CODE LOGIC

kafka_produce_patient_vitals.py

1. Create the topic if not exists to ensure that topic always exists before sending data to the topic with number of partition equal to 1 and replication factor equal to 1.
2. The admin client of kafka is used to create the topic.
3. Sending or pushing data to the topic.
4. The data is fetched from the provided database and each record is first converted into a dictionary and sent to the topic with 1 second of delay between each record.
5. The producer connection is made and json value serializer is used.
6. The data is sent to the topic in partition number 0.

kafka_spark_patient_vitals.py

1. The schema for data format is defined.
2. The spark session is created.
3. The data is read from kafka stream.
4. The data is read in the format of key value pair.
5. The dataframe is converted from json format to single column field with data.
6. The other fields are added to the dataframe.
7. The timestamp field is added to the dataframe.
8. The data is written to the HDFS in parquet file format with 10 rows and processing time of 10 seconds with data being appended.
9. We wait for the termination of stream.

kafka_spark_generate_alert.py

1. The spark session is created.
2. The hivecontext is created to read data from Hive.
3. A verification is done whether there is a successful connection to the hive by using the command for displaying tables and databases.
4. Data is read in streams from the parquet files stored in HDFS.
5. Data is fetched from the patients_contact_info table in hive.
6. An inner join is done between the dataframes patients_vital_info and patients_contact_info.
7. Three columns are added alert_generated_time, to_be_alerted, message_time to the dataframe.
8. Connection with the hbase is made using happybase library.
9. The table threshold_hive is scanned.
10. The anomalies are checked and the column “to_be_alerted” is set to 1 in case of anomalies detection.

11. The corresponding alert message is also set in the column alert_message field.
12. We filter the rows based on the value of to_be_alerted field.
13. We select the appropriate columns and give appropriate aliases to the fields.
14. We write the data to the kafka topic ‘doctor’.

kafka_consume_alerts.py

1. We fetch all the messages from the kafka ‘doctor’ queue.
2. We use the python kafka-python library.
3. We seek to the beginning of the topic partition.
4. Using the boto3 library we made connection to the AWS and send messages to the consumer.

COMMANDS TO RUN FILE

To run kafka_produce_patient_vitals file -

```
python kafka_produce_patient_vitals.py
```

To run kafka_spark_patient_vitals file -

```
export SPARK_KAFKA_VERSION=0.10
spark-submit  --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 kafka_spark_patient_vitals.py
```

To access Hive -

```
hive
```

To access Hbase shell -

```
hbase shell
```

To run kafka_spark_generate_alerts file -

```
export SPARK_KAFKA_VERSION=0.10
spark-submit --conf spark.sql.catalogImplementation=hive --driver-memory 3G --executor-memory 3G --executor-cores 2 --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 kafka_spark_generate_alerts.py
```