Integration of Snowflake with CADP

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 CADP 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

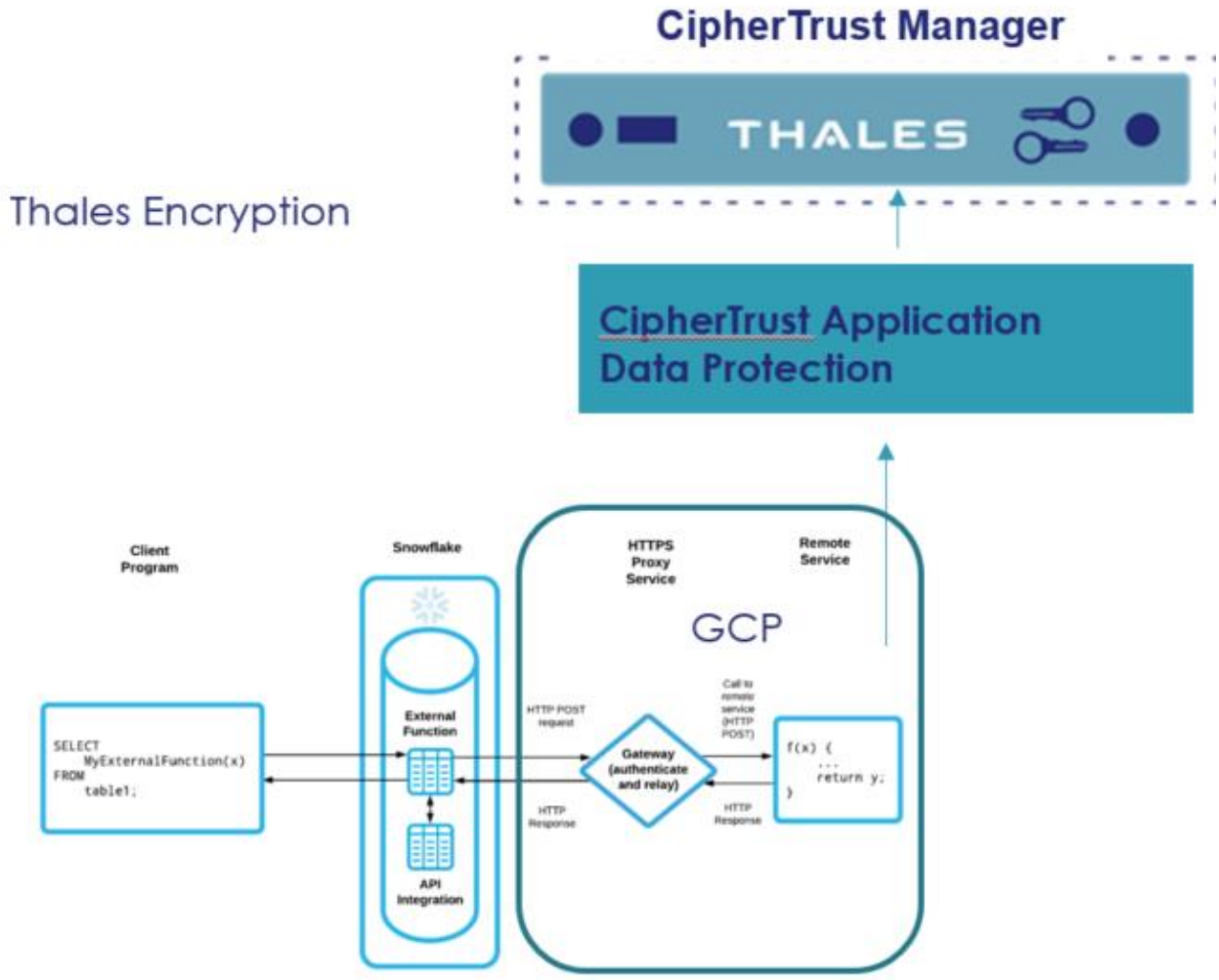
- **Snowflake Snowpipe** – securing private RSA key in Thales CM for connecting to Snowflake using CADP.
- **Snowflake Credentials** – Thales CipherTrust Secrets Manager (Akeyless) for secrets management in Snowfake.

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 CADP to create User Defined Functions (UDF) for encryption and decryption of sensitive data.

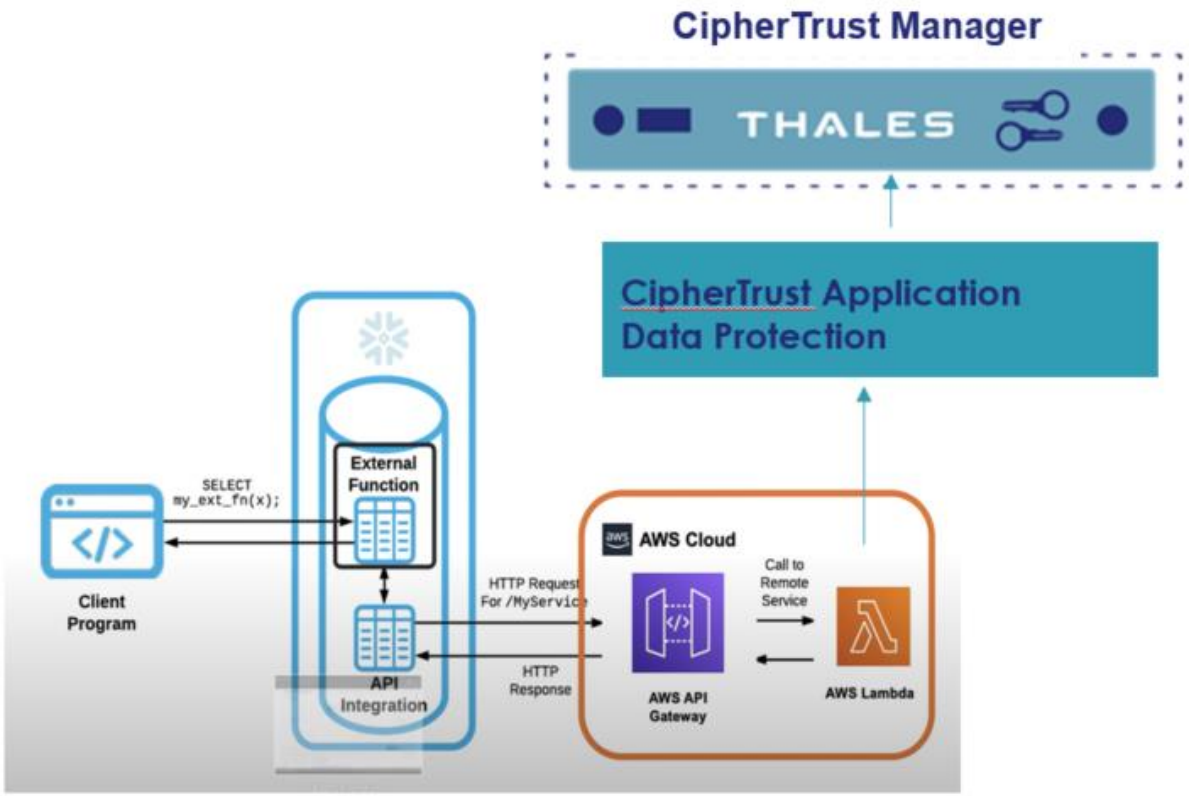
Architecture

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.. The steps provided in this document 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.
- **CADP** CADP Java 8.13 and higher. Note: CADP 8.15.0.001 was tested. If a higher version of CADP is used you may encounter a class not found error with AWS Lambda Functions.
- **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 CADP for Java is installed and configured. Refer to https://thalesdocs.com/ctp/con/cadp/cadp-java/latest/admin/cadp-for-java-quick-start/cadp-for-java-installer/index.html

- Ensure that the CipherTrust Manager is installed and configured. Refer to the **CipherTrust Manager documentation** for details.
- Snowflake communicates with the CipherTrust Manager using the Network Attached Encryption (NAE) Interface. Ensure that the NAE interface is configured. Refer to the **CipherTrust Manager documentation** for details.
- Ensure that the port configured on NAE interface is accessible from Snowflake.

Steps for Integration

- **[Installing and Configuring Thales CADP for Java]**
- **[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]**
- **[Environment Variables]**

Installing and Configuring CADP for Java

To install and configure **CADP for Java,** refer to Quick Start.

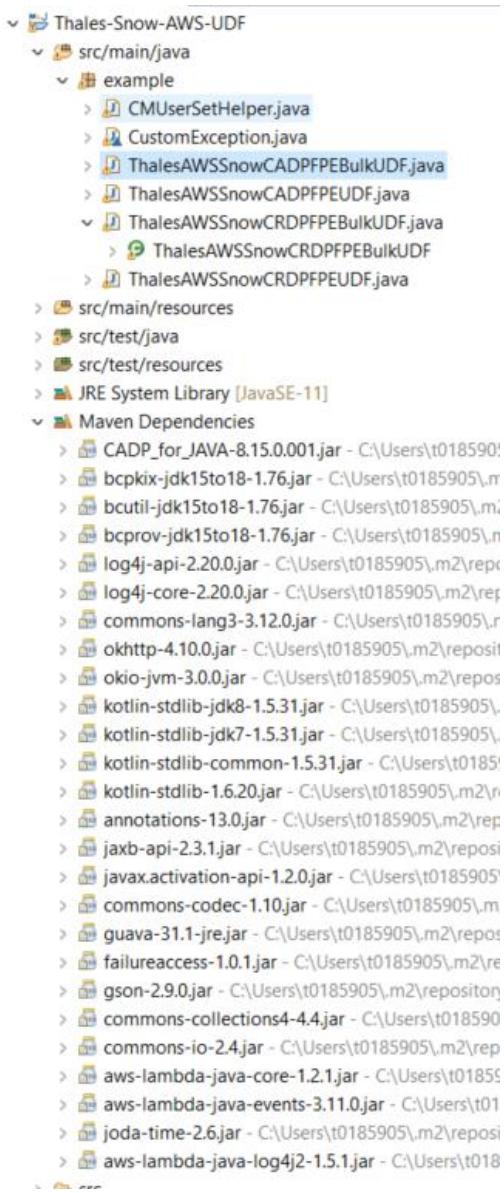
As noted in the link above Thales provides CADP in the Maven repository. This can be found at: https://central.sonatype.com/artifact/io.github.thalescpl-io.cadp/CADP_for_JAVA

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

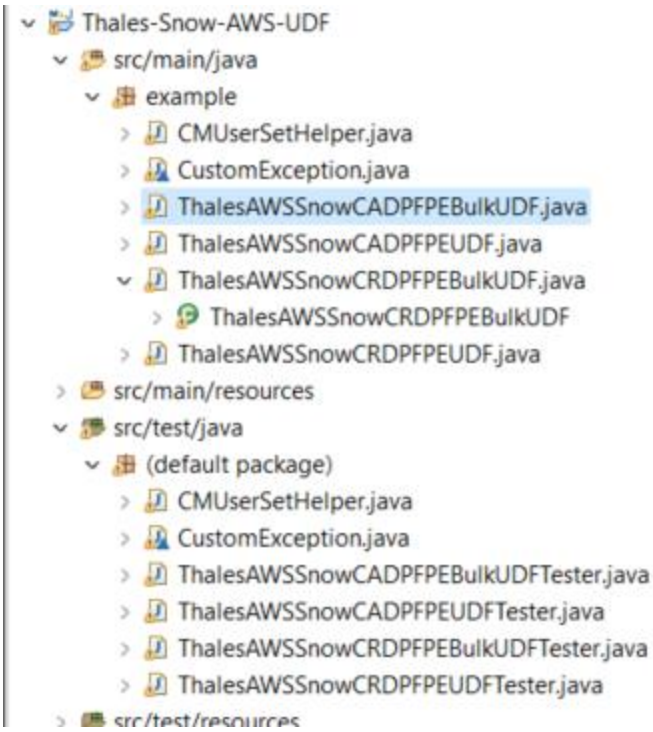
Here is a screenshot in eclipse of the pom file used for these examples:



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.



CADP supports a bulk API which allows for CADP to batch requests before calling encrypt or decrypt. A single class file that accepts environment variables will allow the UDF to be used by more than one snowflake function. There is a limit of 10,000 items to encrypt but it also depends on the size of the items as well so review the CADP documentation to make sure those thresholds are not exceeded. Typically you will reach a point of diminishing returns around 200 as a batchsize.

Assuming you have your CM already configured the `ThalesAWSSnowCADPFPEBulkUDFTester class` 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 CMUSERID, CMPWD, BATCHSIZE and other necessary settings. Please see the appendix for all the environment variables and descriptions. If you have already installed CM then you will need to update the CADP_for_JAVA.properties file with all the necessary settings such as IP/NAE Port, etc. The file is located under the resource's directory in the java project for eclipse.

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 -
--
```

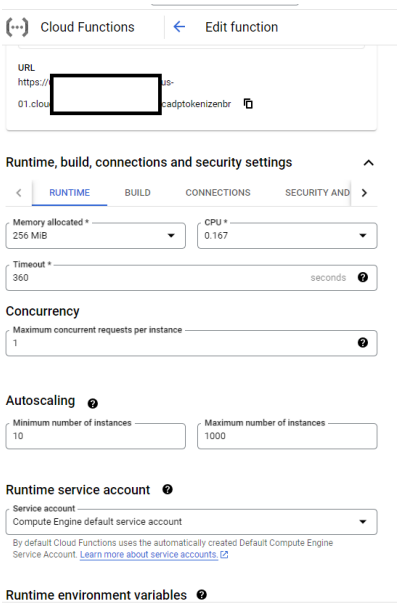
```
[INFO] Installing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\target\Thales-Snow-
AWS-UDF-0.0.5-SNAPSHOT.jar to C:\Users\t0185905\.m2\repository\Thales\Thales-Snow-
AWS-UDF\0.0.5-SNAPSHOT\Thales-Snow-AWS-UDF-0.0.5-SNAPSHOT.jar
[INFO] Installing C:\Users\t0185905\workspace\Thales-Snow-AWS-UDF\pom.xml 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.

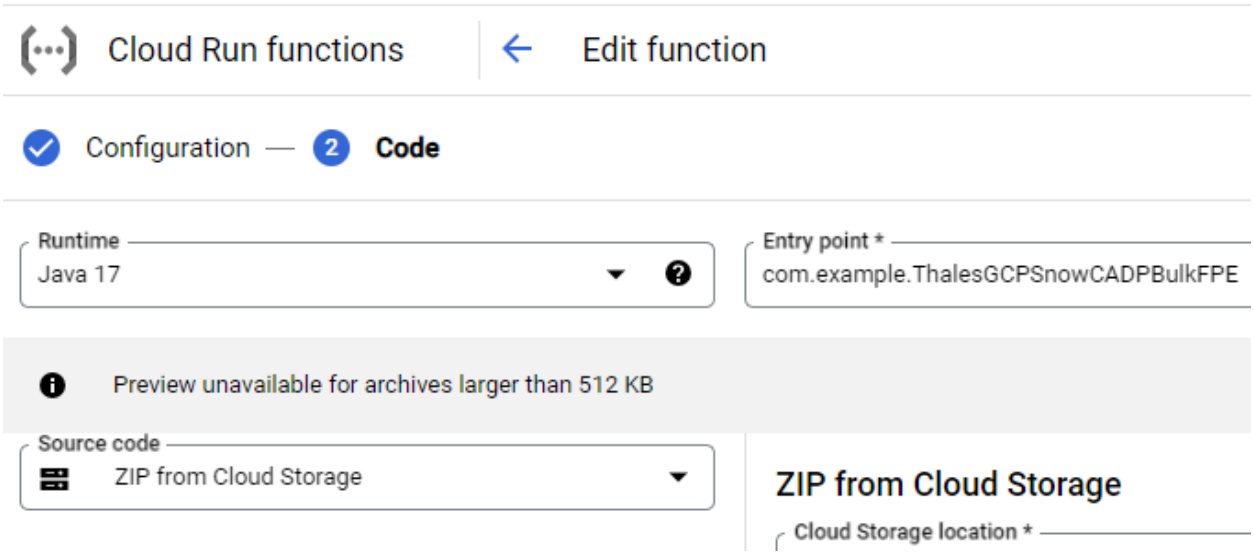
Publish jar/zip file to AWS Lambda Function or Google Cloud Function.

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



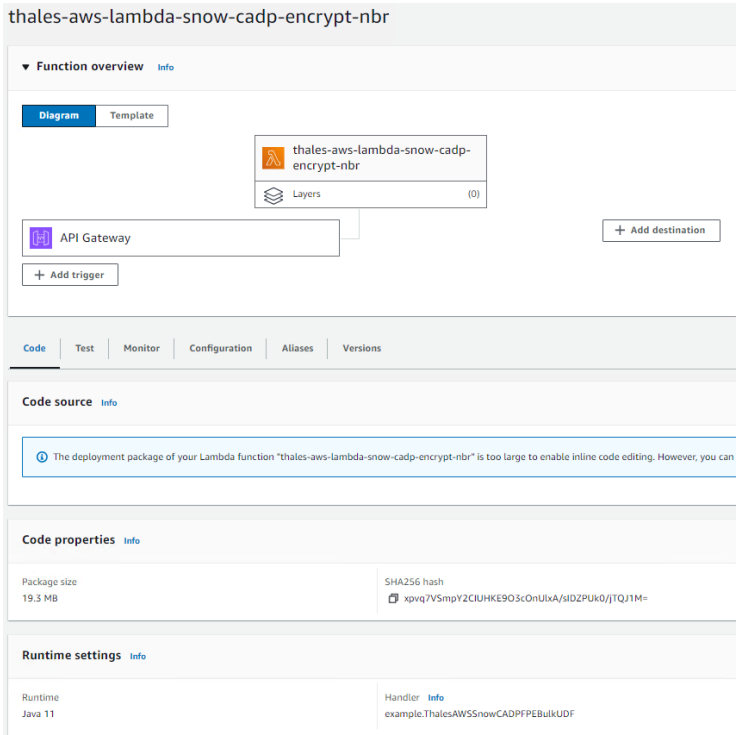
Set environment variables appropriate values (See section on Environment Variables for details) and then select Next. Upload the zip file on the next screen.



Be sure to change the Entry point to the class file you are deploying. The example above is `com.example.ThalesGCPSnowCADPBulkFPE.` 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. Set environment variables appropriate values (See section on Environment Variables for details) and then select Next. Set memory to 256MB. Further testing may allow for lower settings.



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

Once you have created the functions above and if you have already configured and setup CM with the key and all the 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. Create the API Integration to AWS proxy service.
5. Create external function object in Snowflake

Sample Function Definitions.

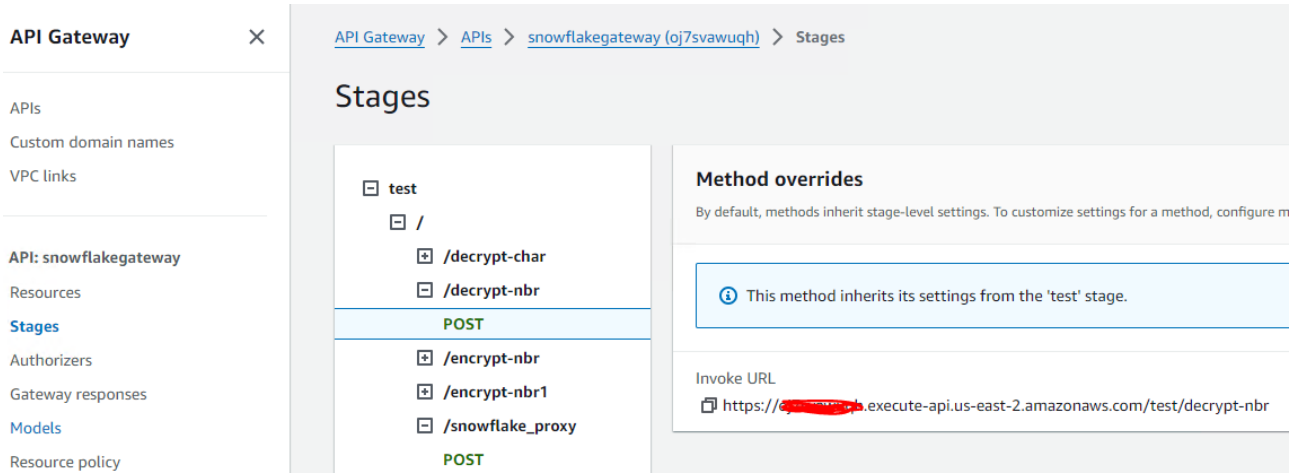
AWS Examples

```
Note: After creating the initial api integration it is important to do an
alter otherwise you will encounter issues.
        alter api integration my_api_integration_aws set
api_allowed_prefixes= ('https://yourcode.execute-api.us-east-
2.amazonaws.com/test/snowflake_proxy','https://yourcode.execute-api.us-east-
2.amazonaws.com/test/encrypt-nbr','https://yourcode.execute-api.us-east-
2.amazonaws.com/test/decrypt-char','https://yourcode.execute-api.us-east-
2.amazonaws.com/test/decrypt-nbr');

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

Note: The values for the allowed prefixes are from the gateway stage. In my case it was from this screen:



Google Examples:

```
    create or replace api integration my_api_integration_name
    api_provider = google_api_gateway
    google_audience = 'demo-api-id-for-external-function1-
yourcode.apigateway.cpl-sales-l-app-us-01.cloud.goog'
    api_allowed_prefixes = ('https://testapi-
yourcode.uc.gateway.dev','https://thalessnowcadp.gateway.dev','https://us-
central1-cpl-sales-l-app-us-01.cloudfunctions.net/echo-1')
    enabled = true;


create or replace external function THALES_CADP_GCP_ENCRYPT_NBR (b varchar)
    returns variant
    api_integration = my_api_integration_name
    as 'https://thalessnowcadp.gateway.dev/thalessnowcadpnbr';

create or replace external function ThalesSnowCADPTokenizeChar (b varchar)
    returns variant
    api_integration = my_api_integration_name
    as 'https://thalessnowcadpchar.gateway.dev/thalessnowcadptokenizechar';
```

Integration with CipherTrust Manager.

Assuming the NAE interface has already been setup based on the prerequisites the only other setup required is to have a user and key created in CM. This user should have the ability to export the key. This user will also be needed to be provided as an environment variable for the function. The examples provided have the key as a hardcoded value, but this can be easily altered to be provided as an environment variable, obtained from a secrets manager or in the header of the Json passed in from the API Gateway.

As noted above there is a test class that can be used to test connectivity with CM without having to publish the Function. If you have not updated the CADP_for_JAVA.properties file with the CM settings such as IP/NAE Port, etc. then do so now. The file is located under the resource's directory in the eclipse project. These properties can also be overwritten with CADP code as well if your desire is to pass them in as environment variables or headers of the JSON request. Here is an example:

```
System.setProperty("com.ingrian.security.nae.NAE_IP.1", "10.20.1.9");
```

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_CADP_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

Environment Variables

Listed below are the environment variables currently required for the Cloud Function.

Key	Value	Description
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)

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_cadp_aws_decrypt_char(val)
    ELSE val
  END;
```

For more information, please refer to the Snowflake documentation.

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 keyname

Currently the sample code uses a hard coded key name. Other options to be considered are:

1.)keyname as an environment variable.
2.)keyname as a GCP/AWS secret.
3.)keyname passed in an attribute to the function. Example with a database view.

```
create view employee as
select first_name, last_name,
thales_cadp_aws_decrypt_char('testfaas', email) as  email
from emp_basic
```