cd /opt/kafka

sudo bin/zookeeper-server-start.sh config/zookeeper.properties

cd /opt/kafka

sudo bin/kafka-server-start.sh config/server.properties

sudo netstat -tulnp | grep 2181 # Check Zookeeper

sudo netstat -tulnp | grep 9092 # Check Kafka

spark-shell

pyspark

python3 --version

netstat -tulnp | grep 27017

sudo apt update

sudo apt install mongodb-clients -- to check

mongo sh

kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic transactions –from-beginning

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.0,org.mongodb.spark:mongo-spark-connector\_2.12:10.2.0 ~/spark\_consumer.py

from pyspark.sql.functions import from\_json, col

from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType

# Define the schema for the JSON messages

schema = StructType([

StructField("transaction\_id", IntegerType(), True),

StructField("user\_id", IntegerType(), True),

StructField("amount", DoubleType(), True),

StructField("location", StringType(), True),

StructField("merchant", StringType(), True),

StructField("timestamp", StringType(), True)

])

# Read from Kafka with startingOffsets set to 'earliest'

df = spark.readStream \

.format("kafka") \

.option("kafka.bootstrap.servers", "localhost:9092") \

.option("subscribe", "transactions") \

.option("startingOffsets", "earliest") \

.load()

# Parse the JSON messages from the Kafka "value" field

parsed = df.selectExpr("CAST(value AS STRING)") \

.select(from\_json(col("value"), schema).alias("data")) \

.select("data.\*")

# Start a streaming query to print all received transactions to the console

query = parsed.writeStream \

.outputMode("append") \

.format("console") \

.start()

# Wait for the streaming query to terminate (this will block and run continuously)

query.awaitTermination()

here are the commands:

zookeeper-server-start.sh config/zookeeper.properties

kafka-server-start.sh config/server.properties

kafka-topics.sh --create --topic payments --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

kafka-topics.sh --list --bootstrap-server localhost:9092

sudo systemctl start mongod

systemctl status mongod

mongosh  
  
use fraud\_detection  
  
db.transactions.insertOne({ "transaction\_id": "12345", "amount": 100, "status": "pending" })  
  
db.transactions.find().pretty()

kafka-topics.sh --create --topic payments --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1  
  
kafka-topics.sh --list --bootstrap-server localhost:9092  
  
kafka-console-producer.sh --broker-list localhost:9092 --topic payments  
  
  
  
consumer code:

from kafka import KafkaConsumer

import json

# Initialize Kafka consumer

consumer = KafkaConsumer(

'payments',

bootstrap\_servers='localhost:9092',

value\_deserializer=lambda v: json.loads(v.decode('utf-8'))

)

print("Listening for transactions...\n")

def is\_fraudulent(transaction):

"""Detect fraudulent transactions based on simple rules."""

return transaction["amount"] > 3000 # Example: Flag transactions above $3000 as fraudulent

for message in consumer:

transaction = message.value

if is\_fraudulent(transaction):

print(f"🚨 FRAUD ALERT! 🚨 Transaction {transaction['transaction\_id']} of ${transaction['amount']} is suspicious! ⚠️")

else:

print(f"✅ Transaction {transaction['transaction\_id']} of ${transaction['amount']} is normal.")

consumer code::

from kafka import KafkaConsumer

import json

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import numpy as np

# Initialize Kafka consumer

consumer = KafkaConsumer(

'payments',

bootstrap\_servers='localhost:9092',

value\_deserializer=lambda v: json.loads(v.decode('utf-8'))

)

print("Listening for transactions...\n")

# Dictionaries to track transaction counts

fraud\_transactions = {}

normal\_transactions = {}

def is\_fraudulent(transaction):

"""Detect fraudulent transactions based on simple rules."""

return transaction["amount"] > 3000 # Flag transactions above $3000 as fraudulent

def update\_graph(i):

"""Update the bar graph with fraud and normal transactions."""

while True:

message = next(consumer)

transaction = message.value

user\_id = transaction.get("user\_id", "Unknown")

if is\_fraudulent(transaction):

print(f"🚨 FRAUD ALERT! 🚨 Transaction {transaction['transaction\_id']} of ${transaction['amount']} detected! ⚠️")

fraud\_transactions[user\_id] = fraud\_transactions.get(user\_id, 0) + 1

else:

print(f"✅ NORMAL TRANSACTION: {transaction['transaction\_id']} of ${transaction['amount']}")

normal\_transactions[user\_id] = normal\_transactions.get(user\_id, 0) + 1

break

# Get unique user IDs

users = list(set(fraud\_transactions.keys()) | set(normal\_transactions.keys()))

fraud\_counts = [fraud\_transactions.get(user, 0) for user in users]

normal\_counts = [normal\_transactions.get(user, 0) for user in users]

# Bar positions

x = np.arange(len(users))

# Clear and replot

plt.cla()

plt.bar(x - 0.2, normal\_counts, width=0.4, color='blue', label="Normal Transactions")

plt.bar(x + 0.2, fraud\_counts, width=0.4, color='red', label="Fraud Transactions")

plt.xlabel("User ID")

plt.ylabel("Transaction Count")

plt.title("Real-Time Fraud & Normal Transactions per User")

plt.xticks(x, users, rotation=45)

plt.legend()

plt.grid(axis='y')

# Set up real-time bar graph

fig = plt.figure()

ani = animation.FuncAnimation(fig, update\_graph, interval=2000)

plt.show()  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
from kafka import KafkaConsumer

import json

import tkinter as tk

from tkinter import scrolledtext

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.animation as animation

import numpy as np

# Initialize Kafka consumer

consumer = KafkaConsumer(

'payments',

bootstrap\_servers='localhost:9092',

value\_deserializer=lambda v: json.loads(v.decode('utf-8'))

)

print("Listening for transactions...\n")

# Dictionaries to track transactions

fraud\_transactions = {}

normal\_transactions = {}

def is\_fraudulent(transaction):

"""Detect fraudulent transactions based on simple rules."""

return transaction["amount"] > 3000 # Fraud if amount > 3000

def update\_graph(i):

"""Update UI and bar graph."""

while True:

message = next(consumer)

transaction = message.value

user\_id = transaction.get("user\_id", "Unknown")

if is\_fraudulent(transaction):

fraud\_transactions[user\_id] = fraud\_transactions.get(user\_id, 0) + 1

log\_transaction(f"🚨 FRAUD: {transaction['transaction\_id']} - ${transaction['amount']} ⚠️", "red")

else:

normal\_transactions[user\_id] = normal\_transactions.get(user\_id, 0) + 1

log\_transaction(f"✅ NORMAL: {transaction['transaction\_id']} - ${transaction['amount']}", "blue")

break

# Get unique users

users = list(set(fraud\_transactions.keys()) | set(normal\_transactions.keys()))

fraud\_counts = [fraud\_transactions.get(user, 0) for user in users]

normal\_counts = [normal\_transactions.get(user, 0) for user in users]

x = np.arange(len(users))

# Clear and replot

ax.clear()

ax.bar(x - 0.2, normal\_counts, width=0.4, color='blue', label="Normal Transactions")

ax.bar(x + 0.2, fraud\_counts, width=0.4, color='red', label="Fraud Transactions")

ax.set\_xlabel("User ID")

ax.set\_ylabel("Transaction Count")

ax.set\_title("Real-Time Fraud & Normal Transactions")

ax.set\_xticks(x)

ax.set\_xticklabels(users, rotation=45)

ax.legend()

ax.grid(axis='y')

def log\_transaction(message, color):

"""Log transaction messages in the UI."""

transaction\_log.config(state=tk.NORMAL)

transaction\_log.insert(tk.END, message + "\n", color)

transaction\_log.tag\_config(color, foreground=color)

transaction\_log.config(state=tk.DISABLED)

transaction\_log.yview(tk.END)

# Create UI Window

root = tk.Tk()

root.title("Fraud Detection System")

root.geometry("800x600")

# Transaction Log UI

transaction\_log = scrolledtext.ScrolledText(root, width=80, height=10, state=tk.DISABLED)

transaction\_log.pack(pady=10)

# Matplotlib Figure for Bar Graph

fig, ax = plt.subplots(figsize=(6, 4))

canvas = FigureCanvasTkAgg(fig, master=root)

canvas.get\_tk\_widget().pack()

# Animate the Graph

ani = animation.FuncAnimation(fig, update\_graph, interval=2000)

# Run Tkinter Main Loop

root.mainloop()