Capstone Project: Instant Health Alert System - Mid-Project Submission

Introduction:

This document outlines the code implementation for the Instant Health Alert System's mid-project submission. The system fetches patients' vital information from a MySQL database and streams it to a Kafka topic. This setup enables real-time data analysis and alerts for healthcare monitoring. This explains the file **kafka_produce_patient_vitals.py**

1. Database Connection parameters and producing the data in JSON format

```
import mysql.connector
from mysql.connector import Error
from kafka import KafkaProducer
import json
import time

HOST = 'upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com'
DATABASE = 'testdatabase'
USER = 'student'
PASSWORD = 'STUDENT123'
TOPIC='patients_vital_info'
KAFKA_BOOTSTRAP_SERVERS='localhost:9092'
producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
value_serializer=lambda x: json.dumps(x).encode('utf-8'))
```

2. Database Connection Check

This function, `check_connection`, tries to establish a connection to the MySQL database. If successful, it confirms connectivity. Otherwise, it prints an error.

```
def check_connection(host, database, user, password):
    try:
        connection = mysql.connector.connect(
            host=host,
            database=database,
            user=user,
            password=password
        )
        if connection.is_connected():
            print("Connected to MySQL database")
            return connection
    except Error as e:
        print(f"Error while connecting to MySQL: {e}")
    return None
```

3. Data Fetching Function

The `fetch_data` function retrieves all records from the `patients_vital_info` table. If the connection is successful, it fetches and returns the data; otherwise, it handles the error and closes the connection.

```
def fetch_data(host, database, user, password):
    connection = check_connection(host, database, user, password)
    if connection is None:
        print("Failed to connect to the database.")
        return None

    try:
        cursor = connection.cursor()
        cursor.execute('SELECT * FROM patients_vital_info')
        result = cursor.fetchall()
        return result
    except Error as e:
        print(f"Error while fetching data: {e}")
        return None
    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()
```

4. Kafka Producer Function

The `produce_to_kafka` function iterates over the fetched data records. Each record is serialized and sent to the Kafka topic. The function includes error handling and rate control to prevent overload.

```
def produce_to_kafka(data, topic):
    if not data:
        print("No data to produce.")
        return
    try:
        for row in data:
            data_dict = {
                'customerId': row[0],
                'heartBeat': row[1],
                'bp': row[2],
            }
            print(data_dict)
            producer.send(topic, data_dict)
            time.sleep(1)
        producer.flush()
```

5. Main Execution

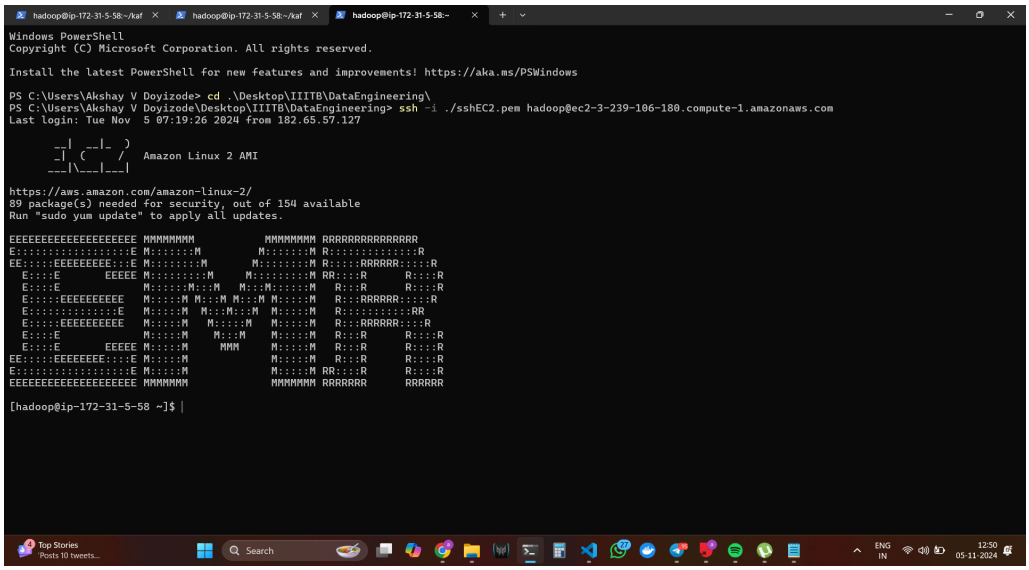
The main section initiates the process by fetching data and passing it to the `produce_to_kafka` function to stream the information to the Kafka topic

```
if __name__ == "__main__":
    data = fetch_data(HOST, DATABASE, USER, PASSWORD)
    produce_to_kafka(data, TOPIC)
```

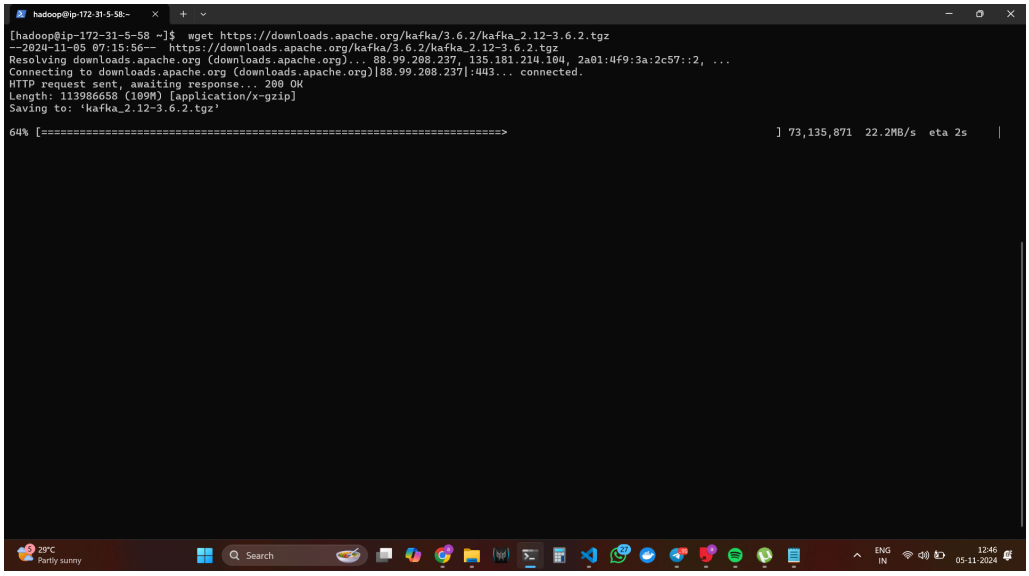
6. Setting Up Apache Kafka on an EMR Cluster

This guide provides step-by-step instructions to download, install, and configure Apache Kafka on an AWS EMR cluster. Additionally, it includes steps to create Kafka topics and install required Python libraries

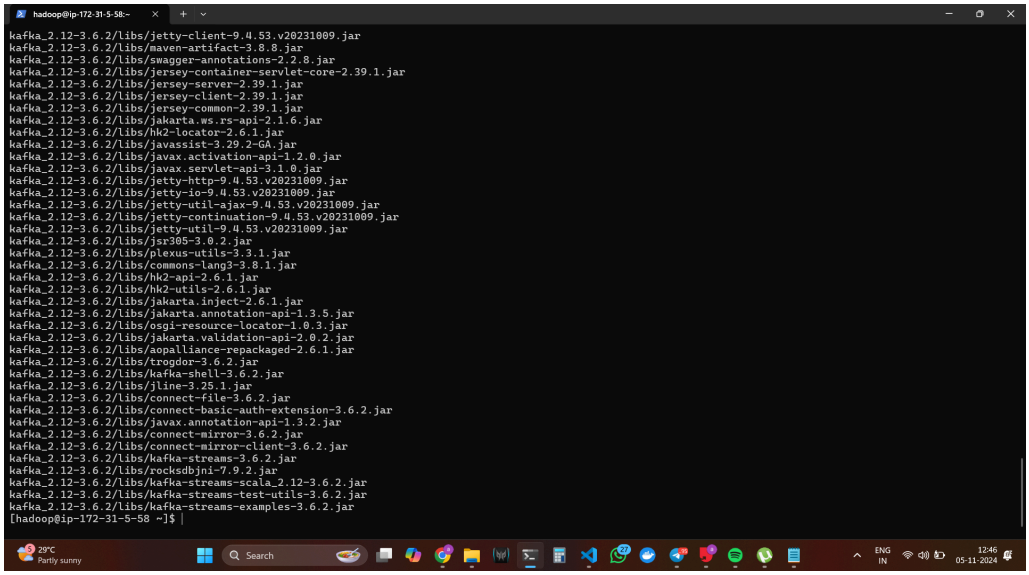
6.1 Connect to your EMR cluster via SSH



6.2 Download Kafka using the wget command



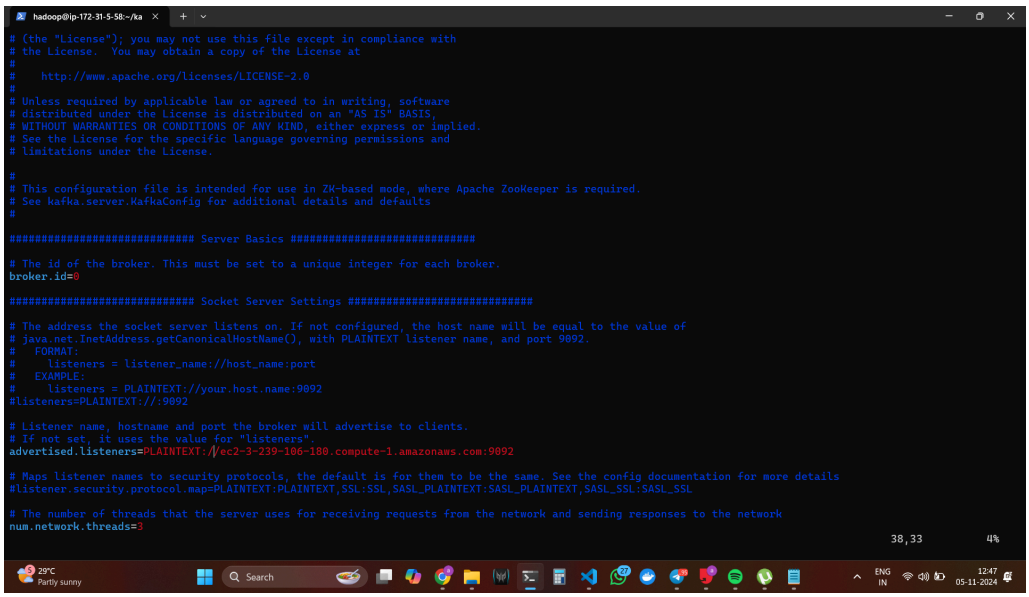
6.3 : Extract the Kafka Files



6.4 : Configure Kafka Server

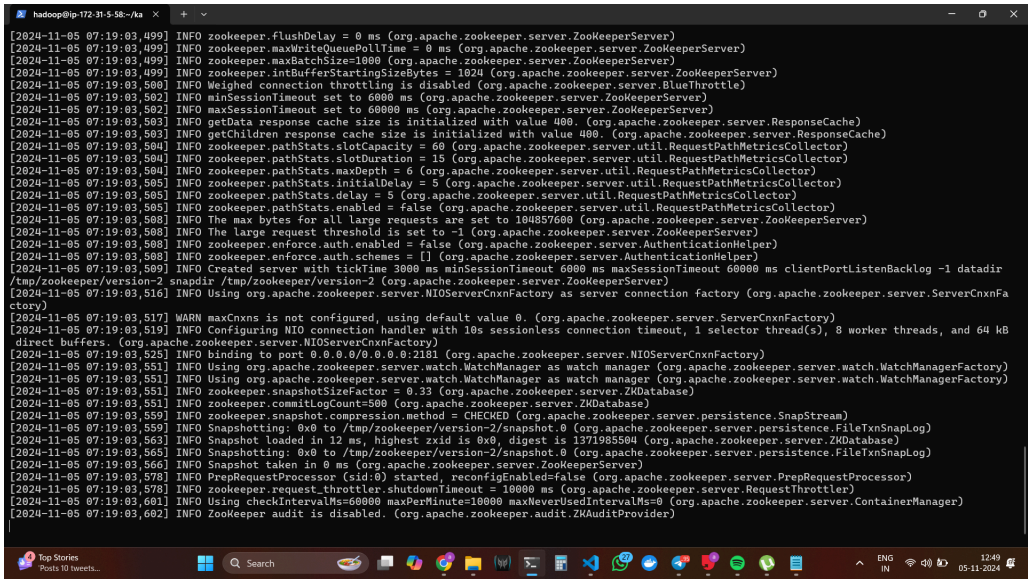
Update the advertised listeners to use our EMR IP address

vi config/server.properties



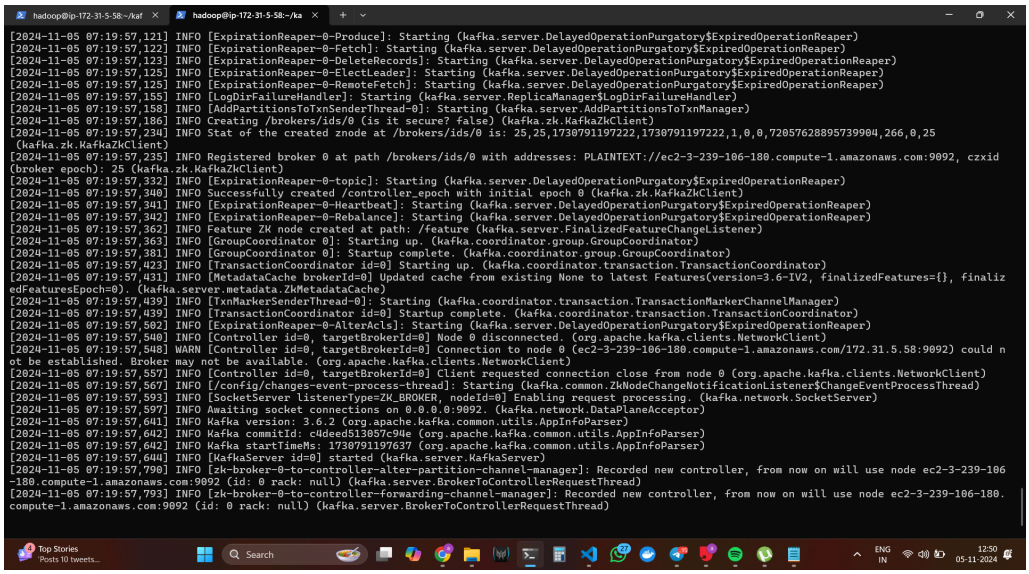
6.5 Start the Zookeeper

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



6.6 Start the server

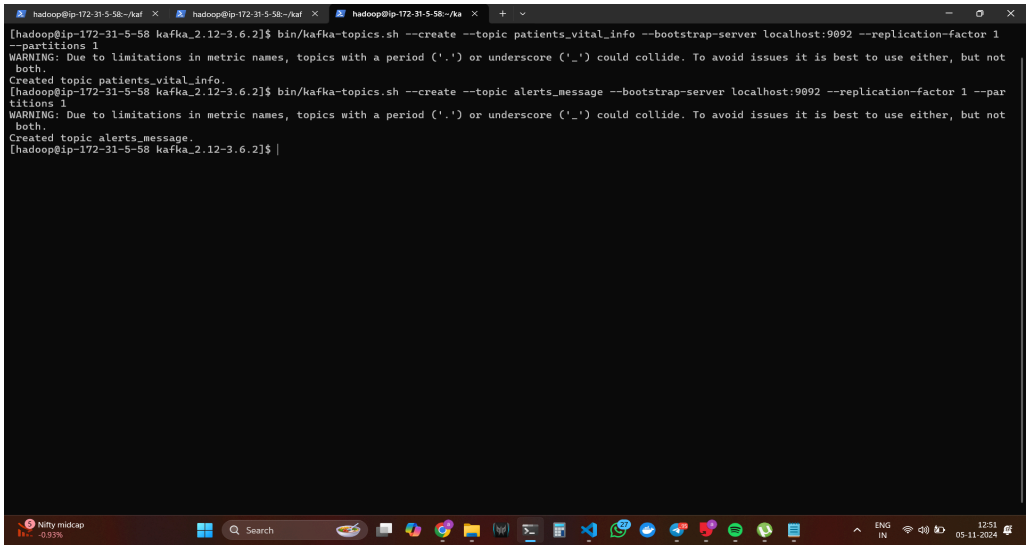
bin/kafka-server-start.sh config/server.properties



6.7 Create Kafka Topics

bin/kafka-topics.sh --create --topic patients_vital_info --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1

bin/kafka-topics.sh --create --topic alerts_message --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1



7. Install the required dependencies

pip install mysql-connector-python kafka-python

```
pip install mysql-connector-python
pip install kafka-python
```

7.1. Run the python producer script

python kafka_produce_patient_vitals.py

This python script reads data from RDS per second and simulates IoT devices to push patient vital data in the JSON format into the Kafka Queue.

Screenshot of JSON vital information written to the console (per second)

```
Connected to MySQL database
{'customerId': 1, 'heartBeat': 74, 'bp': 202}
{'customerId': 2, 'heartBeat': 68, 'bp': 173}
{'customerId': 3, 'heartBeat': 71, 'bp': 152}
{'customerId': 4, 'heartBeat': 72, 'bp': 166}
{'customerId': 5, 'heartBeat': 68, 'bp': 171}
{'customerId': 1, 'heartBeat': 70, 'bp': 189}
{'customerId': 2, 'heartBeat': 72, 'bp': 173}
{'customerId': 3, 'heartBeat': 68, 'bp': 178}
{'customerId': 4, 'heartBeat': 71, 'bp': 152}
{'customerId': 5, 'heartBeat': 73, 'bp': 166}
{'customerId': 1, 'heartBeat': 74, 'bp': 185}
{'customerId': 2, 'heartBeat': 67, 'bp': 177}
{'customerId': 3, 'heartBeat': 66, 'bp': 158}
{'customerId': 4, 'heartBeat': 71, 'bp': 177}
{'customerId': 5, 'heartBeat': 66, 'bp': 155}
{'customerId': 1, 'heartBeat': 71, 'bp': 220}
```