GDPR Compliance BigData Systems

This projects aims at implementing a solution for the European Data Protection Regulation, the research paper **A framework for GDPR compliance in Big Data systems** made by **Mouna Rhahla**, **Sahar Allegue** and **Takoua Abdellatif** in **Proxym-Lab** Refer to Documents directory for more information.

SUPERVISER : Tarek Sghair

Table of Contents

GDPR Overview

Principles

You can check one of the research documents provided for an explanation of each principle .

alt text

Actors

You can check one of the research documents provided for an explanation of each actor .

alt text

Context and Objectives

Key words

- **Service**: Which is one of the programmed services it may be any of the enterprise application built API that needs data to process such as a **Machine Learning** service that demands customer data at REST to perform clustering or client segmentation and an **Analytics** service that performs data aggregations in order to make a statistical dashboard that helps in decision making etc ...
- **Restriction** : It is the order of control of which the subject data is not allowed for any use.

Context

Log data is the information recorded by your application server about when, how, and which visitors are using your website. Application server providers commonly collect the following information about each user:

- **IP address and device identifier**: This is the unique identifying address broadcasted by the browser or device by which each user is accessing your online platform.
- **Request date/time**: The specific date and time of each action the user takes.
- **Personal details**: Including the full name of a user, identity number, and password.
- **Device**: Whether the user is connected from a web or mobile application.
- **Transaction Amount**: The amount of a triggered transaction process.
- **Service**: The service provided by the application.

It is trivial to ignore the **sensitivity** and the **privacy** of the logs generated, a GDPR-based application contains the necessary components stacked in layers in order to build a **trusted** application and ensure **confidentiality** by providing all users the ability to **control** all data usage except the ones of which needed for server or application required operations.

In order to do so, **restrictions** to protect customer data from the application services must be provided to **choose who**(Service?) is allowed and **which**(attribute?) data to consume in processing. All restrictions will be stored in a database for further needs.

Here is an example illustrating a scenario for more understanding of the problem:

Let serviceA be one of the mentioned services in the above paragraph

- **User 1**: The serviceA has a restriction on the last name and the amount of transfer
- **User 2**: The serviceA has a restriction on the customer name, the device, and the service provided to the user
- **User 3**: The serviceA has a restriction on the customer name; customer last name and the device

alt text

Now after **storing** the data for authorization, our application services will try to **process** it, the **scenario** is like the following:

- **Request**: The service will try to get the required data (user1 in the example) from the database.
- **Response**: The response must contain only the authorized data of the required customers.

alt text

Objective

Our two primary objectives are to guarantee secure, end-to-end, real-time data flow by delivering encrypted logs and maintaining an access control layer built on top of the database with respect to all GDPR constraints.

Project Architecture

First of all,at a time **T** and event **E** will occur during the normal or the malfunctioning of the application and this event record will be stored in a log file in the image below the data source representing our application.

Events may be one of the following :

- **Transaction**
- **Application Error**

The data flow manager will be intercepting and ready to process any incoming information to finally merge it into the database.

On the other side, the data protection officer will be managing access control to the database by providing a dynamic token with a limitless lifetime triggered by a request from any existing service in the enterprise.

Once the token is available, the service can get the required data except the ones restricted by the user.

alt text

Database Architecture

For each service we have authorized customer data, a pipeline generates the **View** where all data within it is eligible to use. For example: if we have **n** services the number of collections will be **n + 1 policy collection + all data collection**.

Authorized Data Format :

- 0 : not allowed to use by service
- 1 : allowed to use by service

Technologies Identification

alt text

Project Implementation

Module 1

Data Flow Manager

Apache Kafka Quick Overview

Apache Kafka's concept is very simple and easy to understand, it is essentially a distributed platform for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

It has these core APIs:

- **Producer API**: Applications can publish a stream of records to one or more Kafka topics.
- **Consumer API**: Applications can subscribe to topics and process the stream of records produced to them.

The core abstraction Kafka provides for a stream of records is the **topic**. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber. This means that a topic can have zero, one, or many consumers that subscribe to the data written to it.

For each topic, the Kafka cluster maintains a **partitioned** log like the one shown in the image below.

Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log. The records in the partitions are each assigned a sequential ID number called the offset, that uniquely identifies each record within the partition.

The Kafka cluster also durably persists all published records, whether they have been consumed using a configurable retention period or not,it’s performance is effectively constant with respect to data size, which means storing data for a long time is not a problem.

Roles Identification

- **Producer:** Any integrated Application
- **Consumer:** Kafka Connect

alt text

Zookeeper Quick Overveiw

For now, Kafka services cannot be used in production without first installing ZooKeeper. * This is true even if your use case requires just a single broker, single topic, and single partition.

For any distributed system, there needs to be a way to coordinate tasks. Kafka is a distributed system that was built to use ZooKeeper. However, other technologies like Elasticsearch and MongoDB have their own built-in mechanisms for coordinating tasks.

ZooKeeper has five primary functions. Specifically, ZooKeeper is used for controller election, cluster membership, topic configuration, access control lists, and quotas.

- **1. Controller Election**. The controller is the broker responsible for maintaining the leader/follower relationship for all partitions. If ever a node shuts down, ZooKeeper ensures that other replicas take up the role of partition leaders replacing the partition leaders in the node that is shutting down.

- **2. Cluster Membership**. ZooKeeper keeps a list of all functioning brokers in the cluster.

- **3. Topic Configuration**. ZooKeeper maintains the configuration of all topics, including the list of existing topics, number of partitions for each topic, location of the replicas, configuration overrides for topics, preferred leader node, among other details.

- **4. Access Control Lists (ACLs)**. ZooKeeper also maintains the ACLs for all topics. This includes who or what is allowed to read/write to each topic, list of consumer groups, members of the groups, and the most recent offset each consumer group received from each partition.

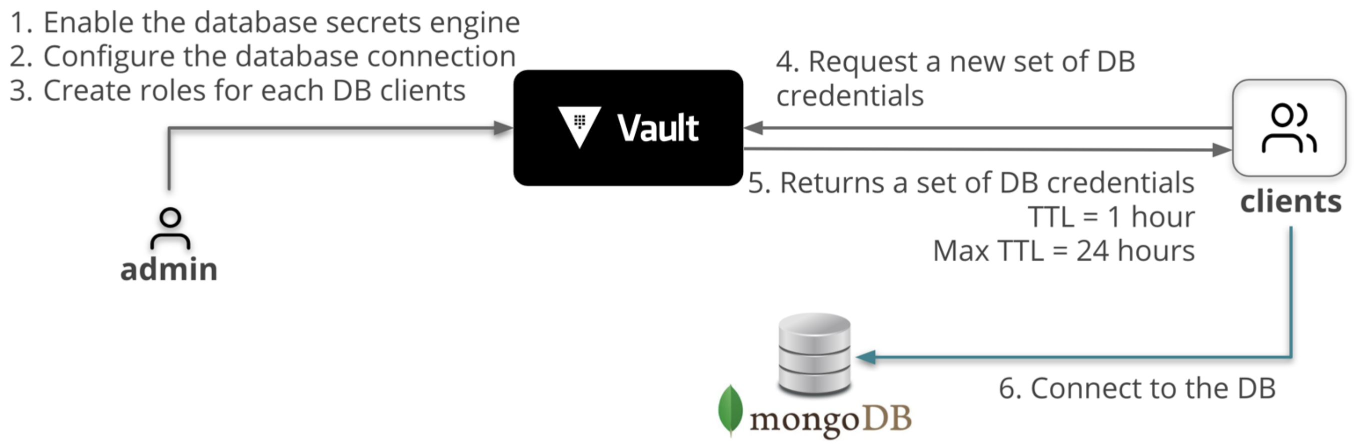
- **5. Quotas**. ZooKeeper accesses how much data each client is allowed to read/write.
  ZooKeeper is used in distributed systems for service synchronization and as a naming registry. When working with Apache Kafka, ZooKeeper is primarily used to track the status of nodes in the Kafka cluster and maintain a list of Kafka topics and messages.

Module 2 Data Protection Officier

Admin : Application admin

Clients : FLASK Services

MongoDB : Application Data Base contains users and Application Data


1. Enable the database secrets engine
2. Configure the database connection
3. Create roles for each DB clients
Vault
admin
4. Request a new set of DB credentials
5. Returns a set of DB credentials
TTL = 1 hour
Max TTL = 24 hours
clients
6. Connect to the DB
mongoDB

Note: when the service access the DB with the provided token the only collection that will be visible is the authorized one which is the View in our case.

Running Module 1

- Before proceeding to run any script or command let's first install the requirements by running **requirements_installation.sh** :

  Note:

  - The script will also add your local machine user to the docker group to be able to run docker without super user privilege because docker uses the Unix sockets for communication other wise it will demand credentials every time you run it .
  - during installation always proceed with yes

```
git clone https://github.com/SabriMahmoud/GDPR-Compliance-Big-Data-Systems.git
cd GDPR-Compliance-Big-Data-Systems/PythonVersion_Consent_Management/Data_Management/Kafka/Kafka_Connect/
./requirements_installation.sh
```

To make the project up and running clone the repository and run the **test_rootless.sh** script by typing these commands .

Note:

- The script will get our docker containers up and running in the server
- Download the MongoDB connect Plugin
- Copy the Plugin inside the connect container
- Create Kafka topics
- Configure the MongoDB Sink connectors
- Produce test data

```
./test_rootless.sh
```

Note: If you encounter an error **bash: ./test_rootless.sh Permission denied** you should give the permision of execution to prevent it from occuring by typing this command

```
chmod +x ./test_rootless.sh
```

Otherwise docker containers are invading the host server at this moment let's check the server status.

Server Status

You can check whether everythiing is good and compatible with the image below type this command .

```
docker ps
```

If everything looks good let's see the MongoDB Sink connector configuration.

Database Sink Connector

Configuration

Kafka connect Sink configuration is already done inside the script **test_rootless.sh** that you've runned earlier this is how the configuration looks like

```
{
    "name": "mongo-sink-connector",
    "config": {
        "connector.class": "com.mongodb.kafka.connect.MongoSinkConnector",
        "tasks.max": "1",
        "topics": "events",
        "connection.uri": "mongodb://mongodb:27017/Bankerise",
        "database": "Bakerise",
        "collection": "AppUsers",
        "max.num.retries": 5,
        "mongo.errors.tolerance": "all",
        "mongo.errors.log.enable": true,
        "errors.log.include.messages": true,
        "errors.deadletterqueue.topic.name": "events.deadletter",
        "errors.deadletterqueue.context.headers.enable": true
    }
}
```

You can clearly see that the connector is confugured as a Sink Connecter by looking to **config** attribute **connector.class**. Here we mentioned one topic which is events and we gave the connector the database **url** and the **collection** to where to transfer.

Same as the policy connecter only the topics and the collection have changed.

Data Factory

After we setteled the environement and created a kafka topic named **events** it is time to produce some logs.So as to achieve, proceed to **GDPR-Compliance-Big-Data-Systems/PythonVersion_Consent_Management/Data_Management/Kafka/** directory and run **producer.py**.

```
python3 producer.py
```

As shown in the image below the producer uses Vault API to encrypt data. you can check **GDPR-Compliance-Big-Data-Systems/PythonVersion_Consent_Management/Data_Management/Kafka/encrypt_with_vault.py** to see how the encryption process is done.

alt text

To verify that data was delivered as planned you can either access MongoDB container or use the MongoDB compass GUI

1. Access MongoDB container

open a new terminal and run this command

```
sudo docker exec -it mongodb /bin/sh
```

to open the shell command line run this command

```
mongo
```

Inside the mongo shell run these commands

```
use Bankerise
```

```
db.AppUsers.find()
```

2. Use MongoDB compass you can follow this link on how to install MongoDB compass : Installation

Now we are happy that we did the job correctly as you can the data was delivered securly encrypted with vault, we can move on tho the second module.

Running Module 2

Build Flask Docker Image

Vault and MongoDB containers are already running in server the final step is to create the Flask docker image and run the container. Proceed tro **GDPR-Compliance-Big-Data-Systems/PythonVersion_Consent_Management/FlaskApi/** and run the **create_flask_image.sh**

```
./create_flask_image.sh
```

MongoDB View Creation

To create the view proceed to **** directory and run the **create_views.py** file

```
python3 create_views.py
```

The view in MongoDB is created by applying this pipeline. You can customise the **create_views.py** as needed since we have severl piplines to perform.

```
    command = [
       {
          "$lookup":{
         "from":"UsersPolicy",
         "let":{
            "id_u":"$id"
         },
         "pipeline":[
            {
               "$match":{
                  "$expr":{
                     "$and":[
                        {
                           "$eq":[
                              "$id",
                              "$$id_u"
                           ]
                        }
                     ]
                  }
               }
            }
         ],
         "as":"same_id"
          }
       },
       {
          "$replaceRoot":{
         "newRoot":{
            "$mergeObjects":[
               {
                  "$arrayElemAt":[
                     "$same_id",
                     0
                  ]
               },
               "$$ROOT"
            ]
         }
          }
       },
       {
          "$project":{
         "same_id":0
          }
       },
       {
          "$project":{
         "id":1,
         "first_name":{
            "$cond":[
               {
                  "$eq":[
                     "$use_first_name",
                     1
                  ]
               },
               "$first_name",
               "$$REMOVE"
            ]
         },
         "last_name":{
            "$cond":[
               {
                  "$eq":[
                     "$use_last_name",
                     1
                  ]
               },
               "$last_name",
               "$$REMOVE"
            ]
         },
         "email":{
            "$cond":[
               {
                  "$eq":[
                     "$use_email",
                     1
                  ]
               },
               "$email",
               "$$REMOVE"
            ]
         },
         "gender":{
            "$cond":[
               {
                  "$eq":[
                     "$use_gender",
                     1
                  ]
               },
               "$gender",
               "$$REMOVE"
            ]
         },
         "ip_address":{
            "$cond":[
               {
                  "$eq":[
                     "$use_ip_address",
                     1
                  ]
               },
               "$ip_address",
               "$$REMOVE"
            ]
         },
         "transfert_amount":{
            "$cond":[
               {
                  "$eq":[
                     "$use_transfert_amount",
                     1
                  ]
```

```
                    },
                    "$transfert_amount",
                    "$$REMOVE"
                ]
            },
            "date":1
             }
        }
    ]
```