Integration of Databricks with CADP

Databricks [Overview]

This document describes how to configure and integrate CipherTrust Manager with Databricks.

Built on open source and open standards, a lakehouse simplifies your data estate by eliminating the silos that historically complicate data and AI. One architecture for integration, storage, processing, governance, sharing, analytics and AI. One approach to how you work with structured and unstructured data. One end-to-end view of data lineage and provenance. One toolset for Python and SQL, notebooks and IDEs, batch and streaming, and all major cloud providers. Thales provides a few different capabilities to protect data in Databricks. The most secure option is to Bring Your Own Encryption (BYOE). There are a couple of ways to utilize this capability with Databricks.

Bring Your Own Encryption (BYOE)

- **Data Ingest** – with Thales Batch Data Transformation (BDT) or User Defined Functions (UDF)
- **Data Access** –user defined functions for column level protection.

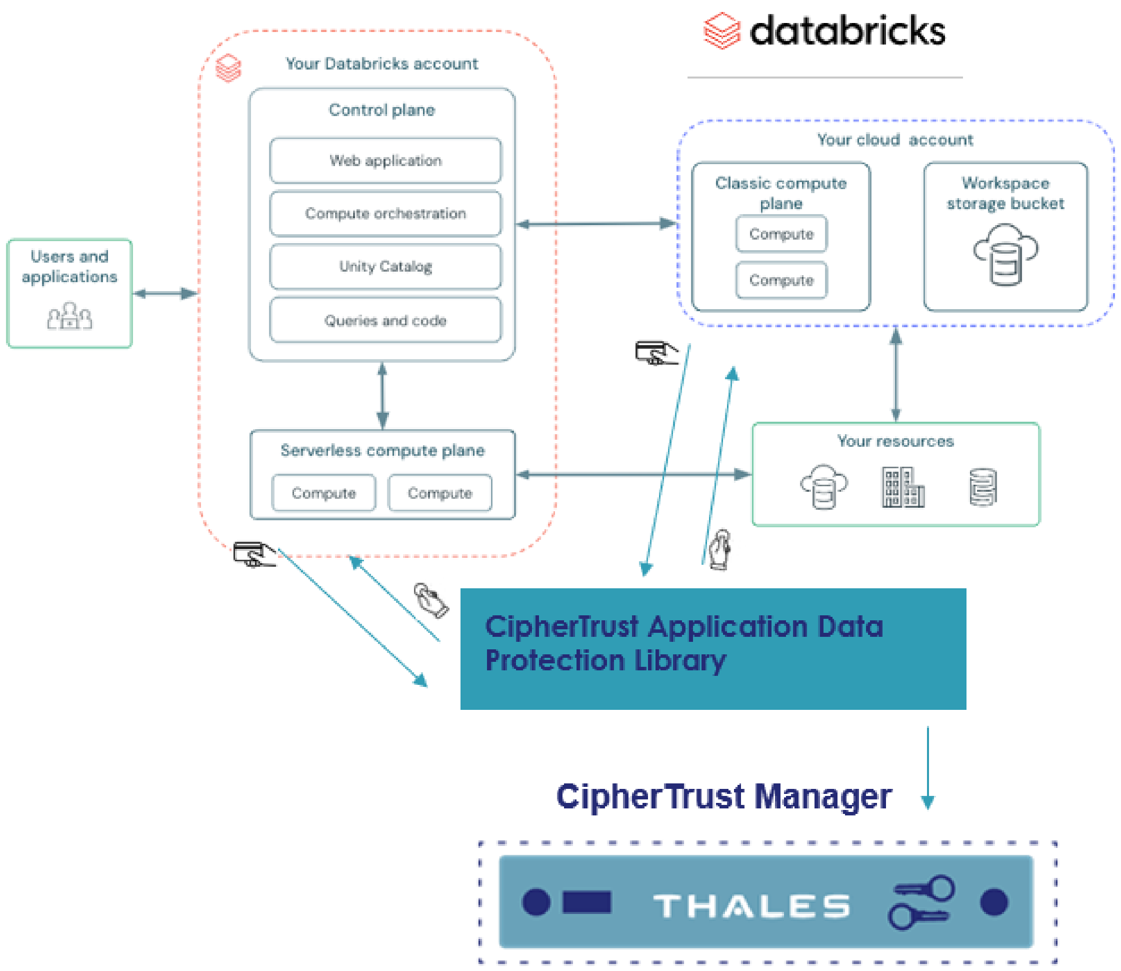
The focus of this integration will be on Data Access protecting sensitive data in Databricks columns by using CipherTrust Application Data Protection(CADP)

THALES GROUP LIMITED DISTRIBUTION - SCOPE

Architecture

Databricks can run on all three major cloud service providers AWS, Azure and GCP.   A nice feature with Databricks is that even though each CSP has its own unique way to create functions and gateways there is only one method to know with Databricks. Listed below is an example of how this integration works.

Databricks Sample Integration



Supported Product Versions

- **CipherTrust Manager** CipherTrust Manager 2.14 and higher.
- **CADP** CADP Java 8.13 and higher. Note: CADP 8.16.0.000 was tested.
- **Databricks Compute LTS 14.3 or higher.**

This integration is validated using Java 1.8.   As of October 16 higher version of Java were not supported by Databricks.

Prerequisites

- Helpful Databricks links:
  - https://docs.databricks.com/en/udf/unity-catalog.html
- 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.
- Databricks 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 Databricks.

Java UDF's are currently only supported in Databricks Compute Cluster.   Although you can execute the Java UDF's in a notebook to execute against a table in Databricks SQL Warehouse they are not supported in Databricks SQL Warehouse as a standalone UDF. Here is an example of what is supported from a Databricks notebook:

```
%sql
select ThalesencryptCharUDF(c_name) as enc_name, c_name from samples.tpch.customer
limit 50
```

Steps for Integration

- **[Installing and Configuring Thales CADP for Java]**
- **[Download code from Thales github and compile]**
- **[Publish jar file to Databricks Compute Engine]**
- **[Define and register the UDF in Databricks Notepad]**
- **[Integration with Thales CipherTrust Manager]**
- **[UDF Environment or property file 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.

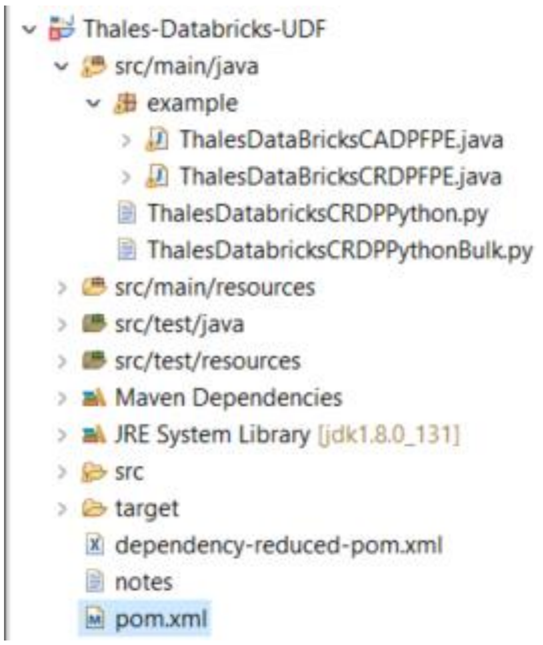
eclipse.buildId=4.15.0.I20200305-0155

m2e - Maven Integration for Eclipse

Download code from github and compile.

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

The database directory has all the code for Databricks.



```
Thales-Databricks-UDF
  src/main/java
    example
      ThalesDataBricksCADPFPE.java
      ThalesDataBricksCRDPFPE.java
    ThalesDatabricksCRDPPython.py
    ThalesDatabricksCRDPPythonBulk.py
  src/main/resources
  src/test/java
  src/test/resources
  Maven Dependencies
  JRE System Library [jdk1.8.0_131]
  src
  target
  dependency-reduced-pom.xml
  notes
  pom.xml
```

Assuming you have your CM already configured each of the classes have a main method which can be used to test locally. Make sure your CM environment is configured correctly. You will need to modify the udfConfig.properties file values such as CMUSERID, CMPWD and other necessary settings. This example uses a configuration file (udfConfig.properties) in the java project resource directory to set variables needed by the UDF. Environment variables or a secrets manager can also be used for this information. (See section on Environment Variables for details).

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 Databricks.

To compile and generate the target jar file to be uploaded to the Databricks compute platform select the project and choose "Run As" "maven install" to generate the target.

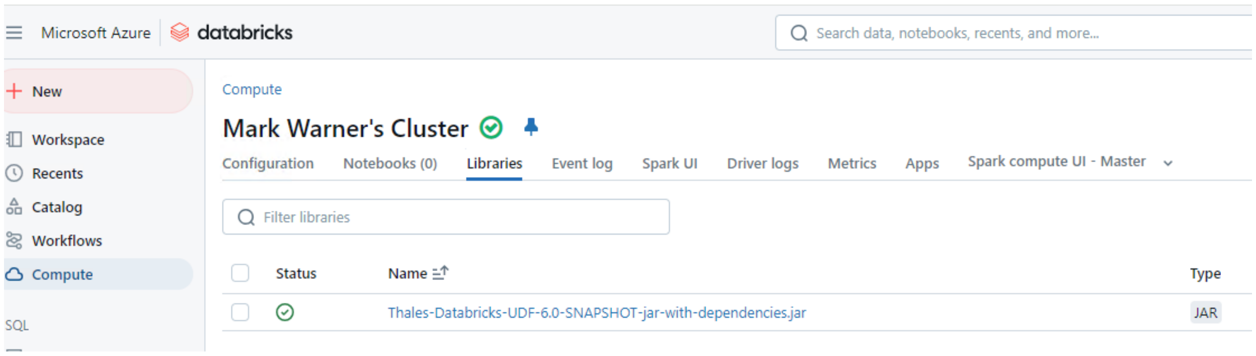
```
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ Thales-Databricks-UDF
---
[INFO] Installing C:\Users\t0185905\workspace\Thales-Databricks-UDF\target\Thales-
Databricks-UDF-6.0-SNAPSHOT.jar to C:\Users\t0185905\.m2\repository\Thales\Thales-
Databricks-UDF\6.0-SNAPSHOT\Thales-Databricks-UDF-6.0-SNAPSHOT.jar
[INFO] Installing C:\Users\t0185905\workspace\Thales-Databricks-UDF\pom.xml to
C:\Users\t0185905\.m2\repository\Thales\Thales-Databricks-UDF\6.0-SNAPSHOT\Thales-
Databricks-UDF-6.0-SNAPSHOT.pom
[INFO] Installing C:\Users\t0185905\workspace\Thales-Databricks-UDF\target\Thales-
Databricks-UDF-6.0-SNAPSHOT-jar-with-dependencies.jar to
C:\Users\t0185905\.m2\repository\Thales\Thales-Databricks-UDF\6.0-SNAPSHOT\Thales-
Databricks-UDF-6.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.482 s
[INFO] Finished at: 2024-10-17T10:28:59-04:00

[INFO] ------------------------------------------------------------------------
```

Publish jar/zip file to Databricks compute engine.

Once you have generated the jar file to upload you can then upload it to the Databricks compute engine. Here are the steps:

- Go to the "Workspace" section.
- Click on "Create" -> "Library".
- Upload your JAR file.

Here is the screen that shows the jar file uploaded:



Mark Warner's Cluster

	Status	Name	Type
	✓	Thales-Databricks-UDF-6.0-SNAPSHOT-jar-with-dependencies.jar	JAR

Once you have uploaded the functions above and if you have already configured and setup CM with the key and all the environment variables you can proceed with the next section.

Note: When using the Unity Catalog(UC) you might encounter permission errors. Check the troubleshooting section for more details on how to resolve.

Define and register the UDF in Databricks Notepad.

A nice feature with Databricks is that even though each CSP has its own unique attributes to create functions and gateways there is only one method to know with Databricks. Listed below is a summary of the entire process.

1. Set Up Your Environment

You will need a Databricks compute cluster of 14.3 or higher. Note as of October 2020 Java 1.8 was supported by Databricks.

2. Create the Thales Function in Java

Should already be done from previous steps.

3. Package the Java Code as a JAR

Should already be done from previous steps.

4. Upload the JAR to Databricks

Should already be done from previous steps. You can do this through the Databricks UI:

5. Register the UDF in Databricks

As you can see from below these functions accept 3 input values.

Key	sampledata	Description
datatype	char	datatype of column (char or nbr)
mode	protect	mode of operation(protect,or reveal
Data	345-23-3345	Can be protected or sensitive information.

Register Examples in Databricks Notebook

Character Data

```
%scala
import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.expressions.UserDefinedFunction

val ThalesencryptCharUDF: UserDefinedFunction = udf((data: String) => {
  try {
    example.ThalesDataBricksCADPFPE.thales_cadp_udf(data,"encrypt","char")
  } catch {
    case e: Exception => null
  }
})

spark.udf.register("ThalesencryptCharUDF", ThalesencryptCharUDF)
// Do not need if using UC. spark.sql("CREATE or replace FUNCTION
ThalesencryptCharUDF AS 'example.ThalesencryptCharUDF'")
spark.sql("SELECT ThalesencryptCharUDF('thisisatest')").show()
```

```
%scala
import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.expressions.UserDefinedFunction


val ThalesdecryptCharUDF: UserDefinedFunction = udf((data: String) => {
  try {
    example.ThalesDataBricksCADPFPE.thales_cadp_udf(data,"decrypt","char")
  } catch {
    case e: Exception => null
  }
})

// Register the UDF

spark.udf.register("ThalesdecryptCharUDF", ThalesdecryptCharUDF)
// Do not need if using UC. spark.sql("CREATE or replace FUNCTION
ThalesdecryptCharUDF AS 'example.ThalesdecryptCharUDF'")
spark.sql("SELECT ThalesdecryptCharUDF('Ma9zhQvxnEX')").show()
```

Number Data

```
%scala
import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.expressions.UserDefinedFunction


val ThalesencryptNbrUDF: UserDefinedFunction = udf((data: String) => {
  try {
    example.ThalesDataBricksCADPFPE.thales_cadp_udf(data,"encrypt","nbr")
  } catch {
    case e: Exception => null
  }
})

spark.udf.register("ThalesencryptNbrUDF", ThalesencryptNbrUDF)
// Do not need if using UC. spark.sql("CREATE or replace FUNCTION ThalesencryptNbrUDF
AS 'example.ThalesencryptNbrUDF'")
spark.sql("SELECT ThalesencryptNbrUDF('-46')").show()

%scala

import org.apache.spark.sql.functions.udf
import org.apache.spark.sql.expressions.UserDefinedFunction

val ThalesdecryptNbrUDF: UserDefinedFunction = udf((data: String) => {
  try {
    example.ThalesDataBricksCADPFPE.thales_cadp_udf(data,"decrypt","nbr")
  } catch {
    case e: Exception => null
  }
})

spark.udf.register("ThalesdecryptNbrUDF", ThalesdecryptNbrUDF)
// Do not need if using UC. spark.sql("CREATE or replace FUNCTION ThalesdecryptNbrUDF
AS 'example.ThalesdecryptNbrUDF'")
spark.sql("SELECT ThalesdecryptNbrUDF('914171854902')").show()
```

Assuming the NAE interface has already been setup based on the prerequisites the only other

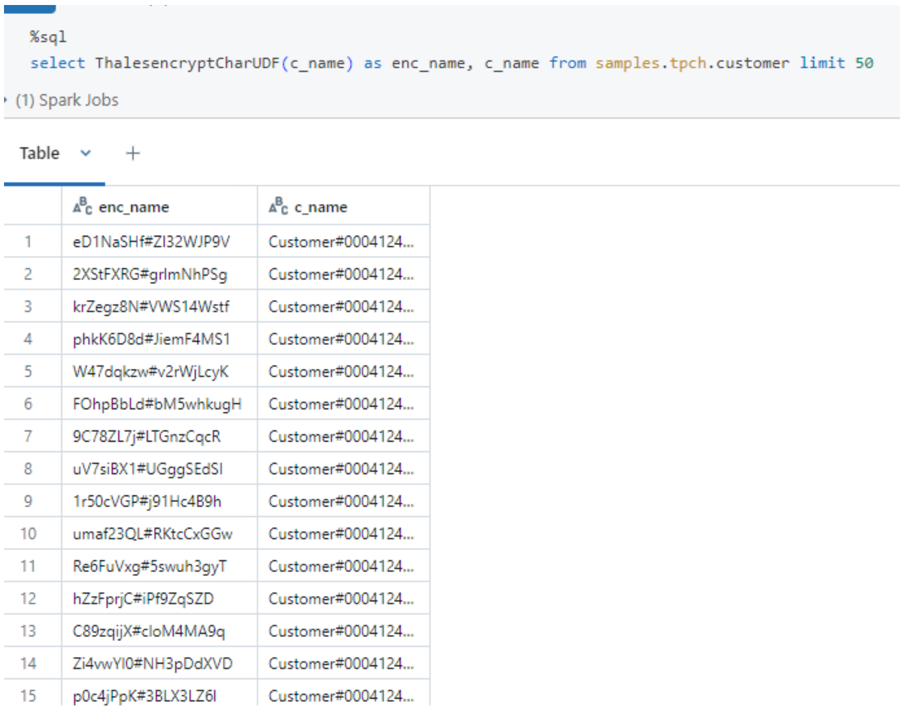
Integration with CipherTrust Manager.

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.

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. 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 be able to execute a query within a session that has already defined the function. Here is a sample query running in a databricks notepad cell using one of the UDF's. Please note this must run after the registration above in the same notebook.



```
%sql
select ThalesencryptCharUDF(c_name) as enc_name, c_name from samples.tpch.customer limit 50
```

(1) Spark Jobs

Table

	enc_name	c_name
1	eD1NaSHf#ZI32WJP9V	Customer#0004124...
2	2XStFXRG#grlmNhPSg	Customer#0004124...
3	krZegz8N#VWS14Wstf	Customer#0004124...
4	phkK6D8d#JiemF4MS1	Customer#0004124...
5	W47dqkzw#v2rWjLcyK	Customer#0004124...
6	FOhpBbLd#bM5whkugH	Customer#0004124...
7	9C78ZL7j#LTGnzCqcR	Customer#0004124...
8	uV7siBX1#UGggSEdSI	Customer#0004124...
9	1r50cVGP#j91Hc4B9h	Customer#0004124...
10	umaf23QL#RKtcCxGGw	Customer#0004124...
11	Re6FuVxg#5swuh3gyT	Customer#0004124...
12	hZzFprjC#iPf9ZqSZD	Customer#0004124...
13	C89zqijX#cloM4MA9q	Customer#0004124...
14	Zi4vwYI0#NH3pDdXVD	Customer#0004124...
15	p0c4jPpK#3BLX3LZ6I	Customer#0004124...

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
datatype	charint	datatype of column (char or charint)
mode	encrypt	mode of operation(protect,reveal,protectbulk,revealbulk) CRDP

Advanced Topics

Databricks also publishes a best practice/performance recommendation link that can also provide some options to improve performance.

https://learn.microsoft.com/en-us/azure/databricks/udf/

https://docs.databricks.com/en/sql/language-manual/sql-ref-functions-udf-scalar.html#language-java

For more information, please refer to the Databricks documentation.

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 Databricks 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
```

Controlling Access to UDF

In Databricks, Java UDFs typically run within the Spark executor context, which doesn't directly expose the user information.

Grants

With Databricks it is possible to manage permissions on UDFs (user-defined functions) in Databricks using **Unity Catalog**. Unity Catalog allows you to control access to UDFs (as well as databases, tables, views, etc.) via GRANT and REVOKE commands. This means you can specify which users or roles are allowed to execute certain UDFs by assigning appropriate privileges.

Steps to Grant Permissions on UDFs in Unity Catalog

1. **Create a UDF in Unity Catalog**: When you create a UDF in a catalog that is managed by Unity Catalog, the UDF becomes an object in the catalog. This allows you to manage access to it.

   Example of creating a UDF in SQL:

   ```
   CREATE FUNCTION catalog_name.schema_name.my_udf AS (
       x INT
   ) RETURNS INT
   COMMENT 'UDF to multiply by 2'
   LANGUAGE PYTHON
   RETURN x * 2;
   ```

2. **Grant Execute Permission**: After creating the UDF, you can grant `EXECUTE` permission on it to specific users, groups, or roles.

THALES GROUP LIMITED DISTRIBUTION - SCOPE

Example SQL to grant execute permission to a specific user:

```
GRANT EXECUTE ON FUNCTION catalog_name.schema_name.my_udf TO
`user@example.com`;
```

If you want to grant permissions to a group or role, you would do something like this:

```
GRANT EXECUTE ON FUNCTION catalog_name.schema_name.my_udf TO
`data_scientist_group`;
```

3. **Revoke Execute Permission**: Similarly, you can revoke permissions if needed:

```
REVOKE EXECUTE ON FUNCTION catalog_name.schema_name.my_udf FROM
`user@example.com`;
```

4. **Check Permissions**: To verify the permissions on the UDF, you can use the `SHOW GRANTS` command:

```
SHOW GRANTS ON FUNCTION catalog_name.schema_name.my_udf;
```

Privileges You Can Grant on UDFs

- **EXECUTE**: Allows the user to execute the UDF.
- **USAGE**: Allows the user to reference the schema or catalog containing the UDF. This might be required alongside the `EXECUTE` privilege, depending on your environment setup.

Unity Catalog Requirement

This type of fine-grained access control for UDFs is only available if you are using **Unity Catalog** in Databricks. If you're not using Unity Catalog, you won’t be able to manage access control for UDFs directly using GRANT/REVOKE commands. In non-Unity Catalog environments, UDFs generally do not have per-user permission management beyond the standard Databricks workspace access control (e.g., cluster or notebook permissions).

Without Unity Catalog

If you are not using Unity Catalog, a workaround for controlling access to UDFs would be to create separate UDFs or notebooks for different user groups and control access via workspace permissions (like notebook permissions or cluster permissions), but this is not as granular as the control Unity Catalog provides.

Summary

- **Unity Catalog** allows for fine-grained access control on UDFs with `GRANT EXECUTE` and `REVOKE EXECUTE` commands.

THALES GROUP LIMITED DISTRIBUTION - SCOPE

- Without Unity Catalog, you'll need to control access through more general workspace or cluster permissions.

Environment or session variables

It is possible to set environment variables or session-level variables in Databricks that are accessible to a UDF, and you can do this without allowing users to modify the values directly. Here's how you can manage this:

1. Using Spark Configuration Properties:

You can set key-value pairs in the **Spark session configuration** which can be accessed by your UDF. This prevents users from directly modifying these variables.

- **Set the variable in the notebook**: You can set these values at the session level using `spark.conf.set`. For example:

  ```
  spark.conf.set("myapp.secretKey", "super_secret_value")
  ```

- **Access the variable in your UDF**: In your UDF, you can access this configuration property using the `SparkContext` or `SparkSession`.

  Example in Java UDF:

  ```
  import org.apache.spark.sql.api.java.UDF1;
  import org.apache.spark.sql.SparkSession;

  public class MyUDF implements UDF1<String, String> {
      @Override
      public String call(String input) throws Exception {
          SparkSession spark = SparkSession.builder().getOrCreate();
          String secretKey = spark.conf().get("myapp.secretKey");
          // Use the secretKey in your logic
          return "Processed with secret: " + secretKey;
      }
  }
  ```

- **Ensure users cannot modify**: Users of the notebook will not be able to modify the `spark.conf` settings unless they have admin-level access to the notebook. If you manage cluster configurations or the session's scope carefully, they won't have direct access to `spark.conf`.

2. Using Databricks Secrets:

Databricks provides **Databricks Secrets**, which are a secure way to store sensitive information (like API keys, database credentials, etc.) and are not visible to the end user. You can configure environment-specific secrets and access them within UDFs without exposing the actual values.

THALES GROUP LIMITED DISTRIBUTION - SCOPE

- **Set up the secret**: In Databricks, you can store secrets in a secrets scope using the UI or the CLI:

  ```
  bash
  databricks secrets create-scope --scope myapp_secrets
  databricks secrets put --scope myapp_secrets --key secretKey
  ```

- **Access the secret in your UDF**: You can access the secret in your UDF via the `dbutils.secrets.get` method.

  Example in Python:

  ```
  secretKey = dbutils.secrets.get(scope="myapp_secrets", key="secretKey")
  ```

  In a Java UDF, you could pass the value of the secret from Python as an argument or configure it in `spark.conf` like this:

  ```
  python
  spark.conf.set("myapp.secretKey", dbutils.secrets.get("myapp_secrets",
  "secretKey"))
  ```

3. Using Global Variables in Notebooks (if controlled by admin):

You could define certain global variables or environment variables directly within the notebooks by setting them at the beginning of the notebook, and the users simply use them without having permission to modify them.

Example:

```
python
# Define global constants
SECRET_KEY = "super_secret_value"
ENDPOINT_URL = "https://api.example.com"

# Use them in the notebook without users being able to modify them
print(f"Using secret key {SECRET_KEY}")
```

This approach, however, is less secure than using `spark.conf` or Databricks secrets.

4. Control Cluster Settings:

If you're managing a cluster that multiple users are accessing, you can set environment variables or configuration settings at the cluster level via the Databricks **cluster configuration** UI. For instance, under the "Advanced Options" > "Spark Config" section, you can add key-value pairs:

```
spark.myapp.secretKey=super_secret_value
```

These configuration settings can be accessed by UDFs and are not exposed to users.

THALES GROUP LIMITED DISTRIBUTION - SCOPE

Summary

- **`spark.conf.set`**: You can use Spark session configuration to set variables, which can be accessed within UDFs.
- **Databricks Secrets**: For sensitive information, you can securely store secrets using Databricks Secrets and retrieve them without exposing the values.
- **Cluster-level settings**: Set environment variables or configuration values at the cluster level for greater control over what users can access.

By using these methods, you ensure that the variables are set automatically for the users and cannot be modified directly.

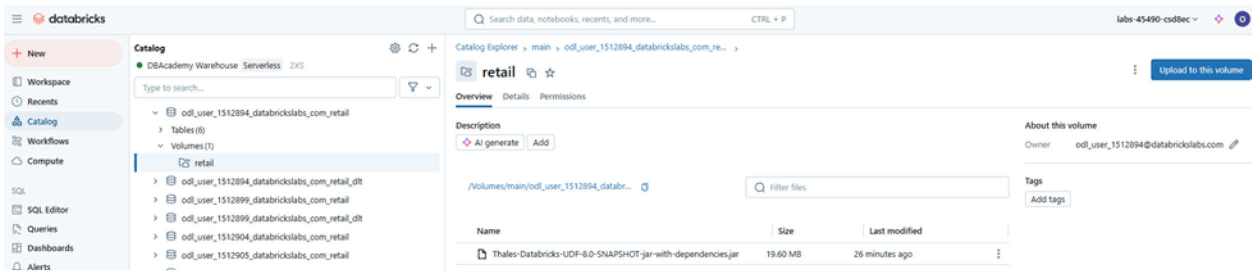
Troubleshooting

Importing jar files.

When using the Unity Catalog(UC) and importing jar files you might encounter permission errors such as:

```
Error code: WSFS_LIBRARY_INSTALLATION_FAILURE. Error message: java.util.concurrent.ExecutionException: com.databricks.backend.daemon.driver.WSFSCredentialForwardingHelper$WorkspaceFilesystemException: Failed to update command details for Files in Repos. Set cluster environment variable WSFS_ENABLE=false if you do not need the feature on this cluster.
```

If so create a UC volume and then upload the jar file to the volume before importing into the library. You must select Volume when doing the import. Here is a screenshot showing the volume located in a UC.



DBAcademy Warehouse Serverless 2XS

odl_user_1512894_databrickslabs_com_retail
Tables (6)
Volumes (1)
retail
odl_user_1512894_databrickslabs_com_retail_dlt
odl_user_1512899_databrickslabs_com_retail
odl_user_1512899_databrickslabs_com_retail_dlt
odl_user_1512904_databrickslabs_com_retail
odl_user_1512905_databrickslabs_com_retail

/Volumes/main/odl_user_1512894_databr...

Name	Size	Last modified
Thales-Databricks-UDF-8.0-SNAPSHOT-jar-with-dependencies.jar	19.60 MB	26 minutes ago

Owner odl_user_1512894@databrickslabs.com