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

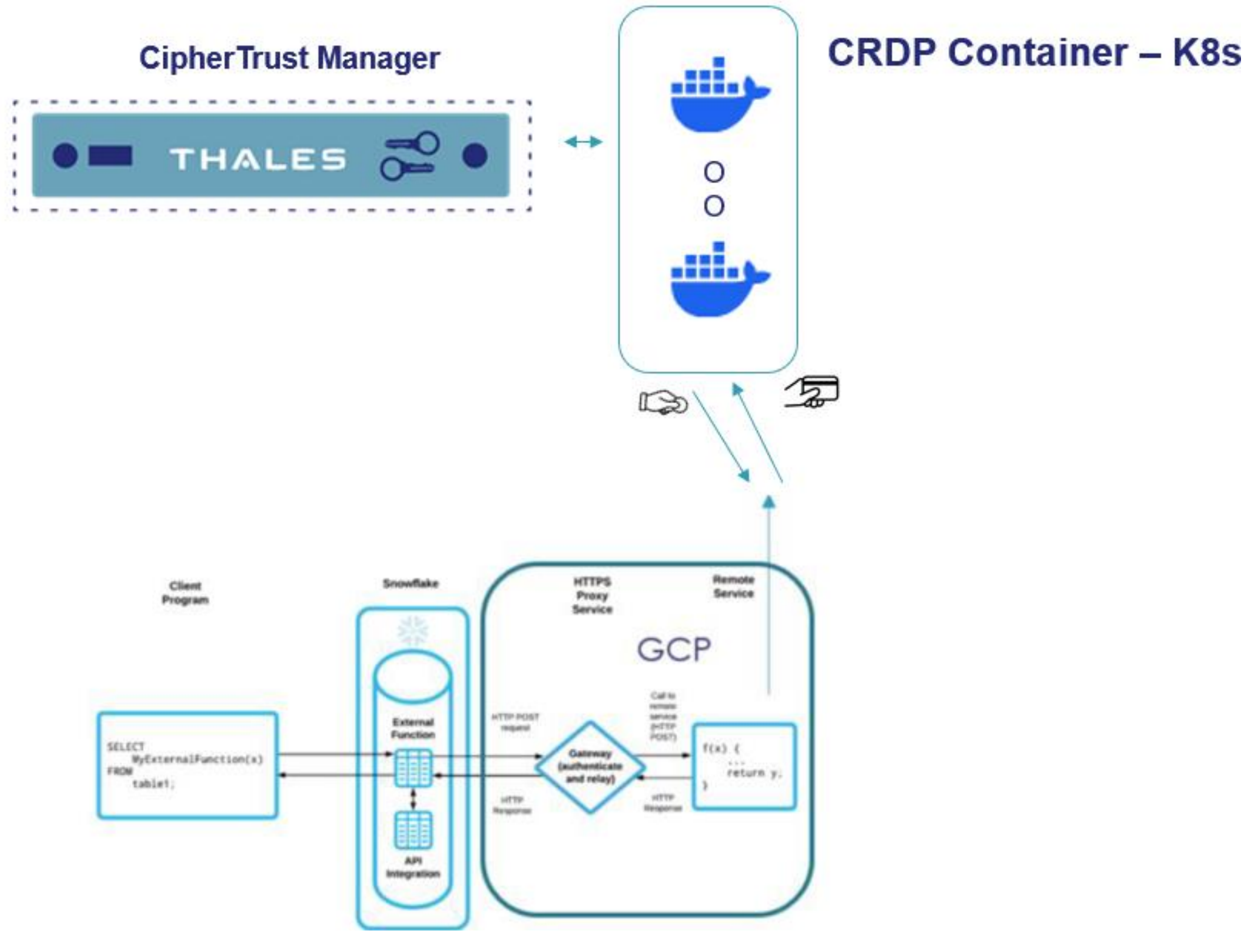
- **Snowflake Snowpipe** – securing private RSA key in Thales CM for connecting to Snowflake using CRDP.
- **Snowflake Credentials** – Thales CipherTrust Secrets Manager (Akeyless) for secrets management in Snowfake.

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.

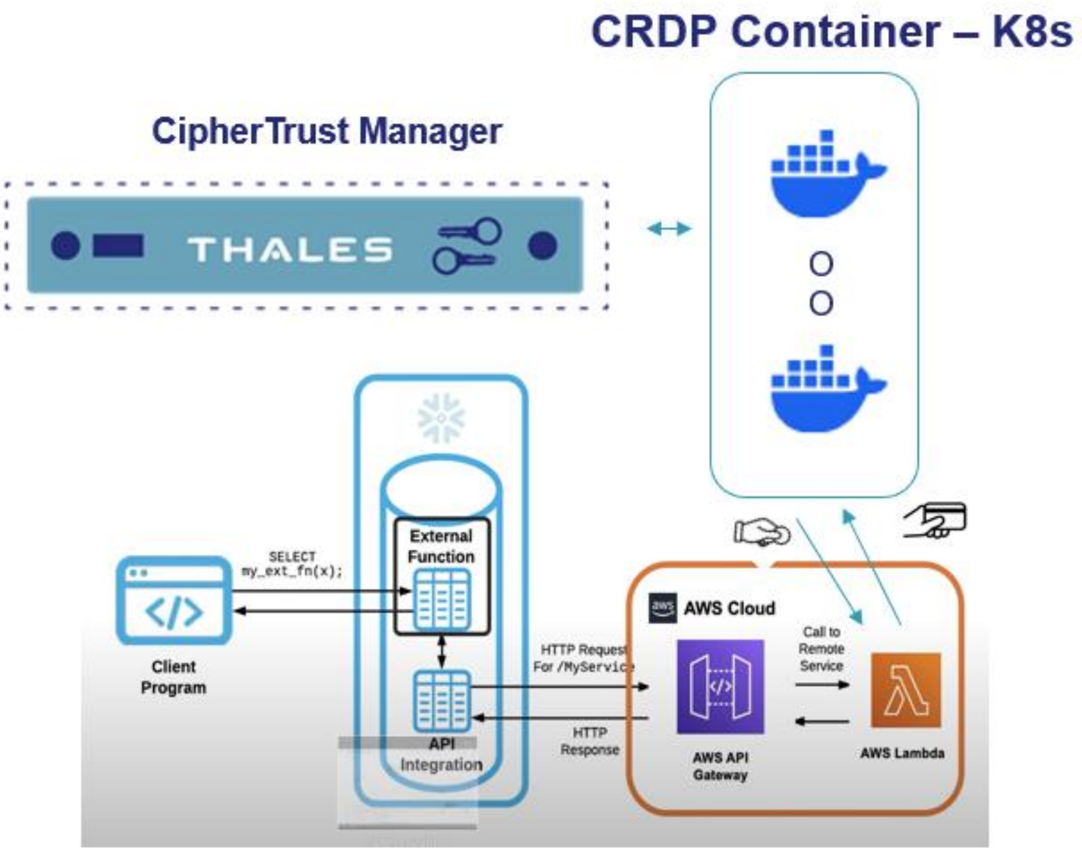
Architecture

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. This document contains examples of both GCP and AWS Functions. The steps provided in this link contain examples of both GCP and AWS Functions.   Listed below are examples of how this integration works.

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.

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]**

Installing and Configuring CRDP

To install and configure **CRDP,** refer to Quick Start.

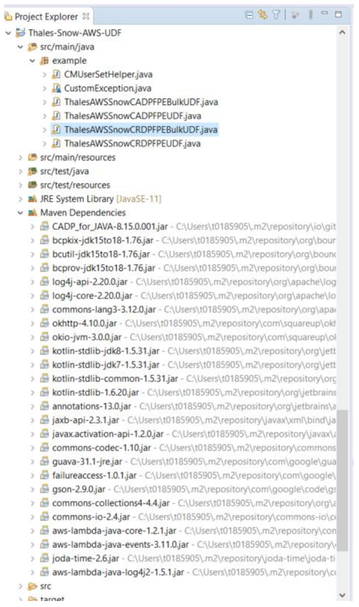
Eclipse development tool was used for these examples. Here is the version used for testing along with the Maven plugin for Eclipse.

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:

```
Packages
(default package)
example
```

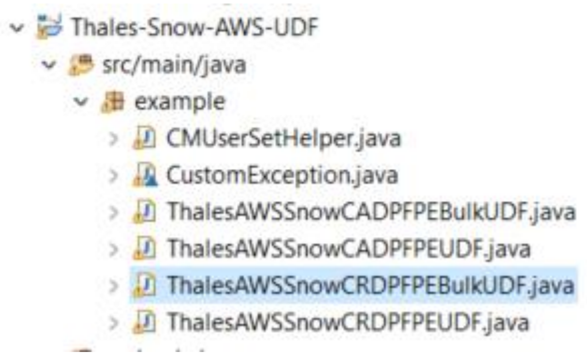


```
Project Explorer
v Thales-Snow-AWS-UDF
  v src/main/java
    v example
      > CMUserSetHelper.java
      > CustomException.java
      > ThalesAWSSnowCADPFPEBulkUDF.java
      > ThalesAWSSnowCADPFPEUDF.java
      > ThalesAWSSnowCRDPFPEBulkUDF.java
      > ThalesAWSSnowCRDPFPEUDF.java
  > src/main/resources
  > src/test/java
  > src/test/resources
  > JRE System Library [JavaSE-11]
  v Maven Dependencies
    > CADP_for_JAVA-8.15.0.001.jar - C:\Users\t0185905\.m2\repository\io\git
    > bcpkix-jdk15to18-1.76.jar - C:\Users\t0185905\.m2\repository\org\bour
    > bcutil-jdk15to18-1.76.jar - C:\Users\t0185905\.m2\repository\org\bounc
    > bcprov-jdk15to18-1.76.jar - C:\Users\t0185905\.m2\repository\org\bour
    > log4j-api-2.20.0.jar - C:\Users\t0185905\.m2\repository\org\apache\log
    > log4j-core-2.20.0.jar - C:\Users\t0185905\.m2\repository\org\apache\lo
    > commons-lang3-3.12.0.jar - C:\Users\t0185905\.m2\repository\org\apac
    > okhttp-4.10.0.jar - C:\Users\t0185905\.m2\repository\com\squareup\okh
    > okio-jvm-3.0.0.jar - C:\Users\t0185905\.m2\repository\com\squareup\ol
    > kotlin-stdlib-jdk8-1.5.31.jar - C:\Users\t0185905\.m2\repository\org\jett
    > kotlin-stdlib-jdk7-1.5.31.jar - C:\Users\t0185905\.m2\repository\org\jett
    > kotlin-stdlib-common-1.5.31.jar - C:\Users\t0185905\.m2\repository\org
    > kotlin-stdlib-1.6.20.jar - C:\Users\t0185905\.m2\repository\org\jetbrains
    > annotations-13.0.jar - C:\Users\t0185905\.m2\repository\org\jetbrains\a
    > jaxb-api-2.3.1.jar - C:\Users\t0185905\.m2\repository\javax\xml\bind\ja
    > javax.activation-api-1.2.0.jar - C:\Users\t0185905\.m2\repository\javax\a
    > commons-codec-1.10.jar - C:\Users\t0185905\.m2\repository\commons
    > guava-31.1-jre.jar - C:\Users\t0185905\.m2\repository\com\google\gua
    > failureaccess-1.0.1.jar - C:\Users\t0185905\.m2\repository\com\google\
    > gson-2.9.0.jar - C:\Users\t0185905\.m2\repository\com\google\code\gs
    > commons-collections4-4.4.jar - C:\Users\t0185905\.m2\repository\org\a
    > commons-io-2.4.jar - C:\Users\t0185905\.m2\repository\commons-io\co
    > aws-lambda-java-core-1.2.1.jar - C:\Users\t0185905\.m2\repository\con
    > aws-lambda-java-events-3.11.0.jar - C:\Users\t0185905\.m2\repository\
    > joda-time-2.6.jar - C:\Users\t0185905\.m2\repository\joda-time\joda-ti
    > aws-lambda-java-log4j2-1.5.1.jar - C:\Users\t0185905\.m2\repository\co
  > src
  > target
```

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.



```
Thales-Snow-AWS-UDF
  src/main/java
    example
      CMUserSetHelper.java
      CustomException.java
      ThalesAWSSnowCADPFPEBulkUDF.java
      ThalesAWSSnowCADPFPEUDF.java
      ThalesAWSSnowCRDPFPEBulkUDF.java
      ThalesAWSSnowCRDPFPEUDF.java
```

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.

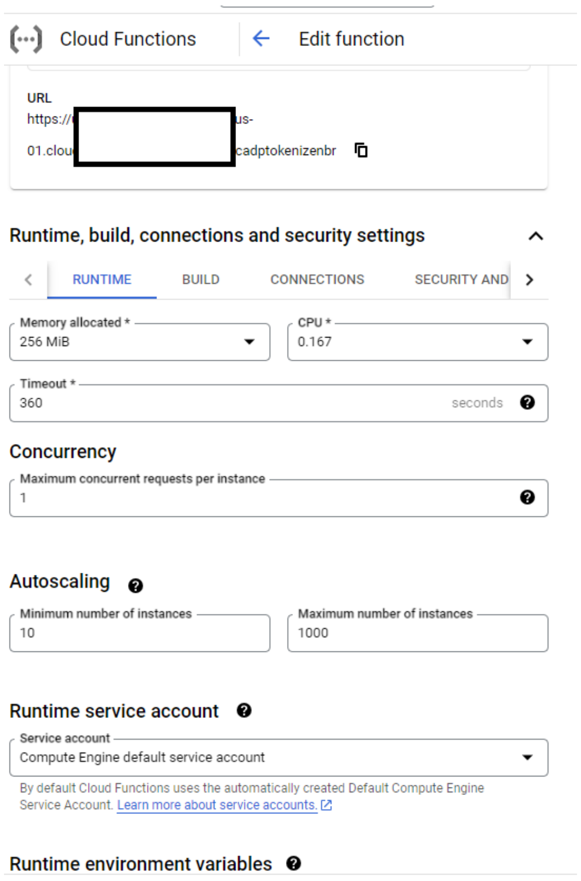
```
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\target\Thales-Snow-
AWS-UDF-0.0.5-SNAPSHOT.jar with C:\Users\t0185905\workspace\Thales-Snow-AWS-
UDF\target\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT-shaded.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (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



URL
https://[redacted]us-
01.clou[redacted]cadptokenizenbr

Runtime, build, connections and security settings

Memory allocated * 256 MiB
CPU * 0.167
Timeout * 360 seconds

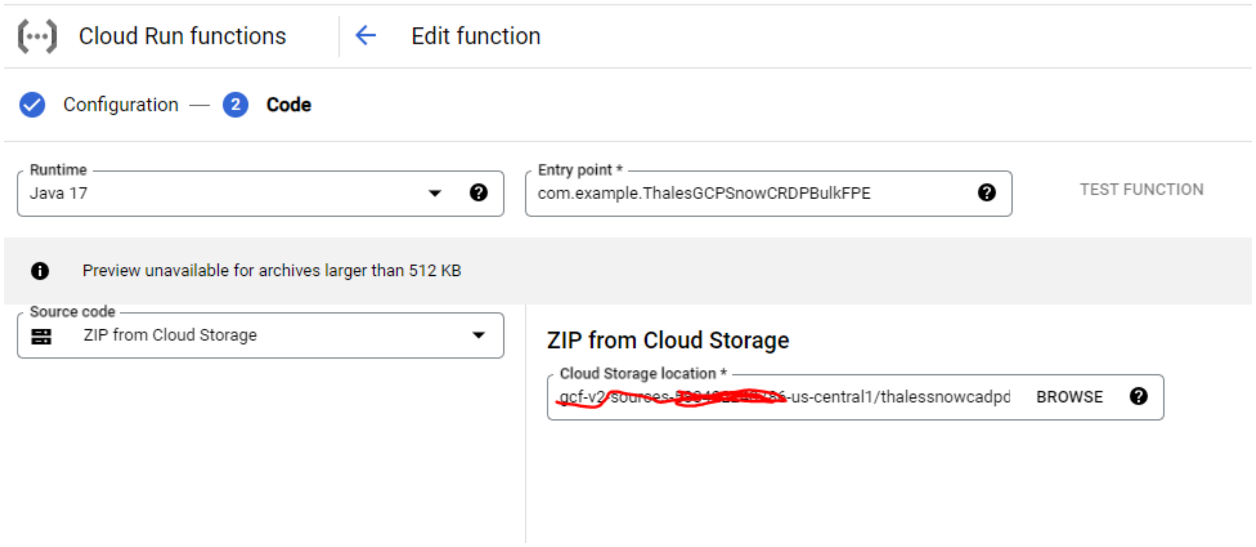
Concurrency
Maximum concurrent requests per instance 1

Autoscaling
Minimum number of instances 10
Maximum number of instances 1000

Runtime service account
Service account Compute Engine default service account
By default Cloud Functions uses the automatically created Default Compute Engine Service Account. Learn more about service accounts.

Runtime environment variables

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


Cloud Run functions

Edit function

Configuration — 2 Code

Runtime: Java 17

Entry point *: com.example.ThalesGCPSnowCRDPBulkFPE

TEST FUNCTION

Preview unavailable for archives larger than 512 KB

Source code: ZIP from Cloud Storage

ZIP from Cloud Storage

Cloud Storage location *: gcf-v2/[redacted]-us-central1/thalessnowcadpd

BROWSE

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.

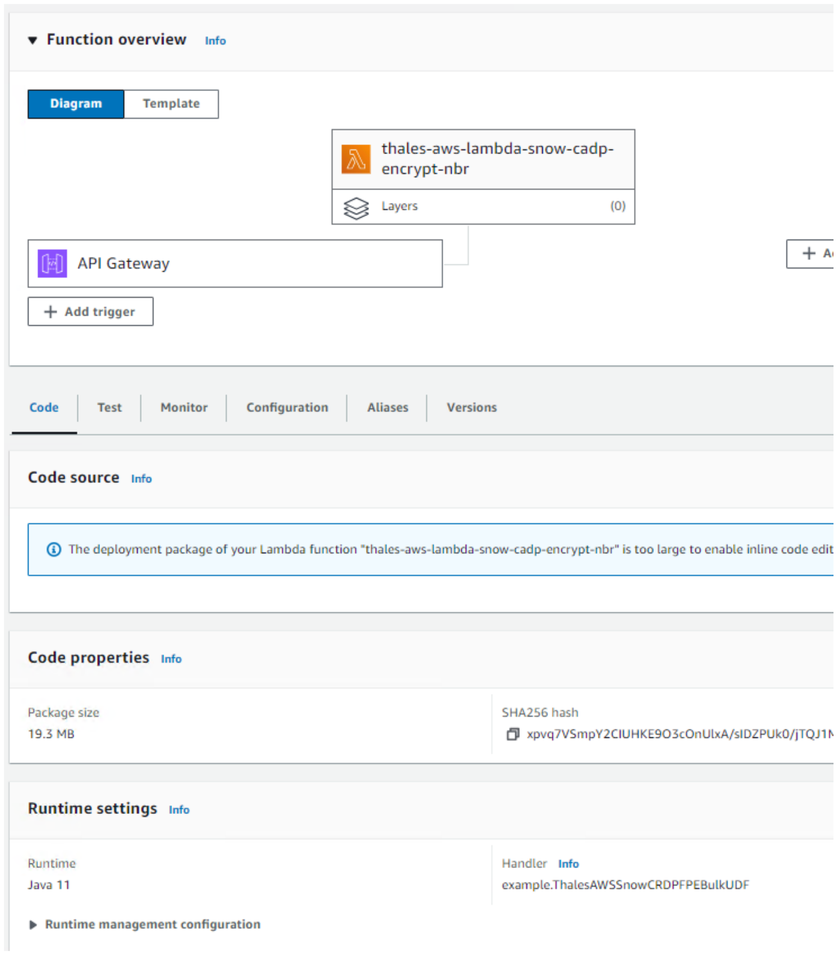

Function overview Info

Diagram Template

thales-aws-lambda-snow-cadp-encrypt-nbr

Layers (0)

API Gateway

Add trigger

Code Test Monitor Configuration Aliases Versions

Code source Info

The deployment package of your Lambda function "thales-aws-lambda-snow-cadp-encrypt-nbr" is too large to enable inline code edit

Code properties Info

Package size
19.3 MB

SHA256 hash
xpvq7VSmpY2ClUHKE9O3cOnUlxA/sIDZPUk0/jTQJ1N

Runtime settings Info

Runtime
Java 11

Handler Info
example.ThalesAWSSnowCRDPFPEBulkUDF

Runtime management configuration

`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

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

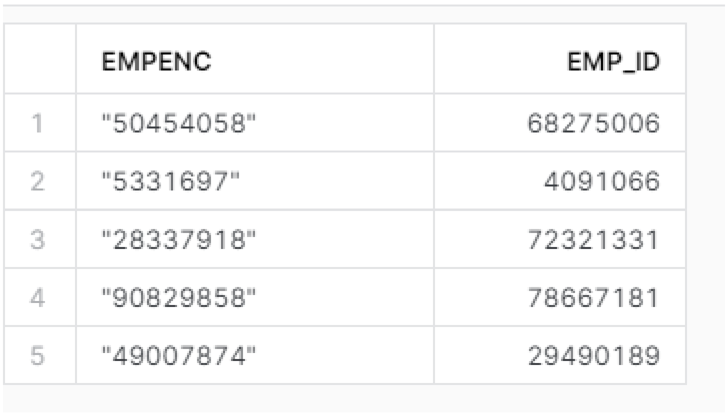
There also is a CRDP setup tutorial (Using Protect/Reveal with CRDP (CM Setup Tutorial)) that explains the assets in the Application Data Protection tile.

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

Advanced Topics

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 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

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)

Sample commands to create 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_CADP_AWS_ENCRYPT_NBR (b varchar)
    returns variant
immutable
    api_integration = my_api_integration_aws
    as 'https://yourcode.execute-api.us-east-2.amazonaws.com/test/encrypt-
nbr';

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

        create or replace external function
SF_TUTS.PUBLIC.THALES_CADP_AWS_ENCRYPT_CHAR (b varchar)
    returns variant
immutable
```

```
    api_integration = my_api_integration_aws
    as 'https://yourcode.execute-api.us-east-2.amazonaws.com/test/encrypt-
char';

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

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.

Option 1. Modify the queries.

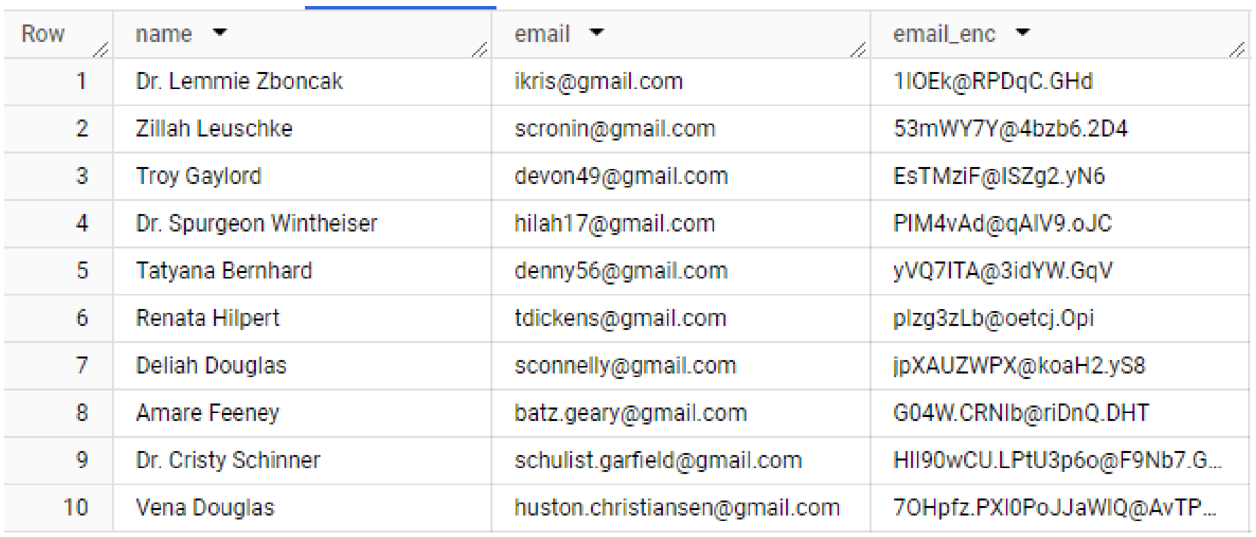
Many times, simply adding a where clause to exclude values that have nulls or less than 2 bytes can avoid query errors. For example: `select * from FROM`
```
  mw_demo_dataset_US.plaintext50cadpemailprotected_nulltest
where email is not null and length(email) > 1;
```
For those scenarios where that is 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@qAIV9.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. As noted above snowflake provides this integration with various providers: