grow quickly if left unattended for a long time. The system administrator is responsible for log rotation.
- Verify the integrity of the KeySafe 5 tar file before executing it. You can verify the integrity of this file with the hash provided with the software download.
- Suitably protect the network environment of KeySafe 5 to maintain its availability, for example using firewalls and intrusion detection and prevention systems.
- Ensure that the KeySafe 5 platform's system clock is set accurately and only authorized system administrators can modify it so that the platform correctly interprets certificate and token lifetimes.
- Ensure that only authorized system administrators have access to the KeySafe 5 system, and only trusted software is run on the platform hosting KeySafe 5.
- Take standard virus prevention and detection measures on the platform hosting KeySafe 5.
- The system administrator should consider whether threats in the KeySafe 5 deployment environment would justify the encryption of the sensitive configuration data held in Kubernetes secrets, see Kubernetes documentation.

9. Manual Install

The following steps provide a step-by-step guide to installing KeySafe 5 and its dependencies into an existing Kubernetes cluster.

An alternative to this guide is the KeySafe 5 Quick Start Guide which provides a scripted means of installing KeySafe 5.

These steps install KeySafe 5 and its dependencies. They should be followed to set up a demo environment for evaluation purposes and should *not* be used for production environments.

Please see Hardening The Deployment for steps to harden the deployment. Entrust recommends these steps as a minimum and that additional hardening may be required dependent on your own requirements.

A production deployment will have as a minimum the following:

- Maintained and patched versions of all the dependencies.
- A secure CA with TLS v1.3 support for certificates. The deploy script can provide a local insecure CA.
- A secure Kubernetes installation. The deploy script can install K3s locally.
- A secure MongoDB database. The deploy script can provide a replicated MongoDB with X.509 authentication running in Kubernetes.
- A secure RabbitMQ message broker. The deploy script can provide a RabbitMQ with X.509 authentication running in Kubernetes.
- A secure means of large object storage. The deploy script can provide local object storage within the Kubernetes cluster or be configured for using an NFS for object storage.
- HTTPS secured by a trusted certificate for the KeySafe 5 endpoints. The deploy script will enable HTTPS connections with a self-signed insecure certificate.
- Require authentication to access KeySafe 5. OIDC & OAUTH2 are currently supported in KeySafe 5. The deploy script will not set up authenticated access.

9.1. Unpack the release

```
$ mkdir ~/keysafe5-install
$ tar -xf nshield-keysafe5-1.2.0.tar.gz -C ~/keysafe5-install
```

```
$ cd ~/keysafe5-install
```

9.2. Docker images

The Docker images need to be loaded in a Docker registry that each node in your Kubernetes cluster can pull the images from.

```
# Load the Docker images to your local Docker
$ docker load < docker-images/codesafe-mgmt.tar
$ docker load < docker-images/hsm-mgmt.tar
$ docker load < docker-images/sw-mgmt.tar
$ docker load < docker-images/ui.tar
```

To run a local private Docker registry, see Deploy a registry server.

Push the KeySafe 5 images to your private docker registry:

```
$ export DOCKER_REGISTRY=private.registry.local

# Tag the Docker images for a private registry
$ docker tag codesafe-mgmt:1.2.0 $DOCKER_REGISTRY/keysafe5/codesafe-mgmt:1.2.0
$ docker tag hsm-mgmt:1.2.0 $DOCKER_REGISTRY/keysafe5/hsm-mgmt:1.2.0
$ docker tag sw-mgmt:1.2.0 $DOCKER_REGISTRY/keysafe5/sw-mgmt:1.2.0
$ docker tag mgmt-ui:1.2.0 $DOCKER_REGISTRY/keysafe5/mgmt-ui:1.2.0

# Log in to ensure pushes succeed
$ docker login $DOCKER_REGISTRY

# And push
$ docker push $DOCKER_REGISTRY/keysafe5/codesafe-mgmt:1.2.0
$ docker push $DOCKER_REGISTRY/keysafe5/hsm-mgmt:1.2.0
$ docker push $DOCKER_REGISTRY/keysafe5/sw-mgmt:1.2.0
$ docker push $DOCKER_REGISTRY/keysafe5/mgmt-ui:1.2.0
```

9.3. Set up a Certificate Authority

You should use your existing CA for a production system. This is simply used as an example for the purposes of having a working demo system.

Either OpenSSL 3.0 or OpenSSL 1.1.1 may be used to create the CA, and the CA may be created in a directory of your choosing. In these examples, `/home/user/keysafe5-install/internalCA` is the example directory used. In that directory, create the file `internalCA.conf` with the contents:

```
[ ca ]
default_ca      = CA_default                        # The default ca section

[ CA_default ]

dir             = /home/user/keysafe5-install/internalCA  # The directory of the CA
```

```
database        = $dir/index.txt                    # index file.
new_certs_dir   = $dir/newcerts                     # new certs dir

certificate     = $dir/cacert.pem                   # The CA cert
serial          = $dir/serial                       # serial no file
#rand_serial    = yes                               # for random serial#'s
private_key     = $dir/private/cakey.pem            # CA private key
RANDFILE        = $dir/private/.rand                # random number file

default_days    = 15                                # how long to certify for
default_crl_days= 5                                 # how long before next CRL
default_md      = sha256                            # Message Digest
policy          = test_root_ca_policy
x509_extensions = certificate_extensions
unique_subject  = no
# This copy_extensions setting should not be used in a production system.
# It is simply used to simplify the demo system.
copy_extensions = copy

[ test_root_ca_policy ]
commonName = supplied
stateOrProvinceName = optional
countryName = optional
emailAddress = optional
organizationName = optional
organizationalUnitName = optional
domainComponent = optional

[ certificate_extensions ]
basicConstraints = CA:false

[ req ]
default_bits       = 4096
default_md         = sha256
prompt             = yes
distinguished_name = root_ca_distinguished_name
x509_extensions    = root_ca_extensions

[ root_ca_distinguished_name ]
commonName = hostname

[ root_ca_extensions ]
basicConstraints       = CA:true
keyUsage               = keyCertSign, cRLSign
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints       = critical,CA:true
```

Remember to update the `dir` value to the directory in which the `internalCA.conf` and the other CA files will be stored.

To generate the long-term CA key and random number source, create a directory called `private`:

```
$ mkdir ~/keysafe5-install/internalCA/private
```

Then run:

```
$ openssl genrsa -out ~/keysafe5-install/internalCA/private/cakey.pem 4096
```

```
$ openssl rand -out ~/keysafe5-install/internalCA/private/.rand 1024
```

The CA needs a self-signed certificate; as this is a short-term demo it will be valid for 90 days:

```
$ openssl req -x509 -new -nodes \
  -key internalCA/private/cakey.pem \
  -subj "/CN=internalCA" -days 90 \
  -out internalCA/cacert.pem \
  -config internalCA/internalCA.conf
$ cp internalCA/cacert.pem ca.crt
```

And finally, to finish off the configuration:

```
$ mkdir internalCA/newcerts
$ echo 01 > internalCA/serial
$ touch internalCA/index.txt
```

9.4. Install and set up the supporting software

9.4.1. Kubernetes namespace

Create a namespace in Kubernetes for KeySafe 5 installation.

```
$ kubectl create namespace nshieldkeysafe5
```

9.4.2. Istio

The version of Istio installed will align with the software version of `istioctl`.

```
$ istioctl install -y
```

9.4.3. RabbitMQ

Entrust recommends that you use your standard secure RabbitMQ installation, along with your policies for authentication and virtual hosts on your production system; this is only a demo system.

First, you must generate the TLS keys and guest password. You must add the network addresses through which RabbitMQ will be accessed to the certificate, and are very dependent on the configuration of the Kubernetes cluster.

```
$ openssl genrsa -out ~/keysafe5-install/rabbit.key 4096
$ export DNS1="*.rabbit-chart-rabbitmq-headless.rabbitns.svc.cluster.local"
$ export DNS2=rabbit-chart-rabbitmq.rabbitns.svc
$ export DNS3=rabbitmq.rabbitns.svc.cluster.local
$ export DNS4=host.docker.internal
$ export LOCALIP=127.0.0.1
$ export HOSTIP=$(hostname -I | cut -f1 -d" ")
$ openssl req -new -key ~/keysafe5-install/rabbit.key \
  -out ~/keysafe5-install/rabbitmq.csr -subj \
  "/CN=rabbitmq/C=GB/L=Cambridge" \
  -addext "keyUsage=digitalSignature" \
  -addext "extendedKeyUsage=serverAuth" \
  -addext
"subjectAltName=DNS:rabbitmq,DNS:${DNS1},DNS:${DNS2},DNS:${DNS3},DNS:${DNS4},DNS:${HOSTNAME},IP:${LOCALIP},IP:${H
OSTIP}"
$ openssl ca -config ~/keysafe5-install/internalCA/internalCA.conf \
  -out rabbit.crt \
  -in rabbitmq.csr -batch
$ rm rabbitmq.csr
```

Now create the kubernetes secrets for the RabbitMQ service:

```
$ kubectl create namespace rabbitns
$ kubectl create secret generic rabbitmq-certificates \
  --namespace=rabbitns \
  --from-file=ca.crt \
  --from-file=tls.crt=rabbit.crt \
  --from-file=tls.key=rabbit.key
$ kubectl -n rabbitns create secret generic rabbitmq-pw \
  --from-literal=rabbitmq-password=guest
```

Then install RabbitMQ.

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami && helm repo update
$ helm install rabbit-chart \
  --namespace=rabbitns \
  --set image.tag=3.11.19-debian-11-r18 \
  --set auth.username=guest \
  --set auth.existingPasswordSecret=rabbitmq-pw \
  --set auth.tls.enabled=true \
  --set auth.tls.existingSecret=rabbitmq-certificates \
  --set replicaCount=2 \
  --set service.type=LoadBalancer \
  --set extraConfiguration='
    listeners.ssl.default = 5671
    ssl_options.versions.1 = tlsv1.3
    ssl_options.depth=0
    ssl_options.verify = verify_peer
    ssl_options.fail_if_no_peer_cert = true
    auth_mechanisms.1 = EXTERNAL
    ssl_cert_login_from = subject_alternative_name
    ssl_cert_login_san_type = dns
    ssl_cert_login_san_index = 0' \
  --set plugins="" \
  --set extraPlugins="rabbitmq_auth_mechanism_ssl" \
  --wait --timeout 10m  bitnami/rabbitmq --version 11.16.2
```

As we have the service.type set to LoadBalancer we use the IP address of this machine for AMQP connections. We have also added `${HOSTNAME}` to the certificate's subjectAltName

as a DNS entry so it may also be used instead of the IP address.

```
$ export RABBIT_URL=${HOSTIP}:5671
```

Add the virtual host that will be used for KeySafe 5 communication.

```
$ export RUN_RABBIT="kubectl -n rabbitns exec rabbit-chart-rabbitmq-0 -c rabbitmq -- "
$ export RABBIT_VHOST=nshieldvhost
$ ${RUN_RABBIT} rabbitmqctl add_vhost ${RABBIT_VHOST}
```

Then add and configure the X.509 user for the KeySafe 5 application to communicate with RabbitMQ.

```
$ export KS5_USER=ks5
$ ${RUN_RABBIT} rabbitmqctl add_user $KS5_USER "ephemeralpw"
$ ${RUN_RABBIT} rabbitmqctl set_permissions -p $RABBIT_VHOST $KS5_USER ".*" ".*" ".*"
$ ${RUN_RABBIT} rabbitmqctl clear_password $KS5_USER
```

You should then create the X.509 key and certificate for this user.

```
openssl genrsa -out $KS5_USER.key 4096
openssl req -new -key $KS5_USER.key -out $KS5_USER.csr \
  -subj "/CN=${KS5_USER}/C=GB/L=Cambridge" \
  -addext "keyUsage=digitalSignature" \
  -addext "extendedKeyUsage=clientAuth" \
  -addext "subjectAltName=DNS:${KS5_USER}"
openssl ca -config ~/keysafe5-install/internalCA/internalCA.conf \
  -out ${KS5_USER}.crt -in ${KS5_USER}.csr -batch
rm ${KS5_USER}.csr
$ kubectl create secret generic ks5-amqptls \
  --namespace nshieldkeysafe5 \
  --from-file=ca.crt \
  --from-file=tls.crt=ks5.crt \
  --from-file=tls.key=ks5.key
```

Now remove access for the default `guest` user.

```
$ ${RUN_RABBIT} rabbitmqctl delete_user guest
```

9.4.4. MongoDB

Entrust recommends that you use your standard secure MongoDB Replica Set installation. This is just an example, and not production-ready.

```
$ kubectl create namespace mongons
$ helm install mongo-chart \
  --set image.tag=5.0.19-debian-11-r3 \
  --set architecture=replicaset \
  --set auth.enabled=true \
```

```
  --set auth.username=dummyuser \
  --set auth.password=dummypassword \
  --set auth.database=authdb \
  --set tls.enabled=true \
  --namespace=mongons \
  --wait --timeout 10m  bitnami/mongodb --version 12.1.31
```

There will be a message listing the MongoDB server addresses. Save the addresses to environment variables for use later.

```
$ export MONGO1=mongo-chart-mongodb-0.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGO2=mongo-chart-mongodb-1.mongo-chart-mongodb-headless.mongons.svc.cluster.local:27017
$ export MONGODB=${MONGO1},${MONGO2}
```

We then pick up the secrets in the MongoDB configuration.

```
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.client-pem}" | base64 --decode | \
  openssl pkey -out mongo-client-key.pem
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.client-pem}" | base64 --decode | \
  openssl x509 -out mongo-client-cert.pem
$ kubectl get secret --namespace mongons mongo-chart-mongodb-ca \
  -o jsonpath="{.data.mongodb-ca-cert}" | base64 --decode > mongo-ca-cert.pem
```

We add those secrets in a format that KeySafe 5 can accept.

```
$ kubectl create secret generic mongodb-client-tls \
  --namespace=nshieldkeysafe5 \
  --from-file=ca.crt=mongo-ca-cert.pem \
  --from-file=tls.crt=mongo-client-cert.pem \
  --from-file=tls.key=mongo-client-key.pem
$ rm mongo-ca-cert.pem mongo-client-key.pem
```

Access the MongoDB shell to create roles with minimal required permissions on the `code-safe-mgmt-db`, `hsm-mgmt-db` and `sw-mgmt-db` collections, followed by creating a user with these roles. Note that the username needs to match the subject of the client certificate, as found by the following command.

```
$ openssl x509 -in mongo-client-cert.pem -subject | head -n 1
```

In this example, `mongo-chart-mongodb.mongons.svc.cluster.local` is used.

Run the Mongo client container.

```
$ export MONGO_RUN="kubectl -n mongons exec mongo-chart-mongodb-0 -c mongodb -- "
$ export TLS_PRIVKEY="$(${MONGO_RUN} bash -c 'cat /certs/mongodb.pem')"
$ export TLS_CERT="$(${MONGO_RUN} bash -c 'cat /certs/mongodb-ca-cert')"
$ export MONGODB_ROOT_PASSWORD=$(kubectl get secret --namespace mongons \
  mongo-chart-mongodb -o jsonpath="{.data.mongodb-root-password}" \
  | base64 --decode)
```

```
$ kubectl run --namespace mongons mongo-chart-mongodb-client \
  --rm --tty -i --restart='Never' --env="MONGODB_ROOT_PASSWORD=$MONGODB_ROOT_PASSWORD" \
  --env="TLS_PRIVKEY=$TLS_PRIVKEY" --env="TLS_CERT=$TLS_CERT" --env="MONGODB=$MONGODB" \
  --image bitnami/mongodb:5.0.19-debian-11-r3 --command -- bash
```

Once inside the Mongo client container, you must set up a connection to the server before starting mongo admin and creating the roles and user. After the user is created, exit Mongo admin and the Mongo client container.

```
$ echo "$TLS_CERT" > /tmp/tls.crt
$ echo "$TLS_PRIVKEY" > /tmp/tls.key
$ mongo admin --tls --tlsCAFile /tmp/tls.crt --tlsCertificateKeyFile /tmp/tls.key \
  --host $MONGODB --authenticationDatabase admin -u root -p $MONGODB_ROOT_PASSWORD

> use admin
> db.createRole(
  {
    role: "hsm-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "hsm-mgmt-db", "collection": ""},
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> db.createRole(
  {
    role: "sw-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "sw-mgmt-db", "collection": ""},
          "actions": ["createIndex", "dropCollection", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> db.createRole(
  {
    role: "codesafe-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "codesafe-mgmt-db", "collection": ""},
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
> use $external
> x509_user = {
   "user" : "CN=mongo-chart-mongodb.mongons.svc.cluster.local",
   "roles" : [
     {"role": "codesafe-mgmt-db-user", "db": "admin" },
     {"role": "hsm-mgmt-db-user", "db": "admin" },
     {"role": "sw-mgmt-db-user", "db": "admin" },
   ]
 }
> db.createUser(x509_user)
> exit
$ exit
```

9.4.5. Object Storage

For large object storage, create a Persistent Volume Claim, in the `nshieldkeysafe5` Kubernetes namespace (the same namespace that we will deploy the application to).

9.4.5.1. Cluster-local Object Storage

If your Kubernetes cluster only has 1 worker node, you can choose to use local storage.

```
$ cat << EOF | kubectl -n nshieldkeysafe5 apply -f -
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: data-nshield-keysafe5
  spec:
    accessModes:
      - ReadWriteOnce
    storageClassName: local-path
    resources:
      requests:
        storage: 2Gi
EOF
```

9.4.5.2. NFS Object Storage

If your Kubernetes cluster has more than 1 worker node, you must use a type of storage that supports distributed access, such as NFS. For details on creating a PVC for NFS object storage, please see NFS Object Storage Configuration.

9.5. Install KeySafe 5

```
# Get Ingress IP address
$ export INGRESS_IP=$(kubectl --namespace istio-system get svc -l app=istio-ingressgateway -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}')

# Install the KeySafe 5 backend services
$ helm install keysafe5-backend \
  --namespace=nshieldkeysafe5 \
  --set codesafe_mgmt.image=$DOCKER_REGISTRY/keysafe5/codesafe-mgmt:1.2.0 \
  --set hsm_mgmt.image=$DOCKER_REGISTRY/keysafe5/hsm-mgmt:1.2.0 \
  --set sw_mgmt.image=$DOCKER_REGISTRY/keysafe5/sw-mgmt:1.2.0 \
  --set database.type=mongo \
  --set database.mongo.hosts="$MONGO1\,$MONGO2" \
  --set database.mongo.replicaSet=rs0 \
  --set database.mongo.auth.type=tls \
  --set database.mongo.auth.authDatabase=authdb \
  --set database.mongo.tls.enabled=true \
  --set database.mongo.tls.existingSecret=mongodb-client-tls \
  --set amqp.URL=${RABBIT_URL}/${RABBIT_VHOST} \
  --set amqp.auth.type=tls \
  --set amqp.tls.enabled=true \
  --set amqp.tls.existingSecret=ks5-amqptls \
  --set objectStore.pvc=data-nshield-keysafe5 \
```

```
  --wait --timeout 10m \
 helm-charts/nshield-keysafe5-backend-1.2.0.tgz

# Install the KeySafe 5 UI
$ helm install keysafe5-ui \
  --namespace=nshieldkeysafe5 \
  --set ui.image=$DOCKER_REGISTRY/keysafe5/mgmt-ui:1.2.0 \
  --set svcEndpoint="https://${INGRESS_IP}" \
  --set authMethod=none \
  --wait --timeout 10m \
helm-charts/nshield-keysafe5-ui-1.2.0.tgz

# Create the TLS secret for the Istio Ingress Gateway
$ openssl genrsa -out istio.key 4096
$ openssl req -new -key istio.key -out istio.csr \
  -subj "/CN=${HOSTNAME}" \
  -addext "keyUsage=digitalSignature" \
  -addext "extendedKeyUsage=serverAuth" \
  -addext "subjectAltName=DNS:${HOSTNAME},IP:${INGRESS_IP}"
$ openssl ca -config ~/keysafe5-install/internalCA/internalCA.conf \
  -out istio.crt -in istio.csr -batch
$ kubectl -n istio-system create secret tls \
  keysafe5-server-credential --cert=istio.crt --key=istio.key

# Configure Istio Ingress Gateway for KeySafe 5
$ helm install keysafe5-istio \
  --namespace=nshieldkeysafe5 \
  --set tls.existingSecret=keysafe5-server-credential \
  --set requireAuthn=false \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.2.0.tgz
```

9.6. Access KeySafe 5

You can now access KeySafe 5 at `https://$INGRESS_IP`.

For example, you could send curl requests as demonstrated below.

```
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/mgmt/v1/hsms | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/mgmt/v1/hosts | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/mgmt/v1/pools | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/mgmt/v1/feature-certificates | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/mgmt/v1/worlds | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/codesafe/v1/images | jq
$ curl -X GET --cacert ca.crt https://${INGRESS_IP}/codesafe/v1/certificates | jq
```

You can access the Management UI in a web browser at `https://$INGRESS_IP`.

9.7. Configure nShield client machines

To configure a host machine to be managed and monitored by this deployment, run the KeySafe 5 agent binary on the nShield client machine containing the relevant Security World or HSMs.

Configure this KeySafe 5 agent to communicate with the RabbitMQ server installed previously.

Ensure no firewall rules are blocking the AMQP port communication between the machine exposing the AMQP port from Kubernetes and the machine running the agent.

```
$ sudo tar -xf ~/keysafe5-install/keysafe5-agent/keysafe5-1.2.0-Linux-keysafe5-agent.tar.gz -C /
$ export KS5CONF=/opt/nfast/keysafe5/conf
$ sudo cp $KS5CONF/config.yaml.example $KS5CONF/config.yaml
$ sudo sed -i "s|^  url: 127.0.0.1:5671|  url: ${INGRESS_IP}:5671/${RABBIT_VHOST}|g"
/opt/nfast/keysafe5/conf/config.yaml
```

Create the amqp/tls configuration directory and copy the `ca.crt` file created in the `keysafe5-install` directory on the deployment machine.

```
$ sudo mkdir -p $KS5CONF/amqp/tls
$ sudo cp ca.crt $KS5CONF/amqp/tls/ca.crt
```

Create the private key and a certificate signing request (CSR) for this specific KeySafe 5 agent.

```
$ sudo /opt/nfast/keysafe5/bin/amqptls -keypath=/opt/nfast/keysafe5/conf/amqp/tls/tls.key -keygen
$ sudo /opt/nfast/keysafe5/bin/amqptls -keypath=/opt/nfast/keysafe5/conf/amqp/tls/tls.key -csrgen
```

The CSR should be provided to a KeySafe 5 administrator who, in a secure location/environment, creates a RabbitMQ service client TLS certificate using the CA trusted by the RabbitMQ server.

On the machine that you created the Demo Certificate Authority on, use the `agentcert.sh` script to create a client TLS certificate for this KeySafe 5 agent using the CSR and the demo CA.

```
$ sudo chmod +r ks5agent_demohost.csr
$ ./agentcert.sh ks5agent_demohost.csr 365
```

Using the username printed in the output of the previous command, configure the RabbitMQ server to allow access for this X.509 user in the appropriate virtual host.

```
$ export x509user=ks5agent_demohost
$ export RUN_RABBIT="kubectl -n rabbitns exec rabbit-chart-rabbitmq-0 -c rabbitmq -- "
$ ${RUN_RABBIT} rabbitmqctl add_user $x509user "ephemeralpw"
$ ${RUN_RABBIT} rabbitmqctl set_permissions -p $RABBIT_VHOST $x509user ".*" ".*" ".*"
$ ${RUN_RABBIT} rabbitmqctl clear_password $x509user
```

Transfer the resulting certificate `ks5agent_demohost.crt` to the nShield client machine at `/opt/nfast/keysafe5/conf/amqp/tls/tls.crt`.

On the nShield client machine, if the hardserver is already running, use the KeySafe 5 install script to not restart it when installing the KeySafe 5 agent.

```
$ sudo /opt/nfast/keysafe5/sbin/install
```

Otherwise, use the nShield install script which will start both the nShield Security World software and the KeySafe 5 agent.

```
$ sudo /opt/nfast/sbin/install
```

9.8. Uninstall

```
helm --namespace nshieldkeysafe5 uninstall keysafe5-istio
helm --namespace nshieldkeysafe5 uninstall keysafe5-backend
helm --namespace nshieldkeysafe5 uninstall keysafe5-ui
helm --namespace rabbitns uninstall rabbit-chart
helm --namespace mongons uninstall mongo-chart
```

To uninstall the KeySafe 5 agent, run the KeySafe 5 uninstaller.

```
$ sudo /opt/nfast/keysafe5/sbin/install -u
```

10. Hardening The Deployment

To harden the demo deployment there are a number of steps to follow. The documentation below requires modifying the configuration of the Helm charts installed by following the Manual Install steps or running the `deploy.sh` script. To obtain the installed configuration for each installed Helm chart, run the following commands:

```
$ helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-backend > keysafe5-backend-values.yaml
$ helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-ui > keysafe5-ui-values.yaml
$ helm -n nshieldkeysafe5 get values --all --output yaml keysafe5-istio > keysafe5-istio-values.yaml
```

Documentation for each configurable value in the KeySafe 5 Helm charts can be found by untarring the chart.tgz and viewing the contents of either README.md or the default values.yaml file.

10.1. Certificates

The Manual Install steps and `deploy.sh` script will generate a local Certificate Authority, and install a number of short-lived demo certificates. These certificates must be replaced to con tinue using the system after their expiry.

Your new certificates will need to adhere to X.509 v3, sometimes known as a multiple-domain certificates, or SAN certificates. The X.509 extension Subject Alternative Name (SAN) allows specifying multiple hostnames, and has replaced Common Name as the source of the hostname.

10.1.1. External KeySafe 5 Server TLS Certificate

To update the TLS certificate used for the KeySafe 5 Server (for HTTPS connections to the REST API or User Interface) you must create a Kubernetes Secret containing the new certifi cate/private key and redeploy the `keysafe5-istio` Helm chart.

For more information on enabling HTTPS for helm-keysafe5-istio, see the Helm Chart Instal lation section of the Installation Guide.

The Manual Install steps and `deploy.sh` script create a Kubernetes Secret called `keysafe5-server-credential`. You can either delete the existing Secret as shown below, or use a dif-ferent name for the Secret containing your new TLS certificate.

```
$ kubectl --namespace istio-system delete secret keysafe5-server-credential

$ kubectl --namespace istio-system create secret tls keysafe5-server-credential \
  --cert=path/to/cert/file \
```

```
  --key=path/to/key/file
```

Before running `helm upgrade`, set the following values in your `keysafe5-istio-values.yaml`:

- httpsEnabled=true
- tls.existingSecret=keysafe5-server-credential (or the name you used when creating the Kubernetes Secret containing your certificate/private key)

```
$ helm upgrade --install keysafe5-istio \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-istio-values.yaml \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.2.0.tgz
```

10.1.2. Internal Certificates

The Manual Install steps and `deploy.sh` script create a Certificate Authority, and install KeySafe 5 using TLS authorised by the CA for the communications between the central plat form and MonogDB/RabbitMQ.

You can refresh these internal certificates by running the `updateinternalcerts.sh` script, specifying the number of days for which the new certificates will be valid.

The help for `updateinternalcerts.sh` is shown with the `-h` option:

```
  Usage: ./updateinternalcerts.sh [OPTION]... [SERVER] DAYS
  Generate certificates for SERVER that expire in DAYS days and update the
  KeySafe 5 internal TLS certificates for a deployment created by deploy.sh

  SERVER                       One of "mongodb" or "rabbitmq".
                               If the server is not specified then both will
                               be updated.
  DAYS                         Number of days before the generated certificates
                               expire
  -n, --namespace=NAMESPACE    The deploy script normally uses nshieldkeysafe5
                               as the namespace for the KeySafe 5 servers,
                               mongons for MongoDB, and rabbitns for RabbitMQ.
                               However you can override these to use a single
                               namespace for all the servers.  If that is the
                               case, then use this option.

  Certificate Addresses

  The certificates contain names and IP addresses that verify the connection.

  The backend servers use Kubernetes internal addressing to access the servers,
  and this script will set these up.

  MongoDB is only used by the backend services so does not require an external
  address.

  RabbitMQ is also accessed by the KeySafe 5 Agents through an external address.
  This script will use the IP address provided by the loadbalancer, but you may
  override this with a different IP address or DNS name.
```

```
-r, --rabbitmq=ADDRESS        The external address for RabbitMQ
-m, --mongodb=ADDRESS         The external address for MongoDB

When you update the certificates for RabbitMQ the script will generate an
updated agent-config.tar.gz and agent-config.zip files containing the updated
TLS certificates. You must then update all nShield client machines with a
KeySafe 5 agent installed to use the updated certificates and restart the
agent so that the new configuration is applied.

Examples

Refresh certificates for both servers for the next 30 days, and set a DNS
entry for RabbitMQ
  ./updateinternalcerts.sh -r rabbitmq.example.com 30

Refresh just MongoDB for the next 4 weeks
  ./updateinternalcerts.sh mongodb 28
```

This script will update both MongoDB and RabbitMQ TLS certificates, though you may choose to update only one set of TLS certificates by specifying only that server.

The `updateinternalcerts.sh` script must be run from a directory containing the KeySafe 5 Helm charts (for example, from the root directory of the untarred KeySafe 5 package). If the CA, created when the original deploy script was run, is not available then a new one will be created and used.

If both certificates have expired when running `updateinternalcerts.sh`, updating the certificates of only one service may return an error. In this case re-running `updateinternalcerts.sh` without specifying any server will update both server certificates at once, and will usually solve the problem.

If an error occurs during certificate update you can restore the previous setup by rolling back Helm chart installations to a previous release, see Helm Chart Upgrade in the Upgrade section of the Installation Guide.

10.1.2.1. MongoDB TLS Certificates

The Manual Install steps and `deploy.sh` script installs the Bitnami MongoDB Helm chart with TLS enabled for the connections between KeySafe 5 and the MongoDB server and also with TLS used for authentication. These certificates will initially be valid for 30 days from the time the process was run.

You can refresh the MongoDB certificates by running the `updateinternalcerts.sh` script, specifying the number of days the new certificates will be valid for.

```
$ updateinternalcerts.sh mongodb 365
```

This script will:

- Generate the new TLS certificates
- Update the MongoDB helm chart to use the new certificates
- Update the `keysafe5-backend` helm chart to use the new certificates

You may specify an external address with the `-m` parameter but this is usually not required.

10.1.2.2. RabbitMQ

The Manual Install steps and `deploy.sh` script installs the Bitnami RabbitMQ Helm chart with TLS enabled.

You can refresh the RabbitMQ certificates by running the `updateinternalcerts.sh` script, specifying the number of days the new certificates will be valid for.

```
$ updateinternalcerts.sh rabbitmq 365
```

This script will:

- Generate the new TLS certificates
- Update the RabbitMQ helm chart to use the new certificates
- Update the `keysafe5-backend` helm chart to use the new certificates
- Output a new `agent-config.tar.gz` that contains the agent configuration file and TLS certificates for authentication to RabbitMQ

The external address that will be put into the certificate, and the configuration file, will be the IP address of the Ingress service provided by the Kubernetes server. If you wish to use a different IP address or DNS name, this can be passed in with the `-r` parameter.

You will need to update your client machines with a KeySafe 5 agent installed to use the updated config and restart the agent so that the new configuration is applied.

```
$ sudo tar xf agent-config.tar.gz -C /opt/nfast/keysafe5/
$ /opt/nfast/scripts/init.d/keysafe5-agent restart
```

10.2. Authentication

If you chose to install the demo deployment without authentication you should enable

authentication for accessing the KeySafe 5 REST API and User Interface.

For how to configure authentication for the KeySafe 5 REST APIs see `Helm Chart Installation: helm-keysafe5-istio authentication` in the Installation Guide.

To update the `keysafe5-istio` Helm chart installed by the demo deployment, set the following values in `keysafe5-istio-values.yaml`.

- requireAuthn=true
- issuer[0].authIssuer="https://foobar.auth0.com"
- issuer[0].authJWKsURI="https://www.googleapis.com/oauth2/v1/certs"
- issuer[0].authAudiences[0]="https://keysafe5.location"

Then run `helm upgrade`.

```
$ helm upgrade --install keysafe5-istio \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-istio-values.yaml \
  --wait --timeout 1m \
helm-charts/nshield-keysafe5-istio-1.2.0.tgz
```

To update the `keysafe5-ui` Helm chart installed by the demo deployment, set the following values in `keysafe5-ui-values.yaml`:

- authMethod=oidc

Untar the chart and copy your OIDC provider config file (`OIDCProviders.json`) into the config directory:

For more details on how to populate `OIDCProviders.json` and how to configure authentication for the KeySafe 5 User Interface see `Helm Chart Installation: Configure UI authentication` in the Installation Guide.

```
$ tar -xf helm-charts/nshield-keysafe5-ui-1.2.0.tgz -C helm-charts
$ cp my-oidc-provider-config.json helm-charts/nshield-keysafe5-ui/config/OIDCProviders.json
```

Then run `helm upgrade`:

```
$ helm upgrade --install keysafe5-ui \
  --namespace=nshieldkeysafe5 \
  --values keysafe5-ui-values.yaml \
  --wait --timeout 3m \
helm-charts/nshield-keysafe5-ui
```

10.3. K3s

If not using your own Kubernetes cluster, the `deploy.sh` script will create one using K3s. To harden this K3s install follow the official documentation at K3s Hardening Guide.

The `deploy.sh` script installs K3s with `traefik` and `metrics-server` explicitly disabled.

11. Troubleshooting

11.1. Central platform

To view the KeySafe 5 application service logs, see Obtaining Logs.

If a Kubernetes resource is not working as expected, use `kubectl describe` to display any errors with that resource.

```
$ kubectl describe pod nshield-keysafe5-0
...
81s         Warning   FailedMount        pod/nshield-keysafe5-0         MountVolume.SetUp failed for volume
"keysafe5-amqp-basicauth-volume" : secret "amqp-credentials" not found
```

You can also use `kubectl get events` to detect errors.

```
$ kubectl get events --all-namespaces
```

For more information on debugging Kubernetes applications, see the Kubernetes documentation here.

11.2. KeySafe 5 agent

If the agent fails to start, ensure that the configuration file is present at `%NFAST_DATA_HOME%/keysafe5/conf/config.yaml`.

If the configuration file is present but the agent still fails to start, see the Logging: KeySafe 5 agent section for instructions on accessing the log.

If you are using TLS, ensure that the private key and certificate files are present in `%NFAST_DATA_HOME%/keysafe5/conf/amqp/tls`.

If you are using TLS authentication, ensure that the username is specified in the X509 field expected by RabbitMQ, and the level of indirection configured on the RabbitMQ server is correct (see RabbitMQ: Certificate Chains and Verification Depth).

12. Obtaining Logs

12.1. Central platform

The KeySafe 5 application is configured to log to `stdout`. This means you can view logs by running standard `kubectl` commands.

To get the KeySafe 5 backend services logs run `kubectl get pods`

By default, the KeySafe 5 backend Helm chart will create multiple replicas of each service. The below example commands only retrieves the logs from the first replica of each service.

```
$ kubectl -n nshieldkeysafe5 logs nshield-keysafe5-0 codesafe-mgmt
$ kubectl -n nshieldkeysafe5 logs nshield-keysafe5-0 hsm-mgmt
$ kubectl -n nshieldkeysafe5 logs nshield-keysafe5-0 sw-mgmt
```

To get the KeySafe 5 UI logs.

```
$ UI_POD=$(kubectl -n nshieldkeysafe5 get pods -l app=keysafe5-ui-app -o jsonpath='{.items[0].metadata.name}')
$ kubectl logs $UI_POD
```

Because all logs are directed to `stdout`, you can integrate the application logs with third-party log monitoring tools such as Prometheus or Splunk.

12.2. KeySafe 5 agent

12.2.1. Linux

The KeySafe 5 agent log file is located at `/opt/nfast/log/keysafe5-agent.log`, unless configured otherwise.

12.2.2. Windows

The KeySafe 5 agent log file is located at `C:\ProgramData\nCipher\Log Files\KeySafe5-agent.log`, unless configured otherwise.

The KeySafe 5 Windows Service actions are emitted to the Windows event log under the `nShieldKeySafe5` source identifier.

You can use the `nshieldeventlog` utility to extract these log entries and output them to the

console or a text file.

```
nshieldeventlog.exe --source=nShieldKeySafe5
```

As required, specify the following parameters.

- `-c` | `--count`: The number of records read from the event log.

  The default is `10000`

- `-f` | `--file`: The output filename.

See the nShield Security World Software documentation for more information on the `nshieldeventlog` utility.

13. Database

All persistent data for KeySafe 5 is stored in the MongoDB database.

13.1. Databases

KeySafe 5 stores data in three different databases within MongoDB. One database for storing HSM Management related data, one database for storing Security World Management related data and one for storing CodeSafe Management related data.

The names of the databases used within MongoDB are defined by the `helm-keysafe5-backend` Helm chart.

Key	Description	Default value
codesafe_mgmt.dbName	Name of the database to use for storing persistent CodeSafe data	codesafe-mgmt-db
hsm_mgmt.dbName	Name of the database to use for storing persistent HSM data	hsm-mgmt-db
sw_mgmt.dbName	Name of the database to use for storing persistent Security World data	sw-mgmt-db

13.2. Collections

13.2.1. HSM Management database

KeySafe 5 stores nShield HSM related data in the following collections:

- hsms
- pools
- hosts
- hardservers
- features

13.2.2. Security World Management database

KeySafe 5 stores nShield Security World data in the following collections:

- worlds

For each Security World known to KeySafe 5, the following collections are automatically cre ated, where each collection name is prefixed by the ID of the Security World database record that the collection corresponds to:

- <id>_actions
- <id>_authorizations
- <id>_authorized_pools
- <id>_cards
- <id>_cardsets
- <id>_domains
- <id>_groups
- <id>_keys
- <id>_module_certs
- <id>_operations
- <id>_p11objects
- <id>_softcards

13.2.3. CodeSafe Management database

KeySafe 5 stores nShield CodeSafe related data in the following collections:

- images
- machines
- certificates
- certificatestatus
- operations
- steps

13.3. User Roles

MongoDB has the notion of roles, where each role has a defined set of allowed actions. A user of a MongoDB database can be given a role which then determines what the user can and cannot do to the data.

For details about MongoDB roles, see the MongoDB documentation.

From a security point of view we want to give KeySafe 5 as a user of the MongoDB data- base the least privileges which suffice for the functionality it requires from the MongoDB

database.

The documentation below details the minimum privileges required for a KeySafe 5 MongoDB user for each database created by KeySafe 5.

13.3.1. HSM Management database

The following actions are required by KeySafe 5 for the operation of MongoDB for the HSM Management collections:

- createIndex
- find
- insert
- remove
- update

The MongoDB administrator will configure the HSM Management database with the following actions and privileges for KeySafe 5 `hsm-mgmt-db-user` role:

```
use admin
db.createRole(
  {
    role: "hsm-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "hsm-mgmt-db", "collection": ""},
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
```

13.3.2. Security World Management database

As KeySafe 5 creates new collections in the Security World Management Database as new Security Worlds are introduced to the system, RBAC (Role-based access control) must be applied at the database level rather than individual collections.

The following actions are required by KeySafe 5 for the operation of MongoDB for the Security World Management collections:

- createIndex
- dropCollection
- find

- insert
- remove
- update

The MongoDB administrator will configure the Security World Management database with the following actions and privileges for KeySafe 5 `sw-mgmt-db-user` role:

```
use admin
db.createRole(
  {
    role: "sw-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "sw-mgmt-db", "collection": ""},
          "actions": ["createIndex", "dropCollection", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
```

13.3.3. CodeSafe Management database

The following actions are required by KeySafe 5 for the operation of MongoDB for the Code Safe Management collections:

- createIndex
- find
- insert
- remove
- update

The MongoDB administrator will configure the CodeSafe Management database with the following actions and privileges for KeySafe 5 `codesafe-mgmt-db-user` role:

```
use admin
db.createRole(
  {
    role: "codesafe-mgmt-db-user",
    privileges: [
        {
          "resource": {"db": "codesafe-mgmt-db", "collection": ""},
          "actions": ["createIndex", "find", "insert", "remove", "update"]
        },
      ],
    roles: []
  }
)
```

13.3.4. Creating a MongoDB user with the User-defined roles

The MongoDB administrator may create a user for the KeySafe 5 application to access the KeySafe 5 databases by using the `db.createUser` command in the MongoDB shell.

```
ks5_user = {
   "user" : "ks5username",
   "roles" : [
     {"role": "codesafe-mgmt-db-user", "db": "admin" },
     {"role": "hsm-mgmt-db-user", "db": "admin" },
     {"role": "sw-mgmt-db-user", "db": "admin" },
   ]
 }
> db.createUser(ks5_user)
```

Note that when using TLS authentication for MongoDB, the username needs to match the subject of the client certificate.

13.4. Authentication Methods

KeySafe 5 supports the following authentication mechanisms for access to the MongoDB server:

- No authentication
- SCRAM
- X.509 certificate authentication

The type of authentication is specified by `database.mongo.auth.type` value in the `helm-keysafe5-backend` Helm chart.

13.4.1. No Authentication

This option is used during development. It should not be used during production.

13.4.2. SCRAM

Using Salted Challenge Response Authentication Mechanism (SCRAM), MongoDB verifies the supplied credentials against the MongoDB's username, password and authentication database.

In the `helm-keysafe5-backend` Helm chart:

- `database.mongo.auth.type` must be set to `pwd`.

- `database.mongo.auth.existingSecret` must be set to the name of an existing Kubernetes Secret that contains the username and password to use (the Secret must contain a value for `username` and `password` keys).
- `database.mongo.auth.authDatabase` must be set to the name of MongoDB's authentication database.

13.4.3. X.509 Certificate Authentication

KeySafe 5 can use X.509 certificates instead of usernames and passwords to authenticate to the MongoDB database.

In the `helm-keysafe5-backend` Helm chart:

- `database.mongo.auth.type` must be set to `tls`.
- `database.mongo.tls.enabled` must be set to `true`.
- `database.mongo.tls.existingSecret` must be set to the name of an existing Kubernetes Secret that contains the TLS certificates to use (the Secret must contain the keys `tls.crt`, `tls.key` and `ca.crt`).

13.5. Backup

To be able to restore the KeySafe 5 application, Entrust recommends that you regularly backup the MongoDB database as suggested in the MongoDB documentation.

13.6. Maintenance

The KeySafe 5 application (`helm-keysafe5-backend` Helm chart) does not support having database collections removed while the application is running.

If deleting collections, or replacing the MongoDB server that KeySafe 5 uses, then please stop the `helm-keysafe5-backend` Helm chart before performing database maintenance and restart the application once the database maintenance is complete.

14. Metrics

The KeySafe 5 metrics endpoints return metrics in OpenMetrics text format (HTTP content-type "application/openmetrics-text"). This format is defined by the OpenMetrics Specification.

The following metric endpoints are available in this release of KeySafe 5.

Endpoint	Minimum KeySafe 5 version	Description	Metric Labels
/codesafe/v1/metrics	1.2	SEE Machine statistics for all running CodeSafe 5 machines	• uuid (Local machine UUID) • esn • package_name • direction (only applicable to metrics for the network link for an SEE Machine)

14.1. Integrations

Data from KeySafe 5 metrics endpoints can be imported into any observability tool capable of consuming the OpenMetrics format.

For most tools this consists of configuring your tooling to poll the metrics HTTP endpoint at a certain interval by providing the API endpoint to query along with any necessary authentication.

This section provides basic guidance for a selection of common tools. For more detailed instructions, consult the documentation for your specific tooling.

14.1.1. Prometheus

For Prometheus you must configure a `scrape config` to directly consume the KeySafe 5 metrics data into Prometheus.

An example scrape config for polling an unauthenticated KeySafe 5 CodeSafe metrics endpoint:

```
scrape_configs:
  - job_name: KeySafe 5 CodeSafe
    scrape_interval: 300s
    static_configs:
      - targets: ["example.keysafe5deployment.com"]
    metrics_path: "/codesafe/v1/metrics"
```

```
    scheme: https
```

14.1.2. Elastic Stack

Integration with the Elastic Stack (Elasticsearch and Kibana) can be achieved by using the OpenMetrics integration within Kibana. This involves configuring Metricbeat to report the metrics data to Elasticsearch via polling the KeySafe 5 metrics endpoint.

For more details, search for OpenMetrics in Elastic integrations or follow the step-by-step OpenMetrics integration guide within Kibana.

14.1.3. Splunk

For Splunk there is not currently a direct OpenMetrics integration. One possible approach is to configure an HTTP Event Collector and use an intermediary script to poll the KeySafe 5 metrics endpoint, then translate the API responses from KeySafe 5 into a format that can be submitted to the HTTP Event Collector endpoint in Splunk.

15. Supported TLS Cipher Suites

This appendix and the helm values.yaml file both use the OpenSSL project's identifiers for TLS Cipher Suites.

Recommended Cipher Suites: The Default List

The following TLS Cipher Suites are supported by KeySafe 5, and are configured for use by default. It is strongly recommended that this default set of cipher suites, or a subset of it, is used.

- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-CHACHA20-POLY1305
- ECDHE-RSA-CHACHA20-POLY1305

Less Secure Cipher Suites: Not Recommended

The following TLS Cipher Suites are supported by KeySafe 5, but only if explicitly configured for use by the user. These are less secure cipher suites and should only be configured for use after a thorough threat analysis of the operating environment.

- ECDHE-RSA-AES256-SHA
- ECDHE-RSA-AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA
- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES256-SHA
- AES128-SHA
- DES-CBC3-SHA

TLSv1.3 Cipher Suites: Not Configurable

The following TLS Cipher Suites are supported by KeySafe 5 and cannot be explicitly configured. These are all secure TLSv1.3 cipher suites.

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

- TLS_AES_128_GCM_SHA256