BeyondTrust

DevOps Secrets Safe 21.1
Getting Started

TC:10/14/2021

Table of Contents

Install DevOps Secrets Safe: Overview **4**
Install DevOps Secrets Safe on the Server 5
Prerequisites 5
Supported Databases 5
Installation Instructions 6
Upgrade Instructions 6
Install with a Certificate 6
Uninstall Instructions 7
Install the DevOps Secrets Safe CLI 8
Prerequisites 8
Install the Package with pip 8
Execute the CLI 8
Configure the Initial Context 8
Bash Autocompletion 9
Quick Start DevOps Secrets Safe 10
Manage DevOps Secrets Safe Users 10
Resource Name Restrictions 12
Manage DevOps Secrets Safe Applications 12
Manage Secrets and Scopes 14
Secret and Scope Maximums 15
Manage Metadata 15
Manage Safelists and IP Ranges 16
Safelist Model 16
IP Range Model 16
Install Nginx Ingress for Safelist Capability 20
Create Event Sink Configurations 20
Manage DevOps Secrets Safe CLI Contexts 20
Environment Variable Overrides 22
Security Model for DevOps Secrets Safe 23
DSS Encryption and the Master Key 23
Authentication and Authorization 23

Auditing 24
Threats to DSS Security 24
DevOps Secrets Safe Sizing and Performance 25
Secrets Safe Test Scenario 25
Deployment Environments 25
Audit Volume 26
Conclusions 26
DevOps Secrets Safe Licensing 27
License Data Contents 27
Apply and Update License - With Internet Access 27
Apply and Update License - Offline 28
View License Data 28
Terminate License 29

Install DevOps Secrets Safe: Overview

This guide provides instructions to install DevOps Secrets Safe on a server and install a client. It also covers how to initialize DevOps Secrets Safe, set up users, applications, and secrets, and manage aspects including metadata, safelists, and event sinks. There are also sections on security and product architecture.

> *For information about integration with third-party products, and configurations, including using APIs and managing access control, please see the How-To Guide at https://www.beyondtrust.com/docs/secrets-safe/index.htm.*

Install DevOps Secrets Safe on the Server

To install DevOps Secrets Safe, review the prerequisites, and then run the install script, as detailed below. Instructions to upgrade and uninstall are also noted below.

Prerequisites

1. Kubernetes cluster with version 1.14, 1.15, 1.16, 1.17. 1.18, 1.19, or 1.20 must be available to host the deployment.
2. Install **kubectl** and configure to allow full permissions to the cluster. The version of **kubectl** must be within one minor version of the cluster (above or below).
3. Install Helm v3 and initialize with the appropriate Role-Based Access Control (RBAC).
4. In order for the application to be reachable, configure an NGINX ingress controller in the cluster.
5. Provide BeyondTrust the DockerHub username of the installing user, for them to be given permission to pull the required images.

Note: *As a reference deployment, DevOps Secrets Safe has been tested on a three-node Kubernetes cluster, each with a minimum of 7GB of RAM.*

For more information, please see the following:

- *Install Kubectl on Linux at https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-linux*
- *Install Helm at https://helm.sh/docs/intro/install/#from-script*

Supported Databases

The following information provides details on supported database providers and any privileges required by the DevOps Secrets Safe database user.

Postgres

DevOps Secrets Safe currently supports Postgres 11+.

Minimum version: 11

The user specified in the DevOps Secrets Safe database connection string requires special privileges:

- **CREATEDB** (Unless the initial run of DevOps Secrets Safe points to a pre-existing database)
- **CREATE** (On the database DevOps Secrets Safe uses)
- **SELECT**
- **INSERT**
- **UPDATE**
- **DELETE**

Microsoft SQL Server

Installation Instructions

The **install.sh** script is a bash entry point that installs DevOps Secrets Safe through a series of **kubectl** calls and then a **helm install** call. Values in the file **values.yml** within the helm chart are the defaults for the install. The **install.sh** script itself can be supplied with values through arguments or environment variables, or interactively. Values passed by arguments override any other form, then environment variables are accepted, and finally, mandatory values not specified otherwise are requested interactively.

To see a list of accepted parameters, run the install script with the **--help** parameter.

```
./install.sh --help
```

Note: *If an installation does not complete successfully, run the uninstaller before running the installer again.*

Example: *Install DevOps Secrets Safe using a postgreSQL database:*

```
./install.sh --docker-hub-username docker-user --docker-hub-password dockerpass --docker-
hub-email docker-user@beyondtrust.com --database-type postgres --connection-string
'Server=secretssafe.database.beyondtrust.com;Database=secrets-safe;Port=5432;User
Id=postgresql-user@secretssafe;Password=postgresql-password;Ssl Mode=Require;'
```

Example: *Install DevOps Secrets Safe using a Microsoft SQL Server database:*

```
./install.sh --docker-hub-username docker-user --docker-hub-password dockerpass --docker-
hub-email docker-user@beyondtrust.com --database-type mssql --connection-string'
Server=10.10.10.10;Database=secrets-safe;User Id=sqluser;Password=sqlpass;'
```

Once the application is installed, a means to access it is also required. Currently, DevOps Secrets Safe is compatible with the NGINX Ingress controller.

Upgrade Instructions

To upgrade an existing DevOps Secrets Safe installation from a cluster, run the install script with the **--upgrade** parameter. This preserves all custom values entered for the release. Additional value overrides can be specified during the upgrade either with additional parameters or by modifying the values file prior to upgrade and specifying the **--values-from-file** flag.

```
./install.sh --upgrade
```

Install with a Certificate

Please see the **Certificates.md** file for instructions on how to mount a custom certificate in a Secrets Safe installation.

Uninstall Instructions

To remove a DevOps Secrets Safe installation from a cluster, run the uninstall script. The uninstall script removes all DSS data, containers, secrets, etc. from the cluster. This does not include removing the database.

```
./uninstall.sh
```

Install the DevOps Secrets Safe CLI

The DevOps Secrets Safe Command Line Interface (CLI), **ssrun**, is a Python package that wraps functionality exposed by the DevOps Secrets Safe API into a convenient client that is used to interact with the system.

Prerequisites

The DevOps Secrets Safe CLI is designed to run on any major platform supported by Python and that has Python 3.6 and pip3 or above available.

Install the Package with pip

The DevOps Secrets Safe CLI package, **secretssafe**, is installed and managed on a client machine by the Python package manager pip, through a WHL file supplied by BeyondTrust, and is located in the **CommandLineInterface** directory of the extracted archive.

Execute the following when running in a virtual environment:

```
pip install secretssafe-<version>-py3-none-any.whl
```

Conversely, execute the following when running outside a virtual environment:

```
pip3 install secretssafe-<version>-py3-none-any.whl
```

Execute the CLI

After a successful installation, the CLI can be run by executing the following from any location on the file system: **ssrun**

> ***Note:*** *If the* ***secretssafe*** *package is installed inside a virtual environment, the environment must be first activated for* ***ssrun*** *to be on the path and thus executable.*

Configure the Initial Context

Contexts allow for multiple DevOps Secrets Safe instances to be easily configured and accessed from a single client machine. On preliminary installation, execute the following to be prompted for details of the initial context:

```
ssrun context create
```

Follow the prompts to configure the DevOps Secrets Safe instance that the CLI initially interacts with. To view your configured clusters, execute the following:

```
ssrun context get
```

```
CURRENT    NAME       HOSTNAME/IP      PORT  API VERSION    SSL CA
*          localhost  localhost        8443  v1             false
```

The initial context is set to **current** (configuration to use during any other CLI action) on creation, and any subsequent contexts created can be configured as **current** with the following command:

```
ssrun context set-current -n <context_name>
```

In addition, specific environment variables can be used to override the current context:

Example:

```
export SECRETSSAFE_HOST=<IP address or hostname of Secrets Safe instance>
export SECRETSSAFE_PORT=<port of Secrets Safe instance>
```

Note: *The following variable is necessary if the certificate authority is not publicly trusted:*

```
export SECRETSSAFE_VERIFY_CA=<path_to_ca_cert>
```

The DevOps Secrets Safe CLI verifies the SSL certificate presented by the DSS instance. The **SECRETSSAFE_VERIFY_CA** environment variable or **SSL CA** context attribute specifies the path to the CA certificate that the DSS certificate is checked against.

If no **SECRETSSAFE_VERIFY_CA** is specified, the default certificate bundles provided by the Python requests library are used.

Certificate verification can be disabled by setting **SECRETSSAFE_VERIFY_CA=false**. We strongly discouraged this practive for production environments.

To use these environment variables by default, rather than by manually managing contexts, you can make them persistent in the shell environment. They can be stored in a users **~/.bashrc** file.

Example:

```
echo 'export SECRETSSAFE_HOST=1.1.1.1' >> ~/.bashrc
echo 'export SECRETSSAFE_PORT=443' >> ~/.bashrc
echo 'export SECRETSSAFE_VERIFY_CA=false' >> ~/.bashrc
source ~/.bashrc
```

*In this example, certificate verification has been set to **false**. While this is convenient for testing, we do not recommend this for production environments.*

Bash Autocompletion

The DevOps Secrets Safe CLI comes with the ability to configure bash autocompletion for ease of use and convenience. To install bash completion globally, execute the following:

```
ssrun completion bash > /etc/bash_completion.d/ssrun
```

This will allow any new bash instances to autocomplete the DevOps Secrets Safe CLI commands on demand. Sudo rights might be required to be able to write to **/etc/bash_completion.d/**.

Quick Start DevOps Secrets Safe

Note: *Before proceeding with this section, please ensure a new instance of DevOps Secrets Safe (DSS) is running and that the DSS Command Line Interface (CLI) is configured to communicate with it.*

1. Enter the following command to initialize DevOps Secrets Safe:

```
ssrun init
```

Set the desired password for the root user account in the DSS instance when prompted. The password must be at least 10 characters long. A successful call to initialize returns the master key for this DSS instance. Save this key to a file.

Note: *The remainder of this guide assumes that the root account password for the DevOps Secrets Safe instance has been set to* ***rootpassword*** *and that the master key has been saved to a file called* ***master.txt****.*

2. Unseal DevOps Secrets Safe:

```
ssrun unseal -f master.txt
```

All CLI commands aside from **Initialize** and **Unseal** are unavailable until the instance is unsealed. This command puts the DevOps Secrets Safe application into a state where secrets may be saved and retrieved.

3. Log in to DevOps Secrets Safe as root:

```
ssrun login -u root -p rootpassword
```

For more information please see the following:

- *"Install DevOps Secrets Safe: Overview" on page 4.*
- *"Install the DevOps Secrets Safe CLI" on page 8.*

Manage DevOps Secrets Safe Users

Before starting this section, ensure you have initialized, unsealed, and logged into DevOps Secrets Safe as root.

1. Create a new user:

```
ssrun user create -n NewUser -p NewUserPassword
```

Note: *Passwords must be a minimum of 10 characters in length.*

2. View the list of users:

```
ssrun user get -v
```

> ***Note:*** *The principal discovery mechanism accepts any subset of the URI* ***{identity_provider}/{principal_type}/{principal_name}/{principal_extension_data}****. Therefore, the URI above returns all internal users. Additionally, the (optional)* ***-v*** *flag can be used to get a full listing of principals or principal containers attributes. Otherwise, a slim view of each principal or principal container is returned.*

3. Create a secret:

```
echo -n "I love my test content" | ssrun secret create testsecret:mytestsecret
```

> ***Note:***
> *Whenever you reference a secret, the URI must be in the format* ***{scopePath}:{secretName}****. For example,* ***path/to/secrets:secretName****.*
>
> *The echo line may only be performed in bash and similar shells.*

4. Authorize the new user to read the secret:

```
ssrun authorization create -p principal/internal/user/NewUser -o read -a allow
secret/testsecret:mytestsecret
```

The create-authorization command accepts the following arguments:

The authorization command accepts the following arguments:

- **-p**: (Required). URI of the principal the access control is being applied to.

  A user's URI can be derived using the principal discovery mechanism detailed in step 2.

- **-o**: (Optional). Operations authorization applies to.

  Options are **create**, **read**, **update**, and **delete**.

  Options are **create**, **read**, **update**, **delete** and **grant**.

- **-a**: (Optional). Set to allow to grant authorization or deny to revoke.

5. Log in as the new user:

```
ssrun login -u NewUser -p NewUserPassword
```

6. Read the secret:

```
ssrun secret get testsecret:mytestsecret
```

7. Log in as root again:

```
ssrun login -u root -p rootpassword
```

8. Delete the new user:

```
ssrun user delete -n NewUser
```

Resource Name Restrictions

DSS enforces restrictions for all resource types.

- The valid characters for resources at large are:
  - a b c d e f g h i j k l m n o p q r s t u v w x y z
  - A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  - 0 1 2 3 4 5 6 7 8 9 @ : $ _ . + ! * ' ( ) -

Additionally, there are specific restrictions on user, application, and group names.

- The maximum number of characters in any user, application, or group name is 120.
- The valid characters for user, application, and group names are:
  - a b c d e f g h i j k l m n o p q r s t u v w x y z
  - A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  - 0 1 2 3 4 5 6 7 8 9 - . _ @ +

Manage DevOps Secrets Safe Applications

Before starting this section, ensure you have initialized, unsealed, and logged into DevOps Secrets Safe as root.

1. Create a new application:

```
ssrun application create -n NewApplication
```

> ***Note:*** *Upon creation, an API key is returned. This is used in any subsequent log in.*

2. View the list of applications:

```
ssrun application get -v
```

> ***Note:*** *The principal discovery mechanism in the API accepts any subset of the URI* ***{identity_provider}/{principal_type}/{principal_name}/{principal_extension_data}****. Therefore, the command above returns all internal applications. Additionally, the (optional)* ***-v*** *flag can be used to get a full listing of principals or principal containers attributes. Otherwise, a slim view of each principal or principal container is returned.*

3. Create a secret:

```
echo -n "I love my test content" | ssrun secret create testsecret:mytestsecret
```

> ***Note:***
> *Whenever you reference a secret, the URI must be in the format **{scopePath}:{secretName}**. For example, **path/to/secrets:secretName**.*
>
> *The echo line may only be performed in bash and similar shells.*

4. Authorize the new application to read the secret:

```
ssrun authorization create -p principal/internal/application/NewApplication -o read -a allow
secret/testsecret:mytestsecret
```

The **authorization** command accepts the following arguments:

- **-p**: (Required). URI of the principal the access control is being applied to.

  An applications URI can be derived using the principal discovery mechanism detailed in step 2.

- **-o**: (Optional). Operations authorization applies to.

  Options are **create**, **read**, **update**, **delete**, and **grant**.

- **-a**: (Required). Set to allow to grant authorization or deny to revoke.

  Options are **allow** and **deny**.

5. Log in as the new application:

```
ssrun login -a NewApplication -k 2a098f21-0b11-4918-b705-7752588d5d8c
```

> ***Note:** The API key **-k** comes from what was returned when the application was created in step 1.*

6. Read the secret:

```
ssrun secret get testsecret:mytestsecret
```

7. Log in as root again:

```
ssrun login -u root -p rootpassword
```

8. Delete the new application:

```
ssrun application delete -n NewApplication
```

> ***Note:*** *The name associated with an application can be determined via the list applications command as detailed in step 2.*

Manage Secrets and Scopes

Before starting this section, ensure you have initialized, unsealed, and logged into DevOps Secrets Safe as root.

The next example assumes there are two files, **myTestSecretData1.txt** and **myTestSecretData2.txt**, containing data you want to store as secrets.

1. Create two secrets:

```
ssrun secret create -f myTestSecretData1.txt path/to/my/secrets:mytestsecret1
```

```
ssrun secret create -f myTestSecretData2.txt path/to/my/secrets:mytestsecret2
```

> ***Note:*** *Whenever you reference a secret, the URI must be in the format* ***{scopePath}:{secretName}****. For example,* ***path/of/scope:secretName****.*

2. Retrieve the list of secret names for a given scope:

```
ssrun scope get path/to/my/secrets
```

The next example assumes there is a file called **updatedMyTestSecretData1.txt** containing the data you want to use to update this secret.

3. Update a secret:

```
ssrun secret update -f updatedMyTestSecretData1.txt path/to/my/secrets:mytestsecret1
```

4. Retrieve a secret:

```
ssrun secret get path/to/my/secrets:mytestsecret1
```

5. Retrieve all secrets under a scope and save them in the directory **my_secret_dir**

```
ssrun secret get path/to/my/secrets -d my_secret_dir
```

6. Remove a secret:

```
ssrun secret delete path/to/my/secrets:mytestsecret1
```

> ***Note:*** *This not only removes the secret but also all metadata that is associated with it.*

7. Remove a scope:

```
ssrun scope delete path/to/my/secrets
```

> ***Note:*** *This not only removes the scope but also all scopes, secrets and metadata that are children of it.*

Secret and Scope Maximums

DSS enforces a maximum size for secret and scope names:

- The maximum number of characters in any path segment is 1024. A segment is a string between two forward-slash (/) characters.
- The maximum number of segments in any scope path is 100.

Manage Metadata

Before starting this section, ensure you have initialized, unsealed, and logged into DevOps Secrets Safe as root.

> ***Note:*** *Metadata is currently supported for secret and scope resource types.*

1. Create metadata for a secret:

```
ssrun metadata create -n mytestsecret1Meta1Name -v meta1Value
secret/path/to/my/secrets:mytestsecret1
```

> ***Note:*** *When managing metadata, to reference a scope, set the URI to its path. For example,* ***path/of/scope****. To reference a secret, use* ***{path}:{secretName}****. For example,* ***path/of/scope:secretName****.*

2. Update metadata for a secret:

```
ssrun metadata update -n mytestsecret1Meta1Name -v updatedMeta1Value
secret/path/to/my/secrets:mytestsecret1
```

3. View metadata for a secret:

```
ssrun metadata get -n mytestsecret1Meta1Name secret/path/to/my/secrets:mytestsecret1
```

> ***Note:*** *The above command retrieves only the information associated with the metadata item named* ***mytestsecret1Meta1Name****. To retrieve the information for all metadata items associated with a scope or secret, omit the* ***-n*** *argument.*

4. Remove metadata:

```
ssrun metadata delete -n mytestsecret1Meta1Name secret/path/to/my/secrets:mytestsecret1
```

Manage Safelists and IP Ranges

Before starting this section, ensure you have initialized, unsealed, and logged into DevOps Secrets Safe as root.

Safelists allow you to explicitly grant or deny access to specific IP addresses for all CLI commands. Safelists and IP ranges must be structured in the following way:

Safelist Model

- **Name**: (Required). Name for this safelist.
- **Description**: (Optional). Details about this safelist.
- **Expiry date**: (Optional). Specifies a day and time when this safelist will expire.

  An empty or null value denotes no expiry.

IP Range Model

- **Name**: (Required). Name for this IP range.
- **Value**: (Required). Specifies a range of IP addresses.

  The supported IP range value patterns are:

  - CIDR range: 192.168.0.0/24, fe80::%lo0/10
  - Single address: 10.101.8.16, fe80::1%23
  - Begin-end range: 10.101.8.10 - 10.101.8.20, fe80::1%23 - fe80::ff%23
- **Allow**: (Required). Specifies whether the defined range of IP addresses allows or denies access.
- **Description**: (Optional). Details about this IP range.
- **Expiry date**: (Optional). Specifies a day and time for the IP range to expire.

  An empty or null value denotes no expiry.

***Note:** A safelist must have at least one IP range associated with it.*

1. Create two safelists:

```
ssrun safelist create -f safelist1.txt
```

```
ssrun safelist create -f safelist2.txt
```

The following examples assume there are two files, **safelist1.txt** and **safelist2.txt**, with the given contents:

Example:

safelist1.txt

```
{
   "ipRanges": [
      {
         "name": "ip_range_1",
         "value": "10.101.8.10-10.101.8.20",
         "allow": true,
         "description": "IP Range 1 Description",
         "expiryDate": "2020-06-21T11:44:31.733Z",
         "xForwardedForHeaderLimit": "2"
      }
   ],
   "name": "safelist_1",
   "description": "Safelist 1 Description",
   "expiryDate": "2020-06-21T11:44:31.733Z"
}
```

Note: *In the above example, the safelist is enforced only until the defined expiry date and allows only IP addresses in the range of 10.101.8.10 to 10.101.8.20.*

Example:

safelist2.txt

```
{
   "ipRanges": [
      {
         "name": "ip_range_2",
         "value": "10.101.8.50-10.101.8.60",
         "allow": false,
         "description": "IP Range 2 Description"
      }
   ],
   "name": "safelist_2",
   "description": "Safelist 2 Description"
}
```

Note: *In the above example, the safelist never expires and denies IP addresses in the range of 10.101.8.50 to 10.101.8.60.*

2. View safelists and IP ranges:

   The **safelist get** command returns all safelists that exist.

```
ssrun safelist get
```

You can also limit the view by passing in the name of the safelist targeted for discovery.

```
ssrun safelist get -n safelist_1
```

The **ip-range get** command returns all the IP ranges that exist for a given safelist.

```
ssrun ip-range get -n safelist_1
```

You can also limit the view by passing in the name of the IP range targeted for discovery.

```
ssrun ip-range get -n safelist_1 -i ip_range_1
```

Views can be further modified by using the following flags:

- **-d**: (Depth). Use this to define the maximum depth of the view to return.
  - A value of **0** returns only the element specified.
  - A value of **1** returns the element specified and all direct child elements
  - A value of **2** returns all child and grandchild elements of the element specified.
- **-v**: (Verbose). Use this to get a full listing of safelists and/or IP range attributes; otherwise, a slim view of each safelist or IP range is returned.

3. Update a safelist:

```
ssrun safelist update -n safelist_2 -f safelist2Update.txt
```

This command updates the safelist with the name **safelist_2**.

The following example assumes there is a file called **safelist2Update.txt** with the given contents:

Example:

safelist2Update.txt

```
{
    "description": "Safelist 2 Description Updated",
    "expiryDate": "2021-06-21T12:17:14.326Z"
}
```

4. Add an IP range to a safelist:

```
ssrun ip-range create -n safelist_2 -f ipRange.txt
```

This command adds an IP range to the safelist with the name **safelist_2**.

The following example assumes there is a file called **ipRange.txt** with the given contents:

Example:

ipRange.txt

```
{
    "value": "10.101.8.70",
    "allow": false,
    "description": "IP Range 3 Description",
    "expiryDate": "2021-06-21T11:58:03.315Z"
}
```

Note: *In the above example, the IP range is only enforced until the defined expiry date and denies IP requests coming from the IP address 10.101.8.70.*

5. Update an IP range of a safelist:

```
ssrun ip-range update -n safelist_2 -i ip_range_2 -f ipRangeUpdate.txt
```

This command updates the IP range with the name **ip_range_2** for the safelist with the name **safelist_2**.

The following example assumes there is a file called **ipRangeUpdate.txt** with the given contents:

Example:

ipRangeUpdate.txt

```
{
    "value": "10.101.8.71",
    "allow": false,
    "description": "IP Range 3 Updated",
    "expiryDate": "2021-06-21T11:58:03.315Z"
}
```

6. Assign a safelist to a user:

```
ssrun authorization create -p principal/internal/user/user1 -o read -a allow
safelist/safelist_2/access
```

This command associates the safelist with the name **safelist_2** to the user with the name **user1**.

7. Delete an IP range from a safelist:

```
ssrun ip-range delete -n safelist_2 -i ip_range_2
```

This command deletes the IP range with the name **ip_range_2** from the safelist with the name **safelist_2**.

8. Delete a safelist:

```
ssrun safelist delete -n safelist_2
```

This command deletes the safelist with the name **safelist_2**.

Install Nginx Ingress for Safelist Capability

The DevOps Secrets Safe application is compatible with the Nginx Ingress Controller.

To install this ingress controller from the official helm chart for a bare metal deployment, run the following command:

```
helm install ingress-nginx ingress-nginx/ingress-nginx --namespace kube-system --set
controller.hostNetwork=true --set rbac.create=true --set controller.kind=DaemonSet --version
3.24.0
```

To install this ingress controller from the official helm chart for an Azure deployment, run the following command:

```
helm install ingress-nginx ingress-nginx/ingress-nginx --namespace kube-system --set
controller.service.externalTrafficPolicy=Local --set controller.replicaCount=3 --version 3.24.0
```

Note: *The* ***--set controller.service.externalTrafficPolicy=Local*** *option is added to the* ***helm*** *install command for safelist enforcement purposes. This enables client source IP preservation for requests to containers in your cluster. If you are not planning on using safelist enforcement, this option can be excluded.*

For more information about the Nginx Ingress Controller, please see Charts at https://charts.helm.sh/stable.

Create Event Sink Configurations

Create an event sink configuration by entering this command:

```
ssrun event-sink create -f myconfig.json
```

This command creates an event sink configuration using the provided JSON file.

For detailed instructions on event sink configuration, please see Event Sinks at https://www.beyondtrust.com/docs/secrets-safe/how-to/configuration/event-sinks.htm.

Manage DevOps Secrets Safe CLI Contexts

Contexts are CLI-specific configurations that allow you to access multiple instances of DevOps Secrets Safe from a single client machine. CLI contexts exist only on the client side and only tell the CLI where to access the DevOps Secrets Safe instance. They do not interact with the instance in any way on their own.

Example: *Assume you want to interact with two instances of DevOps Secrets Safe, one in staging and one in production, and you want to take the value of a secret from your staging DevOps Secrets Safe and save it to your production instance. In this example, your staging instance has an IP address of* ***164.223.32.59*** *and your production instance has an IP address of* ***164.225.37.62****.*

1. *Create a context pointed at staging:*

```
ssrun context create -n staging -a 164.223.32.59 -p 443 -s false -v v1
```

2. *Create a context pointed at production:*

```
ssrun context create -n production -a 164.225.37.62 -p 443 -s true -v v1
```

3. *Set the staging context to active:*

```
ssrun context set-current -n staging
```

4. *List all contexts:*

```
ssrun context get
```

```
CURRENT  NAME       HOSTNAME/IP    PORT  API VERSION  SSL CA
*        staging    164.223.32.59 443   v1           false
         production 164.225.37.62 443   v1           true
```

Note: *The asterisk () in the* ***CURRENT**** *column shows it is the active context.*

5. *Log on to the staging instance:*

```
ssrun login -u my_staging_user -p my_staging_user_password
```

6. *Save secret from staging DevOps Secrets Safe instance to your file system:*

```
ssrun secret get path/to/staging:secret -f mysecret
```

7. *Switch contexts so your CLI is pointed at the production DevOps Secrets Safe instance:*

```
ssrun context set-current -n production
```

8. *Log in as a user from the production DevOps Secrets Safe instance:*

```
ssrun login -u my_production_user -p my_production_user_password
```

9. *Create a new secret on the production instance, storing the value retrieved from the staging instance:*

```
ssrun secret create path/to/production:secret -f mysecret
```

Tip: *To learn more about DevOps Secrets Safe CLI contexts you can use the* ***-h*** *flag (****$ ssrun context -h****).*

Environment Variable Overrides

Specific environment variables can override the current configured context.

IMPORTANT!

If any of the environment variables below are defined, they override what is in the current context.

```
export SECRETSSAFE_HOST=(IP address or hostname of DevOps Secrets Safe instance)
```

```
export SECRETSSAFE_PORT=(port of DevOps Secrets Safe instance)
```

```
export SECRETSSAFE_VERIFY_CA=(bool indicating if ca should be verified)
```

Security Model for DevOps Secrets Safe

The three components considered in the security model are: the DSS client, the DSS server, and the persisted datastore external to the DSS server.

All communication between DSS clients and DSS server is encrypted using Transport Layer Security (TLS). Secrets at rest in the persisted datastore are encrypted using 256-bit Advanced Encryption Standard - Galois/Counter Mode (AES-GCM) encryption.

DSS stores its encrypted and plaintext data in the persisted datastore external to the DSS product. Even though the secret data is encrypted in the datastore, access to the DSS database must be strictly limited to maintain security for the system.

DSS Encryption and the Master Key

DSS encrypts secrets using 256-bit AES-GCM cryptography. The encryption keys used for secrets are themselves protected using RSA encryption with a 3072-bit key. This key is called the *master key*.

The first time a DSS instance is launched, it starts in an uninitialized and sealed state. There is no master key yet, and no secret encryption keys are present. Initialization is required to set the initial root password and to generate the master key.

An initialization request to the DSS API specifies the initial root password and causes generation of the master key that is returned from the API. The master key must be handled securely, as it is the only piece of information required to unseal the DSS instance.

- The master key is returned as a JSON blob containing the RSA private key.
- A copy of this key must be retained by the operator as the DSS instance cannot be unsealed without it.
- Immediately after initialization, the DSS instance is still sealed and most endpoints remain unavailable.

Unsealing DSS using the master key is required after initialization to enable the system's cryptographic services. The system becomes unsealed when the master key is provided to the API in an unseal request. Encryption keys are only available to DSS when the system is operating in its unsealed state.

The DSS instance returns to the sealed state when any of the following occur:

- DSS is restarted
- DSS is upgraded
- When a seal operation is commanded via the API

Authentication and Authorization

DSS enforces that all API requests aside from initialization, unsealing, and login require authentication. Authentication is verified using a token in the header of requests sent to the DSS API. Tokens are obtained for users from the internal identity provider by user name and password, or for applications using API key credentials. Multi-factor authentication can be configured for users. External identity providers can be configured, enabling use of different credential types and use of identity information sourced from external systems.

Authenticated DSS principals can only access resources in DSS they have permissions for. Access rights to DSS resources are computed using Access Control Entries (ACE) that consist of three elements:

- Principal: The identity of a user, application, or group in DSS.
- Resource: The URI path for a resource in DSS.
- Operation: The available operations are **Create**, **Read**, **Update**, **Delete**, and **Grant**.

Each access control entry can indicate that the operation is *allowed* or can be an explicit denial entry. Any denial rule takes precendence over conflicting *allow* rules.

Authorization can be further restricted based on the requesting client's IP address using DSS's safelist functionality.

Auditing

DSS can generate an audit event log for all API requests and responses. Auditing for DSS is configured using the **Event Sink API**. Audit events can be delivered as text to the console or as structured events to an aggregator like ELK or syslog.

All system actions undertaken to service a request are audited, and audit events for a particular request are associated by a common Correlation ID.

Log aggregation and visualization tools such as **Elasticsearch** and **Kibana** provide a convenient means for exploring audit information sent from DSS.

Threats to DSS Security

The following threats to DSS security cannot be mitigated by the application itself. Mitigation of these threats requires proper administration and security controls for the DSS server and its external datastore.

1. DSS cannot prevent against disruption of the system in the event that an attacker gains control of the persisted datastore external to the DSS server. In the event that the security of the DSS datastore is compromised, an operator with arbitrary control of the DSS database can circumvent the product's security in ways that cannot be mitigated. For example:
    - The attacker could delete the database entirely, or selectively.
    - The attacker could roll back database changes to an earlier state.
    - The attacker could infer the existence of secret material even though the secret contents remain encrypted.

IMPORTANT!

Security of the external datastore must be strictly maintained to protect against these threats.

2. DSS cannot protect against memory analysis of the DSS server application software. An attacker with root access to the DSS cluster nodes and the ability to perform runtime analysis could feasibly compromise sensitive data secured by DSS.

DevOps Secrets Safe Sizing and Performance

DevOps Secrets Safe architecture was designed from inception to provide flexibility and scalability. The system is made up of a series of distributed services deployed as containers using Kubernetes. By its nature, the performance of the system varies greatly depending on the environment it is running in. To provide an idea of the performance that can be expected from the system, a reference deployment was used to gather performance statistics.

Secrets Safe Test Scenario

A test scenario was created using JMeter. JMeter used 200 threads to simultaneously iterate a list of users. For each user in the list, an authentication was performed and a secret was retrieved. The test continued iterating the list of users for a period of ten minutes.

The following data was loaded into DevOps Secrets Safe prior to the execution of the test. After the data was loaded, the resulting database was approximately 100MB in size.

- 20,000 secrets, each 1024 bytes in size
- 1000 local user accounts
- Access to the secrets was granted to each user

> ***Note:*** *All audit, logs, and performance telemetry generated by Secrets Safe during the test are forwarded to an external Elasticsearch instance.*

Deployment Environments

DevOps Secrets Safe was tested in both a cloud hosted as well as an on premises virtualized environment.

Azure Environment

The Azure environment consisted of the following resources:

Service	Version	VM Host Specs
Azure Kubernetes	1.14.8	• 3 x D4s_v3 • 4 vCPU • 16GB RAM • 6400 Max IOPS
Azure Database for PostgreSQL	11	• General Purpose • 4 vCores • Local Redundant

Using this configuration, DevOps Secrets Safe can handle approximately 270 incoming secret requests per second, or approximately 170,000 requests over a 10 minute period.

On-Premises ESXi Environment

The on-premises environment consisted of the following resources:

Service	Version	VM Host Specs
Kubernetes cluster	1.15.5	• 3 x CentOS 7.7 • 4 Cores • 16GB RAM
PostgreSQL	10.10	• 1 x CentOS 7.7 • 4 Cores • 16GB RAM

Using this configuration, DevOps Secrets Safe can handle approximately 270 incoming secret request per second, or approximately 170,000 request over a 10 minute period.

Audit Volume

Action	Number of Audit Events
User Authentication	3
Secret Reterival	3
Full Performance test	~500,000 (325MB in Elasticsearch)

Conclusions

The performance of DevOps Secrets Safe's reference deployment should be enough for most small to medium-sized customers. For larger customers, some further horizontal scaling might be required. Due to DevOps Secrets Safe's underlying architecture, this is easily achievable by adding additional Kubernetes nodes and increasing the number of replicas for the appropriate services.

DevOps Secrets Safe Licensing

DevOps Secrets Safe can run unlicensed for an evaluation period of 30 days upon being unsealed for the first time. To continue using DSS after this evaluation period, you must generate licensing information using a serial number from BeyondTrust.

License Data Contents

Each of the licensing commands described below return a common data structure detailing the current state of the DevOps Secrets Safe instance license.

```
{
  "SerialNumber": "AAAAA-11111-BBBBB-22222-CCCCC-33333",
  "LicenseKey": "ZZZZZZZZ-YYYYYYYY-11111111-99999999-XXXXXXXX-88888888",
  "ExpiryDate": "2020-10-14T16:39:17Z",
  "IsEvaluation": false,
  "ActiveLicenseCount": 2,
  "AllowedLicenseCount": 2147483647
}
```

Each attribute describes the following:

- **SerialNumber**: The serial number used to license this instance of DSS.
- **LicenseKey**: License key generated using this serial number.
- **ExpiryDate**: The expiry date of this serial number.
- **IsEvaluation**: **True** if this serial number is an evaluation serial number. Otherwise **false**.
- **ActiveLicenseCount**: Number of currently active instances of DSS using this serial number.
- **AllowedLicenseCount**: Number of instances of DSS allowed for this serial number.

Apply and Update License - With Internet Access

Apply License During Unseal

Enter the serial number using the **-s** parameter with the unseal command, to update your current serial number, or to apply a serial number for the first time.

Example:

```
ssrun unseal -f masterkey -s AFZTG-FDRJC-6WQFU-2KIHC-G44BS-EAF65
```

Note: *If an existing serial number is in place, it is decremented for this instance of DevOps Secrets Safe before the new serial number is applied.*

Update License

Enter the serial number using the **s** parameter with the **license update** command to update your current serial number.

Example:

```
ssrun license update -s AAAAA-BBBBB-11111-22222-CCCCC-33333
```

Note: *If an existing serial number is in place, it is decremented for this instance of DevOps Secrets Safe before the new serial number is applied.*

Apply and Update License - Offline

If internet access is not available or restricted from your DevOps Secrets Safe instance, you can generate a license key to use offline.

To generate an offline license key:

1. Access a machine that has Internet access.
2. Browse to the offline licensing form at licensing.beyondtrust.com.
3. Enter the serial number that was provided to you and submit.
4. Copy the license key that was generated for your instance of DevOps Secrets Safe.

Apply License During Unseal

Enter the serial number using the **-s** parameter, and the license key using the **l** parameter, with the unseal command, to update your current serial number or to apply a serial number for the first time.

Example:

```
ssrun unseal -f masterkey -s AAAAA-BBBBB-11111-22222-CCCCC-33333 -l ZZZZZZZZ-YYYYYYYY-
11111111-99999999-XXXXXXXX-88888888
```

Update License

Enter the serial number using the **s** parameter, and the license key using the **l** parameter, with the **license update** command, to update your current serial number.

Example:

```
ssrun license update -s AAAAA-BBBBB-11111-22222-CCCCC-33333 -l ZZZZZZZZ-YYYYYYYY-
11111111-99999999-XXXXXXXX-88888888
```

View License Data

The current state of the license of your instance of Secrets Safe can be retrieved by using the **license get** command:

```
ssrun license get
```

You can also use the force flag **-f** to make the system request an update of the licensing information from the licensing server.

Terminate License

To terminate your existing license, use the **license delete** command:

```
ssrun license delete
```

Note: *This removes all licensing data from this instance of DevOps Secrets Safe and decrements the active instance count for the associated serial number.*