Integration of Snowflake with CRDP

Snowflake [Overview]

This document describes how to configure and integrate CipherTrust Manager with Snowflake. Snowflake's Data Cloud is powered by an advanced data platform provided as a self-managed service. Snowflake enables data storage, processing, and analytic solutions that are faster, easier to use, and far more flexible than traditional offerings.

The Snowflake data platform is not built on any existing database technology or "big data" software platforms such as Hadoop. Instead, Snowflake combines a completely new SQL query engine with an innovative architecture natively designed for the cloud. Snowflake provides all of the functionality of an enterprise analytic database, along with many additional special features and unique capabilities.

Thales provides three different methods to protect sensitive data in Snowflake.

Bring Your Own Encryption (BYOE)

- **Data Ingest** – with Thales Batch Data Transformation (BDT)
- **Data Access** – external remote user defined functions for column level encrypt and decryption using Thales CRDP and tokenization using Thales CT-VL.

Bring/Hold Your Own Key (BYOK) (HYOK)

- **Snowflake Tri-Secret Secure** – with Thales CM CCKM BYOK and HYOK.

Secrets Management

- **Snowflake Snowpipe** – securing private RSA key in Thales CM for connecting to Snowflake using CRDP.
- **Snowflake Credentials** – Thales CipherTrust Secrets Manager (Akeyless) for secrets management in Snowfake.

The above methods are NOT mutually exclusive. All three methods can be used to build a strong defense in depth strategy to protect sensitive data in the cloud. The focus of this integration will be on Data Access protecting sensitive data in snowflake columns by using CRDP to create User Defined Functions (UDF) for encryption and decryption of sensitive data.

Overview

Snowflake has provided for many years the ability to use external functions to provide custom code to protect sensitive data residing in snowflake. Snowflake can run on all three major cloud service providers AWS, Azure and GCP. All three major CSP's provide the ability to create a function as a service (FAAS). AWS refers this as AWS Lambda Functions Google calls this GCP Cloud Functions and Azure calls them Azure Functions.

More recently snowflake has built a new capability called Snowpark Containerized Services (SPCS) is a fully managed container offering designed to facilitate the deployment, management, and scaling of containerized applications within the Snowflake ecosystem. This service enables users to run containerized workloads directly within Snowflake, ensuring that data doesn't need to be moved out of the Snowflake environment for processing. Snowpark Container Services reached General Availability (GA) on a regional basis, starting in August 2024.

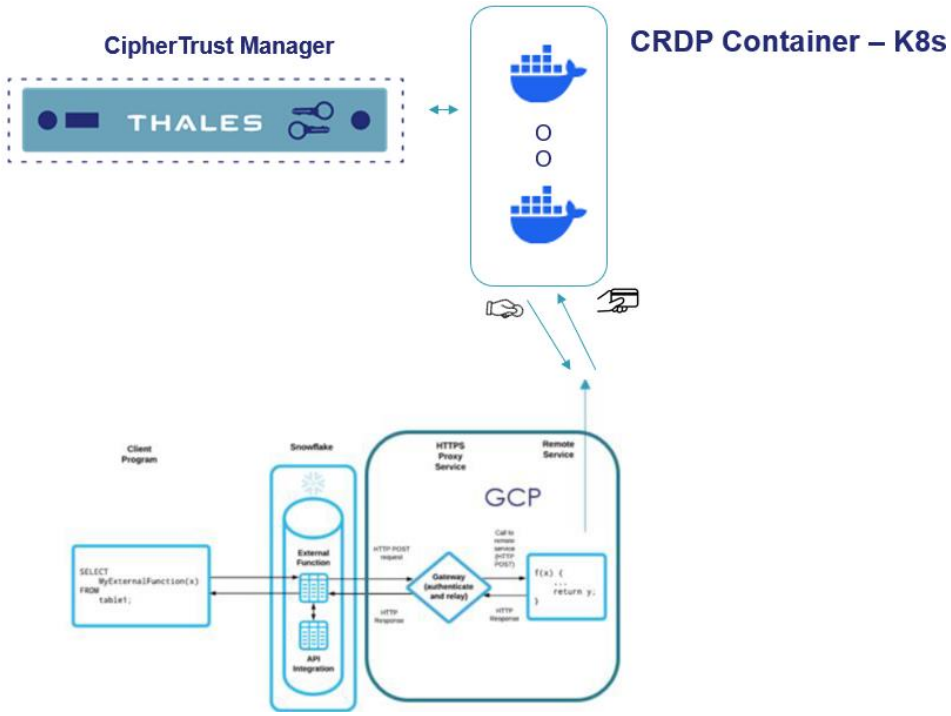
- **August 1, 2024: Snowflake announced the GA of Snowpark Container Services to accounts in all commercial AWS regions.**
- **February 3, 2025: Snowflake Native Apps with Snowpark Container Services support for Azure became generally available.**
- **August 1, 2025: Snowflake announced the GA of Snowpark Container Services in Google Cloud commercial regions.**

It's important to note that the GA date can vary depending on the cloud provider (AWS, Azure, GCP) and the specific region. The original Thales User Defined Functions were built using the External User Defined function method. This document will now include the steps needed to deploy either the External User Defined Functions or the Containerized Option. There are expected performance benefits to use the SPCS as documented by Snowflake. For more information on the differences see the Appendix section of this document.

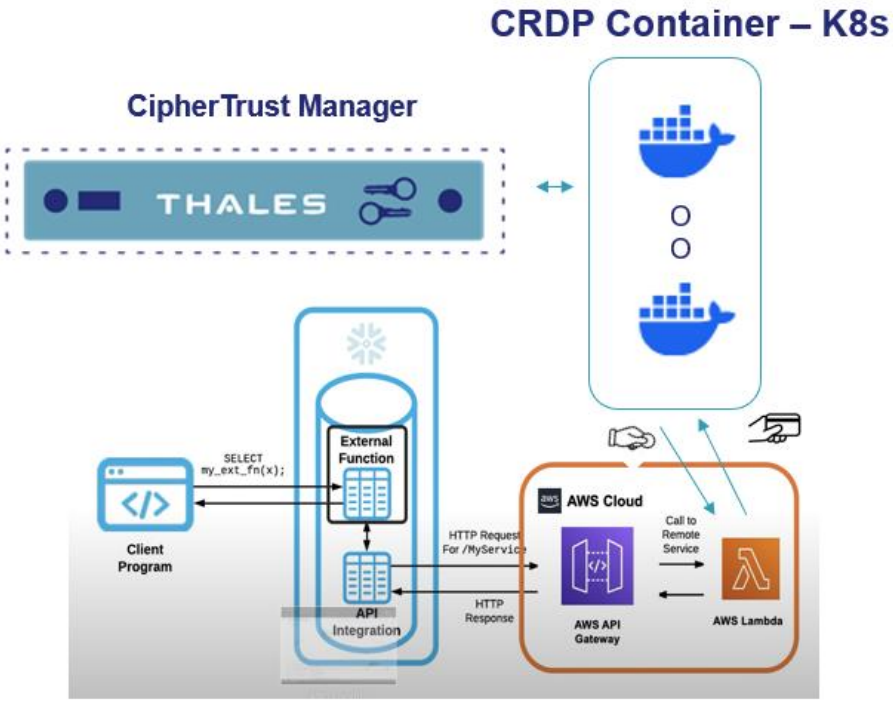
External User Defined Functions

The steps provided in this link contain examples of both GCP and AWS Functions. Listed below are examples of how this integration works for External User Defined Functions.

GCP Example Integration



AWS Example Integration



Supported Product Versions

- **CipherTrust Manager** CipherTrust Manager 2.14 and higher
- **CRDP** CRDP 1.0 and higher
- **Snowflake**

This integration is validated using AWS Lambda and Google Cloud Functions Java 11.

Prerequisites

- Steps performed for this integration were provided by this Snowflake link: https://docs.snowflake.com/en/sql-reference/external-functions
- Ensure that CRDP container is installed and configured. Refer to https://thalesdocs.com/ctp/con/crdp/latest/admin/crdp-deploy_alternative/index.html
- Ensure that the CipherTrust Manager is installed and configured. Refer to the **CipherTrust Manager documentation** for details.
- Integration with CipherTrust Manager. Assuming the CRDP container has already been deployed setup based on the prerequisites the only other setup required is to have the policies and key created in CM. There is a brief demo that shows how to set up the user sets, access policies and protection polices in CM.
  thales.navattic.com/thalesprotectreveal

Steps for Integration

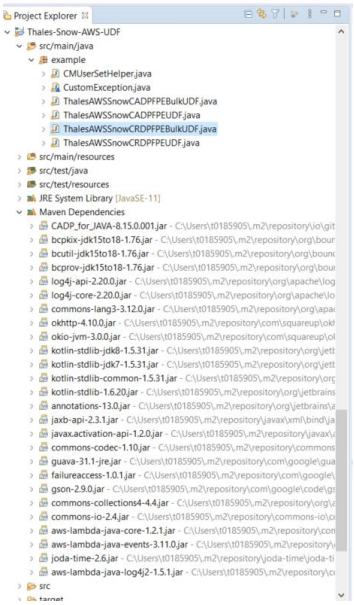
- **[Installing and Configuring Thales CRDP container]**
- **[Download code from Thales github and compile]**
- **[Publish jar/zip file to AWS Lambda Function or GCP Cloud Function]**
- **[Create and configure API Gateway, Snowflake API Integration and Snowflake External Function]**
- **[Integration with Thales CipherTrust Manager. See prerequisites for more information]**

Installing and Configuring CRDP

To install and configure **CRDP,** refer to Quick Start.
Eclipse development tool was used for these examples. Here is the version used for testing along with the Maven plugin for Eclipse.

eclipse.buildId=4.15.0.I20200305-0155
m2e - Maven Integration for Eclipse

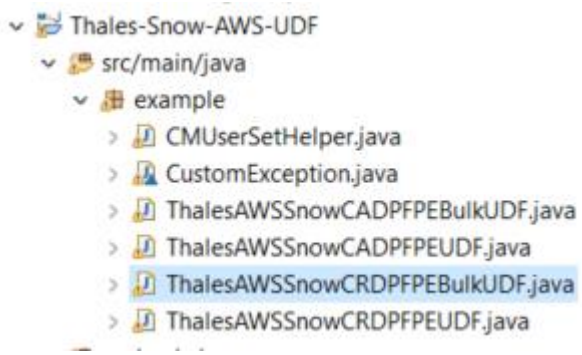
Here is a screenshot in eclipse of the jar files from the pom file used for these examples:



Download code from github and compile.

git clone https://github.com/ThalesGroup/CipherTrust_Application_Protection.git

The database directory has all the code for snowflake. The AWS Lambda examples should have the following class files in your project. Google Functions will have a similar number of class files.



CRDP supports a bulk API which allows for CRDP to batch requests before calling protect or reveal. A single class file that accepts environment variables will allow the UDF to be used by more than one snowflake function.

Assuming you have your CM already configured the `ThalesAWSSnowCRDPFPEUDFTester` can be used to test basic connection to CM to make sure your CM environment is configured correctly.

You will need to modify environment variables such as CRDPIP, BATCHSIZE and other necessary settings. Please see the appendix for all the environment variables and descriptions.

Generate the jar file to upload to the CSP.

To compile and generate the target jar file to be uploaded to AWS Lambda select the project and choose “Run As” “maven install” to generate the target.

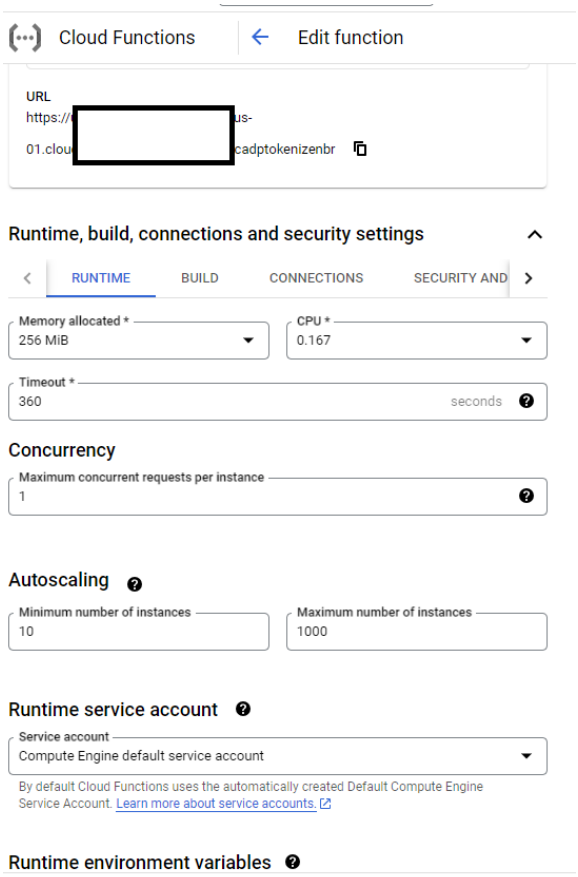
```
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\target\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT.jar with C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\target\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT-shaded.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ Thales-Snow-AWS-UDF ---
[INFO] Installing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\target\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT.jar to C:\Users\t0185905\.m2\repository\Thales\Thales-Snow-AWS-UDF\0.0.5-SNAPSHOT\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT.jar
[INFO] Installing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\pom.xml to C:\Users\t0185905\.m2\repository\Thales\Thales-Snow-AWS-UDF\0.0.5-SNAPSHOT\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.041 s
[INFO] Finished at: 2024-09-10T11:43:20-04:00
[INFO] ------------------------------------------------------------------------
```

The process would be the same for GCP Cloud Functions or Azure Functions.

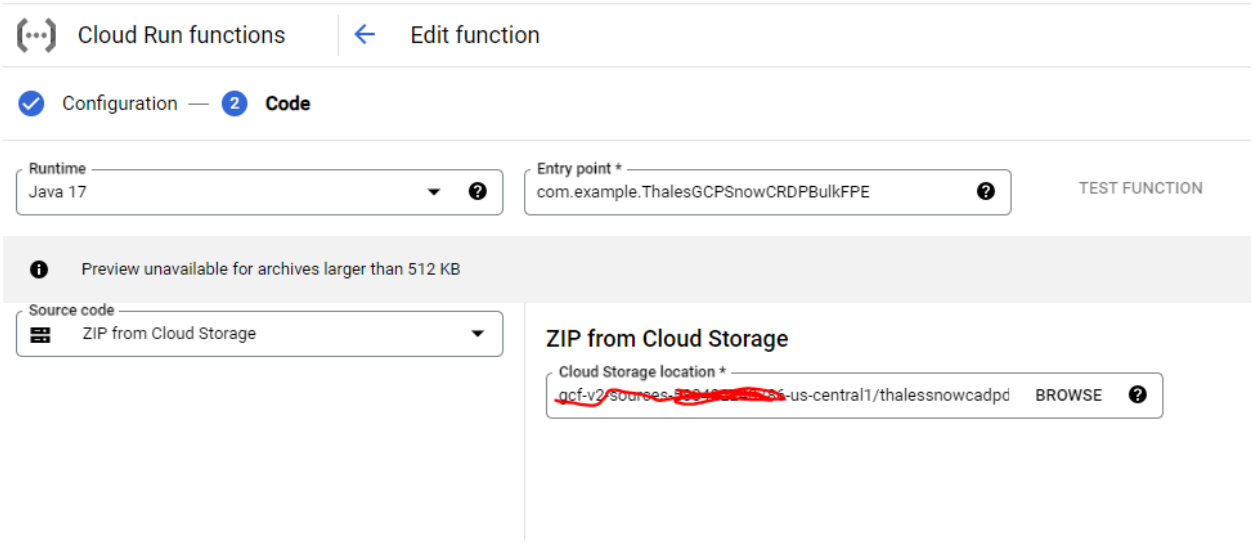
Publish jar/zip file to AWS Lambda Function or Google Cloud Function.

Once you have generated the jar file to upload you can then create the CSP function. Google requires a zip file so zip up the jar file in the target directory of your eclipse project.

GCP Cloud Function



Set the environment variables to the appropriate values and then select Next. Upload the zip file on the next screen.

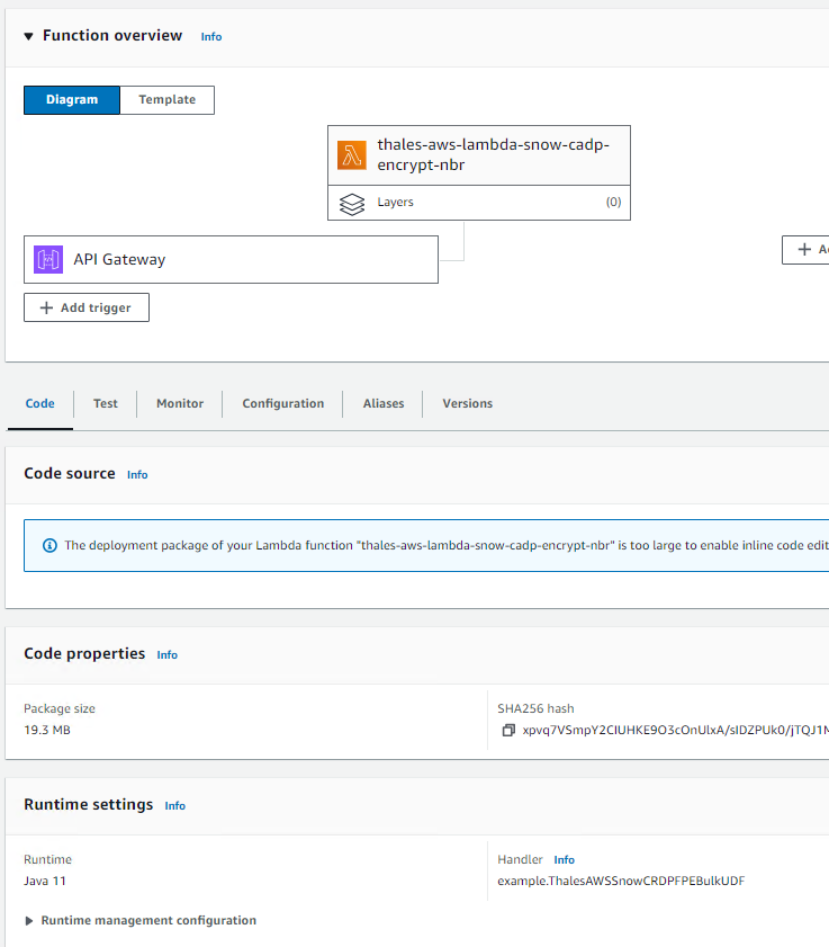


Be sure to change the Entry point to the class file you are deploying. The example above is `com.example.ThalesGCPSnowCRDPBulkFPE.` Click Deploy to deploy the function.

AWS Lambda Function

To create a lambda function upload your give your function a name upload the jar file, make sure to provide the appropriate class file name and select the configuration tab to set the

environment variables such as CRDPIP and BATCHSIZE. Set memory to 256MB. Further testing may allow for lower settings.



Note: If you notice the name of this function is thales-aws-lambda-snow-cadp-encrypt-nbr but the runtime settings are set for example.ThalesAWSSnowCRDPFPEBulkUDF which means that it will execute this class file which is code for CRDP not CADP. So, the name of the function really does not matter for execution what matters is the actual runtime class.

Be sure to change the Handler to the class file you are deploying. The example above is example.ThalesAWSSnowCRDPFPEBulkUDF.

Once you have created the functions above and if you have already configured and setup CM with the appropriate environment variables you can test the function with the test tab. You will need to provide the appropriate json to test. The AWS examples use the format from the API Gateway which includes all the headers and other attributes that can be parsed by the UDF.

Create and configure API Gateway, Snowflake API Integration and Snowflake External Function.

Each CSP has its own unique attributes to create functions and gateways. Snowlake has provided a worksheet that can be used to capture the necessary settings to help organize the

setup. Please follow the instructions on the worksheet to create the necessary settings in the CSP.

For GCP use this link.

https://docs.snowflake.com/en/sql-reference/external-functions-creating-gcp.html#

As noted above the steps are:

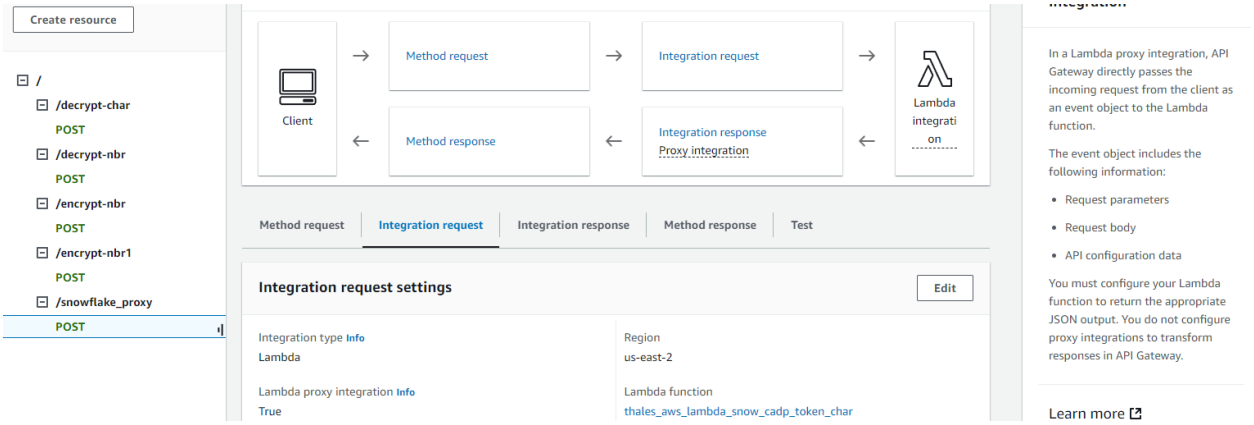
1. Create the GCP Cloud Function. (should already be done from above)
2. Create API Gateway in GCP
3. Create API Integration in Snowflake
4. Create external function object in Snowflake

For AWS use this link.

https://docs.snowflake.com/en/sql-reference/external-functions-creating-aws

1. Create the AWS Lambda Function. (should already be done from above)
2. Create API Gateway in AWS
3. Create API Integration in Snowflake
4. Linke the API Integration to AWS proxy service.
5. Create external function object in Snowflake

The UDF's are written to accept the json in a certain format. For AWS it is necessary to select the Lambda proxy integration to true to have the right format from snowflake.



As noted above there is a test class (`ThalesAWSSnowCRDPFPEBulkUDFTester`) located in the test directory that can be used to test connectivity with CM without having to publish the Function. When all the above steps are performed you should see your UDF's in Snowflake under Routines in the UI. Here is a sample query using one of the UDF's.

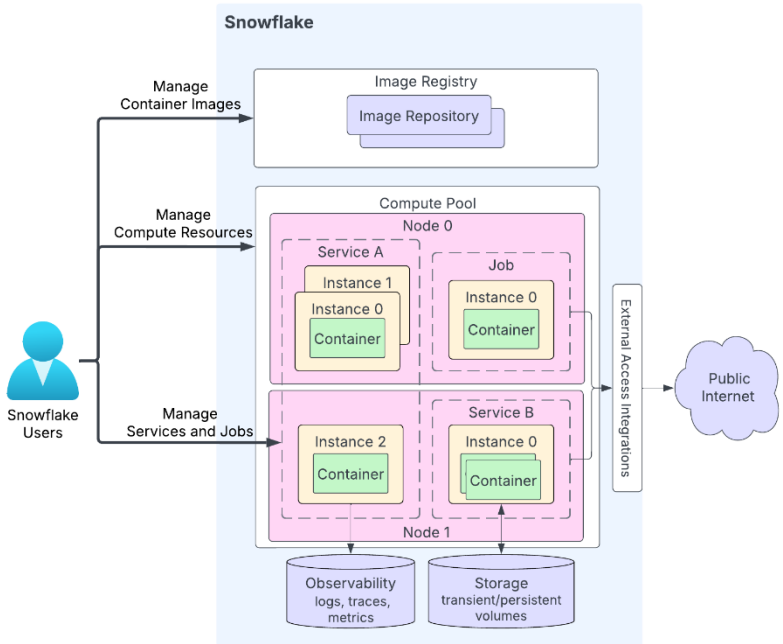
Sample Results:

```
select THALES_CRDP_GCP_ENCRYPT_NBR(emp_id) as EMPENC, emp_id from emp_big
limit 5
```

	EMPENC	EMP_ID
1	"50454058"	68275006
2	"5331697"	4091066
3	"28337918"	72321331
4	"90829858"	78667181
5	"49007874"	29490189

Snowpark Container Service (SPCS)

The steps provided in this section contain a couple of options for the deployment. As noted above all 3 CSP's are now supported although it is region dependent. What is nice about this option is that it will be the same process for any of the CSPs. Here is an overview on how this service works. https://docs.snowflake.com/en/developer-guide/snowpark-container-services/working-with-services

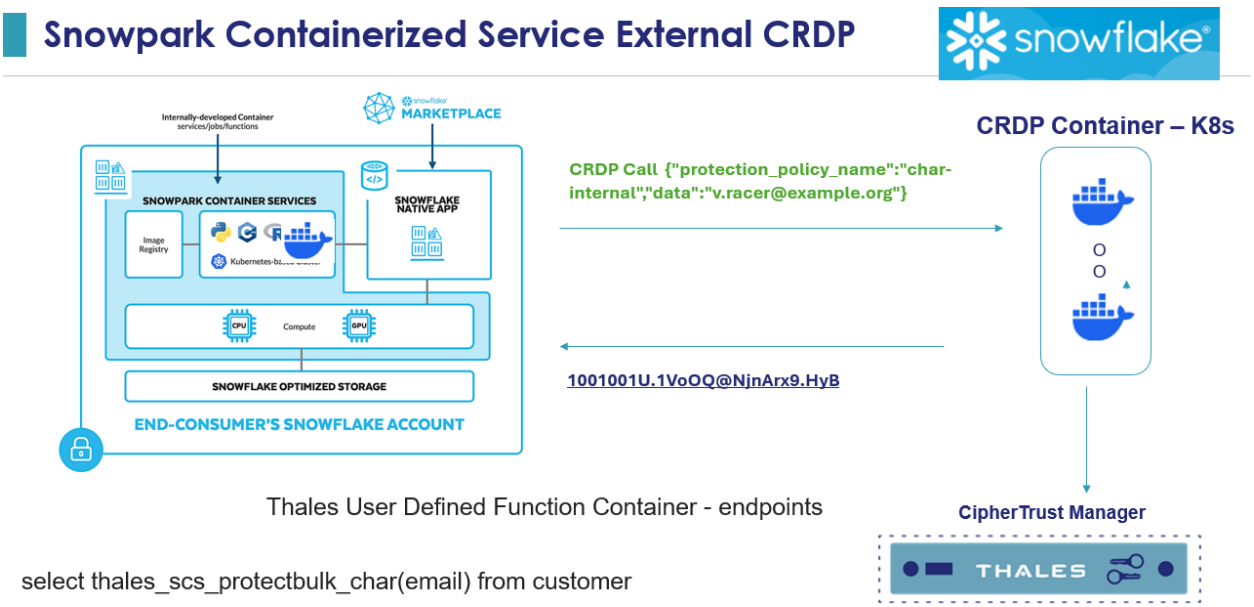


CRDP & Thales CRDP EndPoint Service in SPCS

This example is for both the CRDP Container and the CRDP EndPoint Service in SPCS.

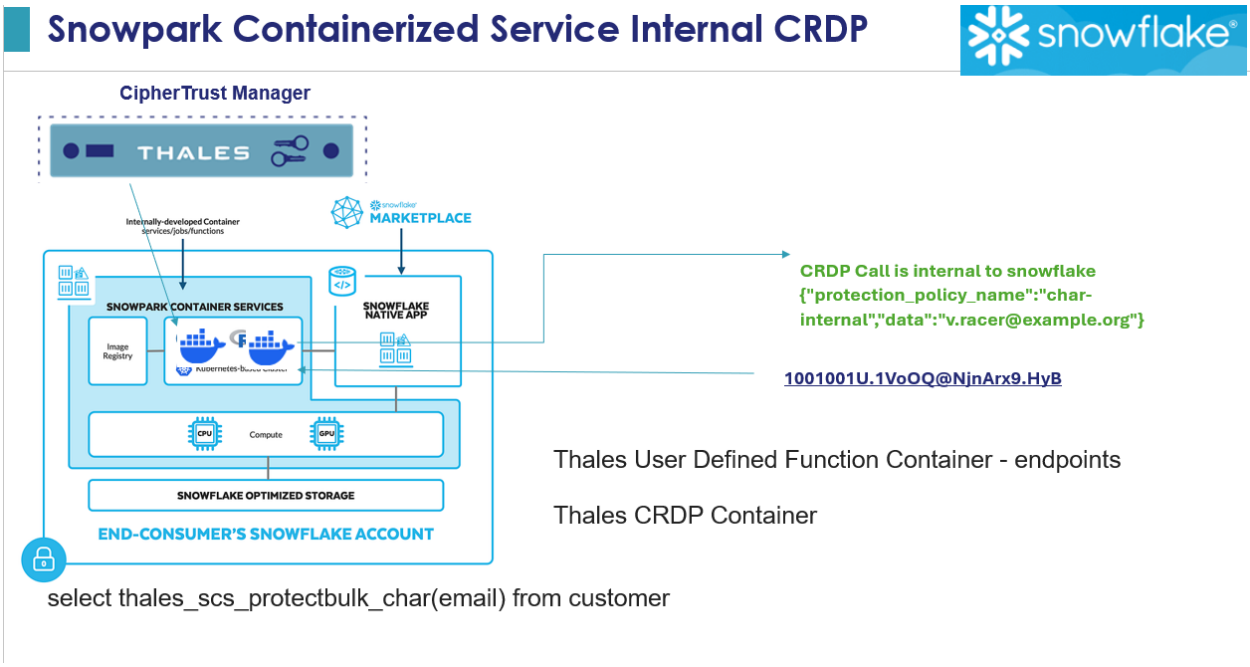
SPCS External CRDP

Customers who have a CRDP environment already up and running and may be restricted to licensing can use a preexisting CRDP deployment. The diagram below highlights an example of that configuration.



SPCS Internal for both CRDP and Thales Endpoints

Customers can deploy both CRDP and the Thales CRDP Endpoint Application to the Snowflake registry. The diagram below highlights an example of that configuration.



Supported Product Versions

- **CipherTrust Manager** CipherTrust Manager 2.20 and higher
- **CRDP** CRDP 1.2.1 and higher
- **Snowflake CLI 3.10.1**
- **Docker 28.3 API Version 1.51**
- **Spring Tool Suite 4**

This integration is validated using Java 17.

Prerequisites

- Steps performed for this integration were provided by this Snowflake link: https://docs.snowflake.com/en/developer-guide/snowpark-container-services/overview
- Spring Tool Suite 4
- Ensure that CRDP container is installed and configured. Refer to https://thalesdocs.com/ctp/con/crdp/latest/admin/crdp-deploy_alternative/index.html
- Ensure that the CipherTrust Manager is installed and configured. Refer to the **CipherTrust Manager documentation** for details.
- Integration with CipherTrust Manager. Assuming the CRDP container has already been deployed setup based on the prerequisites the only other setup required is to have the policies and key created in CM. There is a brief demo that shows how to set up the user

sets, access policies and protection polices in CM.
thales.navattic.com/thalesprotectreveal

- Snowflake CLI.
- github personal access token or docker equivalent.
- Snowflake accountadmin or role with appropriate permissions. See link at https://quickstarts.snowflake.com/guide/intro_to_snowpark_container_services/index.html for an example. (Note this example was created using accountadmin).

Note for this section you will need to have Docker installed and snowflake CLI. Please follow these instructions on how to install snowflake cli. https://docs.snowflake.com/en/developer-guide/snowflake-cli/installation/installation

Also follow these instructions to set up the config file.
https://docs.snowflake.com/en/developer-guide/snowflake-cli/connecting/configure-cli

The examples provided below used github as the location for the docker image repository. When using github as a location for docker images a personal access token is required, and the GITHUP_TOKEN environment variable must be set with that token.

Example:
source snow-env/bin/activate
export GITHUB_TOKEN=ghp_yourtoken
docker login ghcr.io -u yourgithubid

When prompted for pwd enter the ghp_yourtoken token
Note: When using github as a repository for images they show up under packages.

Here is a link for more information https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

Steps for Integration

Step-by-Step Guide to Publish Your Images to Snowflake Container Services. The CRDP image must be published first so you can obtain the correct internal URL to be used by the Thales CRDP SpringBoot Endpoint Image.

CRDP Image deployment

This guide will walk you through pulling the public Thales CipherTrust image, tagging it for your Snowflake account's repository, and pushing it so it can be used by your Snowpark Container Service.

Install the snowflake cli if not already done and configure the config file and decide what repository name to use for the images. In this example we had two separate repositories.

Create the repositories

if not already done in snowflake. Two repos were create in snowflake one for the Thales Snowflake Springboot Endpoints App (my_repo) and the other for CRDP (my_backend_repo)

```
create image repository if not exists my_repo;
create image repository if not exists my_backend_repo;
```

Log in to the Snowflake Image Registry.

First, you need to authenticate your Docker client with your Snowflake Image Registry. You will be prompted for your Snowflake password. Replace <orgname>-<acctname> with your Snowflake organization and account name (e.g., myorg-myaccount). You can find this in the URL you use to access Snowflake.

docker login <orgname>-<acctname>.registry.snowflakecomputing.com -u <your_snowflake_username>
or
Example: snow --config-file snow.toml spcs image-registry login

Pull the Thales CRDP image from Docker Hub to your local machine.

Example: docker pull thalesciphertrust/ciphertrust-restful-data-protection:1.2.1

Tag the Image for Snowflake.

Now, you need to tag the local image with the full path to your Snowflake Image Registry. This new tag tells Docker where to push the image. Replace <orgname>-<acctname> with your Snowflake organization and account name. The path /sf_tuts/public/my_backend_repo should match the database, schema, and image repository you've created in Snowflake.

Example: docker tag thalesciphertrust/ciphertrust-restful-data-protection:1.2.1 yoursnowflake-account.registry.snowflakecomputing.com/sf_tuts/public/my_backend_repo/ciphertrust-restful-data-protection:1.2.1

Push the Image to the Snowflake Registry.

With the image correctly tagged, you can now push it to your Snowflake repository.

Example: docker push yoursnowflake-account.registry.snowflakecomputing.com/sf_tuts/public/my_backend_repo/ciphertrust-restful-data-protection:1.2.1

After completing these steps you should proceed to the next section to deploy the CRDP Snowflake Sprinboot Application Image.

Thales CRDP Snowflake Springboot Endpoint Application Image Deployment

This section will walk you through creating and tagging the Thales CRDP Snowflake SpringBoot Application image for your Snowflake account's repository and pushing it so it can be used by your Snowpark Container Service.

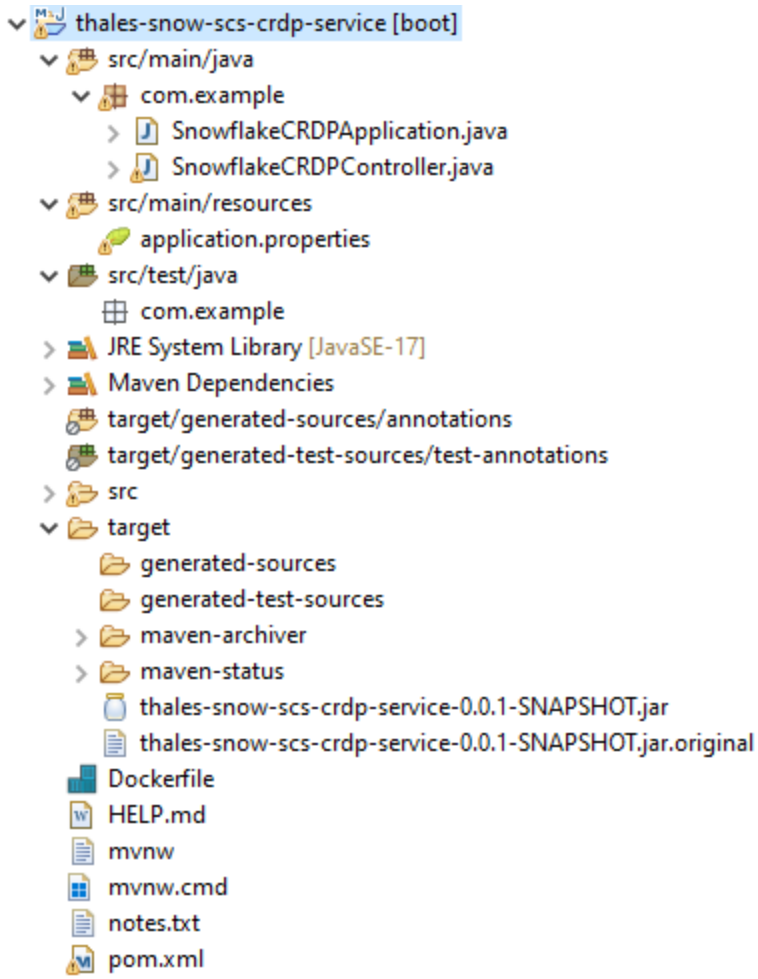
Download and build the Thales CRDP Snowflake Sringboot Endpoint Application.

Since you may be modifying the provided code best practices is to first test locally and then push the docker image to your repository. You will need to modify environment variables such as CRDPIP, BATCHSIZE and other necessary settings. Please see the appendix for all the environment variables and descriptions.

Spring Tool Suite 4 development tool was used for these examples.

git clone https://github.com/ThalesGroup/CipherTrust_Application_Protection.git

The database directory has all the code for snowflake. The code is at this location:



CRDP supports a bulk API which allows for CRDP to batch requests before calling protect or reveal. This API was used for these examples. A single class file that contains 6 different end points depending on the data type to be protected.

Assuming you have your CM already configured the app can be used to test. Make sure your CM environment is configured correctly. You will need to modify environment variables such as CRDPIP, BATCHSIZE and other necessary settings. Please see the appendix for all the environment variables and descriptions.

In addition, you will need to find the URL needed to connects to the CRDP container. Here are a couple of commands to identify the correct hash value.

```
SELECT SYSTEM$GET_SERVICE_DNS_DOMAIN('SF_TUTS.PUBLIC');
DESC SERVICE thales_backend_service;
```

and use the dns_name column value: Example:

```
thales-backend-service.yourhashcode.svc.spcs.internal
```

There are 3 ways you can provide this value and any other environment variable settings.

Code change.

Dockerfile change

Create Service command override.

The order of precedence is as follows:

1. **Value from Snowflake service spec (env: block) → highest precedence.**
2. **Value from Dockerfile ENV → used only if service spec doesn't set it.**
3. **Value from Java default (getOrDefault) → used only if neither service spec nor Dockerfile provide**

Generate the jar file to build your docker image.

To compile and generate the target jar file to be used to build a docker image select the project and choose "Run As" "maven install" to generate the target.

```
[INFO] [1;32mTests run: [0;1;32m1 [m, Failures: 0, Errors: 0, Skipped: 0, Time
elapsed: 3.238 s -- in com.example. [1mThalesSnowUdfApplicationTests [m
[INFO]
[INFO] Results:
[INFO]
[INFO] [1;32mTests run: 1, Failures: 0, Errors: 0, Skipped: 0 [m
[INFO]
[INFO]
[INFO] [1m--- [0;32mjar:3.4.2:jar [m [1m(default-jar) [m @ [36mthales-snow-scs-crdp-
service [0;1m --- [m
[INFO] Building jar: D:\workspace-spring-tool-suite-4-4.25.0.RELEASE\thales-snow-scs-
crdp-service\target\thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] [1m--- [0;32mspring-boot:3.5.4:repackage [m [1m(repackage) [m @ [36mthales-
snow-scs-crdp-service [0;1m --- [m
[INFO] Replacing main artifact D:\workspace-spring-tool-suite-4-
4.25.0.RELEASE\thales-snow-scs-crdp-service\target\thales-snow-scs-crdp-service-
0.0.1-SNAPSHOT.jar with repackaged archive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to D:\workspace-spring-tool-suite-4-
4.25.0.RELEASE\thales-snow-scs-crdp-service\target\thales-snow-scs-crdp-service-
0.0.1-SNAPSHOT.jar.original
[INFO]
[INFO] [1m--- [0;32minstall:3.1.4:install [m [1m(default-install) [m @ [36mthales-
snow-scs-crdp-service [0;1m --- [m
[INFO] Installing D:\workspace-spring-tool-suite-4-4.25.0.RELEASE\thales-snow-scs-
crdp-service\pom.xml to C:\Users\root\.m2\repository\com\example\thales-snow-scs-
crdp-service\0.0.1-SNAPSHOT\thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.pom
[INFO] Installing D:\workspace-spring-tool-suite-4-4.25.0.RELEASE\thales-snow-scs-
crdp-service\target\thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.jar to
```

```
C:\Users\root\.m2\repository\com\example\thales-snow-scs-crdp-service\0.0.1-
SNAPSHOT\thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.jar
[INFO] [1m------------------------------------------------------------------------ [m
[INFO] [1;32mBUILD SUCCESS [m
[INFO] [1m------------------------------------------------------------------------ [m
[INFO] Total time: 9.016 s
[INFO] Finished at: 2025-09-08T10:37:48-04:00
[INFO] [1m------------------------------------------------------------------------
```

Copy jar file and Dockerfile to your build machine and create the image.

Locate the `thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.jar` file in the target directory. It should be about 25MB. Copy the jar file to your build machine and then proceed with the docker build commands as noted in the following steps below.

Dockerfile

```
FROM eclipse-temurin:17-jdk-alpine

WORKDIR /app

COPY thales-snow-scs-crdp-service-0.0.1-SNAPSHOT.jar app.jar

EXPOSE 8083

ENV CRDPIP=thales-backend-service.yourhashcode.svc.spcs.internal
ENV BATCHSIZE=1000
ENV DEFAULTMETADATA=1001000
ENV DEFAULTMODE=internal
ENV DEFAULTCHARPOLICY=char-internal
ENV DEFAULTNBRCHARPOLICY=nbr-char-internal
ENV DEFAULTNBRNBRPOLICY=nbr-nbr-internal
ENV DEFAULTINTERNALCHARPOLICY=char-internal
ENV DEFAULTINTERNALNBRCHARPOLICY=nbr-char-internal
ENV DEFAULTINTERNALNBRNBRPOLICY=nbr-nbr-internal
ENV DEFAULTEXTERNALCHARPOLICY=char-external
ENV DEFAULTEXTERNALNBRCHARPOLICY=nbr-char-external
ENV DEFAULTEXTERNALNBRNBRPOLICY=nbr-nbr-external
ENV DEFAULTREVEALUSER=admin
```

Sample code can be found at: ?????
git pull ????
Example

docker build -t thales-snow-scs-crdp-service -t ghcr.io/yourgithubaccount/thales-snow-scs-crdp-service .

Create the repositories in snowflake.

Two repos were create in snowflake one for the Thales Snowflake Springboot Endpoints App (my_repo) and the other for CRDP (my_backend_repo). This should already have been done from the prior step.

```
create image repository if not exists my_repo;
create image repository if not exists my_backend_repo;
```

Log in to the Snowflake Image Registry.

First, you need to authenticate your Docker client with your Snowflake Image Registry. You will be prompted for your Snowflake password. Replace <orgname>-<acctname> with your Snowflake organization and account name (e.g., myorg-myaccount). You can find this in the URL you use to access Snowflake.

docker login <orgname>-<acctname>.registry.snowflakecomputing.com -u <your_snowflake_username>
or
Example: snow --config-file snow.toml spcs image-registry login

Note: you should be prompted in DUO for MFA approval for snowflake.

Tag the Image for Snowflake.

Now, you need to tag the local image with the full path to your Snowflake Image Registry. This new tag tells Docker where to push the image.Replace <orgname>-<acctname> with your Snowflake organization and account name. The path /sf_tuts/public/my_repo/thales-snow-scs-crdp-service should match the database, schema, and image repository you've created in Snowflake.

Example: docker tag ghcr.io/yourgithubaccount/thales-snow-scs-crdp-service:latest yoursnowflake-account.registry.snowflakecomputing.com/sf_tuts/public/my_repo/thales-snow-scs-crdp-service:latest

Push the Image to the Snowflake Registry.

With the image correctly tagged, you can now push it to your Snowflake repository.

Example: docker push yoursnowflake-account.registry.snowflakecomputing.com/sf_tuts/public/my_repo/thales-snow-scs-crdp-service:latest

Configure snowflake

Create snowflake network rules.

1. Create Network Rules that specify the external destinations your containers need to access:

```
CREATE OR REPLACE NETWORK RULE thales_network_rule
  MODE = EGRESS
  TYPE = HOST_PORT
  VALUE_LIST = ('ciphertrust.manager.com:443',
onlyifCRDPisexternal.northcentralus.cloudapp.com:8092');
```

2. Create an External Access Integration that references these network rules:

```
CREATE EXTERNAL ACCESS INTEGRATION thales_access_integration
ALLOWED_NETWORK_RULES = (thales_network_rule)
ENABLED = true;
```

3. Grant usage on the integration to the role that will create the service: (if needed)

```
GRANT USAGE ON INTEGRATION thales_access_integration TO ROLE ;
```

https://docs.snowflake.com/en/developer-guide/snowpark-container-services/additional-considerations-services-jobs#label-working-with-services-jobs-egress

Create compute pool.

```
CREATE COMPUTE POOL my_compute_pool
  MIN_NODES = 1
  MAX_NODES = 1   -- one node is fine for single-user testing
  INSTANCE_FAMILY = CPU_X64_M   -- or even L if you need more horsepower
  AUTO_SUSPEND_SECS = 3600;
```

Note: The above was only for testing. Your settings will most likely be different to obtain better performance. Please see Appendix on tips to consider for performance. Be aware that charges will start once the pool is running. Here are some commands to stop and resume:

```
ALTER COMPUTE POOL my_compute_pool STOP ALL;
```

```
ALTER COMPUTE POOL my_compute_pool SUSPEND;
describe compute pool  my_compute_pool
ALTER COMPUTE POOL my_compute_pool RESUME;
```

Create services

https://docs.snowflake.com/en/developer-guide/snowpark-container-services/working-with-services

Update Your Service Specification Finally, make sure the image path in your CREATE SERVICE DDL specification exactly matches the new tag you just pushed. The path in the spec should not include the registry URL (<orgname>-<acctname>.registry.snowflakecomputing.com). Snowflake automatically resolves this. Your image line in the service specification should look like this:image: /sf_tuts/public/my_backend_repo/ciphertrust-restful-data-protection:1.2.1

```
CREATE SERVICE thales_udf_service
  IN COMPUTE POOL my_compute_pool
  FROM SPECIFICATION $$
    spec:
      containers:
        - name: udf
          image: /sf_tuts/public/my_repo:latest
          env:
            PORT: 8083
            DEFAULTMODE: "internal"
      endpoints:
        - name: udfendpoint
          port: 8083
          public: false
  $$
  EXTERNAL_ACCESS_INTEGRATIONS = (thales_access_integration)
  MIN_INSTANCES = 1
  MAX_INSTANCES = 1;

  CREATE SERVICE thales_backend_service
  IN COMPUTE POOL my_compute_pool
  FROM SPECIFICATION $$
    spec:
      containers:
        - name: ciphertrust-service
          image: /sf_tuts/public/my_backend_repo/ciphertrust-restful-data-
protection:1.2.1
          env:
            KEY_MANAGER_HOST: "ciphertrust.manager.com"
            REGISTRATION_TOKEN: "yourcmapplicationregtoken"
            SERVER_MODE: "no-tls"
      endpoints:
        - name: backendendpoint
```

```
        port: 8090
        public: false
$$
EXTERNAL_ACCESS_INTEGRATIONS = (thales_access_integration)
MIN_INSTANCES = 1
MAX_INSTANCES = 1;
```

Note: The above was only for testing. Your settings will most likely be different to obtain better performance. Please see Snowflake documentation on services for more details.

For environment variables the order of precedence is as follows.

4. **Value from Snowflake service spec (env: block) → highest precedence.**
5. **Value from Dockerfile ENV → used only if service spec doesn't set it.**
6. **Value from Java default (getOrDefault) → used only if neither service spec nor Dockerfile provide**

Create user defined functions

https://docs.snowflake.com/en/sql-reference/sql/create-function-spcs

Note batch size settings can be altered depending on use cases.

```
-- Protect bulk (CHAR)
CREATE or replace FUNCTION thales_crdp_scs_protectbulk_char(input VARCHAR)
RETURNS VARCHAR
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/protectbulkchar';

-- Protect bulk (NBR -> CHAR)
CREATE FUNCTION thales_crdp_scs_protectbulk_nbr_char(input VARCHAR)
RETURNS VARCHAR
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/protectbulknbrchar';

-- Protect bulk (NBR -> NBR)
CREATE FUNCTION thales_crdp_scs_protectbulk_nbr_nbr(input VARCHAR)
```

```
RETURNS VARIANT
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/protectbulknbrnbr';

-- Reveal bulk (CHAR)
CREATE FUNCTION thales_crdp_scs_revealbulk_char(input VARCHAR)
RETURNS VARCHAR
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/revealbulkchar';

-- Reveal bulk (NBR -> CHAR)
CREATE FUNCTION thales_crdp_scs_revealbulk_nbr_char(input VARCHAR)
RETURNS VARCHAR
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/revealbulknbrchar';

-- Reveal bulk (NBR -> NBR)
CREATE FUNCTION thales_crdp_scs_revealbulk_nbr_nbr(input VARCHAR)
RETURNS VARIANT
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/revealbulknbrnbr';
```

Also, additional UDFS can also be made like small, medium and large and just alter the batchsizes. Amounts should be tested to ensure no timeouts occur.

Note: Your context_headers can contain more values like CURRENT_ROLE. Unfortunately, HEADERS are not supported as of Sept 2025.

Here is an example:

```
CONTEXT_HEADERS = ( CURRENT_USER, CURRENT_ROLE )
(Not supported for SPCS) HEADERS = ('classification'='PII',
'department'='HR')
```

*Note: Since HEADERS for the DDL syntax in the create function command are **not** supported to support use cases for both internal and external policies one method is to create a different service and pass in the DEFAULTMODE to "internal" or "external".*

Once all the above is done you can then run queries using the service functions above.

```
select address ,thales_crdp_scs_protectbulk_char(address ) as
address_enc from emp_big where emp_id > 100  limit 5
```

ADDRESS	ADDRESS_ENC
38868 Sarah Court Suite 980	dFuMR CGVsW QnHSt lo2fj jjH
47631 Miller Center	YjU8h 50lbfZ NAvr5t
04515 Wendy Roads	sl2wz 6Gav1 ERZ8U
83671 Alvarado Brook	ohduf 0Mk5B3Dt HxS78
279 Becker Underpass Apt. 041	RmQ w1DoZA ST7vCJf9A KBd. eXB

This is an example of a python worksheet in snowflake

```
import snowflake.snowpark as snowpark
from snowflake.snowpark.functions import col

def main(session: snowpark.Session):
    query = """
    SELECT
      emp_id,
      address,
      thales_crdp_scs_protectbulk_char(address) AS address_enc
    FROM emp_big
    WHERE emp_id > 100
    LIMIT 5
    """

    # 2. Use the session.sql() method to execute the query
    # and return the results as a DataFrame.
    dataframe = session.sql(query)

    # 3. Print a sample of the dataframe to standard output.
    dataframe.show()

    # 4. Return value will appear in the Results tab.
    return dataframe
```

Results

EMP_ID	ADDRESS	ADDRESS_ENC
3475140	38868 Sarah Court Suite 980	dFuMR CGVsW QnHSt lo2fj jjH
27685464	47631 Miller Center	YjU8h 50lbfZ NAvr5t
42388111	04515 Wendy Roads	sl2wz 6Gav1 ERZ8U
27350577	83671 Alvarado Brook	ohduf 0Mk5B3Dt HxS78
83874796	279 Becker Underpass Apt. 041	RmQ w1DoZA ST7vCJf9A KBd. eXB

Advanced Topics

External UDF's Best Practices

Snowflake also publishes a best practice/performance recommendation link that can also provide some options to improve performance.
https://docs.snowflake.com/en/sql-reference/external-functions-implementation
https://docs.snowflake.com/en/sql-reference/external-functions-best-practices
When creating the functions make them immutable which should improve performance for certain types of queries. It is also important to ensure you allow enough Cloud Function instances to run in order to handle the queries with large results sets.

Snowflake masking policies

Snowflake has the ability to create column masks which can invoke a remote external function. Doing this will allow ease of use and control who can run the functions. Here is an example:

```
CREATE OR REPLACE MASKING POLICY thales_mask AS (val string) RETURNS string ->
  CASE
    WHEN CURRENT_ROLE() IN ('ANALYST') THEN thales_crdp_aws_decrypt_char(val)
    ELSE val
  END;
```

For more information please refer to the Snowflake documentation.

UDF Environment Variables for External Functions

Listed below are the UDF environment variables with their descriptions. Note these are all of the variables for both CADP and CRDP examples.

Key	Value	Desc
BATCHSIZE	200	Nbr of rows to chunk when using batch mode
CMPWD	Yourpwd!	CM PWD if using CADP
CMUSER	apiuser	CM USERID if using CADP
CRDPIP	20.221.216.666	CRDP Container IP if using CRDP
datatype	charint	datatype of column (char or charint)
keymetadata	1001000	policy and key version if using CRDP
keymetadatalocation	external	location of metadata if using CRDP (internal,external)
mode	revealbulk	mode of operation(protect,reveal,protectbulk,revealbulk) CRDP
protection_profile	plain-nbr-ext	protection profile in CM for CRDP
returnciphertextforuserwithnokeyaccess	yes	if user in CM not exist should UDF error out or retur ciphertext
usersetidincm	716f01a6-5cab-4799-925a-6dc2d8712fc1	userset in cm if user lookup is done
usersetlookup	no	should uselookup be done (yes,no)
usersetlookupip	20.241.70.666	userset lookup
showrevealinternalkey	yes	show keymetadata when issuing a protect call (CRDP)

UDF Environment Variables for Thales Snowflake Springboot Endpoint App.

Listed below are the UDF environment variables with their descriptions. Note these are all of the variables for both CADP and CRDP examples.

Key	Value	Desc
BATCHSIZE	1000	Nbr of rows to chunk when using batch mode
BADDATATAG	999999999	Value to be used for bad data
CRDPIPPORT	8090	CRDP Port
CRDPIP	thales-backend-service.yourhashcode.svc.spcs.internal	CRDP Container IP if using CRDP
DEFAULTREVEALUSER	admin	Default user for revealing data
DEFAULTCHARPOLICY	char-internal	Default value for alphanumeric data
DEFAULTNBRCHARPOLICY	nbr-char-internal	Input is a number but may contain special characters. Results will be digits except for special characters.
DEFAULTNBRNBRPOLICY	nbr-nbr-internal	Input is number results should be digits.
DEFAULTINTERNALCHARPOLICY	char-internal	Default value for alphanumeric data
DEFAULTINTERNALNBRCHARPOLICY	nbr-char-internal	Input is a number but may contain special characters. Results will be digits except for special characters.
DEFAULTINTERNALNBRNBRPOLICY	nbr-nbr-internal	Input is number results should be digits.
DEFAULTEXTERNALCHARPOLICY	char-external	Default value for alphanumeric data
DEFAULTEXTERNALNBRCHARPOLICY	nbr-char-external	Input is a number but may contain special characters. Results will be digits except for special characters.
DEFAULTEXTERNALNBRNBRPOLICY	nbr-nbr-external	Input is number results should be digits.

For environment variables the order of precedence is as follows .

1. **Value from Snowflake service spec (env: block) → highest precedence.**
2. **Value from Dockerfile ENV → used only if service spec doesn't set it.**
3. **Value from Java default (getOrDefault) → used only if neither service spec nor Dockerfile provide**

The properties file contains logging and input format from snowflake.

Current format for spcs is the same as external UDF's so do not change this setting

@Value("${app.INPUT_FORMAT:external}")

Sample commands:

```
curl -X POST http://localhost:8083/protectbulknbrnbr   -H "Content-Type:
application/json"   -d '{
        "data": [
          [0, 858888],
          [1, 1567856785678]
        ]
      }'
RESPONSE

{"data":[[0,"277473"],[1,"5302143509393"]]}

curl -X POST http://localhost:8083/revealbulknbrnbr -H "Content-Type:
application/json" -H "sf-context-current_user: Ym9i" -d
'{"data":[[0,"277473"],[1,"5302143509393"]]}'
RESPONSE

{"data":[[0,"XX8888"],[1,"XXXXXXXXX5678"]]}
```

Sample commands to create external UDF in snowflake.

Here are some examples of the create statements for the AWS UDF's.

```
 create or replace external function
SF_TUTS.PUBLIC.THALES_CRDP_AWS_PROTECT_NBR (b varchar)
    returns variant
    api_integration = my_api_integration_aws
    as 'https://yoursnowproj.execute-api.us-east-
2.amazonaws.com/test/encrypt-nbr';

     create or replace external function
SF_TUTS.PUBLIC.THALES_CRDP_AWS_REVEAL_NBR (b varchar)
    returns variant
    api_integration = my_api_integration_aws
```

```
    as 'https://yoursnowproj.execute-api.us-east-
2.amazonaws.com/test/decrypt-nbr';

   create or replace external function
SF_TUTS.PUBLIC.THALES_CRDP_AWS_REVEAL_CHAR (b varchar)
    returns variant
    api_integration = my_api_integration_aws
    as 'https://yoursnowproj.execute-api.us-east-
2.amazonaws.com/test/decrypt-char';

    create or replace external function
SF_TUTS.PUBLIC.THALES_CRDP_AWS_PROTECT_CHAR (b varchar)
    returns variant
    api_integration = my_api_integration_aws
    as 'https://yoursnowproj.execute-api.us-east-
2.amazonaws.com/test/snowflake_proxy';
```

In addition, normal grants must be applied like any other custom function in snowflake. See this link https://docs.snowflake.com/en/sql-reference/external-functions-creating-aws-call

Application Data Protection UserSets

Application Data Protection UserSets are currently used for DPG and CRDP to control how the data will be revealed to users. These UserSets can also be independent of any Access Policy. Most cloud databases have some way to capture who is running the query and this information can be passed to CM to be verified in a UserSet to ensure the person running the query has been granted proper access. In github there is a sample class file called CMUserSetHelper that can be used to load a userset with values from an external identity provider such as LDAP. The name of this method is `addAUserToUserSetFromFile`. Once users have been loaded into this userset the usersetid must be captured and used as an environment variable to the Function. The function has a number of environment variables that must be provided for the function to work. Please review the section on Environment Variables for more details.

Options for handling null values.

Since it is not possible to encrypt a column that contains null values or any column that has 1 byte it is necessary to skip those to avoid getting an error when running the query. There are a couple of ways to handle this use case.

For Null values this option can be handled by the create UDF function attribute called `RETURNS NULL ON NULL INPUT`

Here is an example:

```
CREATE FUNCTION thales_crdp_scs_revealbulk_char(input VARCHAR)
RETURNS VARCHAR
RETURNS NULL ON NULL INPUT
SERVICE = thales_udf_service
ENDPOINT = udfendpoint
CONTEXT_HEADERS = ( CURRENT_USER )
MAX_BATCH_ROWS = 5000
AS '/revealbulkchar';
```

CRDP will be supporting a new feature to return the original value if it is under 2 bytes but as of Sept 2025 this option is not available.

Option 1. Modify the queries.

Many times, simply adding a where clause to exclude values that have nulls or less than 2 bytes can avoid query errors. For example: `select * from FROM mw_demo_dataset_US.plaintext50cadpemailprotected_nulltest where email is not null and length(email) > 1;`

For those scenarios where that is not suffice some other examples are listed below. Here is an example of a select statement that can be modified to handle null values:

```
SELECT
  name,
  CASE
    WHEN email IS NULL THEN 'null'
    WHEN length(email) < 2 then email
    WHEN email = 'null' then email
    ELSE `your-project.mw_demo_dataset_US.thales_crdp_protect_char`(email)
  END AS email
FROM
  mw_demo_dataset_US.plaintext50cadpemailprotected_nulltest;
```

The above use case was for situations where the column contained both null and the word ‘null’.

Here is an example of a select statement that can be modified to handle null values in the where clause.

```
    SELECT name, email, email_enc
FROM
  mw_demo_dataset_US.plaintext50cadpemailprotected_nulltest
  where CASE
    WHEN email_enc IS NULL THEN 'null'
    WHEN length(email_enc) < 2 then email_enc
    WHEN email_enc = 'null' then email_enc
    ELSE  `your-project.mw_demo_dataset_US.thales_crdp_reveal_char`(email_enc)
  END  like "%gmail%"
```

Row	name	email	email_enc
1	Dr. Lemmie Zboncak	ikris@gmail.com	1IOEk@RPDqC.GHd
2	Zillah Leuschke	scronin@gmail.com	53mWY7Y@4bzb6.2D4
3	Troy Gaylord	devon49@gmail.com	EsTMziF@ISZg2.yN6
4	Dr. Spurgeon Wintheiser	hilah17@gmail.com	PIM4vAd@qAlV9.oJC
5	Tatyana Bernhard	denny56@gmail.com	yVQ7ITA@3idYW.GqV
6	Renata Hilpert	tdickens@gmail.com	plzg3zLb@oetcj.Opi
7	Deliah Douglas	sconnelly@gmail.com	jpXAUZWPX@koaH2.yS8
8	Amare Feeney	batz.geary@gmail.com	G04W.CRNIb@riDnQ.DHT
9	Dr. Cristy Schinner	schulist.garfield@gmail.com	HII90wCU.LPtU3p6o@F9Nb7.G...
10	Vena Douglas	huston.christiansen@gmail.com	7OHpfz.PXI0PoJJaWIQ@AvTP...

Option 2. Modify the Cloud Function to skip encrypting these values.

Please review the method checkvalid for the logic of handling these use cases.

Note: When using this logic it is important to know that for any column that is null will return 'null'. This should be fine for use cases where the query is not updating the column. For use cases where the source system is expecting null vs the word 'null' additional testing should be conducted on the systems that rely on this data type as being null vs the word 'null'.

User context

Snowflake support for External IDP.

Snowflake supports the following types of external identity providers.

External OAuth integrates the customer's OAuth 2.0 server to provide a seamless SSO experience, enabling external client access to Snowflake. Snowflake supports the following external authorization servers OAuth 2.0, custom clients, and partner applications:

- Okta
- Microsoft Entra ID
- Ping Identity PingFederate
- External OAuth Custom Clients
- Sigma

AWS Example:

AWS support for method authentication

- IAM-Role
- OIDC
- AWS Cognito
- API Key

- AWS Lambda Authorizer

As noted above although AWS API Gateway supports various types of method authentication API Private Endpoint Gateway only supports AWS-IAM which **does not** provide the end user running the query (`user: AROA#4GGF3PW#$@#KPX7G:snowflake`).
Here is the identity json passed in from snowflake via the AWS API Gateway.

```
"identity": {
            "cognitoIdentityPoolId": "Cognito identity pool ID",
            "accountId": "Caller account ID",
            "cognitoIdentityId": "Cognito identity ID",
            "caller": "Caller value",
            "sourceIp": "Source IP address",
            "principalOrgId": "Principal organization ID",
            "accessKey": "Access key ID",
            "cognitoAuthenticationType": "Cognito authentication type",
            "cognitoAuthenticationProvider": "Cognito authentication provider",
            "userArn": "User ARN",
            "userAgent": "User agent string",
            "user": "User value"
        },
```

You can see the other methods available such as Cognito, api key, oidc etc. If a private end point was **not** used and IAM-Role was **not** used it is possible that the user value in the json could be the snowflake user logged on vs the IAM value of `AROA#4GGF3PW#$@#KPX7G:snowflake`.
One option is to use the built in snowflake current_user() function and pass that into the Thales UDF. It is recommended to use a snowflake capability called column masking to implement the handling of passing in the current_user() to the UDF. Here is an example of how it would be created.

```
CREATE OR REPLACE MASKING POLICY thales_mask_ssn_sin AS (val
string) RETURNS string ->
CASE
WHEN CURRENT_ROLE() IN ('ANALYST') THEN
thales_unprotect_ssn_sin(val,current_user())
ELSE val
END;
```

This would make the invocation of the UDF totally transparent to the end user. By using the column mask it could also be the first level of security since the user must also belong in the analyst role.

Include user information in function header and modify UDF

The external function can be altered to include the current_user with this syntax.
https://docs.snowflake.com/en/sql-reference/sql/create-external-function

```
   create or replace external function
SF_TUTS.PUBLIC.THALES_CRDP_AWS_REVEAL_CHAR (b varchar)
```

```
    returns variant
    api_integration = my_api_integration_aws
    CONTEXT_HEADERS = (current_user)
    as 'https://yoursnowproj.execute-api.us-east-
2.amazonaws.com/test/decrypt-char';
```

A Brief Comparison: Snowflake External Functions vs. Snowpark Container Services

Both Snowflake External Functions and Snowpark Container Services allow you to run custom code that extends the functionality of Snowflake. However, they are designed for fundamentally different architectures and use cases. Understanding their trade-offs in terms of performance, cost, and security is crucial for choosing the right approach.

Feature	External Functions	Snowpark Container Services (SPCS)
Architecture	Remote service; data is sent from Snowflake to an external endpoint (e.g., AWS Lambda, Azure Function) and the result is sent back.	In-database; the containerized service runs directly within the Snowflake ecosystem, next to your data.
Data Movement	**Required**. Data is egressed from Snowflake to an external service over the network.	**Minimal to Zero**. The container runs on the same virtual network as your Snowflake data, eliminating the need to move large datasets.
Primary Use Case	Lightweight, stateless, and simple operations on small datasets. Examples include data enrichment, sentiment analysis, or calling a third-party API.	Complex, long-running, or resource-intensive workloads. Examples include custom machine learning models, ETL/ELT pipelines, real-time APIs, or full-stack applications.
Scaling	The external service (e.g., AWS Lambda, Kubernetes) is responsible for its own scaling. Snowflake can	**Built-in**. Snowflake manages the scaling of services running on a dedicated **compute pool**. You

Feature	External Functions	Snowpark Container Services (SPCS)
	increase the number of concurrent requests sent to the external service based on warehouse size, but it is limited by the external service's concurrency and capacity.	define minimum and maximum instances, and Snowflake automatically scales the number of containers up and down based on demand.

Performance and Cost

- **External Functions**:
    - **Performance**: Can introduce significant latency, especially for large datasets. The primary bottleneck is the network latency of moving data out of Snowflake and the cold start time of the external function (e.g., Lambda). Performance is highly dependent on the external service's architecture and scaling capabilities.
    - **Cost**: You pay for both Snowflake warehouse usage (for the SQL call) and the compute costs of the external service (e.g., AWS Lambda, API Gateway, Kubernetes cluster). You also incur data egress charges for transferring data out of Snowflake. This can be cost-prohibitive for high-volume data processing.
- **Snowpark Container Services**:
    - **Performance**: **Superior** for data-intensive workloads. By running the containerized service adjacent to the data, SPCS eliminates data egress latency. This is a game-changer for machine learning, data processing, and other applications that require fast access to vast amounts of data.
    - **Cost**: You pay for a dedicated Snowflake compute pool (a cluster of virtual machines) and the running time of your containers. This is often more cost-effective than External Functions for large-scale operations because you avoid data egress fees and the dual-billing model of separate compute resources. You get predictable, usage-based billing directly within your Snowflake account.

Security

- **External Functions**:
    - Requires you to manage security and access for an external service.
    - Data is transmitted over the network, even if it's a private network, creating a potential point of vulnerability.

    - You need to set up and manage API integrations, IAM roles, and network security policies on your cloud provider's side to ensure secure communication between Snowflake and your service. This adds administrative overhead.

- **Snowpark Container Services**:
    - **Simplified and more secure**. The containers run within the Snowflake security boundary and are governed by the same security model.
    - Data remains in the Snowflake cloud, and container access to data is controlled by standard Snowflake RBAC (Role-Based Access Control).
    - You can manage secrets, network policies (e.g., outbound egress), and access control all from within Snowflake, consolidating your security posture. This eliminates the need to manage security configurations on a separate platform.

How SPCS Handles Container Scaling

The same kind of auto-scaling you might see in a Kubernetes cluster can take place with a container in Snowpark Container Services, but the mechanism is different and fully managed by Snowflake.

In a traditional Kubernetes setup, you define a HorizontalPodAutoscaler that monitors CPU or memory usage and automatically adds or removes pods. In SPCS, scaling is tied to a **Compute Pool** and a **Service**.

1. **Create a Compute Pool**: First, you define a COMPUTE POOL, which is a cluster of virtual machines that will host your containers. You specify the MIN_NODES and MAX_NODES for this pool.
2. **Define a Service**: You then create a SERVICE (a long-running containerized application) and associate it with a compute pool. In the service specification, you can define MIN_INSTANCES and MAX_INSTANCES.
3. **Automatic Scaling**: Snowflake's internal orchestration engine monitors the workload on the compute pool.
    - If the workload increases and more containers are needed to handle incoming requests, Snowflake will automatically start new container instances within the compute pool, up to the MAX_INSTANCES limit you defined.
    - If the existing nodes in the pool are fully utilized, Snowflake will automatically scale up the compute pool by adding more nodes, up to the MAX_NODES limit.
    - When the workload decreases, Snowflake will scale down by removing unused container instances and eventually suspending compute pool nodes to save costs.

This managed approach to scaling means you don't have to configure or manage the underlying Kubernetes cluster, as Snowflake handles all the complexities of orchestration, node provisioning, and scaling for you.

```
create image repository if not exists my_repo;
SHOW IMAGES IN IMAGE REPOSITORY my_repo;
create image repository if not exists my_backend_repo;
SHOW IMAGES IN IMAGE REPOSITORY my_backend_repo;
```

Troubleshooting containers

Here are some examples on how to view the container logs in snowflake.

```
  SELECT value AS log_line
FROM TABLE(
  SPLIT_TO_TABLE(
    SYSTEM$GET_SERVICE_LOGS('thales_backend_service', 0, 'ciphertrust-
service', 1000),
    '\n'
  )
);

SELECT value AS log_line
FROM TABLE(
  SPLIT_TO_TABLE(
    SYSTEM$GET_SERVICE_LOGS('thales_udf_service', 0, 'udf', 1000),
    '\n'
  )
);
```