ENSET

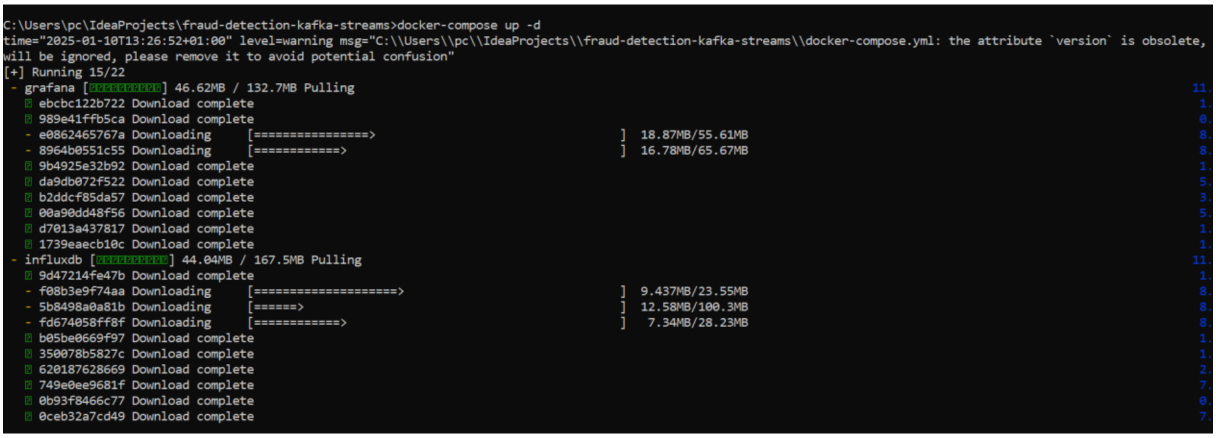
ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Détection de Fraudes en Temps Réel avec Kafka Streams et Tableau de Bord Grafana

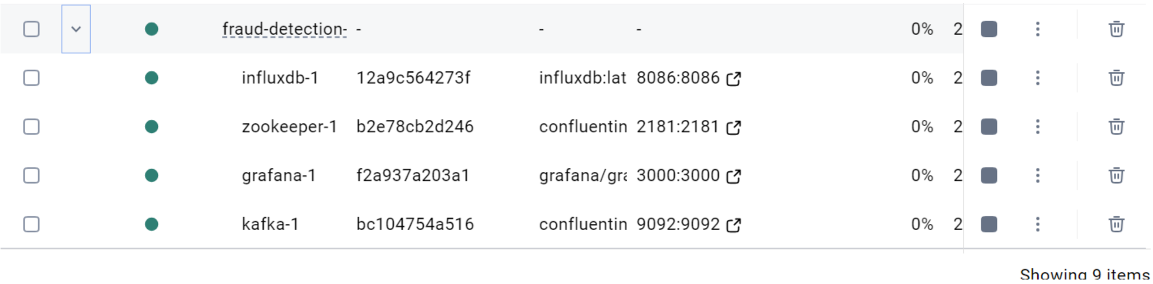
Réalisé par : **AIT JEDDI Assia**

Encadré par : **Abdelmajid BOUSSELHAM**

1. Configuration des Topics Kafka



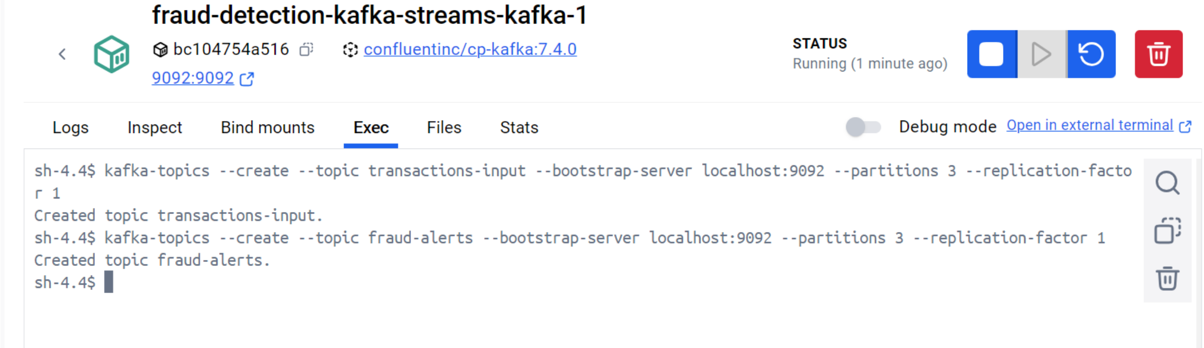
```
C:\Users\pc\IdeaProjects\fraud-detection-kafka-streams>docker-compose up -d
time="2025-01-10T13:26:52+01:00" level=warning msg="C:\\Users\\pc\\IdeaProjects\\fraud-detection-kafka-streams\\docker-compose.yml: the attribute `version` is obsolete,
will be ignored, please remove it to avoid potential confusion"
[+] Running 15/22
 - grafana [▯▯▯▯▯▯▯▯▯▯▯] 46.62MB / 132.7MB Pulling                                                                                                              11.
   ▯ ebcbc122b722 Download complete                                                                                                                               1.
   ▯ 989e41ffb5ca Download complete                                                                                                                               0.
   - e0862465767a Downloading     [================>                                 ]  18.87MB/55.61MB                                                              8.
   - 8964b0551c55 Downloading     [============>                                     ]  16.78MB/65.67MB                                                              8.
   ▯ 9b4925e32b92 Download complete                                                                                                                               1.
   ▯ da9db072f522 Download complete                                                                                                                               5.
   ▯ b2ddcf85da57 Download complete                                                                                                                               3.
   ▯ 00a90dd48f56 Download complete                                                                                                                               5.
   ▯ d7013a437817 Download complete                                                                                                                               1.
   ▯ 1739eaecb10c Download complete                                                                                                                               1.
 - influxdb [▯▯▯▯▯▯▯▯▯▯▯] 44.04MB / 167.5MB Pulling                                                                                                             11.
   ▯ 9d47214fe47b Download complete                                                                                                                               1.
   - f08b3e9f74aa Downloading     [====================>                             ]  9.437MB/23.55MB                                                              8.
   - 5b8498a0a81b Downloading     [======>                                           ]  12.58MB/100.3MB                                                              8.
   - fd674058ff8f Downloading     [============>                                     ]   7.34MB/28.23MB                                                              8.
   ▯ b05be0669f97 Download complete                                                                                                                               1.
   ▯ 350078b5827c Download complete                                                                                                                               1.
   ▯ 620187628669 Download complete                                                                                                                               2.
   ▯ 749e0ee9681f Download complete                                                                                                                               7.
   ▯ 0b93f8466c77 Download complete                                                                                                                               0.
   ▯ 0ceb32a7cd49 Download complete                                                                                                                               7.
```



□	⌄	●	fraud-detection-	-	-	-	0%	2	■	⋮	🗑
□		●	influxdb-1	12a9c564273f	influxdb:lat	8086:8086 ↗	0%	2	■	⋮	🗑
□		●	zookeeper-1	b2e78cb2d246	confluentin	2181:2181 ↗	0%	2	■	⋮	🗑
□		●	grafana-1	f2a937a203a1	grafana/grı	3000:3000 ↗	0%	2	■	⋮	🗑
□		●	kafka-1	bc104754a516	confluentin	9092:9092 ↗	0%	2	■	⋮	🗑

Showing 9 items

- **Création de deux topics Kafka à l'aide de la CLI Kafka :**
  - **transactions-input** : Contient les transactions brutes.
  - **fraud-alerts** : Stocke les transactions suspectes.



fraud-detection-kafka-streams-kafka-1

bc104754a516 confluentinc/cp-kafka:7.4.0

9092:9092

STATUS
Running (1 minute ago)

```
sh-4.4$ kafka-topics --create --topic transactions-input --bootstrap-server localhost:9092 --partitions 3 --replication-facto
r 1
Created topic transactions-input.
sh-4.4$ kafka-topics --create --topic fraud-alerts --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1
Created topic fraud-alerts.
sh-4.4$ █
```

2. Configuration des Topics Kafka

- **Dépendances Maven :**

Ajout des dépendances suivantes dans pom.xml :

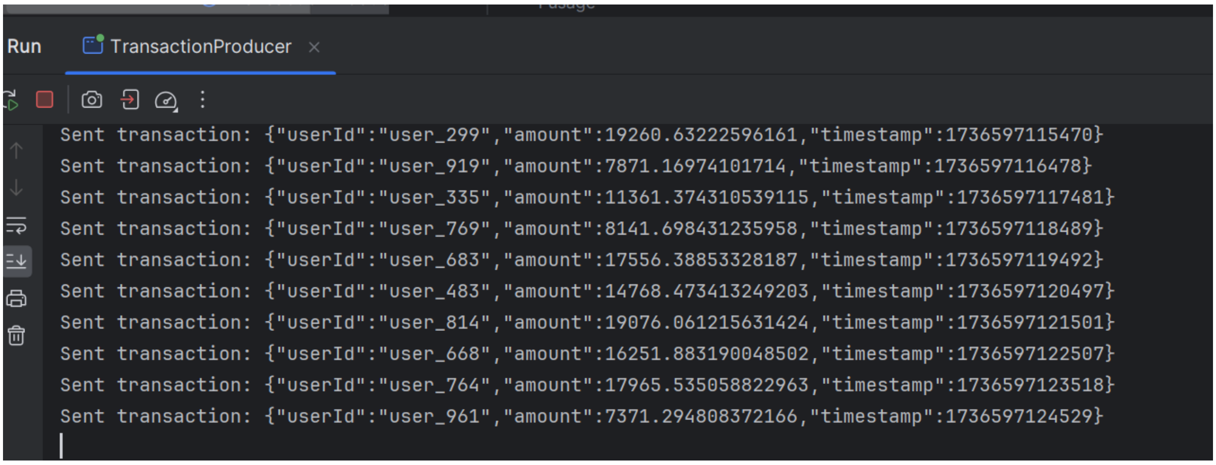
```
<dependencies>
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
```

```
      <version>3.5.0</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.15.2</version>
    </dependency>
  </dependencies>

</project>
```

3. Application Kafka Streams

1- Produire les transactions financières (Producer) :

```
package ma.enset;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;
import java.util.Random;

public class TransactionProducer {
    private static final String TOPIC = "transactions-input";
    private static final Random RANDOM = new Random();
    private static final ObjectMapper MAPPER = new ObjectMapper();

    public static void main(String[] args) {
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

        try (KafkaProducer<String, String> producer = new KafkaProducer<>(props)) {
            while (true) {
                Transaction transaction = generateTransaction();
                String json = MAPPER.writeValueAsString(transaction);

                ProducerRecord<String, String> record =
                        new ProducerRecord<>(TOPIC, transaction.getUserId(), json);

                producer.send(record, (metadata, exception) -> {
                    if (exception != null) {
                        System.err.println("Error sending message: " + exception.getMessage());
                    } else {
                        System.out.println("Sent transaction: " + json);
                    }
                });
                Thread.sleep(1000); // Wait 1 second between messages
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static Transaction generateTransaction() {
        String userId = "user_" + RANDOM.nextInt(1000);
        double amount = 5000 + RANDOM.nextDouble() * 15000; // Random
amount
        long timestamp = System.currentTimeMillis();
        return new Transaction(userId, amount, timestamp);
    }
}

class Transaction {
    private String userId;
    private double amount;
    private long timestamp;

    public Transaction(String userId, double amount, long timestamp) {
        this.userId = userId;
        this.amount = amount;
        this.timestamp = timestamp;
    }

    public String getUserId() {
        return userId;
    }

    public double getAmount() {
        return amount;
    }

    public long getTimestamp() {
        return timestamp;
    }
}
```

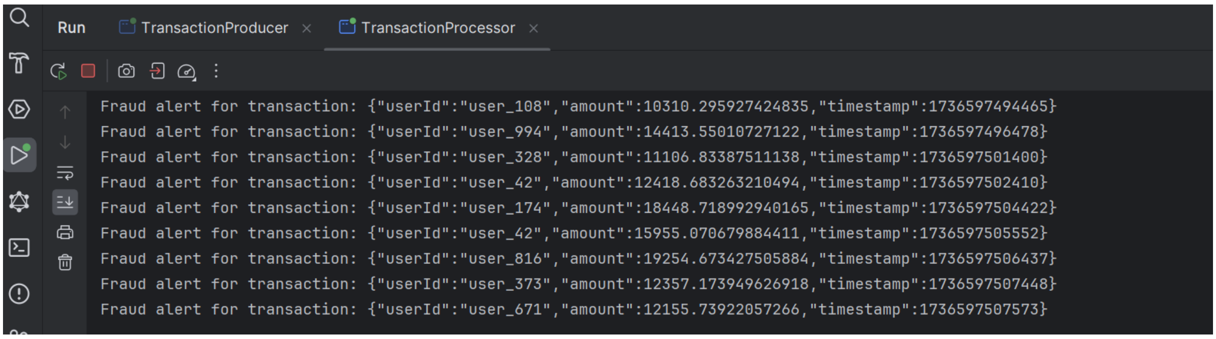


```
Sent transaction: {"userId":"user_299","amount":19260.63222596161,"timestamp":1736597115470}
Sent transaction: {"userId":"user_919","amount":7871.16974101714,"timestamp":1736597116478}
Sent transaction: {"userId":"user_335","amount":11361.374310539115,"timestamp":1736597117481}
Sent transaction: {"userId":"user_769","amount":8141.698431235958,"timestamp":1736597118489}
Sent transaction: {"userId":"user_683","amount":17556.38853328187,"timestamp":1736597119492}
Sent transaction: {"userId":"user_483","amount":14768.473413249203,"timestamp":1736597120497}
Sent transaction: {"userId":"user_814","amount":19076.061215631424,"timestamp":1736597121501}
Sent transaction: {"userId":"user_668","amount":16251.883190048502,"timestamp":1736597122507}
Sent transaction: {"userId":"user_764","amount":17965.535058822963,"timestamp":1736597123518}
Sent transaction: {"userId":"user_961","amount":7371.294808372166,"timestamp":1736597124529}
```

2- Filtrer les transactions suspectes (Processor) :

```
package ma.enset;

import com.fasterxml.jackson.databind.ObjectMapper;
```

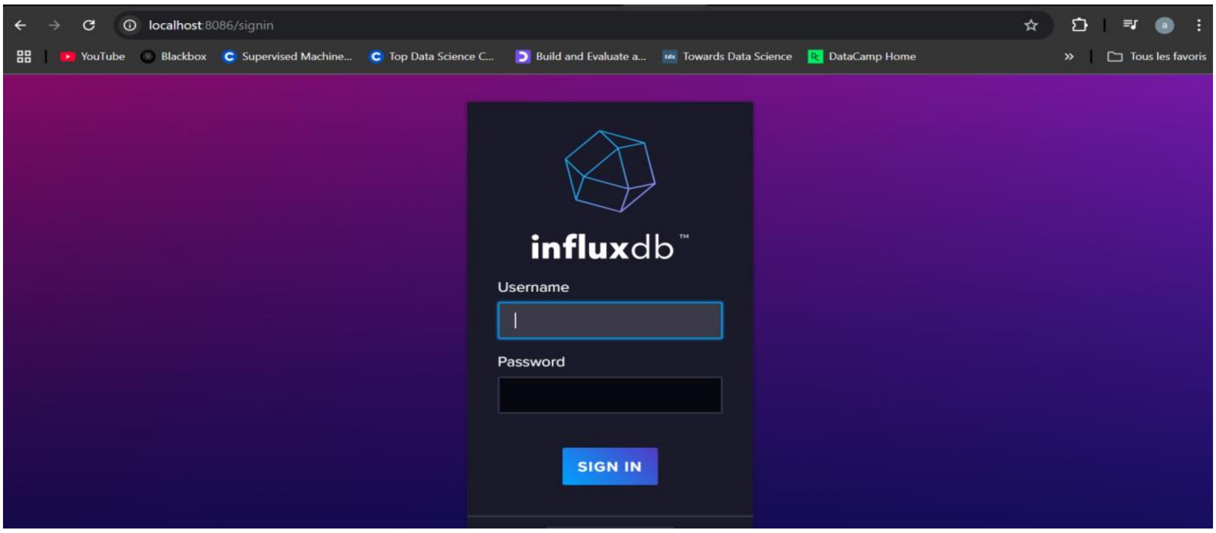
```java
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;

import java.util.Properties;

public class TransactionProcessor {
    private static final String INPUT_TOPIC = "transactions-input";
    private static final String OUTPUT_TOPIC = "fraud-alerts";
    private static final double SUSPICIOUS_AMOUNT = 10000.0;
    private static final ObjectMapper MAPPER = new ObjectMapper();

    public static void main(String[] args) {
        Properties config = new Properties();
        config.put(StreamsConfig.APPLICATION_ID_CONFIG, "transaction-processor");
        config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        config.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
        config.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

        StreamsBuilder builder = new StreamsBuilder();

        // Create stream from input topic
        KStream<String, String> inputStream = builder.stream(INPUT_TOPIC);

        // Process transactions
        inputStream
                .mapValues(value -> {
                    try {
                        return MAPPER.readValue(value, Transaction.class);
                    } catch (Exception e) {
                        System.err.println("Error parsing transaction: " + e.getMessage());
                        return null;
                    }
                })
                .filter((key, transaction) -> transaction != null)
                // Split the stream into fraudulent and normal transactions
                .branch((key, transaction) -> transaction.getAmount() > SUSPICIOUS_AMOUNT)[0]  // Get the fraudulent branch
                .mapValues(transaction -> {
                    try {
                        String json = MAPPER.writeValueAsString(transaction);
                        System.out.println("Fraud alert for transaction: " + json);
                        return json;
                    } catch (Exception e) {
                        System.err.println("Error serializing transaction: " + e.getMessage());
                        return null;
                    }
                })
                .filter((key, value) -> value != null)
                .to(OUTPUT_TOPIC);
        KafkaStreams streams = new KafkaStreams(builder.build(), config);
        streams.start();

        // Shutdown hook
        Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
    }
}
```

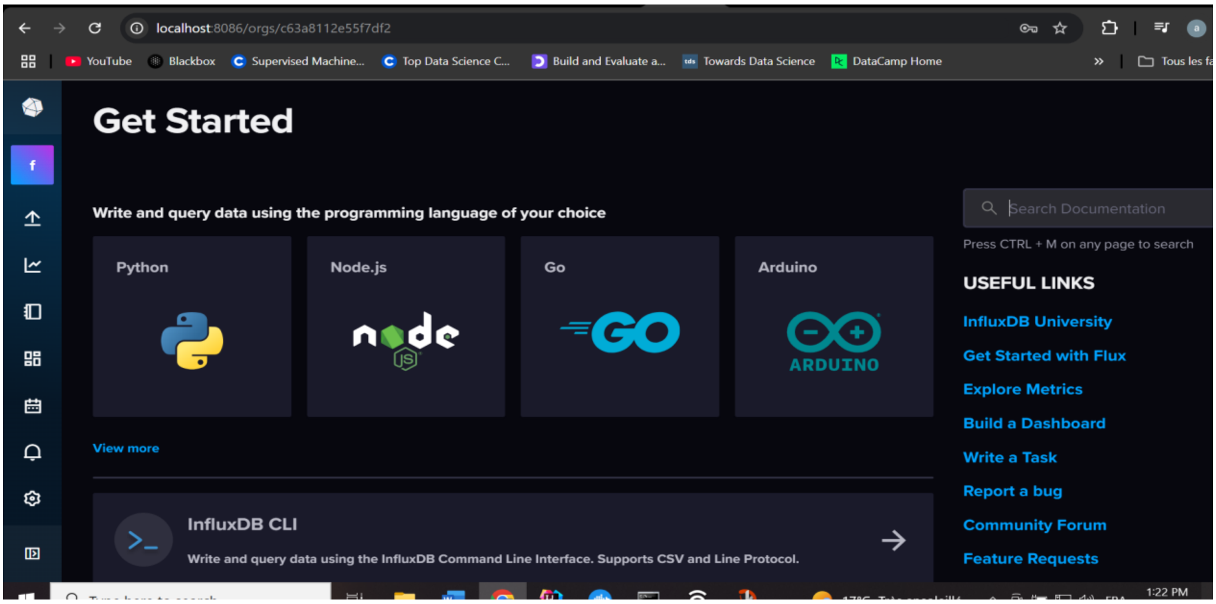


```
Fraud alert for transaction: {"userId":"user_108","amount":10310.295927424835,"timestamp":1736597494465}
Fraud alert for transaction: {"userId":"user_994","amount":14413.55010727122,"timestamp":1736597496478}
Fraud alert for transaction: {"userId":"user_328","amount":11106.83387511138,"timestamp":1736597501400}
Fraud alert for transaction: {"userId":"user_42","amount":12418.683263210494,"timestamp":1736597502410}
Fraud alert for transaction: {"userId":"user_174","amount":18448.718992940165,"timestamp":1736597504422}
Fraud alert for transaction: {"userId":"user_42","amount":15955.070679884411,"timestamp":1736597505552}
Fraud alert for transaction: {"userId":"user_816","amount":19254.673427505884,"timestamp":1736597506437}
Fraud alert for transaction: {"userId":"user_373","amount":12357.173949626918,"timestamp":1736597507448}
Fraud alert for transaction: {"userId":"user_671","amount":12155.73922057266,"timestamp":1736597507573}
```

3- Consommer les transactions suspectes (Consumer) :

- Démarrer Influxdb :



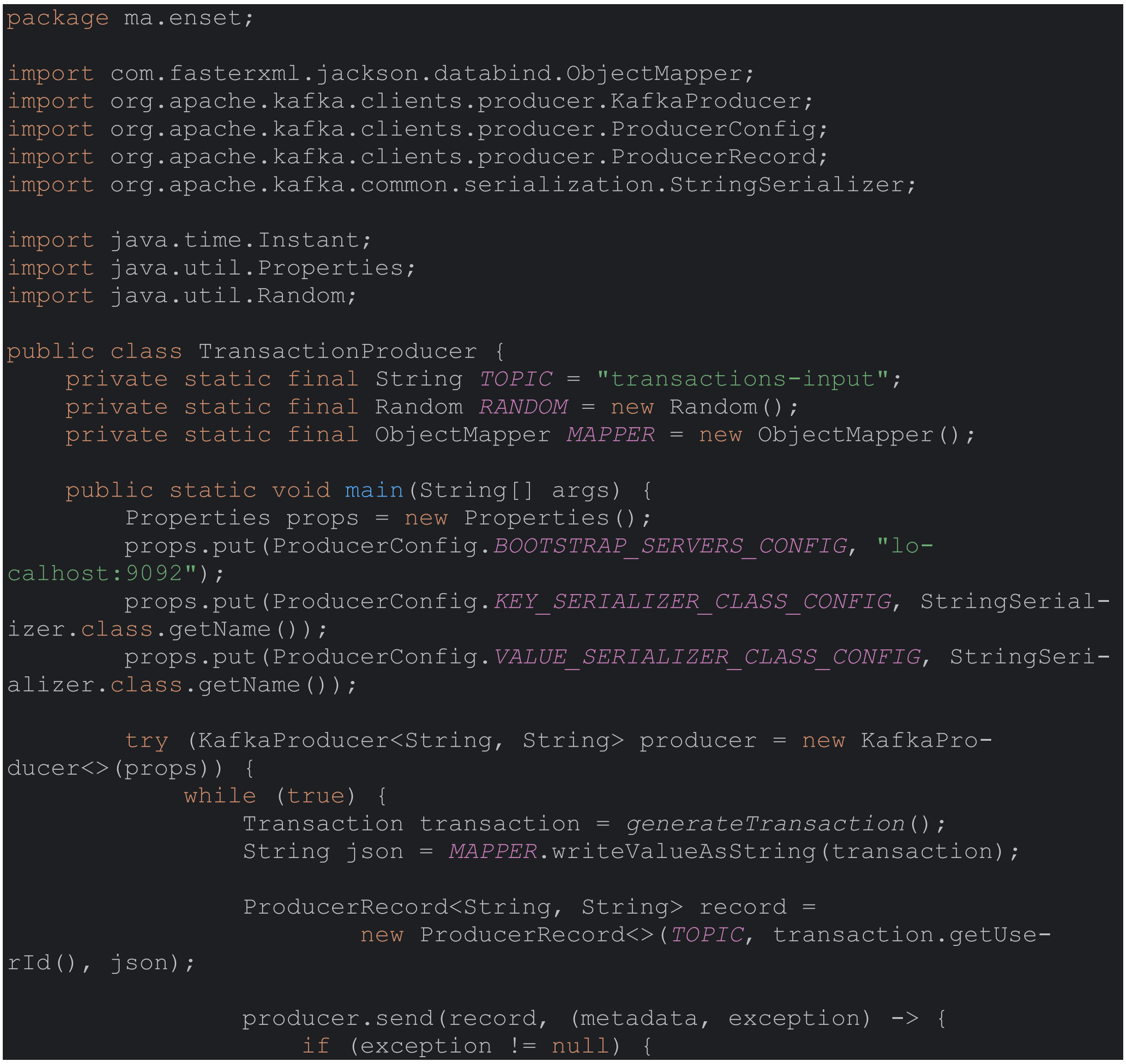


Get Started

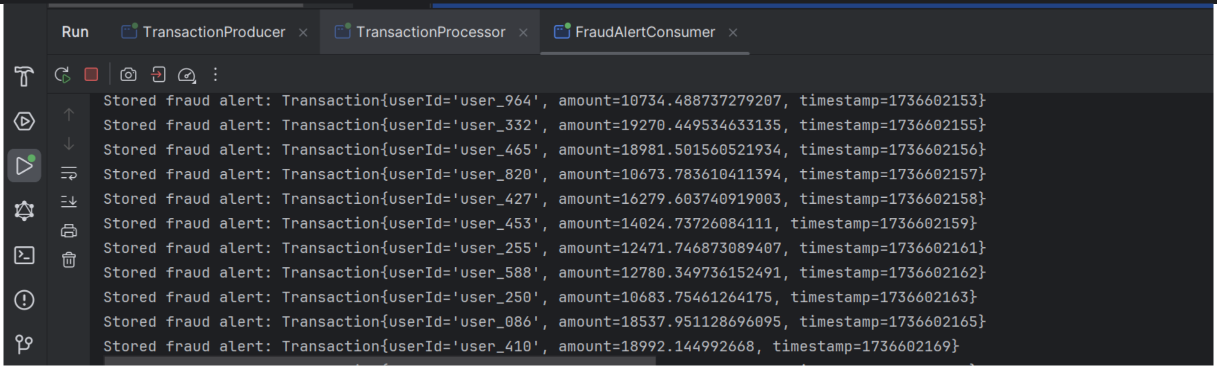
Write and query data using the programming language of your choice

InfluxDB CLI

Write and query data using the InfluxDB Command Line Interface. Supports CSV and Line Protocol.



```java
package ma.enset;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

import java.time.Instant;
import java.util.Properties;
import java.util.Random;

public class TransactionProducer {
    private static final String TOPIC = "transactions-input";
    private static final Random RANDOM = new Random();
    private static final ObjectMapper MAPPER = new ObjectMapper();

    public static void main(String[] args) {
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

        try (KafkaProducer<String, String> producer = new KafkaProducer<>(props)) {
            while (true) {
                Transaction transaction = generateTransaction();
                String json = MAPPER.writeValueAsString(transaction);

                ProducerRecord<String, String> record =
                        new ProducerRecord<>(TOPIC, transaction.getUserId(), json);

                producer.send(record, (metadata, exception) -> {
                    if (exception != null) {
                        System.err.println("Error sending message: " + exception.getMessage());
                    } else {
                        System.out.println("Sent transaction: " + json);
                    }
                });

                Thread.sleep(1000); // Wait 1 second between messages
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static Transaction generateTransaction() {
        String userId = String.format("user_%03d", RANDOM.nextInt(1000));
        double amount = 1000 + RANDOM.nextDouble() * 19000; // Random amount between 1000 and 20000
        int timestamp = (int) (System.currentTimeMillis() / 1000); // Current Unix timestamp
        return new Transaction(userId, amount, timestamp);
    }
}
```

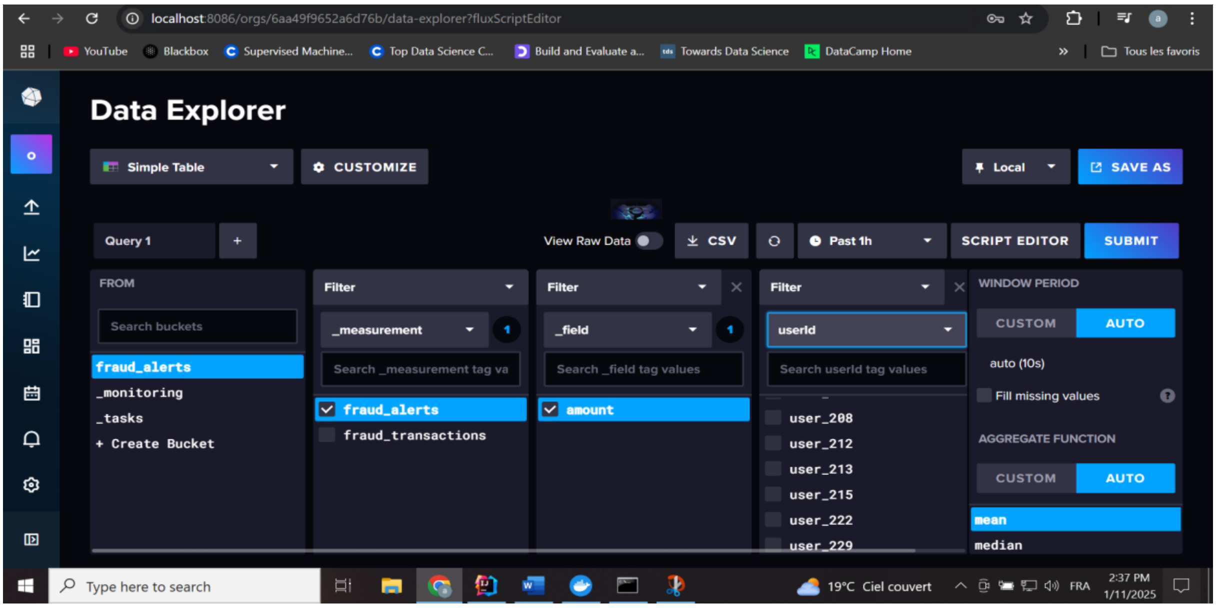


```
Stored fraud alert: Transaction{userId='user_964', amount=10734.488737279207, timestamp=1736602153}
Stored fraud alert: Transaction{userId='user_332', amount=19270.449534633135, timestamp=1736602155}
Stored fraud alert: Transaction{userId='user_465', amount=18981.501560521934, timestamp=1736602156}
Stored fraud alert: Transaction{userId='user_820', amount=10673.783610411394, timestamp=1736602157}
Stored fraud alert: Transaction{userId='user_427', amount=16279.603740919003, timestamp=1736602158}
Stored fraud alert: Transaction{userId='user_453', amount=14024.73726084111, timestamp=1736602159}
Stored fraud alert: Transaction{userId='user_255', amount=12471.746873089407, timestamp=1736602161}
Stored fraud alert: Transaction{userId='user_588', amount=12780.349736152491, timestamp=1736602162}
Stored fraud alert: Transaction{userId='user_250', amount=10683.75461264175, timestamp=1736602163}
Stored fraud alert: Transaction{userId='user_086', amount=18537.951128696095, timestamp=1736602165}
Stored fraud alert: Transaction{userId='user_410', amount=18992.144992668, timestamp=1736602169}
```

4. Stocker les Transactions Suspectes dans InfluxDB

- Insertion des transactions suspectes directement dans InfluxDB avec des champs tels que :

  - userId
  - amount
  - timestamp



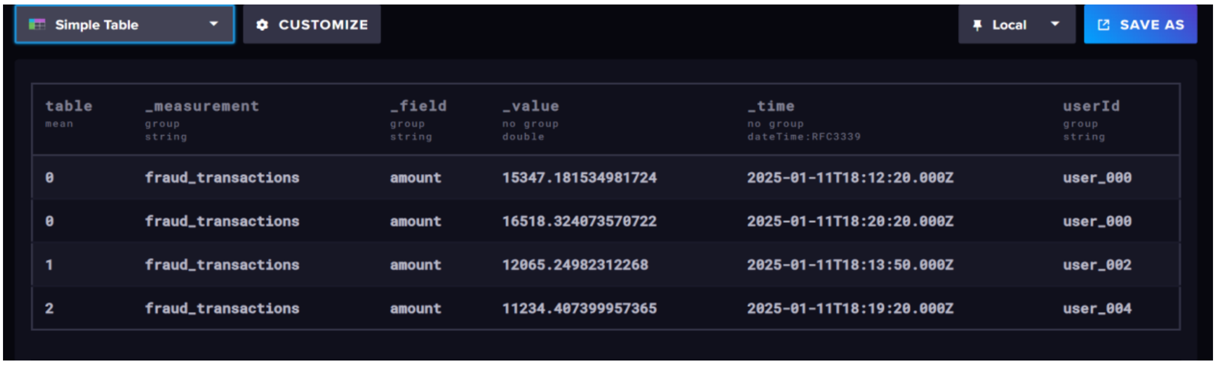
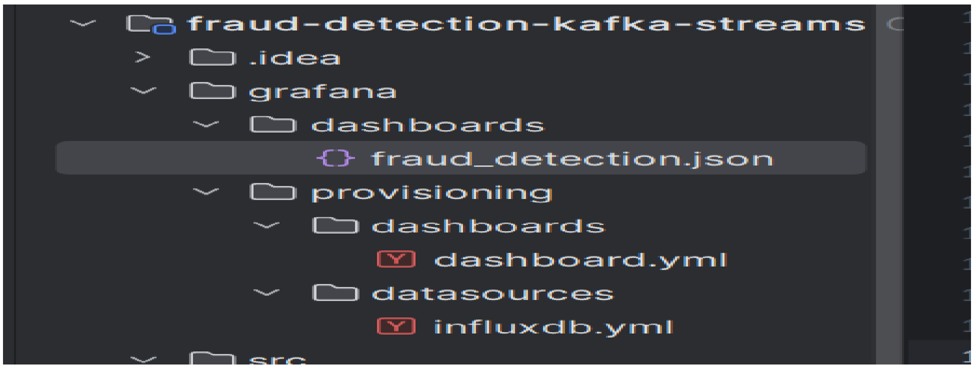


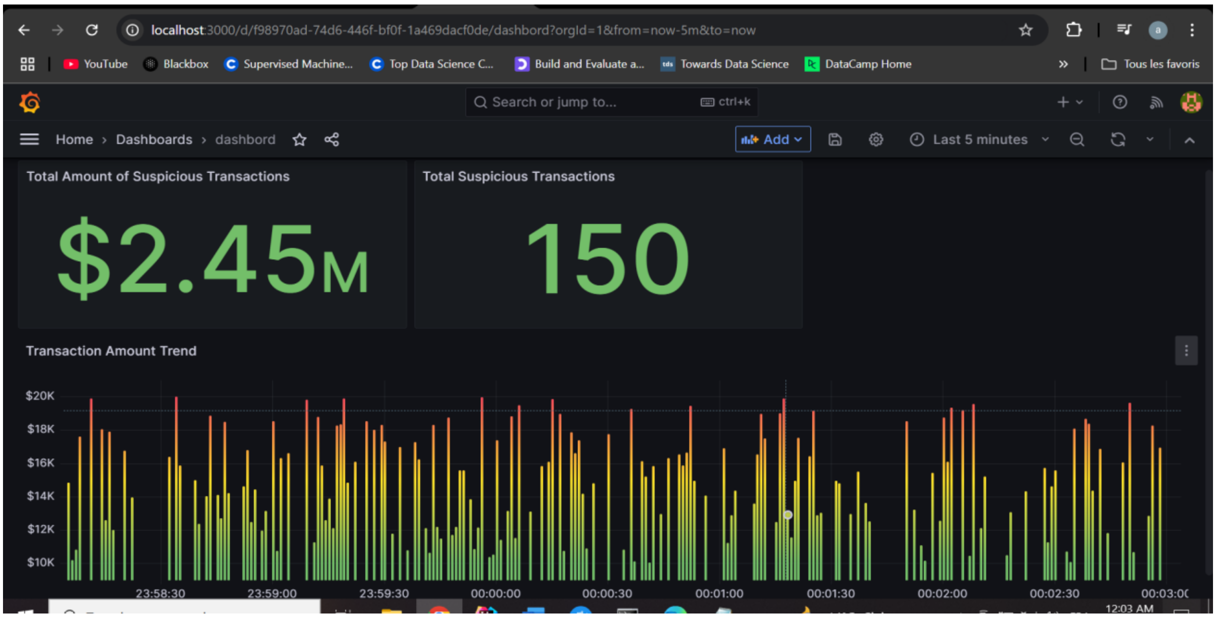
table mean	_measurement group string	_field group string	_value no group double	_time no group dateTime:RFC3339	userId group string
0	fraud_transactions	amount	15347.181534981724	2025-01-11T18:12:20.000Z	user_000
0	fraud_transactions	amount	16518.324073570722	2025-01-11T18:20:20.000Z	user_000
1	fraud_transactions	amount	12065.24982312268	2025-01-11T18:13:50.000Z	user_002
2	fraud_transactions	amount	11234.407399957365	2025-01-11T18:19:20.000Z	user_004

4. Tableau de Bord Grafana

- Configuration de Grafana pour se connecter à InfluxDB et visualiser les transactions suspectes.



```
fraud-detection-kafka-streams
  .idea
  grafana
    dashboards
      fraud_detection.json
    provisioning
      dashboards
        dashboard.yml
      datasources
        influxdb.yml
  src
```



Total Amount of Suspicious Transactions

$2.45M

Total Suspicious Transactions

150

Transaction Amount Trend

5. Déploiement

- Docker-compose.yml :

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"

  influxdb:
    image: influxdb:2.7
    ports:
      - "8086:8086"
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=assia1234
      - DOCKER_INFLUXDB_INIT_ORG=myorg
      - DOCKER_INFLUXDB_INIT_BUCKET=fraud_alerts
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=assia1234

  grafana:
    image: grafana/grafana:9.5.2
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=assia1234
      - GF_AUTH_ANONYMOUS_ENABLED=true
    volumes:
      - ./grafana/provisioning:/etc/grafana/provisioning
      - ./grafana/dashboards:/var/lib/grafana/dashboards
    depends_on:
      - influxdb
```