Collibra and Snowflake

Policy enforcement v1.0.2

Name	Collibra and Snowflake for Data Access Policy enforcement
Category	Integration
Use Case	To share controlled and secure data access across Snowflake analytic, machine learning and ETL services.
Target Audience	Policy enforcement
Who can set it up	Integration Engineer
Target Business Functions	Any
Target Industries	Any
License Requirements	Collibra Platform Collibra Catalog
Dependencies	Collibra Platform v2021+ Snowflake JDBC Driver 3.13.3 Java Development Kit v1.11 Spring Boot framework v2.5.0
Developer	Collibra
License	Binary Code License Agreement
Short Description	A Spring Boot integration that takes requests from Collibra and consume Snowflake jdbc connections to create and update object tags, attach, and detach object tags to resources (columns), grant and revoke roles to and from Snowflake account users.

Overview

Snowflake access control is **a service that makes it easy to set up, secure, and manage your data lake**. Snowflake Access control privileges determine who can access and perform operations on specific Snowflake objects. Snowflake's approach to access control combines aspects from both the following two models:

- **Discretionary Access Control (DAC):** Each object has an owner, who can in turn grant access to that object.
- **Role-based Access Control (RBAC):** Access privileges are assigned to roles, which are in turn assigned to users.

In the Snowflake model, access to securable objects is allowed via privileges assigned to roles, which are in turn assigned to other roles or users. In addition, each securable object has an owner that can grant access to other roles. This model is different from a user-based access control model, in which rights and privileges are assigned to users. The Snowflake model is designed to provide a significant amount of both control and flexibility.

Roles are the entities to which privileges on securable objects can be granted and revoked. Roles are assigned to users to allow them to **perform actions required for business functions** in their organization. This allows users to switch to and from roles (i.e. choose which role is active in the current Snowflake session) to perform different actions using separate sets of privileges.

This integration will map roles to Collibra **business processes** such as direct marketing or customer churn analysis but could well be extended and applied to any available asset type with little to no effort. If you wish to do that, please note that access requesters will need to understand what role they need to be granted access to, and access approvers will need to be know exactly what that access entails.

Requirements

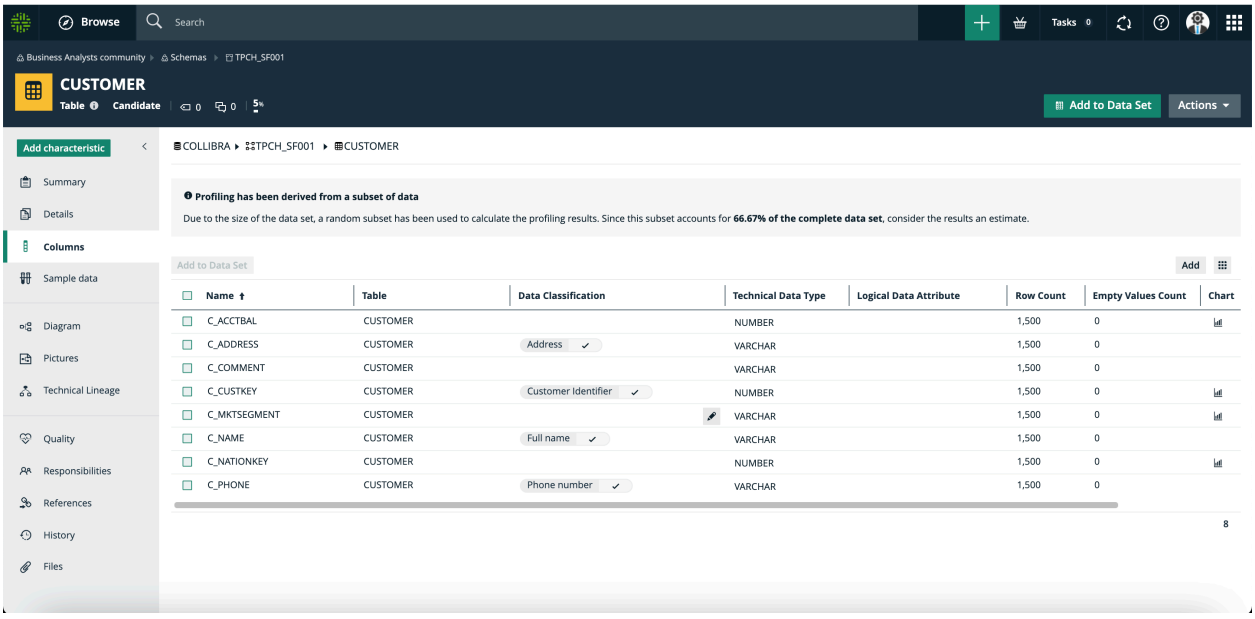
☑ As the data lake admin, I want to add and remove data classifications to Snowflake resources so I can enable fine-grained access controls

☑ As the data lake admin, I want to add and remove data categories to Snowflake resources so I can enable fine-grained data access controls

- ☑ As the data lake admin, I want to know what classifications and categories are and have been granted to each available Snowflake resource
- ☑ As the data consumer I want to request and have access granted to given named resources so I can leverage data for competitive advantage
- ☑ As the data consumer I want to know of all my data access requests which data access requests have or have not been granted and the reasons
- ☑ As the data steward or data owner I want to share controlled secure access to the right data to the right user, so I protect available sensitive data
- ☑ As the data steward or data owner I want to know of all the data access requests which requests have or have not been granted and the reasons

Data classification

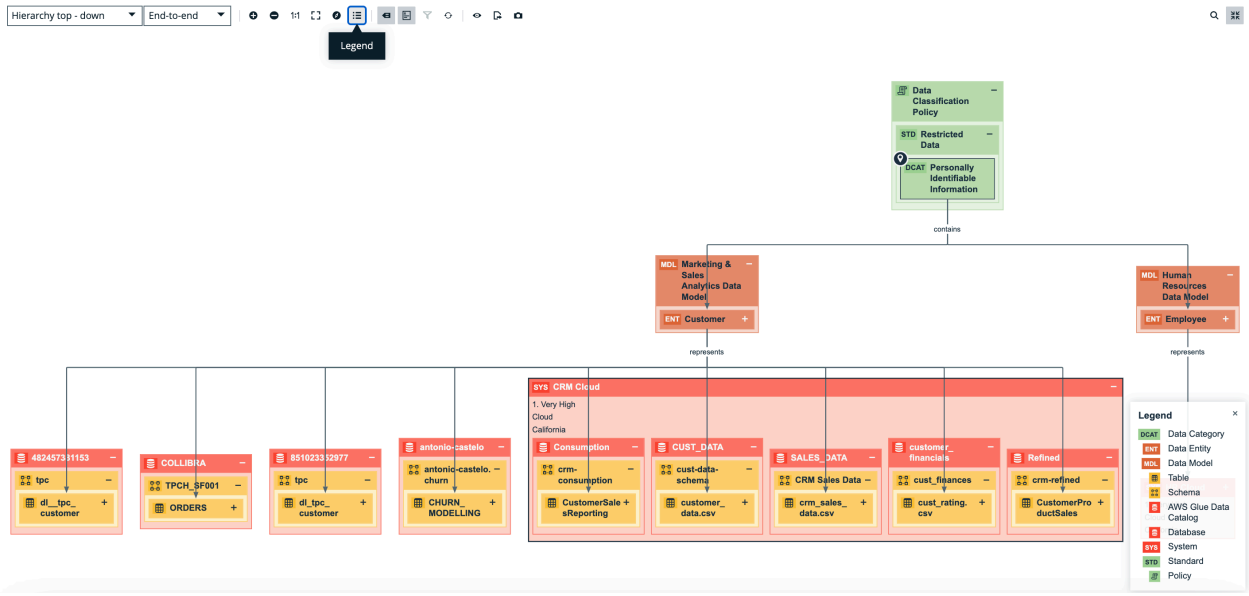
After a data table is ingested, a data steward is required to navigate to the table asset page and run classification. Data Classification leverages Machine Learning to help **classify and tag** the data in the various columns of a table. Stewardship takes those recommendations and let a Steward map them to a conceptual model so they can ultimately be connected to concepts familiar to the busines users. Workflows can be built to further automate this process by automatically accepting recommendations when above a given confidence threshold. Please refer to the Collibra Product documentation here for more information about data classification. For a complete list of the packaged data classes please check here.



Tags by themselves don't enforce any security controls but applying a good tagging strategy is a great way to describe the data. Tags are key-value pairs that you can apply for your Snowflake resources, including table and columns in your data lake. This integration applies a very simple tagging strategy: for the columns that have been classified as Name, make Classification tag equals Name; for columns that contain Personally Identifiable Information, make Category tag equals Personally Identifiable Information. The tagging strategy can be extended and applied to any available asset type with little to no effort.

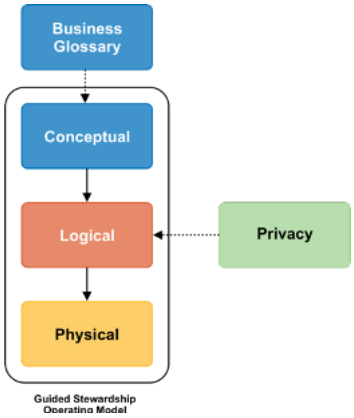
Data categories

Data Categories allow Privacy Stewards to classify data, specifically as it concerns its level of sensitivity, value and criticality. They help determine privacy and security controls for data protection. Take for example the Personally Identifiable Information (PII) data category. PII is classified as Restricted Data and unauthorized disclosure, alteration or destruction may result in significant risk to the organisation and its data subjects. It requires the highest level of access control and security protections whether in storage or in transit. The diagram below shows where your customer PII restricted data is found.



Before granting access to the required data, it is vital to make sure that identity and access management rules are adhered to and the principals of regulatory concerns, if appropriate, are enforced. When data access policies are straightforward and can be addressed at the time of extraction (e.g., Social Security Number must be masked or removed), it can and should be enforced as part of the extraction instructions, where it is most efficient. Some enforcement though cannot be achieved prior to or at the time of

data extraction. For example, some attributes by themselves (e.g., Last Name or Date of Birth) are not considered Personally Identifiable Information (PII); however, in combination with one or more attributes (e.g., Last Name + Date of Birth), can become PII. Thus, the request for and availability of identity attributes must be determined both at time of data extraction and after in a continuous exercise ensuring policies are enforced.



You are required to connect the physical data layer to the logical data layer by filtering on the conceptual data layer. Unless fully connected, no access should be granted unless explicitly approved by the right authorities tasked to ensure that sensitive data is not lost, misused, or accessed by unauthorized users.

Data usage requests

As new users and workloads are onboarded to your data lake, security and governance become more of a priority - and in many cases, a hindrance to the data scientists and analysts seeking to leverage data for competitive advantage and business innovation. The costs of failing to protect sensitive data are high and can include regulatory penalties, reputational damage, even a direct loss of customers.

Collibra and Snowflake enables data platform teams to provide secure data access at scale, service the enterprise governance requirements, and enable self-service data analytics to increase usage and adoption of the data lake and ensure its success within the enterprise. Collibra and Snowflake enforces data access policies at run-time, so each user will only see the data they are authorized to view. Access restrictions can be applied at the table and column, and Snowflake allows for a variety of de-identification types, including masking, redaction, tokenization. Technical stewards (and owners if no steward assigned) are prompted to grant the requesting user access to resources.

Snowflake access security

Snowflake ensures the highest levels of governance for your account and users, as well as all the data you store and access in Snowflake.

Features	Snowflake Editions
Support for masking column data in tables and views using masking policies.	Enterprise (or higher)
Support for determining which rows are visible in a query result using row access policies.	Enterprise (or higher)
Support for auditing the user access history through the Access History view.	Enterprise (or higher)

Note: Row access policies and access history services are out of scope of this document.

Check out Using Dynamic Data Masking with Snowflake for more details here and here.

Column-level Security allows the application of a masking policy to a column within a table or view and currently comprises two features:

- Dynamic Data Masking
- External Tokenization

External Tokenization allows organizations to tokenize sensitive data before loading that data into Snowflake and dynamically detokenize data at query runtime using masking policies with External Functions. It is important to notice that external tokenization cannot be used with Snowflake Data Sharing today.

Factor	Dynamic Data Masking	External Tokenization	Notes
Preserve analytical value after de-identification.		✔	Since tokenization provides a unique value for a given set of characters, it is possible to group records by a tokenized value without revealing the sensitive information. For example, group medical records by diagnosis code with the patient diagnosis code tokenized. Data analysts can then query a view on the diagnosis code to obtain a count of the number of patients with a unique diagnosis code.
Pre-load tokenized Data.		✔	Unauthorized users never see the real data value. Requires third-party tokenization provider.
Pre-load unmasked data.	✔		Only need built-in Snowflake functionality, no third-parties required.
Data Sharing.	✔		External functions cannot be used with Data Sharing, therefore no External Tokenization.
Ease of use and Change management.	✔	✔	Write a policy once and have it apply to thousands of columns across databases and schemas.
Data administration and SoD.	✔	✔	A security or privacy officer decides which columns to protect, not the object owner. Masking policies are easy to manage and support centralized and decentralized administration models.
Data authorization and governance.	✔	✔	
Contextual data access by role or custom entitlements.	✔	✔	Supports data governance as implemented by security or privacy officers and can prohibit privileged users with the ACCOUNTADMIN or SECURITYADMIN role from unnecessarily viewing data.

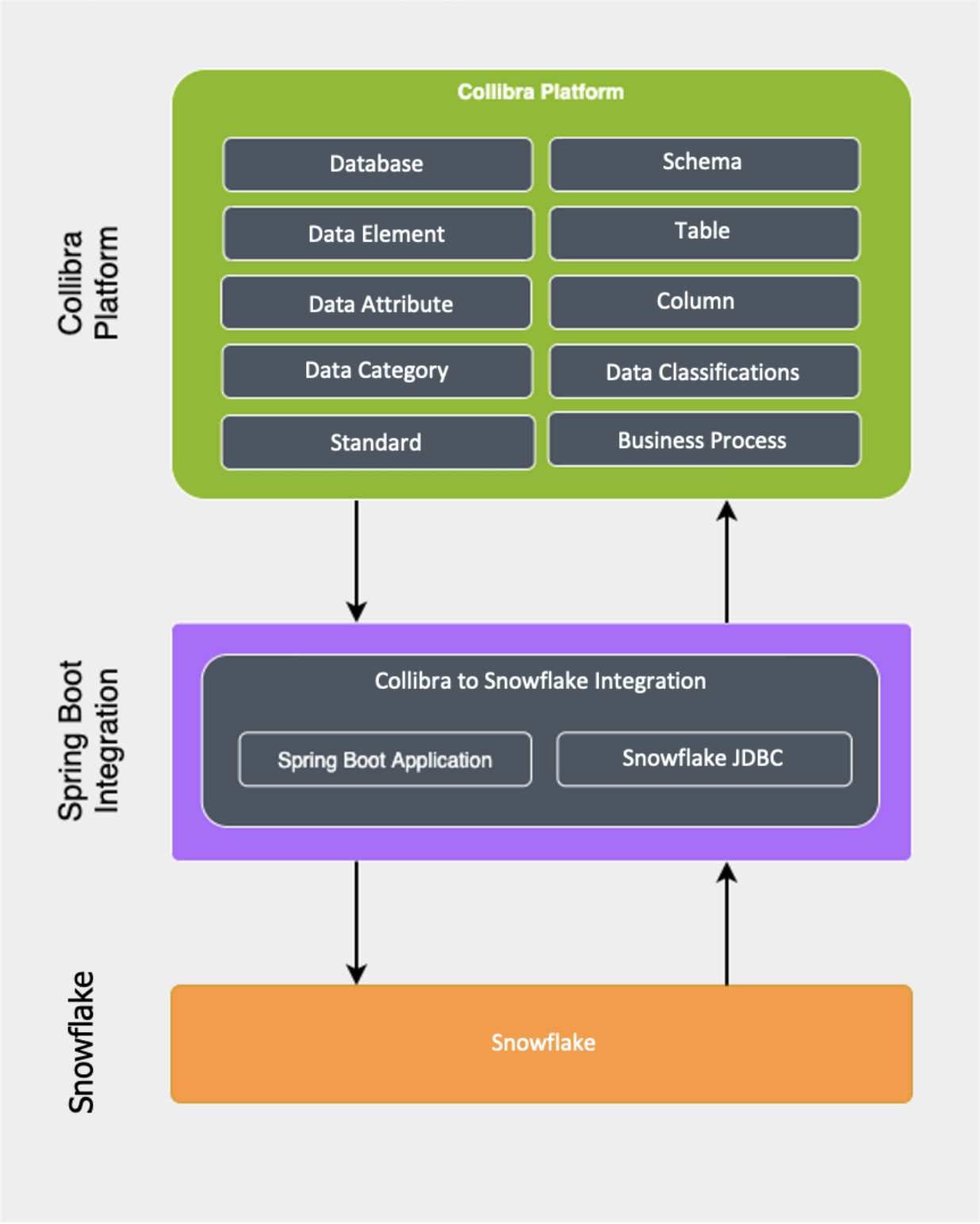
Note: External Tokenization is out of the scope of this document.

Constraints worth noting

- ☑ Snowflake tags and roles must be unique for your schema.
- ☑ Snowflake tags and roles must start with an alphabetic character and cannot contain spaces or special characters unless the entire identifier string is enclosed in double quotes. Identifiers enclosed in double quotes are also case-sensitive.
- ☑ Snowflake stored procedures use JavaScript and, in most cases SQL where JavaScript provides the control structures (branching and looping) and SQL is executed by calling functions in a JavaScript API.
- ☑ Snowflake limits tags to one tag value per tag key.
- ☑ **Snowflake does not support different input and output data types in a masking policy**, such as defining the masking policy to target a timestamp and return a string (e.g. ***MASKED***); the input and output data types must match.
- ☑ **Snowflake applies the masking policy to table or view column, not an object tag.**
- ☑ Currently, can't tell from within the masking policy which column this masking policy is set to or running against and there's only an handful number of functions available.

- ☑ Requires **Collibra Catalog Guided Stewardship Workflow + Sample content** here.

This integration targets those customers who have their data in Snowflake and aim at delivering controlled secure access to the right data, to the right persona, at the right time.



This integration will map roles to Collibra business processes such as direct marketing or customer churn analysis but could well be extended and applied to any available asset type with little to no effort. If you need or wish to do that, please note that data access

requesters will need to understand what roles or groups they need to be granted access to, and data access approvers will need to be know exactly what that data access entails.

☑ Supports basic authentication only

Assets

Collibra Data Intelligence Platform assets

Attributes	Description
Catalog or Database Identifier	The Snowflake catalog Identifier (account Id).
Principal or User Identifier	The user being granted or revoked access.
Tag Key	The Snowflake object tag key name.
Tag Values	The Snowflake object tag key values. *
Tag Values To Add	The Snowflake object tag key values to add. *
Tag Values To Delete	The Snowflake object tag key values to delete.
Database Name	The Snowflake catalog Identifier (account Id).
Schema Name	The Snowflake catalog Database name or Id.
Table Name	The Snowflake catalog Table name identifier.
Column Names	A list of Snowflake catalog Column identifiers.

* Snowflake limits to one tag value per tag key

Asset Type	Description
Add Tags To Snowflake Resource	An issue or ticket issued when requesting adding an object tag to a given Snowflake catalog named resource such as a column of a table.
Grant Snowflake Permissions to User	An issue or ticket issued when requesting access to given Snowflake catalog named resources such as column of table, or existing user role.
Remove Tags from Snowflake Resource	An issue or ticket issued when requesting removing an object tag from a given Snowflake catalog named resource such as a column of a table.

Revoke Snowflake Permissions from User	An issue or ticket issued when requesting revoking access to given Snowflake catalog named resources such as column of table, or existing user role
Update Snowflake Tag Key and Values	**Not required**. An issue or ticket issued when requesting creating or updating a given Snowflake tag key and values. Can be fulfilled by the Add Tags to Snowflake Resource above if not already found.

Workflows	Description
Add Category to Snowflake Resource	Attach a data category to an existing Snowflake catalog named resource.
Add Classification to Snowflake Resource	Attach a classification to an existing Snowflake catalog named resource.
Remove Classification from Snowflake Resource	Detach a classification from an existing Snowflake catalog named resource.
Grant Access To Snowflake Resources	Grant user access to an existing Snowflake catalog named resource.
Grant Access To Snowflake Roles	Grant user access to an existing Snowflake catalog policy tag value.
Request Access To Snowflake Roles	Request user access to an existing Snowflake catalog policy tag.
Revoke Access to Snowflake Roles	Revoke user access to an existing Snowflake catalog policy tag value.
Data Provisioning Process	Request access to all data sets referenced in the shopping cart. The owners of the related data will have to approve the request and access provisioned.
Post Http Requests to Snowflake	Post http requests when an issue or ticket issued has been successfully created and all its required attributes successfully updated. This workflow is making use of java.net which is last resort, not recommended and will soon be disallowed. Instead, we will, soon enough, make a springboot app available which will be pulling ticket requests instead. Check the next paragraphs for more details.

Collibra Data Intelligence Integration assets

Spring Boot starters	Description
spring-boot-starter-web	Starter for building web, including RESTful, applications using Spring MVC.
spring-boot-starter-log4j2	Starter for using Log4j2 for logging.
spring-boot-starter-test	The primary dependency that contains most elements required for our tests.
spring-boot-starter-thymeleaf	Starter for building MVC web applications using Thyme leaf views.
spring-boot-starter-data-jpa	Starter for using Spring Data JPA with Hibernate.
spring-boot-starter-security	Starter for using Spring Security.
spring-boot-starter-actuator	Starter for using Spring Boot's Actuator which provides production ready features to help you monitor and manage your application.
micrometer-registry-prometheus	Application monitoring instrumentation façade to Prometheus systems monitoring and alerting toolkit.
spring-boot-admin-starter-server	Starter for using Spring Boot Admin Server.
spring-boot-admin-server-ui-login	The Spring Boot Admin Server Login UI.
spring-boot-admin-starter-client	Starter for registering client Spring Boot apps.
springdoc-openapi-ui	Starter for helping to automate the generation of API documentation using spring boot projects.
jasypt-spring-boot-starter	Starter for Java Simplified Encryption.
snowflake-jdbc	Snowflake JDBC Driver
commons-io	The Apache Commons IO library contains utility classes, stream implementations, file filters, file comparators, endian transformation classes, and much more.

Spring Boot packages	Description
com.collibra.snowflake.template.config	Bean definitions for this simple web application context including springdoc OpenApi 3 configuration for automatically generating swagger OAS3 documentation in JSON/YAML and swagger UI.

com.collibra.snowflake.template.controller	The web controllers for this simple web application. Includes controllers for the command, role, and tag services. Annotations ensure http get and post requests are mapped to the right service methods.
com.collibra.snowflake.template.model	The JPA entities for this simple web application. Entities in JPA are nothing but POJOs representing data (commands issued in our case) that can be persisted to a relational database like Snowflake.
com.collibra.snowflake.template.repository	The repository interfaces for each domain entity in this simple web application. It contains methods for performing CRUD operations including some of count, delete, deleteById, save, findById, and findAll.
com.collibra.snowflake.template.security	Enable Spring Security's web security support and provide the Spring MVC integration. Defines which URL paths should be secured and which should not. Paths have been configured not to require any authentication, but these could/should be changed.
com.collibra.snowflake.template.service	The service components holding this simple web application business logic.
Apllication	This simple Web Application

Spring Boot resources	**Description**
static/*	UI objects and styling aspects can be located here.
templates/*	Template files. All the SQL scripts do set, or unset tags, grant or revoke roles can be located here.
application.properties	A list of common Spring Boot properties, additional Spring configuration metadata and references to the underlying classes that consume them. Sample list: server.port=8080 management.server.port=8081 spring.application.name=Snowflake agent spring.boot.admin.context-path=/ springdoc.swagger-ui.path=/swagger-ui.html

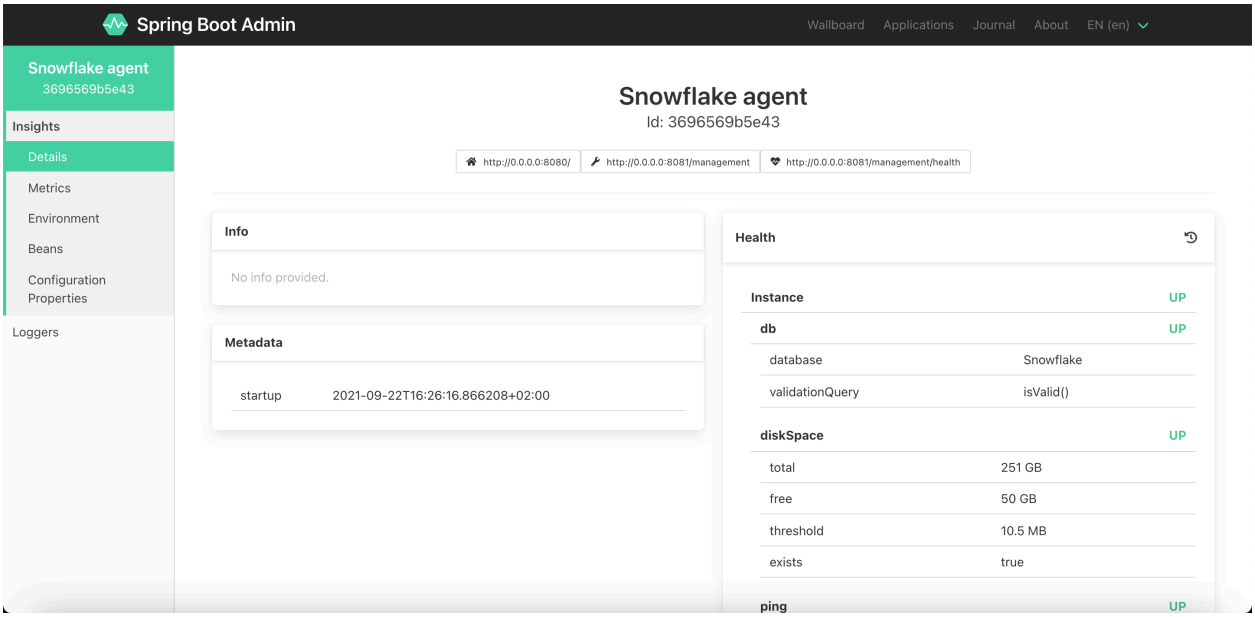
Configuration resources	**Description**
pom.xml	XML file that contains configuration details used by Maven to build this simple web application project.

Dockerfile	Text document that contains all the commands to assemble a Docker image.

Expose and use Spring Boot Admin Server

Admin endpoint	Description
/	Expose operational information about the running application including health, metrics, info, env, etc

Spring Boot Admin is a community project under Apache License 2.0 to manage and monitor your Spring Boot applications.

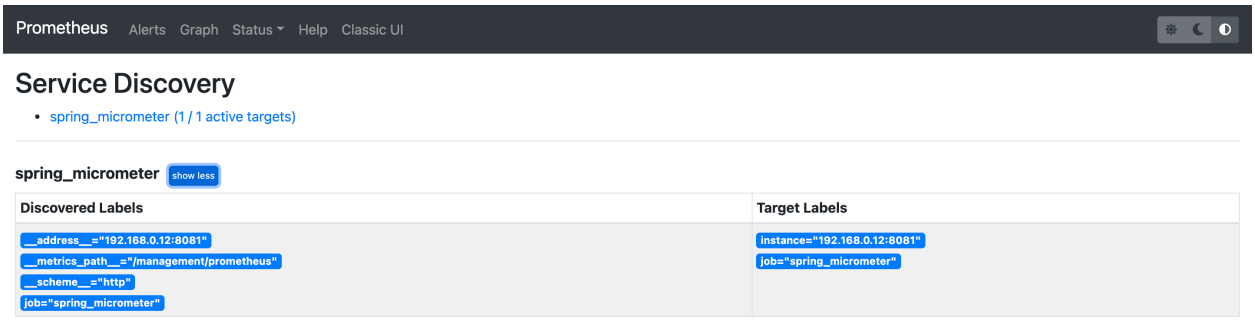


The UI is just a Vue.js application on top of the Spring Boot Actuator endpoints.

Expose and monitor operational information

Management endpoint	Description
/management	Expose operational information about the running application including health, metrics, info, env, etc

Spring Boot Actuator endpoints let you monitor and interact with your application. Each individual endpoint can be enabled and made remotely accessible over HTTP and metrics fed to monitoring systems like Prometheus and to graphing solutions like Grafana.



Prometheus is usually used along side Grafana. Grafana is a visualization tool that pulls Prometheus metrics and makes it easier to monitor.

Grafana also lets you set alert rules based on your metrics data.

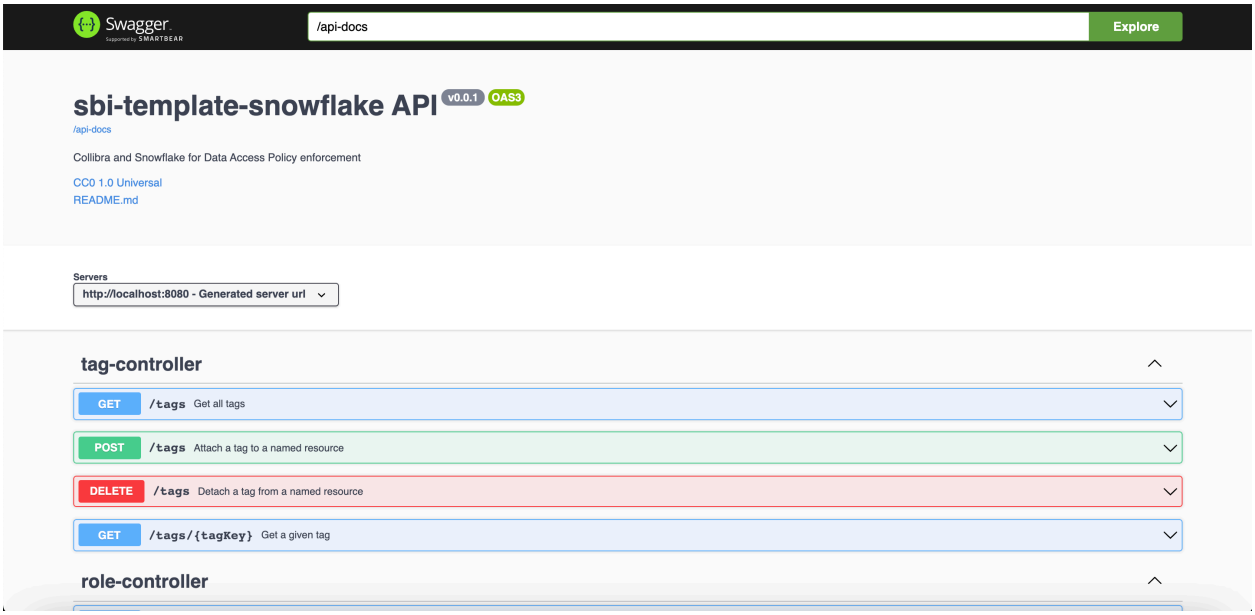
Spring Boot uses Micrometer, an application metrics facade to integrate actuator metrics with external monitoring systems. It supports several monitoring systems like Netflix Atlas, AWS Cloudwatch, Datadog, InfluxData, SignalFx, Graphite, Wavefront, Prometheus etc.

Visualize and interact with OAS3 resources

OAS3 Documentation	Description

/swagger-ui.html	Visualize and interact with the OpenAPI's resources.

Swagger UI allows you to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from the OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back-end implementation and client-side consumption.



Install the Collibra Metadata Archive (cma)

Migration allows an administrator to export parts of the Operating Model from one Collibra instance, and import it into another Collibra instance, while retaining the resource IDs. It is highly recommended not to make changes directly to the metamodel of your Production environment. Instead, we recommend you use a separate Development environment and Testing environment and make sure these changes work well for you before pushing changes to the metamodel of your Production environment.

1. Create a backup of your production environment.
2. Restore the backup in your Development and Testing environments.
3. Make the necessary changes in your development environment if any.
4. Re-apply the changes from the Development environment to the Testing environment.
5. Thoroughly test the changes in the Testing environment.
6. If required, repeat steps 3 to 5.

7. When satisfied with the changes, re-apply the changes to your Production environment.

To apply changes and import the resources simply upload the included **cma** archive file.

Install the Spring Boot Application archive

Spring Boot's flexible packaging options provide a great deal of choice when it comes to deploying your application. You can deploy Spring Boot applications to a variety of cloud platforms, to container images (such as Docker), or to virtual/real machines.

This distribution includes the compiled code and package it in its distributable format JAR for you. The archive can be found under the target folder **target/sbi-template-snowflake-1.0.2-SNAPSHOT.jar**. Check the Dockerfile file if you choose to deploy to Containers.

Spring Boot's executable jars are ready-made for most popular cloud PaaS (Platform-as-a-Service) providers and have everything that it needs to run packaged within these.

In addition to running Spring Boot applications by using **java -jar**, it is also possible to make fully executable applications for Unix systems. A fully executable jar can be executed like any other executable binary or it can be registered with **init.d** or **systemd**.

To create a 'fully executable' JAR with Maven, use the following plugin configuration:

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <executable>true</executable>
    </configuration>
</plugin>
```

Check the next paragraphs for details on how to compile and package your JAR archive.

Package your app in a distributable format

Spring Boot is compatible with Apache Maven 3.2 or above. If you don't already have Maven, first download it from https://maven.apache.org. If your machine doesn't already have Java 11 or above, first download Java 11 or above from the official Oracle website.

This distribution creates a project structure and a Maven build specification for you. The pom file lists the minimum requirement for this simple Spring Boot web application. If you need or wish to package your app yourself, import this project to your favorite IDE. If you are using Visual Studio Code IDE, you may want to check the workspace file available.

Alternatively, find your pom.xml file and run **mvn clean package** from that same folder.

```
$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.collibra:sbi-template-snowflake >-----------------
[INFO] Building demo 1.0.2-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ sbi-template-snowflake ---
[INFO] Deleting /Users/sbi.template/Documents/github/sbi_template_snowflake/target
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ sbi-template-
snowflake ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 9 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ sbi-template-
snowflake ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 17 source files to
/Users/sbi.template/Documents/github/sbi_template_snowflake/target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ sbi-
template-snowflake ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory
/Users/sbi.template/Documents/github/sbi_template_snowflake/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ sbi-
template-snowflake ---
[INFO] Changes detected - recompiling the module!
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ sbi-template-snowflake -
--
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ sbi-template-snowflake ---
[INFO] Building jar:
/Users/sbi.template/Documents/github/sbi_template_snowflake/target/sbi-template-
snowflake-1.0.2-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.5:repackage (repackage) @ sbi-template-
snowflake ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.039 s
[INFO] Finished at: 2021-09-20T22:36:41+02:00
[INFO] ------------------------------------------------------------------------
```

Prerequisites to run your Spring Boot app

Properties that must be specified inside your Spring Boot application.properties file.

Application.properties	
server.port	Specifies the HTTP port of a Spring Boot application. **Default 8080**.
management.server.port	Specifies the HTTP port of the management server. **Default 8081**.
spring.datasource.username	Specifies the login of the user for the connection. **Default SBI_TEMPLATE_SNOWFLAKE**.
spring.datasource.password	The jasypt encrypted password for the specified user
sbi_template_snowflake.datasource.userrole	Specifies the default access control role to use in the Snowflake session initiated by the driver. The specified role should be an existing role that has already been assigned to the specified user for the driver. If the specified role has not already been assigned to the user, the role is not used when the session is initiated by the driver. **Default SBI_TEMPLATE_SNOWFLAKE**.
sbi_template_snowflake.datasource.warehouse	Specifies the virtual warehouse to use once connected, or specifies an empty string. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
sbi_template_snowflake.datasource.database	Specifies the default database to use once connected or specifies an empty string. The specified database should be an existing database for which the specified default role has privileges.
sbi_template_snowflake.datasource.schema	Specifies the default schema to use for the specified database once connected or specifies an empty string. The specified schema should be an existing schema for which the specified default role has privileges.
sbi_template_snowflake.datasource.account	Specifies the default snowflake account identifier.

Use the jasypt.jar java archive found in the distribution to get your encrypted password:

```
java -cp jasypt-1.9.3.jar org.jasypt.intf.cli.JasyptPBEStringEncryptionCLI
algorithm=PBEWithHmacSHA512AndAES_256
```

```
ivGeneratorClassName=org.jasypt.iv.RandomIvGenerator password=<your encryption
password> input=<the password to be encrypted>
```

Your application.properties jasypt encrypted password can be found in the output section.

```
----ENVIRONMENT-----------------

Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM 13.0.2+8

----ARGUMENTS-------------------

input: <the password to be encrypted>
password: <your encryption password>
ivGeneratorClassName: org.jasypt.iv.RandomIvGenerator
algorithm: PBEWithHmacSHA512AndAES_256

----OUTPUT----------------------

BfyOlXMYg3kvbaMSmIwupisRnsPs/3nk8QM68d5BdP39WVNC7s484s3KEf4rvo/F
```

Before running this simple app, get your data source ready to accept incoming requests:

- ☑ create the database user
- ☑ create the default access control role
- ☑ create the default database shema to use
- ☑ create the table holding a list of commands issued
- ☑ create the table holding the masking policies to apply
- ☑ grant all table, schema, tag and apply masking policy grants
- ☑ create a few sample masking policies for illustrating how these work
- ☑ create a couple of procedures illustrating how set and unset masking policies work
- ☑ import privileges on the Snowflake database

This distribution includes a simple **init.sql** file that will alleviate most of the work for you.

A couple of paragraphs about the sample masking policies and masking procedures:

- ☑ sample masking policies provided
  - ☑ policies for address, customer identifier, name, phone number data classifications
  - ☑ clears the values if protection method not found or undefined for a given active role
  - ☑ otherwise protects with either sha2 or hash depending on if text or number values
  - ☑ an entitlements table helps the policies decide what to protect and unprotect when

- ☑ sample procedures for attaching and detaching masking policies provided
  - ☑ the first sets the masking policy depending on if classification and category found
  - ☑ does nothing if no classification and no category tag set for a given table column
  - ☑ does nothing if no masking policy found for a given classification and/or category

☑ the second detaches the masking policy if no remaining object tags are assigned

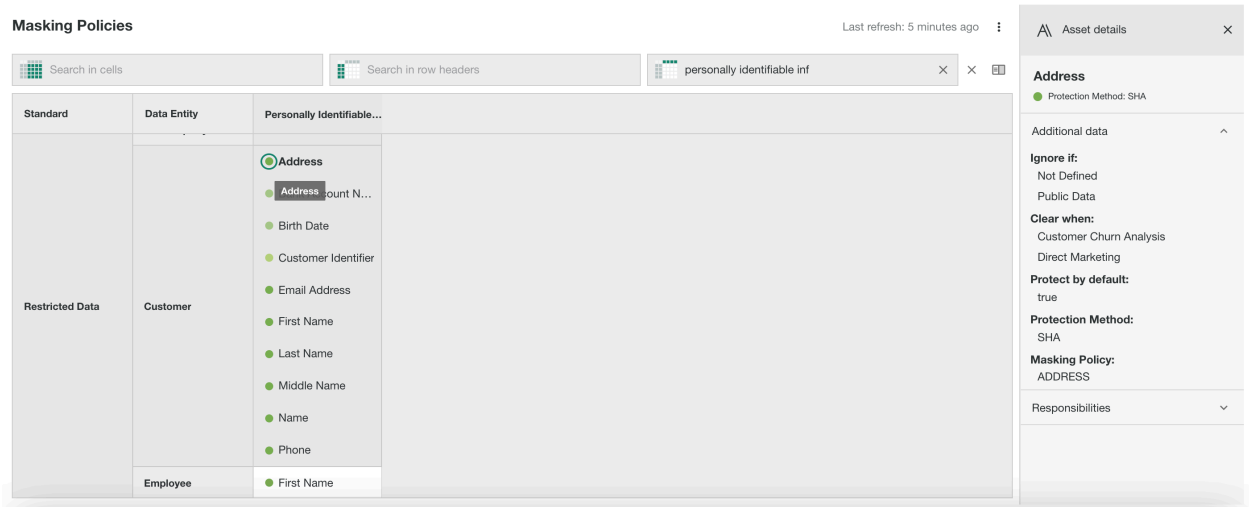
☑ protection methods limited to sha2 and hash for now but could possibly be extended.

A good option is using a custom entitlement table for determining when values should be protected, what protection method to use and under what circumstances. We may want to protect our customer's names in most cases and clear them when granting access for conducting direct marketing activities where we potentially would need to address to an individual by its name for example. Representative examples showing how to create masking policies using a mask, a hash, a regular expression, and UDF can be found here.

Delete Move Validate

	Data Domain	Ignore if	Clear when	Protection Method	Masking Policy
Address	Customer, Employee, Organization, Event, Count...	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	ADDRESS
Customer Identifier	Order, Customer	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	HASH	CUSTOMERIDENTIFIER
Email Address	Organization, Employee, Counterparty, Customer	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	MAIL
Last Name	Name	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	NAME
First Name	Name	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	NAME
Middle Name	Name	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	NAME
Cardholder Name	Counterparty, Customer, Employee	Not Defined, Public Data		SHA	NAME
Name	Counterparty, Employee, Customer	Not Defined, Public Data	Customer Churn Analysis Direct Marketing	SHA	NAME
Phone	Counterparty, Organization, Place, Employee, Cu...	Not Defined, Public Data	Direct Marketing	SHA	PHONENUMBER

global view



asset grid view

Results Data Preview Open History

Query ID SQL 65ms 14 rows

Filter result... Copy Columns

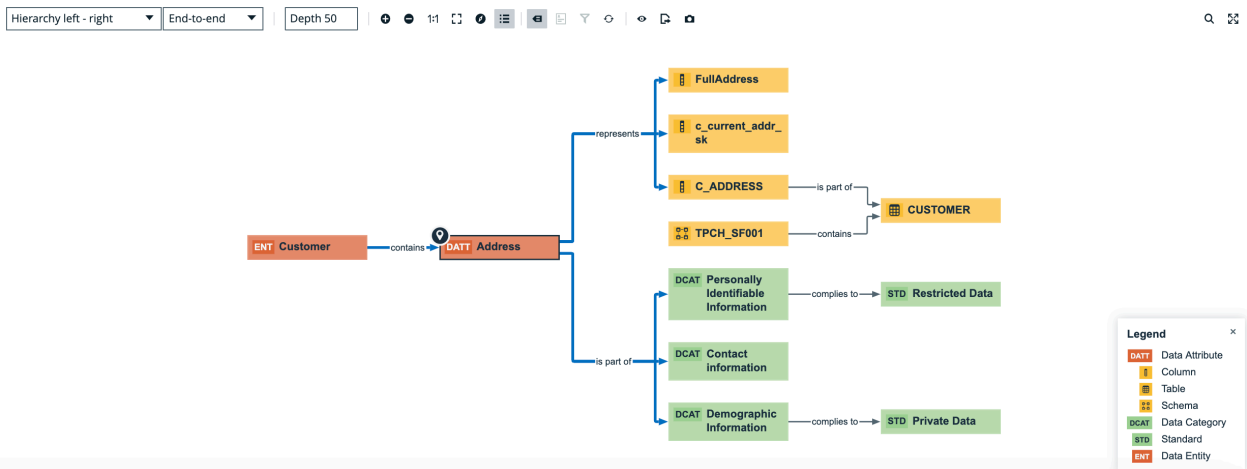
Row	DATA_CLASSIFICATIO	STANDARD	IGNORE_IF	CLEAR_WHEN	DATA_TYPE	PROTECTION_METHO	MASKING_POLICY	UUID
1	Phone number	NULL	Not Defined, Publi...	Direct Marketing	NULL	SHA	PHONENUMBER	NULL
2	Name	NULL	Not Defined, Publi...	Direct Marketing,C...	NULL	SHA	NAME	NULL
3	Middle name	NULL	Not Defined, Publi...	Direct Marketing,C...	NULL	SHA	NAME	NULL
4	Last name	NULL	Not Defined, Publi...	Customer Churn A...	NULL	SHA	NAME	NULL
5	First name	NULL	Not Defined, Publi...	Customer Churn A...	NULL	SHA	NAME	NULL
6	Email	NULL	Not Defined, Publi...	Direct Marketing,C...	NULL	SHA	MAIL	NULL
7	Customer Identifier	NULL	Not Defined, Publi...	Direct Marketing,C...	NULL	SHA	CUSTOMERIDENTI...	NULL
8	Address	NULL	Not Defined, Publi...	Direct Marketing,C...	NULL	SHA	ADDRESS	NULL
9	NULL	Restricted Data			NUMBER	HASH	RESTRICTEDDATA...	NULL
10	NULL	Restricted Data			TEXT	SHA	RESTRICTEDDATA...	NULL
11	NULL	Public Data			NUMBER	HASH	PUBLICDATA_NU...	NULL
12	NULL	Public Data			TEXT	SHA	PUBLICDATA_TEXT	NULL
13	NULL	Private Data			NUMBER	HASH	PRIVATEDATA_NU...	NULL
14	NULL	Private Data			TEXT	SHA	PRIVATEDATA_TEXT	NULL

entitlements table

Note: protection methods depend on the data classification and data category (standard). We may have a name classified column which by default should be protected and cleared once realized this same name column holds the name of an event for example and not restricted. Can't tell from the masking policy which column this policy is set to and there's only a handful number of functions available. One way to go about this is to map masking policies to data classifications and categories and have these unset and set accordingly.

For example:

1. Connect a given column to the customer address data attribute and concept suggesting that column is of restricted access – set default restricted masking policy.
2. Add an Address data classification to that same column and being more specific about what is is to be protected and secured – set the address default masking policy.
3. A given user was granted access to the 'Direct Marketing' business process or role. When using that role (active role), that same column would be cleared for that user.



address diagram view explore

Results Data Preview — Open History

Query ID SQL 282ms 7 rows

Filter result... Copy Columns

Row	TAG_DATABASE	TAG_SCHEMA	TAG_NAME	TAG_VALUE	LEVEL	OBJECT_DATABA	OBJECT_SCHEMA	OBJECT_NAME	DOMAIN	COLUMN_NAME
1	COLLIBRA	SBI_TEMPLAT...	Address	CLASSIFICATI...	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
2	COLLIBRA	SBI_TEMPLAT...	Contact infor...	CATEGORY	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
3	COLLIBRA	SBI_TEMPLAT...	Customer	DOMAIN	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
4	COLLIBRA	SBI_TEMPLAT...	Demographic ...	CATEGORY	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
5	COLLIBRA	SBI_TEMPLAT...	Personally Ide...	CATEGORY	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
6	COLLIBRA	SBI_TEMPLAT...	Private Data	STANDARD	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS
7	COLLIBRA	SBI_TEMPLAT...	Restricted Data	STANDARD	COLUMN	COLLIBRA	TPCH_SF001	CUSTOMER	COLUMN	C_ADDRESS

address column tag references

Results Data Preview — Open History

Query ID SQL 287ms 4 rows

Filter result... Copy Columns

Row	CREATED_ON	POLICY_DB	POLICY_SCHEM	POLICY_NAME	POLICY_KIND	REF_DATABASE_	REF_SCHEMA_N	REF_ENTITY_NA	REF_ENTITY_DO	REF_COLUMN_N	REF_ARG_C
1	2021-10-18 ...	COLLIBRA	SBI_TEMPLA...	ADDRESS	MASKING_P...	COLLIBRA	TPCH_SF001	CUSTOMER	TABLE	C_ADDRESS	NULL
2	2021-10-17 ...	COLLIBRA	SBI_TEMPLA...	CUSTOMERI...	MASKING_P...	COLLIBRA	TPCH_SF001	CUSTOMER	TABLE	C_CUSTKEY	NULL
3	2021-10-17 ...	COLLIBRA	SBI_TEMPLA...	NAME	MASKING_P...	COLLIBRA	TPCH_SF001	CUSTOMER	TABLE	C_NAME	NULL
4	2021-10-17 ...	COLLIBRA	SBI_TEMPLA...	PHONENUM...	MASKING_P...	COLLIBRA	TPCH_SF001	CUSTOMER	TABLE	C_PHONE	NULL

customer table policy references

Note: When connecting that same column to the address data concept of an event data domain or any other data category of public access – unset that column masking policy.

Run your Spring Boot app

The Spring Boot Maven plugin includes a run goal which can be used to quickly compile and run your application with **mvn spring-boot:run.** Before running your application:

Set your jasypt encryption password by exporting JASYPT_ENCRYPTOR_PASSWORD

```
export JASYPT_ENCRYPTOR_PASSWORD=<your encryption password>
```

or set the jasypt.encryptor.password property when launching your application jar archive

```
-Djasypt.encryptor.password=<your encryption password>
```

```
$ mvn spring-boot:run -Djasypt.encryptor.password=<your encryption password>
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.collibra:sbi-template-snowflake >-----------------
[INFO] Building demo 1.0.2-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.4.5:run (default-cli) > test-compile @ sbi-
template-snowflake >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ sbi-template-
snowflake ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 9 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ sbi-template-
snowflake ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ sbi-
template-snowflake ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /Users/sbi.template/Documents/sbi-template-
snowflake/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ sbi-
template-snowflake ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.4.5:run (default-cli) < test-compile @ sbi-
template-snowflake <<<
[INFO]
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.5:run (default-cli) @ sbi-template-snowflake -
--
[INFO] Attaching agents: []
…
[2021-09-21 10:45:52.674][INFO ][http-nio-0.0.0.0-8081-exec-
1][org.springframework.web.servlet.DispatcherServlet] Completed initialization in 0 ms
```

Alternatively, you can simply run your application using **java -jar**. For example

```
$ java -Djasypt.encryptor.password=<your encryption password> -jar target/sbi-template-snowflake-1.0.2-SNAPSHOT.jar
```

Coming soon

- ☐ Cost effectiveness with fast deployment and ability to run anywhere with docker containers.
- ☐ Productized integration and updated UX to enforce access control rules and access insights.