

# 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**

## Context and Objectives

---

### Key words

- **Service:** Which is one of the programmed services it may be any API that needs data to process such as a **Machine Learning** service that demands customer data at REST to perform clustering or client segmentation,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

**Customer Data Restrictions**

	Customer Name	Customer Last Name	Device	Ip_address	Service	Transfer Amount
1	✓	✗	✓	✓	✓	✗
2	✗	✓	✗	✓	✗	✓
3	✗	✗	✗	✓	✓	✓

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 main two goals are to guarantee end to end secure data flow in real time by delivering encrypted logs and maintaining access control layer built on top of the database with respect to all GDPR constraints.

## Project Architecture

 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. 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

 alt text

- Before proceeding to run any script or command let's first install requirements:

**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.

- **Docker** installation for Linux :

```
sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-  
compose-plugin
```

To verify that Docker is installed on your machine please try to run this command. if every thing is setted the installation of the image will take effect and the engine will run it then it will exit

```
sudo docker run hello-world
```

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

```
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_Connec  
t/  
./test_rootless.sh
```

**Note:** If you encounter an error **bash: ./test\_rootless.sh Permission denied** you should give the permission 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 everything is good and compatible with the image below type this command

```
docker ps
```



## Database Sink Connector

### Configuration

Kafka connect 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/my_events",
    "database": "my_events",
    "collection": "kafka_events",
    "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 configured as a Sink Connector 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.

### Data Factory


After we settled the environment 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

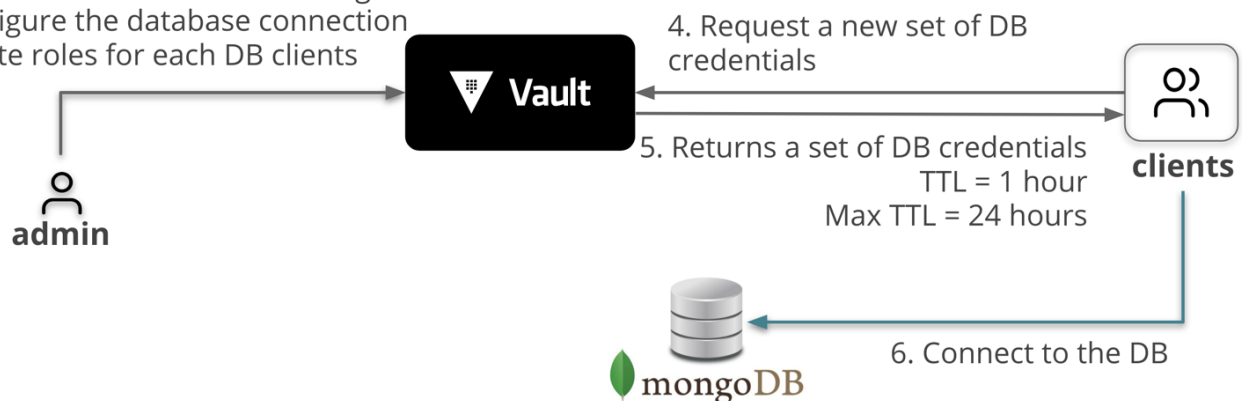
## Module 1 Data Protection Officer

**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



**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.

### Steps to run Module 2 properly

Before running the Spring Boot Application there are two steps that you need to perform

- **1** : start Vault Docker container using this command

```
docker run --net BigData --name vault1 -d --cap-add=IPC_LOCK -e
'VAULT_DEV_ROOT_TOKEN_ID=myroot' -e 'VAULT_DEV_LISTEN_ADDRESS=0.0.0.0:8200' -p
8200:8200 vault
```

- **2** : start Mongoddb container using this command

```
docker run -d -p 0.0.0.0:27017:27017 --name=mongoddb -e
MONGO_INITDB_ROOT_USERNAME="mdbadmin" -e
MONGO_INITDB_ROOT_PASSWORD="hQ97T9JJKZoqnFn2NXE" mongo
```

**Note :**

- For more information about building Vault image visit the official Docker image on Docker Hub
- Specifying the container network is certain to assure the communication between other containers within the Application