Data Security Platform Integration with Google Big Query

May 16, 2023

www.comforte.com

Overview

The comforte Data Security Platform is a scalable, fault-tolerant enterprise data-centric security solution that protects sensitive data with minimal effort and little to no impact on existing applications. The Data Security Platform allows organizations to achieve end-to-end data protection, helps with compliance to standards like GDPR, CCPA, or PCI DSS, as well as significantly reduces the impact and liability of data breaches.

For a comprehensive and detailed description of the Data Security Platform or DSP product, please refer to the SecurDPS Enterprise Protection Cluster reference manual. Throughout this document the term SecurDPS will be used to refer the Data Security Platform protection Cluster.

This document describes specifically the integration between the Data Science Platform with Google Big Query Remote Functions to enable Big Query to protect the data using simple REST API using SecurDPS.

In the following integration example the following components are required:

1. Google Big Query Remote Function
2. REST API Endpoint implemented with Spring Boot and SecurDPS Filter API containerized as Cloud Run
3. A running SecurDPS Protection Cluster is available and reachable from within Google Cloud components, like Cloud Run. This includes the external IP address and user/key used in the REST API instance.

Google Big Query Remote Function

This chapter shows the steps to integrate GCP Big Query Remote Functions to the SecurDPS REST API endpoint to enable Big Query to protect and reveal data using comforte technology.

```
bq mk --connection --location=europe-west3 --project_id=gcp-comforte --
connection_type=CLOUD_RESOURCE bq-sdps-rest-api
```

To enable Big Query to call remote API's we need to create a Big Query Connection configuration.

Here is a sample of a command line that creates a new BQ connection:

```
CREATE FUNCTION 'gcp-comforte.comforte'.sdps_protect_PAN(x BYTES)
RETURNS BYTES
REMOTE WITH CONNECTION 'gcp-comforte.europe-west3.bq-sdps-rest-api'
OPTIONS (
  endpoint = 'https://gcp-rest-api-b3mc3bbyuq-ey.a.run.app',
  user_defined_context = [("X-Operation", "PROTECT"),("X-Strategy", "PAN"),
("Authorization", "")]
)
```

You have to modify the `--location` and `--project_id` parameters accordingly and give an ID to the connection, in the above example `bq-sdps-rest-api`.

After creating the connection, we finally can create the Remote Function in Big Query. See the example below:

REST API Endpoint

The following example describes the implementation of a Spring Boot REST API endpoint to translate data from JSON messages generated with Google Big Query using SecurDPS as a Google Cloud Run function.

Developing and deploying a Google Cloud Run function is easily done using the Google Cloud Run deployment services integrated in an IDE like IntelliJ. For more information about this please see the example on Deploy a Cloud Run service with Cloud Code for IntelliJ.

Requirements

- JDK 11 (It is recommended to use openJDK Eclipse Temurin build from the Eclipse Adoptium project)
- SecurDPS File/Stream Filter version 7.4.0 (available on
-

https://comforte.com) Docker engine Dependencies

For the implementation the build tool **Gradle** is used. A complete `build.gradle` file can be found below.

The following dependencies are required for the project:

```
implementation(
        [name:'securdps-file-stream-filter-7.4.0'],
        [group: 'org.springframework.boot', name: 'spring-boot-dependencies',
version: '3.0.+'],
        [group: 'org.springframework.boot', name: 'spring-boot-starter-web',
version: '3.0.+']
)
```

Gradle Build

The project for this integration example has the following project structure:

```
.
│   build.gradle
│   Dockerfile
│   gradlew
│   gradlew.bat
│   README.md
│
├───libs
│       securdps-file-stream-filter-7.4.0.jar
│
```

```
└──src
    ├──main
    │   ├──java
    │   │   └──com
    │   │       └──comforte
    │   │           └──app
    │   │               └──rest
    │   │                       InputJsonMessage.java
    │   │                       OutputValues.java
    │   │                       RestApplication.java
    │   │
    │   └──resources
    │       │   application.properties
    │       │
    │       └──config
    │               sdf.yaml
    │
    └──test
        └──java
            └──com
                └──comforte
                    └──app
                        └──rest
                                InputJsonMessageTest.java
                                OutputValuesTest.java
```

- `libs` folder contains the SecurDPS File/Stream Filter jar file
- `src/main/java` is the default Java source path and contains the Java sources,
- `src/main/resources` is the default resource path and contains all non-Java sources
- `src/test/java` is the default JUnit source path and contains the JUnit sources
- `src/test/resources` is the default JUnit resource path and contains all non-Java sources
- `build.gradle` is the Gradle build file
- `Dockerfile` is a simple text file that consists of instructions to build the Docker image,
- `gradlew` is the gradle run shell script,
- `gradlew.bat` is the gradle run batch script
- `Readme.md` is this file

The tasks defined in the `build.gradle` file are used to compile and build the application in the `Dockerfile`. The listing below shows the complete `build.gradle` sample here:

```
import java.text.SimpleDateFormat

plugins {
    id 'java'
    id 'org.springframework.boot' version '2.7.+'
    id 'com.github.johnrengelman.shadow' version '7.+'
    id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}
```

```
group = 'com.comforte.securdps.gbq' version
= '0.1.0-SNAPSHOT'
 java {     toolchain {         languageVersion =
JavaLanguageVersion.of(11)
    }
}    repositories  {
flatDir dirs: 'libs'
mavenLocal()
mavenCentral()
}  dependencies {       def
springBootVersion =
"2.7.+"     def junitVersionNumber
= '5.4+'
implementation(
         [group: '', name: 'securdps-file-stream-filter-7.4.0', version: ''],
[group: 'org.springframework.boot', name: 'spring-boot-dependencies', version:
springBootVersion],
         [group: 'org.springframework.boot', name: 'spring-boot-starter-web',
version: springBootVersion],
    )

    // Test Deps
testImplementation(
         [group: 'org.springframework.boot', name: 'spring-boot-starter-test',
version: springBootVersion],
         [group: 'org.junit.jupiter', name: 'junit-jupiter-api', version:
junitVersionNumber],
         [group: 'org.junit.jupiter', name: 'junit-jupiter-engine', version:
junitVersionNumber]
    )  }
tasks.withType(JavaCompile).configureEach
{     options.encoding = "UTF-8"
}  shadowJar
{
    // ensure that all service files are merged
properly     mergeServiceFiles()      def
manifestClasspath = '.. config/ ../config/ lib/
../lib/'

    manifest.attributes('Implementation-Title': 'comforte SecurDPS Spring Boot REST
API',
            'Implementation-Version': project.version,
            'Implementation-Vendor': 'comforte AG',
            'Specification-Version': project.version,
            'Build-Timestamp': new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSSZ").format(new Date()),
            'Build-Revision': project.version,
            'Created-By': "Gradle ${gradle.gradleVersion}",
            'Application': "SecurDPS Cloud Run",
            'Class-Path': manifestClasspath,
            'Multi-Release': true,
            'Main-Class': 'com.comforte.gbc.RestApplication'
    )
    excludes = [
            'config/**',
            'log4j2.xml'
    ]
    archiveClassifier.set('')
}

test {
    useJUnitPlatform()
    testLogging.showStandardStreams = true
}
```

Note: The SecurDPS File/Stream Filter jar is loaded from a local `./libs` folder. See task declarations `repositories` and `dependencies`.

Running

The Spring Boot application listens on port 8080 is started with the following command locally:

```
./gradlew --info clean bootJar
```

Note: In case of Java compatibility issues, make sure that your Java Home environment variable is set to the correct Java 11 version.

Testing

The JUnit test is called with the following gradle command:

```
./gradlew --INFO clean test
```

Note: In case of Java compatibility issues, make sure that your Java Home environment variable is set to the correct Java 11 version.

REST API Endpoint Sample

The Sample is using Spring Boot Application interface and the SecurDPS Filter API embedded in the SecurDPS File/Stream Filter. The application creates a REST API endpoint for Http POST requests to the path `/`. The endpoint consumes and generates messages of media type `application/json`.

Input Format

This endpoint is used to process JSON messages generated by Big Query Remote Function. BigQuery sends Http POST requests with JSON body in the following format:

Field name	Description	Field type
requestId	Id of the request. Unique over multiple requests sent to this endpoint in a GoogleSQL query.	Always provided. String.
caller	Job full resource name for the GoogleSQL query calling the remote function.	Always provided. String.
sessionUser	Email of the user executing the GoogleSQL query.	Always provided. String.
userDefinedContext	The user defined context that was used when creating the remote function in BigQuery.	Optional. A JSON object with key-value pairs.
calls	A batch of input data.	Always provided. A JSON array.Each element itself is a JSON array, which is a JSON encoded argument list of one remote function call.

(Source: [Big Query - Working with Remote Functions](Working with Remote Functions | BigQuery | Google Cloud))

The listing below shows a JSON Message sample to protect credit card numbers using a protection strategy with the name PAN64.

```
{
    "requestId": "124ab1c",
    "caller":
"//bigquery.googleapis.com/projects/myproject/jobs/myproject:US.bquxjob_5b4c112c_1
7961fafeaf",
    "sessionUser": "test-user@test-company.com",
    "userDefinedContext": {
        "operation": "PROTECT",
        "strategy": "PAN64"
    },
    "calls": [
        [
            "123456789123456789"
        ],
        [
            "789456123789456123"
        ],
        [
            "987654321987654321"
        ]
    ]
}
```

Note: The `operation` and `strategy` are required in `userDefinedContext`, otherwise the service will fail. Valid values for `operation` are `PROTECT | REVEAL`. The value for `strategy` must match exactly the name of a strategy as defined in the SecurDPS Definition File (SDF) on the Protection Cluster.

Output Format

BigQuery expects the endpoint to return an Http response in the following format. If it doesn't receive a response in the correct format BigQuery can't consume it and will fail the query calling the remote function.

Field name	Description	Value Range
replies	A batch of return values.	Required for a successful response. A JSON array. Each element corresponds to a JSON encoded return value of the external function. Size of the array must match the size of the JSON array of `calls` in the HTTP request. For example if the JSON array in `calls` has 4 elements this JSON array needs to have 4 elements as well.
errorMessage	Error message when the HTTP response code other than 200 is returned. For non-retryable errors, we return this as part of the BigQuery job's error message to the user.	Optional. String. Size should be less than 1KB.

An example of a successful response:

```
{
    "replies": [{
            "data": "123456789123456789"
        },
        {
            "data": "789456123789456123"
        },
        {
            "data": "987654321987654321"
        }
    ]
}
```

An example of a failed response:

```
{
  "errorMessage": "No strategy is passed in userDefinedContext!".
}
```

The service response with `400 BAD REQUEST` if:

1. the strategy is not in `userDefinedContext`
2. the operation is not in `userDefinedContext`
3. the input JOSN message doesn't contain any `calls`

Source Samples

In the SecurDPS integration sample the Spring Boot Application main class is `com.comforte.app.rest.RestApplication`. The listing below shows the source code:

```
package com.comforte.app.rest;

/*
* Copyright 2020 Google LLC
 *
* Licensed under the Apache License, Version 2.0 (the "License");  * You may
  not use this file except in compliance with the License.
* You may obtain a copy of the License at
 *
* http://www.apache.org/licenses/LICENSE-2.0
 *
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  *
  See the License for the specific language governing permissions and  *
  limitations under the License.
 */  import com.comforte.securdps.ConfigurationException; import
com.comforte.securdps.Operation; import
com.comforte.securdps.TranslateException; import
com.comforte.securdps.filter.Filter; import
com.comforte.securdps.filter.FilterConfig; import
com.comforte.securdps.filter.processor.FormatPreProcessor; import
com.comforte.securdps.filter.processor.SimpleFormatPreProcessor; import
org.springframework.boot.SpringApplication; import
org.springframework.boot.autoconfigure.SpringBootApplication; import
org.springframework.http.HttpStatus; import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestBody; import
org.springframework.web.bind.annotation.RestController;
 import
java.io.IOException;
```

```
import java.util.Collection; import
java.util.HashMap; import java.util.List;
import java.util.Locale; import
java.util.concurrent.ExecutionException; import
java.util.concurrent.Executors; import
java.util.concurrent.Future; import
java.util.concurrent.ThreadPoolExecutor;
 @SpringBootApplication public
class RestApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestApplication.class, args);
    }      private final Filter
filter;

    public RestApplication() throws ConfigurationException {
        String sdf = System.getProperty("sdf", "config/sdf.yaml");
        FilterConfig filterConfig = (FilterConfig) FilterConfig.loadFromFile(sdf);
filter = Filter.create(filterConfig);
    }

    @RestController     class SDPSRestFilterController {
private static final String STRATEGY = "strategy";
private static final String OPERATION = "operation";
@PostMapping(path = "/",             consumes =
MediaType.APPLICATION_JSON_VALUE,             produces =
MediaType.APPLICATION_JSON_VALUE)          public ResponseEntity<String>
doPost(@RequestBody InputJsonMessage input)
{
            try {
                HashMap<String, String> context =
verifyAndReturnUserDefinedContext(input);
                OutputValues output =
doTranslate(context, input.getFirstColumn());
return new ResponseEntity<>(output.toString(),
HttpStatus.OK);             } catch (Exception e) {
return get400Error(e.getMessage());
            }          }           private OutputValues
doTranslate(final HashMap<String, String> context, final List<String>
data)
            throws TranslateException, IOException, ExecutionException,
            InterruptedException, ConfigurationException {
            ThreadPoolExecutor workerThreadPool = (ThreadPoolExecutor)
Executors.newFixedThreadPool(1);
```

```
            Operation operation =
Operation.valueOf(context.get(OPERATION).toUpperCase(Locale.ENGLISH));
            FormatPreProcessor formatPreProcessor =
SimpleFormatPreProcessor.Builder.create(context.get(STRATEGY))
                .setOperation(operation)
                .build();

            Future<Collection<String>> future =
workerThreadPool.submit(filter.doExchange(data, formatPreProcessor));
Collection<String> protectedCalls = future.get();
workerThreadPool.shutdown();            return new
OutputValues(protectedCalls);
        }

        private  ResponseEntity<String>  get400Error(final  String  errorMessage)  {
return new ResponseEntity<>("{ " + "\"errorMessage\":" + " \"" + errorMessage + "\"
}",
                HttpStatus.BAD_REQUEST);
        }

        private HashMap<String, String>
verifyAndReturnUserDefinedContext(final InputJsonMessage bqJsonMessage)
throws IllegalArgumentException {
             if (null == bqJsonMessage) {                 throw new
IllegalArgumentException("Request contains no data!");
}
             HashMap<String, String> userDefinedContext =
bqJsonMessage.getUserDefinedContext();           if
(userDefinedContext.size() == 0) {               throw new
IllegalArgumentException("No userDefinedContext passed!");
            }
            HashMap<String, String> context = new HashMap<>();
            String[] keys = userDefinedContext.keySet().toArray(new String[0]);
boolean isStrategy = false;            boolean isOperation = false;
for (String key : keys) {                 if
(key.toLowerCase(Locale.ENGLISH).equals(STRATEGY)) {
isStrategy = true;                     context.put(STRATEGY,
userDefinedContext.get(key));
                }                 if
(key.toLowerCase(Locale.ENGLISH).equals(OPERATION)) {
isOperation = true;                     context.put(OPERATION,
userDefinedContext.get(key));
                }
}
             if (!isStrategy) {                 throw new
IllegalArgumentException("No strategy is passed in userDefinedContext!");
            }             if
(!isOperation) {
```

```
                throw new IllegalArgumentException("No operation is passed in
userDefinedContext!");
            }
            if (null == bqJsonMessage.getCalls()) {
                throw new IllegalArgumentException("Request contains no calls!");
            }
            return context;
        }
    }
}
```

The bean class for the JSON request body is `com.comforte.app.rest.InputJsonMessage`. The listing below shows the source code:

```
package com.comforte.app.rest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;

/**
 * @author R. Modde (r.modde@comforte.com)
 */
public class InputJsonMessage {
    private final String requestId;
    private final String caller;
    private final String sessionUser;
    private final HashMap<String, String> userDefinedContext;

    private final List<String[]> calls;

    private final List<String> firstColumn;

    /**
     *
     * @param requestId request id, unique over multiple requests sent to this
endpoint in a GoogleSQL query
     * @param caller job full resource name for the GoogleSQL query calling the
remote function
     * @param sessionUser Email of the user executing the GoogleSQL query
     * @param userDefinedContext the user defined context that was used when
creating the remote function in BigQuery
     * @param calls a batch of input data
     * @throws IllegalArgumentException if calls contains empty arrays
     */
    public InputJsonMessage(String requestId, String caller, String sessionUser,
HashMap<String, String> userDefinedContext, List<String[]> calls) throws
IllegalArgumentException {
        this.requestId = requestId;
        this.caller = caller;
        this.sessionUser = sessionUser;

        String[] keys = userDefinedContext.keySet().toArray(new String[0]);
        this.userDefinedContext = new HashMap<>(keys.length);
        for (String key : keys) {
            this.userDefinedContext.put(key.toLowerCase(Locale.ENGLISH),
userDefinedContext.get(key));
        }

        this.calls = calls;
        firstColumn = new ArrayList<>(calls.size());
        setRows();
    }

    private void setRows() throws IllegalArgumentException {
        for(String[] strings: calls) {
            if (strings.length > 0) {
                firstColumn.add(strings[0]);
            } else {
                throw new IllegalArgumentException("Calls contains empty
arrays!");
            }
        }
    }

    public String getRequestId() {
        return requestId;
    }

    public String getCaller() {
        return caller;
    }

    public String getSessionUser() {
        return sessionUser;
    }

    public HashMap<String, String> getUserDefinedContext() {
        return userDefinedContext;
    }

    public List<String[]> getCalls() {
        return calls;
    }

    public List<String> getFirstColumn() {
        return firstColumn;
    }
}
```

The bean class for the JSON response body is `com.comforte.app.rest.OutputValues`. The code below shows the source code:

```java
package com.comforte.app.rest;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;

import java.util.Collection;

/**
 * @author R. Modde (r.modde@comforte.com)
 */
public class OutputValues {

    private final Collection<String> replies;

    public OutputValues(final Collection<String> replies) {
        this.replies = replies;
    }

    public Collection<String> getReplies() {
        return replies;
    }

    public String toString() {
        ObjectWriter ow = new ObjectMapper().writer().withDefaultPrettyPrinter();
        try {
            return ow.writeValueAsString(this);
        } catch (JsonProcessingException e) {
            return "Error: " + e.getMessage();
        }
    }
}
```

Deploy Cloud Run Service Sample with IntelliJ

In your IDE, for example IntelliJ, with Google Cloud Code you can develop the SecurDPS Cloud Run services locally in a Cloud Run-like environment. Cloud Code watches for changes in your source and quickly updates the running service to reflect these changes.

Develop and Test the Service Sample locally in Cloud Code for IntelliJ

The Spring Boot REST Endpoint application with the SecurDPS Integration can be developed and tested locally using for example IntelliJ from JetBrains.

Specifying the RUN configuration In IntelliJ:

Before you run your service you must create your run configuration:

1. Navigate to Run/Debug configurations selector in the Navigation bar and click **Edit Configurations**.
2. Under Cloud Code: Cloud Run, choose **Cloud Run: Run Locally**.

3. If you want Cloud Code to redeploy your application automatically after your changes are saved, under **Watch mode - rebuild and redeploy**, select **On file save**. New Cloud Run services have **On demand** selected by default. For more information about watch modes, see Watch modes. 4. Set in **Build Settings** the `Builder` to `Dockerfile` and specify the `Dockerfile`
5. Click **OK**.

For more details see [Develop a service locally in Cloud Code for IntelliJ](Develop a service locally in Cloud Code for IntelliJ | Google Cloud).

The following code shows the Dockerfile:

```
# Use the official maven/Java 11 image to create a build artifact.
# https://hub.docker.com/_/maven
FROM eclipse-temurin:11-jdk-alpine AS build-env

# Set the working directory to /app
WORKDIR /comforte
# Copy the gradle files
COPY build.gradle ./
COPY settings.gradle ./
COPY gradlew ./
RUN chmod +x ./gradlew

COPY gradle ./gradle
# Copy the SecurDPS File/Stream Filter jar
COPY libs ./libs

# Copy local code to the container image.
COPY src ./src

# Download dependencies and build a release artifact.
RUN ./gradlew --info clean bootJar

# Use OpenJDK for base image.
# https://hub.docker.com/_/openjdk
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-
stage-builds
FROM eclipse-temurin:11-jdk-alpine

# Copy the jar to the production image from the builder stage.
COPY --from=build-env /comforte/build/libs/securdps-bq-cloud-run-*.jar /securdps-
bq-cloud-run.jar

COPY src/main/resources/config/sdf.yaml /config/sdf.yaml

# Run the web service on container startup.
CMD ["java", "-Dsdf=/config/sdf.yaml", "-jar", "/securdps-bq-cloud-run.jar"]
```

Running the service in IntelliJ

To run your service, follow these steps:

1. Choose the **Cloud Run: Run Locally** run target from the Run/Debug configurations selector in the Navigation bar.
2. Click ▶ **Cloud Run: Run Locally**.
3. View the logs from your running service, streamed directly to the **output window**.
4. Once deployment is complete, you can view your running service by following the URL displayed in your Event Log panel. In the example below, this URL is http://localhost:8080.

Deploy the Service Sample

To deploy the Service Sample follow the introduction on [Deploy a service to Cloud Run in Cloud Code for IntelliJ](Deploy a service to Cloud Run in Cloud Code for IntelliJ | Google Cloud)