

Advanced Distributed Systems

Higher Institute for Applied Sciences and Technology

Student: ضياء حنا
Teacher: محمد بشار دسوقي
Date: 15/3/2026

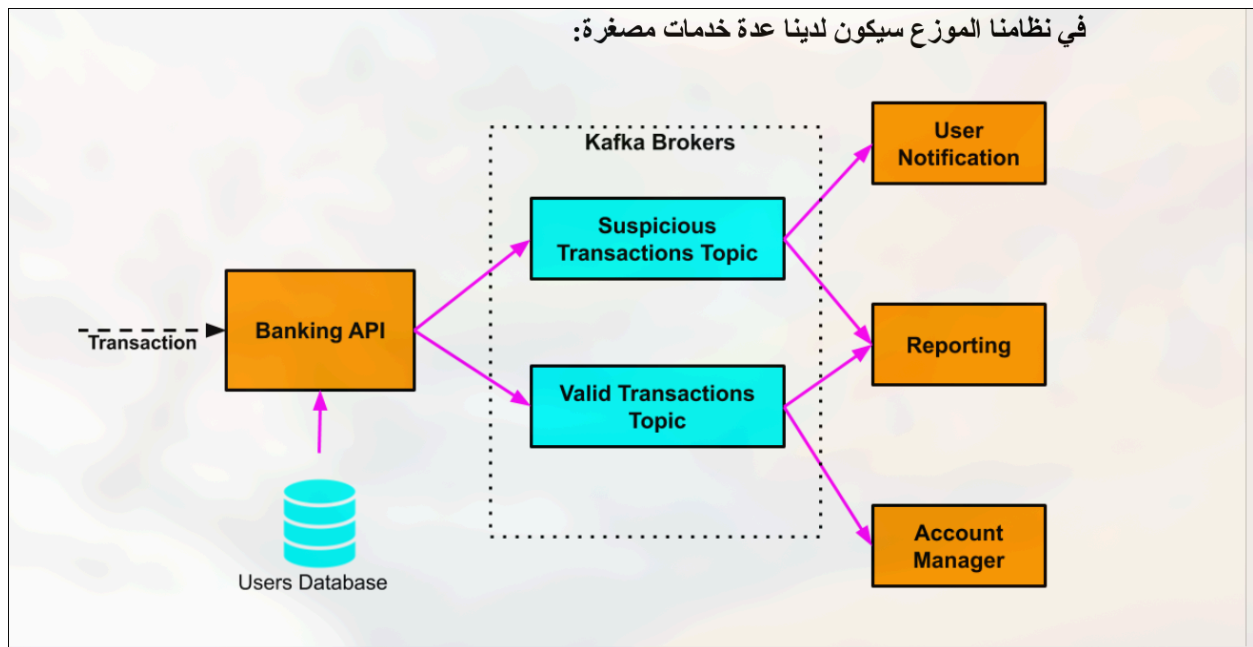
OurBank Report

March 15 2026

The goal of this file is to answer the questions about our written code for the homework and to prove that we actually accomplished what is asked.

Part 1 (كما هو واضح في Kafka تتم جميع الاتصالات داخل نظامنا الموزع من خلال موضوعات التصميم).

first we should understand our system



The kafka component the student decided that the this homework should have two message brokers

Our system should have two topics as the homework requested

1.sus Transactions topic 2.valid translation topic

Each topic is divided into two partitions for parallelism (we are striving the simulate the lab class we did that in class)

The replication factor is 2 (according to class two)

In this way we have

A total of 8 partitions each message broker contains 4 partitions

Message broker one :

Leader of sus transaction

```

x Thu 19 Mar - 21:09 /opt/kafka
@drnull bin/kafka-topics.sh --describe --topic suspicious-transactions --bootstrap-server localhost:9092
Topic: suspicious-transactions TopicId: vFnCqmq9Tve6mwPWqX5qEQ PartitionCount: 2 ReplicationFactor: 2 Configs:
,1 Topic: suspicious-transactions Partition: 0 Leader: 0 Replicas: 0,1 Isr: 0
,0 Topic: suspicious-transactions Partition: 1 Leader: 1 Replicas: 1,0 Isr: 1

Thu 19 Mar - 21:09 /opt/kafka
@drnull bin/kafka-topics.sh --describe --topic valid-transactions --bootstrap-server localhost:9092
Topic: valid-transactions TopicId: Fv-xeBbmTse2Z0BeSyaD0g PartitionCount: 2 ReplicationFactor: 2 Configs:
,1 Topic: valid-transactions Partition: 0 Leader: 0 Replicas: 0,1 Isr: 0
,0 Topic: valid-transactions Partition: 1 Leader: 1 Replicas: 1,0 Isr: 1

Thu 19 Mar - 21:09 /opt/kafka
@drnull

```

Message broker one contains partition one of each topic and replicas for partition 1

And message broker 2 contains partition 2 of each topic and replicas of 1

Explaining the services

Banking services

First it populate the redis database from a text file the contains the users locations

Then it Reads incoming-transations.txt

And act as a producer it is nearly identical to the code

<https://git.hiast.edu.sy/mohamadbashar.disoki/java-kafka-produce>

The other services:

Reporting service : it just create text file report under the directory

Reports

Same thing notification service is the only difference is that reporting service listen fro both topics

Account manager service : it only consume valid transactions and print it to the screen

Important note:

In the code we consumer group so if we fire multiple consumers for example notification service

It will be guaranteed that on message is send to the entire consumer group this way we guarantee parallelism and validity of execution

Lets try to fire tow notification services

One account manager service one reporting service

And two producers

```

Authorizing transaction for user: smith_teresa
Action: Transferring $10.00 from User Account to the Merchant at Spain
Status: SUCCESS - Funds Dispatched.

-----

Authorizing transaction for user: andrewb1999
Action: Transferring $30.00 from User Account to the Merchant at Turkey
Status: SUCCESS - Funds Dispatched.

-----

Authorizing transaction for user: cameron-dion
Action: Transferring $150.00 from User Account to the Merchant at Texas
Status: SUCCESS - Funds Dispatched.

-----

Authorizing transaction for user: arjun4444
Action: Transferring $15.00 from User Account to the Merchant at Canada
Status: SUCCESS - Funds Dispatched.

-----

Authorizing transaction for user: annabecker_1997
Action: Transferring $15.00 from User Account to the Merchant at Sweden
Status: SUCCESS - Funds Dispatched.

-----

ohson98
Successfully processed transaction for: smith_teresa
Successfully processed transaction for: andrewb1999
Successfully processed transaction for: cameron-dion
Successfully processed transaction for: arjun4444
Successfully processed transaction for: rehansh_jk2001
Successfully processed transaction for: mia_fisher
Successfully processed transaction for: annabecker_1997
SKIPPING TRANSACTION: User sara_lee doesn't exist in Redis database
SKIPPING TRANSACTION: User pablo_g doesn't exist in Redis database
SKIPPING TRANSACTION: User yuki_tanaka doesn't exist in Redis database

Thu 19 Mar - 21:37 ~/UNI/adistr/OurBank/bank-api-service origin @ master 7* 22

ohson98
Successfully processed transaction for: smith_teresa
Successfully processed transaction for: andrewb1999
Successfully processed transaction for: cameron-dion
Successfully processed transaction for: arjun4444
Successfully processed transaction for: rehansh_jk2001
Successfully processed transaction for: mia_fisher
Successfully processed transaction for: annabecker_1997
SKIPPING TRANSACTION: User sara_lee doesn't exist in Redis database
SKIPPING TRANSACTION: User pablo_g doesn't exist in Redis database
SKIPPING TRANSACTION: User yuki_tanaka doesn't exist in Redis database

Thu 19 Mar - 21:37 ~/UNI/adistr/OurBank/bank-api-service origin @ master 7* 22

Sending user annabecker_1997 notification about a suspicious transaction of $90.00 in their account originating in Brazil
Sending user john1967 notification about a suspicious transaction of $200.00 in their account originating in France
Sending user arthur_wilson notification about a suspicious transaction of $250.00 in their account originating in UK
Sending user msmith2015 notification about a suspicious transaction of $50.00 in their account originating in California
Sending user cameron-dion notification about a suspicious transaction of $100.00 in their account originating in Florida
Sending user annabecker_1997 notification about a suspicious transaction of $90.00 in their account originating in Brazil
Sending user john1967 notification about a suspicious transaction of $200.00 in their account originating in France
Sending user arthur_wilson notification about a suspicious transaction of $250.00 in their account originating in UK

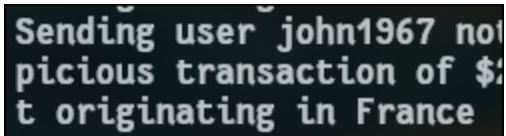
Valid transaction for user rehansh_jk2001, amount $100.00
Valid transaction for user mia_fisher, amount $45.00
Valid transaction for user john1967, amount $30.00
Valid transaction for user msmith2015, amount $200.00
Valid transaction for user msmith2015, amount $10.00
Valid transaction for user jialiang1988, amount $200.00
Valid transaction for user arthur_wilson, amount $45.00
Valid transaction for user smith_teresa, amount $10.00
Valid transaction for user andrewb1999, amount $30.00
Valid transaction for user cameron-dion, amount $150.00
Valid transaction for user arjun4444, amount $15.00
Valid transaction for user annabecker_1997, amount $15.00

```

The bottom left terminal is the two producers and the rest of the terminal are the promised consumers

If we check out the folder notifications and reports folder we can check that that our code worked perfectly

If we laser focus on one message we see it arrived one time



Sending user john1967 not
suspicious transaction of \$:
t originating in France

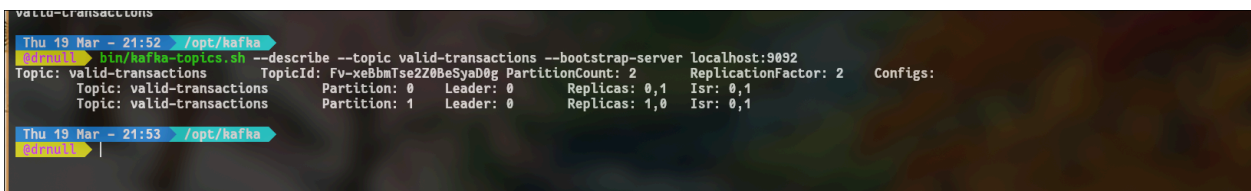
Which is the expected behavior

Part 2(stress testing)

We will start with stress testing because it is the easiest

The goal of this section is to know what are my limits of my laptop

First we verify the topics



```

Thu 19 Mar - 21:52 /opt/kafka
$ bin/kafka-topics.sh --describe --topic valid-transactions --bootstrap-server localhost:9092
Topic: valid-transactions   TopicId: Fv-w6BhmTse2Z0beSya00g PartitionCount: 2   ReplicationFactor: 2   Configs:
  Topic: valid-transactions   Partition: 0   Leader: 0   Replicas: 0,1   Isr: 0,1
  Topic: valid-transactions   Partition: 1   Leader: 0   Replicas: 1,0   Isr: 0,1

Thu 19 Mar - 21:53 /opt/kafka
$
  
```

```
Topic: valid-transactions Partition: 1 Leader: 0 Replicas: 1,0 Isr: 0,1
Thu 19 Mar - 21:53 > /opt/kafka
ndrcnll> bin/kafka-topics.sh --describe --topic suspicious-transactions --bootstrap-server localhost:9092
Topic: suspicious-transactions TopicId: vFnCqmQ9TveGmwPWqX5qEQ PartitionCount: 2 ReplicationFactor: 2 Configs:
Topic: suspicious-transactions Partition: 0 Leader: 0 Replicas: 0,1 Isr: 0,1
Topic: suspicious-transactions Partition: 1 Leader: 0 Replicas: 1,0 Isr: 0,1
Thu 19 Mar - 21:53 > /opt/kafka
ndrcnll> |
```

2. Step 1: The "Baseline" Test (Steady Load)

First, let's send **100,000 messages** at a steady rate of **1,000 messages per second**. Each message will be **100 bytes** (which is roughly the size of your `Transaction JSON`).

```

Thu 19 Mar - 21:53 /opt/kafka
$ bin/kafka-producer-perf-test.sh \
  --topic valid-transactions \
  --num-records 100000 \
  --record-size 100 \
  --throughput 1000 \
  --producer-props bootstrap.servers=localhost:9092,localhost:9093 acks=all
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 23.5 ms avg latency, 474.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 1.2 ms avg latency, 6.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 1.0 ms avg latency, 3.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 13.0 ms max latency.
5002 records sent, 1000.0 records/sec (0.10 MB/sec), 0.7 ms avg latency, 4.0 ms max latency.
4999 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 16.0 ms max latency.
5002 records sent, 1000.4 records/sec (0.10 MB/sec), 0.6 ms avg latency, 7.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.5 ms avg latency, 3.0 ms max latency.
5001 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 25.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.6 ms avg latency, 3.0 ms max latency.
5001 records sent, 1000.2 records/sec (0.10 MB/sec), 0.6 ms avg latency, 7.0 ms max latency.
5000 records sent, 999.8 records/sec (0.10 MB/sec), 0.7 ms avg latency, 11.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.7 ms avg latency, 16.0 ms max latency.
5001 records sent, 1000.0 records/sec (0.10 MB/sec), 0.6 ms avg latency, 5.0 ms max latency.
5003 records sent, 1000.4 records/sec (0.10 MB/sec), 1.2 ms avg latency, 24.0 ms max latency.
5001 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 13.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 0.6 ms avg latency, 2.0 ms max latency.
4999 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 8.0 ms max latency.
5002 records sent, 1000.4 records/sec (0.10 MB/sec), 0.6 ms avg latency, 2.0 ms max latency.
[2026-03-19 19:56:15,209] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46087 on topic-partition valid-transactions-1, retrying (2147483646
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,213] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,221] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46088 on topic-partition valid-transactions-1, retrying (2147483646
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,221] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,255] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46090 on topic-partition valid-transactions-1, retrying (2147483645
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,257] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
100000 records sent, 999.940004 records/sec (0.10 MB/sec), 3.55 ms avg latency, 1151.00 ms max latency, 1 ms 50th, 2 ms 95th, 18 ms 99th, 1057 ms 99.9th.
Thu 19 Mar - 21:56 /opt/kafka
$

```

- **Average Latency:** 3.55 ms. This is incredibly fast. It means, on average, it takes less than 4 milliseconds for a transaction to be sent, replicated to the second broker (`acks=all`), and confirmed.
- **95th Percentile:** 2 ms. This tells you that 95% of all transactions were confirmed in just 2 milliseconds.

Step 2: Finding the "Max" Limit (Full Speed)

```
Thu 19 Mar - 21:56 /opt/kafka
$ bin/kafka-producer-perf-test.sh \
  --topic valid-transactions \
  --num-records 1000000 \
  --record-size 100 \
  --throughput -1 \
  --producer-props bootstrap.servers=localhost:9092,localhost:9093 acks=all
000000 records sent, 239750.659314 records/sec (22.86 MB/sec), 884.82 ms avg latency, 1818.00 ms max latency, 828 ms 50th, 1612 ms 95th, 1773 ms 99th, 1815 ms 99.9th.
Thu 19 Mar - 21:59 /opt/kafka
```

We will be able to send it in one shot using latency of 2 second which is not bad

And my laptop maximum throughput is : Throughput: 239,750 records/sec

Part 3(Exactly-Once Semantics (EOS))

Step 1 defining EOS

In a distributed system, messages can fail. There are usually three ways to handle this:

At-Most-Once: The producer sends a message and doesn't care if it arrives. (Result: Data Loss – bad for banks).

At-Least-Once: The producer sends a message and retries until it gets a "Success." If the network blips, the producer might send the same message twice. (Result: Duplicates – user gets charged twice).

Exactly-Once (EOS): Even if the producer retries 100 times due to network errors, the broker ensures that the message is written to the topic exactly one time.

We want to achieve EOS in our code

Which happens as the producer ends

The producer should find a way to recognise that an error occurred and not to produce the same value twice

Kafka achieves this :

It assigns a Producer ID (PID) and a Sequence Number to every message. If the broker receives the same PID and Sequence Number again, it simply says "I already have this" and discards the duplicate.

Step 2 changing the code to implement EOS

We achieve this by changing the Application.java in bank-api micro service

```
// Inside Banking API - Application.java
public static Producer<String, Transaction>
createKafkaProducer(String bootstrapServers) {
    Properties props = new Properties();

    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
bootstrapServers);

    // --- THE EOS SETTINGS ---
```

```
// 1. Enable Idempotence: This is the core of
EOS.
// It prevents duplicates from the same producer
instance.

props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG,
"true");

// 2. ACKS = ALL: The producer won't consider a
message "sent" until
// BOTH brokers have written it to disk. This
prevents data loss.
props.put(ProducerConfig.ACKS_CONFIG, "all");

// 3. Max In-Flight Requests: Setting this to 5
(or lower) ensures
// that messages stay in the correct order
during retries.

props.put(ProducerConfig.MAX_IN_FLIGHT_REQUESTS_PER_
CONNECTION, "5");
```

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());

props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
Transaction.TransactionSerializer.class.getName());

return new KafkaProducer<>(props);
}
```

Step 3 verifying that EOS worked

```

Thu 19 Mar - 21:53 /opt/kafka
$ bin/kafka-producer-perf-test.sh \
--topic valid-transactions \
--num-records 100000 \
--record-size 100 \
--throughput 1000 \
--producer-props bootstrap.servers=localhost:9092,localhost:9093 acks=all
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 23.5 ms avg latency, 474.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 1.2 ms avg latency, 6.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 1.0 ms avg latency, 3.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 13.0 ms max latency.
5002 records sent, 1000.0 records/sec (0.10 MB/sec), 0.7 ms avg latency, 4.0 ms max latency.
4999 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 16.0 ms max latency.
5002 records sent, 1000.4 records/sec (0.10 MB/sec), 0.6 ms avg latency, 7.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.5 ms avg latency, 3.0 ms max latency.
5001 records sent, 1000.0 records/sec (0.10 MB/sec), 0.8 ms avg latency, 25.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.6 ms avg latency, 3.0 ms max latency.
5001 records sent, 1000.2 records/sec (0.10 MB/sec), 0.6 ms avg latency, 7.0 ms max latency.
5000 records sent, 999.8 records/sec (0.10 MB/sec), 0.7 ms avg latency, 11.0 ms max latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 0.7 ms avg latency, 16.0 ms max latency.
5001 records sent, 1000.0 records/sec (0.10 MB/sec), 0.6 ms avg latency, 5.0 ms max latency.
5003 records sent, 1000.4 records/sec (0.10 MB/sec), 1.2 ms avg latency, 24.0 ms max latency.
5001 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 13.0 ms max latency.
5002 records sent, 1000.2 records/sec (0.10 MB/sec), 0.6 ms avg latency, 2.0 ms max latency.
4999 records sent, 999.8 records/sec (0.10 MB/sec), 0.6 ms avg latency, 8.0 ms max latency.
5002 records sent, 1000.4 records/sec (0.10 MB/sec), 0.6 ms avg latency, 2.0 ms max latency.
[2026-03-19 19:56:15,209] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46087 on topic-partition valid-transactions-1, retrying (2147483646 a
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,213] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common.
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this e
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,221] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46088 on topic-partition valid-transactions-1, retrying (2147483646 a
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,213] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common.
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this e
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,255] WARN [Producer clientId=perf-producer-client] Got error produce response with correlation id 46090 on topic-partition valid-transactions-1, retrying (2147483645 a
tempts left). Error: NOT_LEADER_OR_FOLLOWER (org.apache.kafka.clients.producer.internals.Sender)
[2026-03-19 19:56:15,257] WARN [Producer clientId=perf-producer-client] Received invalid metadata error in produce request on partition valid-transactions-1 due to org.apache.kafka.common.
errors.NotLeaderOrFollowerException: For requests intended only for the leader, this error indicates that the broker is not the current leader. For requests intended for any replica, this e
ror indicates that the broker is not a replica of the topic partition.. Going to request metadata update now (org.apache.kafka.clients.producer.internals.Sender)
100000 records sent, 999.940004 records/sec (0.10 MB/sec), 3.55 ms avg latency, 1151.00 ms max latency, 1 ms 50th, 2 ms 95th, 18 ms 99th, 1057 ms 99.9th.
Thu 19 Mar - 21:56 /opt/kafka

```

Looking at the first screenshot

WARN ... retrying (2147483646 attempts left). Error:
NOT_LEADER_OR_FOLLOWER.

During that test, the connection to the broker was interrupted. The producer retried sending those messages.

At the end of the test, the total count was exactly 100000. If EOS wasn't working, that number would have been higher (e.g., 100005) because of the retries. The fact that the final count matched the input exactly proves Kafka discarded the duplicates during the retries.

Part 4 (لضمان الترتيب داخل بالنسبة Key-Based Routing واعتماد (Partitioning) تقسيم - لنفس المستخدم)

Step 1: Definition of the key-based routing

Why do we need it ?

Imagine a user, Alice, makes two transactions:

Transaction #1: Deposits \$100.

Transaction #2: Withdraws \$50.

If Transaction #2 is processed before #1 because they were in different lines, the bank might reject the withdrawal because of "insufficient funds."

The Goal: We want to make sure all of Alice's transactions stay in the exact same line so they are processed in the correct order.

Step 2 : implementing the key-based routing

In bank-api-service project, inside Application.java, look at the producer.send() line. This is where the magic happens:

```
// Logic: new ProducerRecord<>(TOPIC, KEY, VALUE)
// We pass 'transaction.getUser()' as the KEY.
ProducerRecord<String, Transaction> record =
    new ProducerRecord<>(targetTopic,
        transaction.getUser(), transaction);
```

```
producer.send(record);
```

How it works internally:

Kafka takes the String "john1967", turns it into a number (a Hash), and does some math:

$\text{Hash}(\text{"john1967"}) \% 2 \text{ Partitions} = \text{Partition } 0.$

Every time "john1967" comes in, the math result is always 0.

Therefore, he is pinned to Partition 0 forever.

Step 3 verifying that key-based routing works:

