Capstone - Instant Health Alert System :

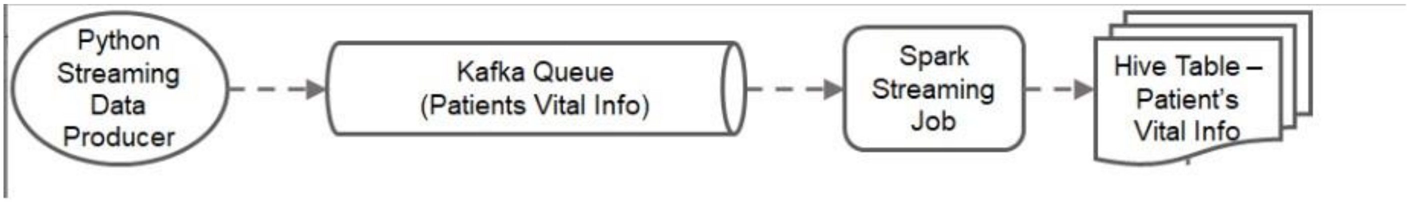
Code Logic :

Here we develop three pipeline for whole project

1. First pipeline for patient vital info (Real-time Pipeline)
2. Second pipeline for contact information (Batch Pipeline)
3. Third pipeline for sending Anamolies to email.

Now here we explain each pipeline one by one :

1. For extracting the patient vital info from RDS to HDFS system:


Python Streaming Data Producer
Kafka Queue (Patients Vital Info)
Spark Streaming Job
Hive Table – Patient's Vital Info

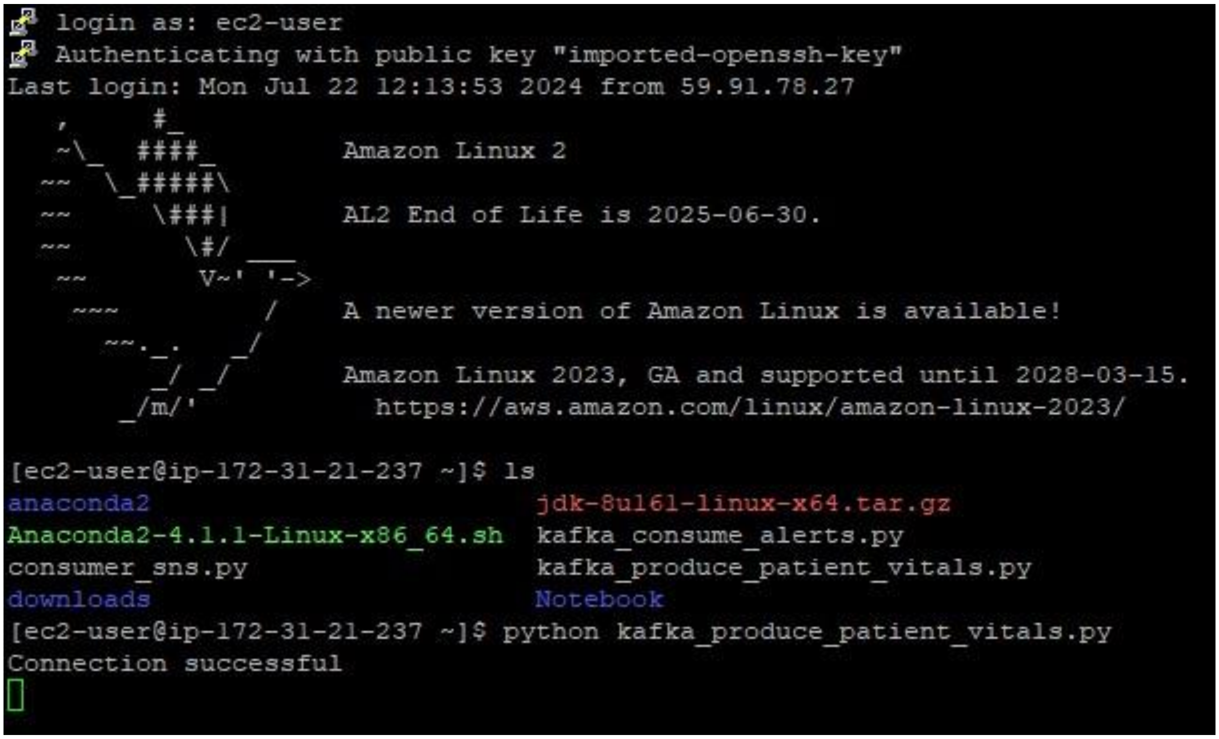
Firstly we start kafka for pulling the data from RDS to Spark Streaming job, then goes to HDFS in parquet format.

A. You can see the command for start/run kafka server, create topic in **kafka.pdf** file in attachment

B. Now we start producer application which having python code to pull the Data from RDS to kafka topic, code present in **kafka_produce_patient_vitals.py**

   Code run by this command:

   ```
   Python kafka_produce_patient_vitals.py
   ```



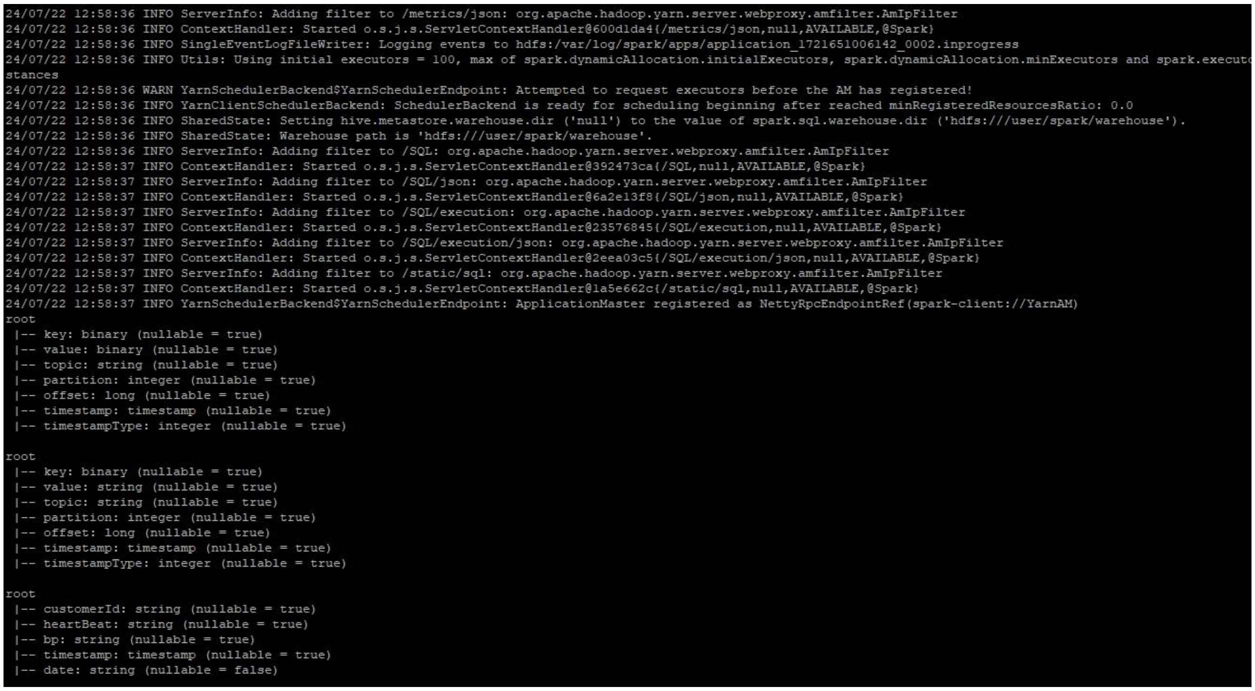
```
login as: ec2-user
Authenticating with public key "imported-openssh-key"
Last login: Mon Jul 22 12:13:53 2024 from 59.91.78.27
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-21-237 ~]$ ls
anaconda2                        jdk-8u161-linux-x64.tar.gz
Anaconda2-4.1.1-Linux-x86_64.sh  kafka_consume_alerts.py
consumer_sns.py                  kafka_produce_patient_vitals.py
downloads                        Notebook
[ec2-user@ip-172-31-21-237 ~]$ python kafka_produce_patient_vitals.py
Connection successful
```

C. After that we start spark streaming application which receive the data from kafka topic(vital-info) and apply transformation over data so that lastly stored the data in parquet file format.
Python file for this step is **kafka_spark_patient_vitals.py** which is also present in attachment

Code run by this command in our EMR :

Spark-submit –package org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 kafka_spark_patient_vitals.py 34.204.124.95 9092 vital-info

Note: In Spark streaming code, we have added a new column using lit('2024-07-21') and while writing the data we have partitioned the data by this newly added constant column, thus all of the parquet files get written inside this folder /home/hadoop/data1/vitals.parquet/date=2024-07-21. This was done because _spark_metadata directory is by default created on the HDFS output directory. In our case, it will get created on path



```
24/07/22 12:58:36 INFO ServerInfo: Adding filter to /metrics/json: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:36 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@600d1da4{/metrics/json,null,AVAILABLE,@Spark}
24/07/22 12:58:36 INFO SingleEventLogFileWriter: Logging events to hdfs:/var/log/spark/apps/application_1721651006142_0002.inprogress
24/07/22 12:58:36 INFO Utils: Using initial executors = 100, max of spark.dynamicAllocation.initialExecutors, spark.dynamicAllocation.minExecutors and spark.executo
stances
24/07/22 12:58:36 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
24/07/22 12:58:36 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
24/07/22 12:58:36 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir ('hdfs:///user/spark/warehouse').
24/07/22 12:58:36 INFO SharedState: Warehouse path is 'hdfs:///user/spark/warehouse'.
24/07/22 12:58:36 INFO ServerInfo: Adding filter to /SQL: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:37 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@392473ca{/SQL,null,AVAILABLE,@Spark}
24/07/22 12:58:37 INFO ServerInfo: Adding filter to /SQL/json: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:37 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@6a2e13f8{/SQL/json,null,AVAILABLE,@Spark}
24/07/22 12:58:37 INFO ServerInfo: Adding filter to /SQL/execution: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:37 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@23576845{/SQL/execution,null,AVAILABLE,@Spark}
24/07/22 12:58:37 INFO ServerInfo: Adding filter to /SQL/execution/json: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:37 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@2eea03c5{/SQL/execution/json,null,AVAILABLE,@Spark}
24/07/22 12:58:37 INFO ServerInfo: Adding filter to /static/sql: org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter
24/07/22 12:58:37 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@1a5e662c{/static/sql,null,AVAILABLE,@Spark}
24/07/22 12:58:37 INFO YarnSchedulerBackend$YarnSchedulerEndpoint: ApplicationMaster registered as NettyRpcEndpointRef(spark-client://YarnAM)
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)

root
 |-- key: binary (nullable = true)
 |-- value: string (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)

root
 |-- customerId: string (nullable = true)
 |-- heartBeat: string (nullable = true)
 |-- bp: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- date: string (nullable = false)
```

/home/hadoop/data1/vitals.parquet, and due to partition by functionality, another folder is created named ‘2024-07-21’ inside which the parquet files are present.

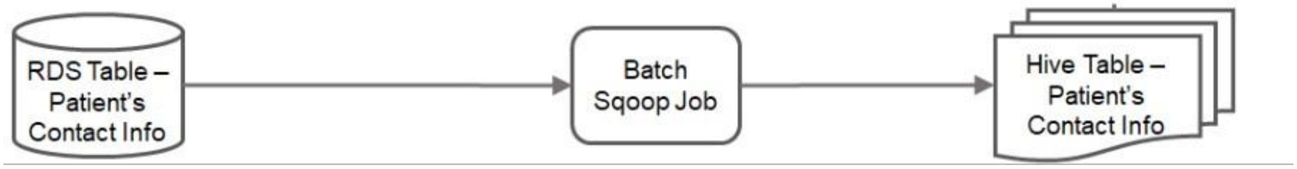
D. Now here we build hive external table over that parquet format data present in **vital.parquet** directory

   Script for this present in **hive2.pdf** file which is also present in attachment.

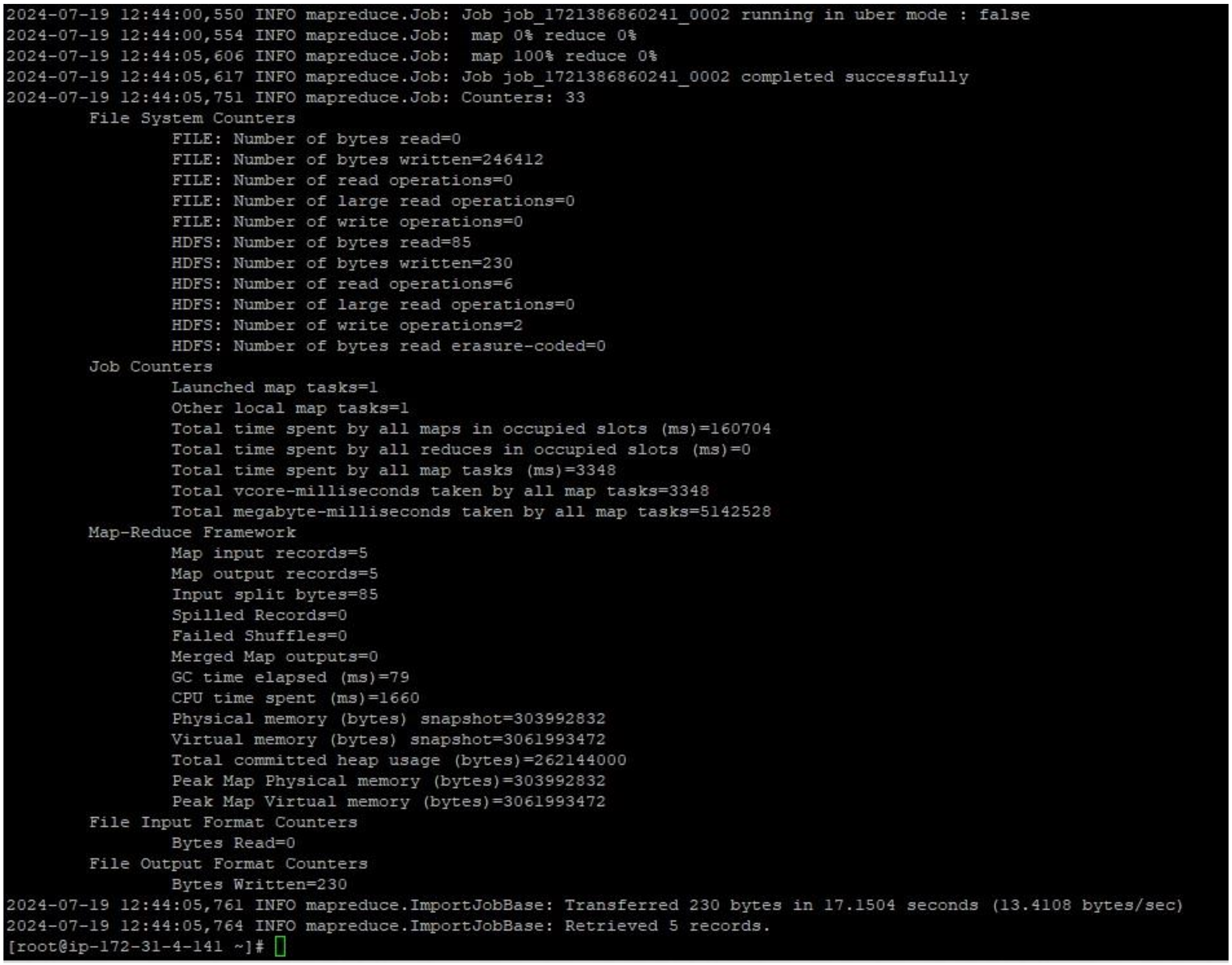
Here we completed our first pipeline which is real-time-data pipeline

2. Extracting the data of patient contact info from RDS to HDFS system by using sqoop

   Here we use sqoop job for extracting the data from RDS(patient_information)

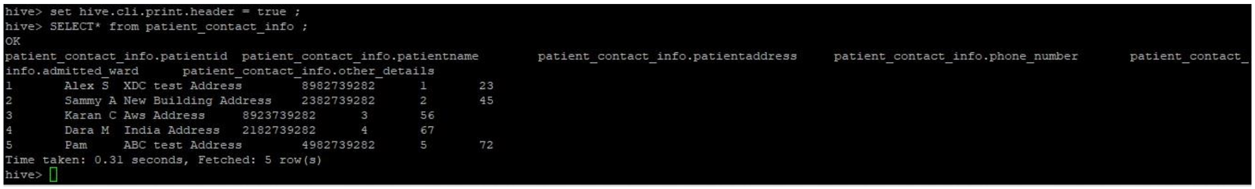

RDS Table – Patient's Contact Info
Batch Sqoop Job
Hive Table – Patient's Contact Info

Script for sqoop job and how to make hive table over it is in **sqoop.pdf** which is also present in attachment.



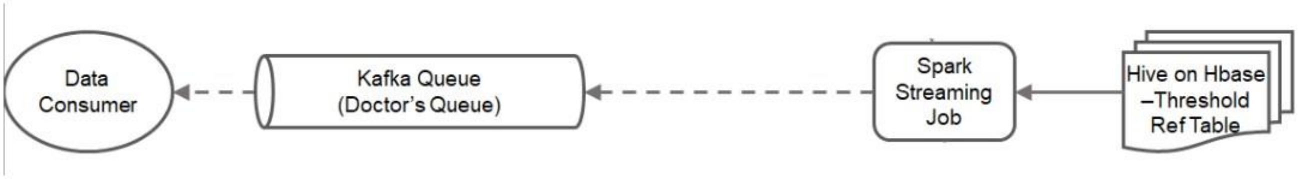
```
2024-07-19 12:44:00,550 INFO mapreduce.Job: Job job_1721386860241_0002 running in uber mode : false
2024-07-19 12:44:00,554 INFO mapreduce.Job:  map 0% reduce 0%
2024-07-19 12:44:05,606 INFO mapreduce.Job:  map 100% reduce 0%
2024-07-19 12:44:05,617 INFO mapreduce.Job: Job job_1721386860241_0002 completed successfully
2024-07-19 12:44:05,751 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=246412
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=85
                HDFS: Number of bytes written=230
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=160704
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=3348
                Total vcore-milliseconds taken by all map tasks=3348
                Total megabyte-milliseconds taken by all map tasks=5142528
        Map-Reduce Framework
                Map input records=5
                Map output records=5
                Input split bytes=85
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=79
                CPU time spent (ms)=1660
                Physical memory (bytes) snapshot=303992832
                Virtual memory (bytes) snapshot=3061993472
                Total committed heap usage (bytes)=262144000
                Peak Map Physical memory (bytes)=303992832
                Peak Map Virtual memory (bytes)=3061993472
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=230
2024-07-19 12:44:05,761 INFO mapreduce.ImportJobBase: Transferred 230 bytes in 17.1504 seconds (13.4108 bytes/sec)
2024-07-19 12:44:05,764 INFO mapreduce.ImportJobBase: Retrieved 5 records.
[root@ip-172-31-4-141 ~]# 
```

And after that build the hive table over that data present in patient information



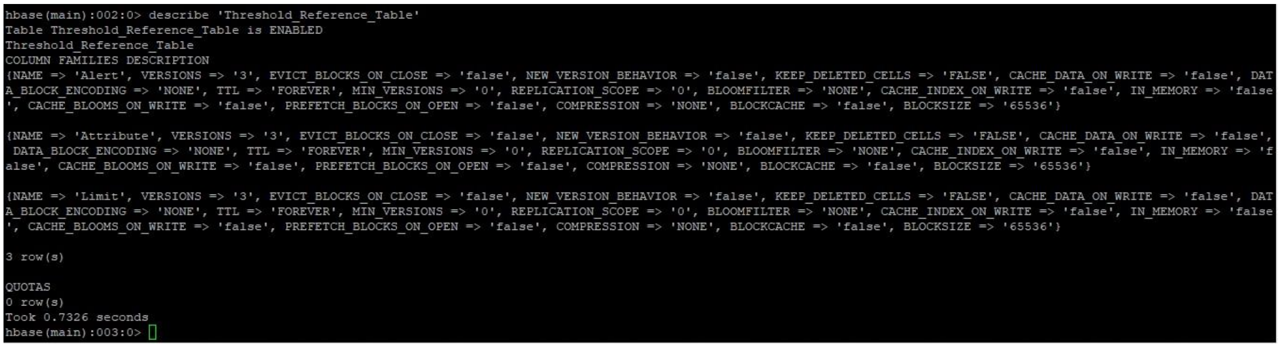
```
hive> set hive.cli.print.header = true ;
hive> SELECT* from patient_contact_info ;
OK
patient_contact_info.patientid  patient_contact_info.patientname        patient_contact_info.patientaddress     patient_contact_info.phone_number       patient_contact_
info.admitted_ward      patient_contact_info.other_details
1       Alex S  XDC test Address        8982739282      1       23
2       Sammy A New Building Address    2382739282      2       45
3       Karan C Aws Address     8923739282      3       56
4       Dara M  India Address   2182739282      4       67
5       Pam     ABC test Address        4982739282      5       72
Time taken: 0.31 seconds, Fetched: 5 row(s)
hive> 
```

3. Third pipeline for sending Alerts to email.

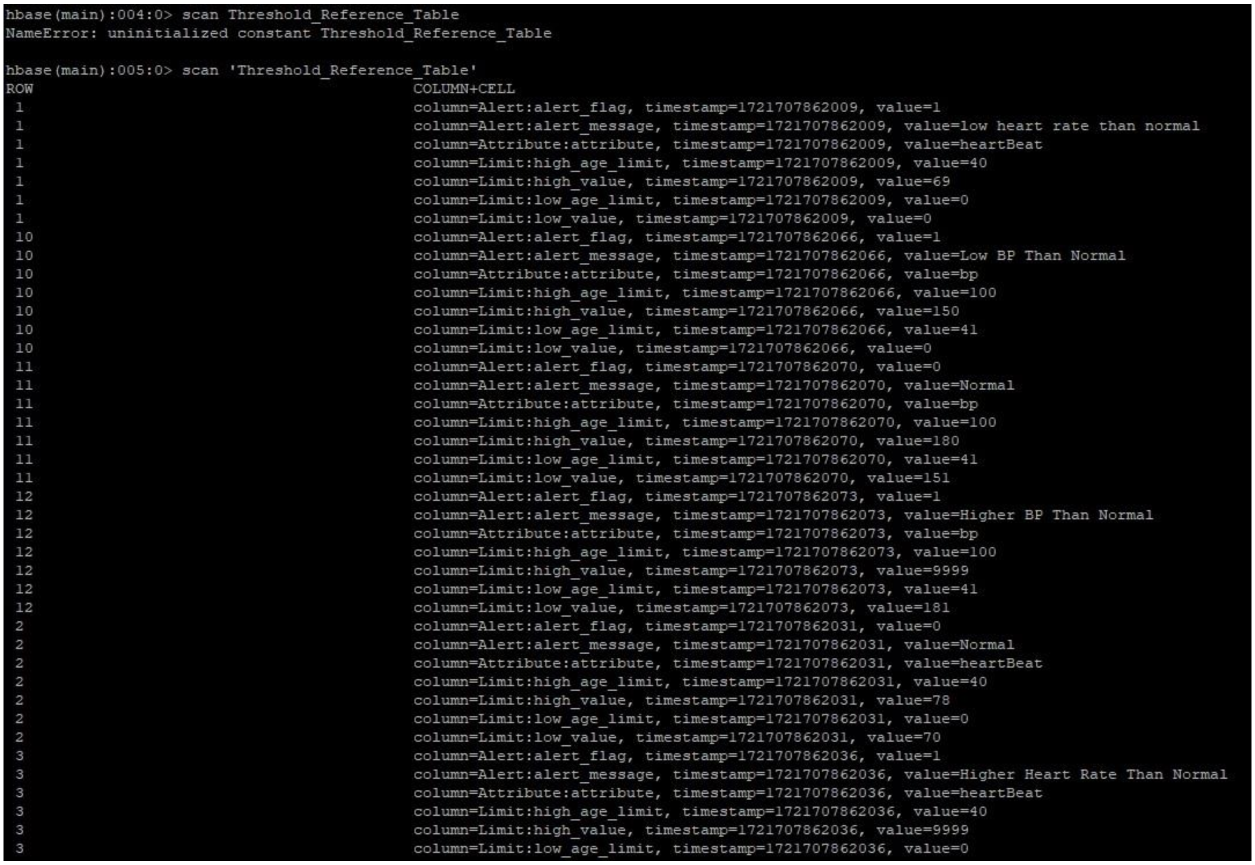

Data Consumer
Kafka Queue (Doctor's Queue)
Spark Streaming Job
Hive on Hbase –Threshold Ref Table

A. In This Step first we stored data of Threshold Reference data into Hbase table and build hive table over it.

Script for building hbase table present in **hbase.pdf** and script for building hive over that hbase data present in **hive1.pdf**, both of the files present in attachment



```
hbase(main):002:0> describe 'Threshold_Reference_Table'
Table Threshold_Reference_Table is ENABLED
Threshold_Reference_Table
COLUMN FAMILIES DESCRIPTION
{NAME => 'Alert', VERSIONS => '3', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DAT
A_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'NONE', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'false
', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'false', BLOCKSIZE => '65536'}

{NAME => 'Attribute', VERSIONS => '3', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false',
 DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'NONE', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'f
alse', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'false', BLOCKSIZE => '65536'}

{NAME => 'Limit', VERSIONS => '3', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR => 'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DAT
A_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER => 'NONE', CACHE_INDEX_ON_WRITE => 'false', IN_MEMORY => 'false
', CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE', BLOCKCACHE => 'false', BLOCKSIZE => '65536'}

3 row(s)

QUOTAS
0 row(s)
Took 0.7326 seconds
hbase(main):003:0> 
```



```
hbase(main):004:0> scan Threshold_Reference_Table
NameError: uninitialized constant Threshold_Reference_Table

hbase(main):005:0> scan 'Threshold_Reference_Table'
ROW                                          COLUMN+CELL
 1                                           column=Alert:alert_flag, timestamp=1721707862009, value=1
 1                                           column=Alert:alert_message, timestamp=1721707862009, value=low heart rate than normal
 1                                           column=Attribute:attribute, timestamp=1721707862009, value=heartBeat
 1                                           column=Limit:high_age_limit, timestamp=1721707862009, value=40
 1                                           column=Limit:high_value, timestamp=1721707862009, value=69
 1                                           column=Limit:low_age_limit, timestamp=1721707862009, value=0
 1                                           column=Limit:low_value, timestamp=1721707862009, value=0
 10                                          column=Alert:alert_flag, timestamp=1721707862066, value=1
 10                                          column=Alert:alert_message, timestamp=1721707862066, value=Low BP Than Normal
 10                                          column=Attribute:attribute, timestamp=1721707862066, value=bp
 10                                          column=Limit:high_age_limit, timestamp=1721707862066, value=100
 10                                          column=Limit:high_value, timestamp=1721707862066, value=150
 10                                          column=Limit:low_age_limit, timestamp=1721707862066, value=41
 10                                          column=Limit:low_value, timestamp=1721707862066, value=0
 11                                          column=Alert:alert_flag, timestamp=1721707862070, value=0
 11                                          column=Alert:alert_message, timestamp=1721707862070, value=Normal
 11                                          column=Attribute:attribute, timestamp=1721707862070, value=bp
 11                                          column=Limit:high_age_limit, timestamp=1721707862070, value=100
 11                                          column=Limit:high_value, timestamp=1721707862070, value=180
 11                                          column=Limit:low_age_limit, timestamp=1721707862070, value=41
 11                                          column=Limit:low_value, timestamp=1721707862070, value=151
 12                                          column=Alert:alert_flag, timestamp=1721707862073, value=1
 12                                          column=Alert:alert_message, timestamp=1721707862073, value=Higher BP Than Normal
 12                                          column=Attribute:attribute, timestamp=1721707862073, value=bp
 12                                          column=Limit:high_age_limit, timestamp=1721707862073, value=100
 12                                          column=Limit:high_value, timestamp=1721707862073, value=9999
 12                                          column=Limit:low_age_limit, timestamp=1721707862073, value=41
 12                                          column=Limit:low_value, timestamp=1721707862073, value=181
 2                                           column=Alert:alert_flag, timestamp=1721707862031, value=0
 2                                           column=Alert:alert_message, timestamp=1721707862031, value=Normal
 2                                           column=Attribute:attribute, timestamp=1721707862031, value=heartBeat
 2                                           column=Limit:high_age_limit, timestamp=1721707862031, value=40
 2                                           column=Limit:high_value, timestamp=1721707862031, value=78
 2                                           column=Limit:low_age_limit, timestamp=1721707862031, value=0
 2                                           column=Limit:low_value, timestamp=1721707862031, value=70
 3                                           column=Alert:alert_flag, timestamp=1721707862036, value=1
 3                                           column=Alert:alert_message, timestamp=1721707862036, value=Higher Heart Rate Than Normal
 3                                           column=Attribute:attribute, timestamp=1721707862036, value=heartBeat
 3                                           column=Limit:high_age_limit, timestamp=1721707862036, value=40
 3                                           column=Limit:high_value, timestamp=1721707862036, value=9999
 3                                           column=Limit:low_age_limit, timestamp=1721707862036, value=0
```

B. Now after above these steps we get three tables patient_vital_info, patient_contact_info, Threshold_Reference_Table, over these hive tables we wants to build spark streaming application which takes data from these tables and detect the anamolies which is values below or above the low_limit and high_limit in Threshold_Reference_Table. Logic and code for this present in

kafka_spark_generate_alerts.py file which is also present in attachment provided by me

Code run by this command in our EMR :

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 --jars /home/hadoop/hbase/hbase-client-1.4.13.jar,

```
/home/hadoop/hbase/hbase-common-1.4.13.jar,
/home/hadoop/hbase/hbase-server-1.4.13.jar,
/home/hadoop/hbase/hbase-hadoop-compat-1.4.13.jar,
/home/hadoop/hbase/hadoop-common-2.7.4.jar,
/home/hadoop/hbase/hive-hbase-handler-3.1.2.jar,
/home/hadoop/hbase/hbase-mapreduce-2.4.10.jar,
/home/hadoop/hbase/hadoop-mapreduce-client-core-2.7.4.jar,
/home/hadoop/hbase/guava-12.0.1.jar
kafka_spark_generate_alerts.py
```

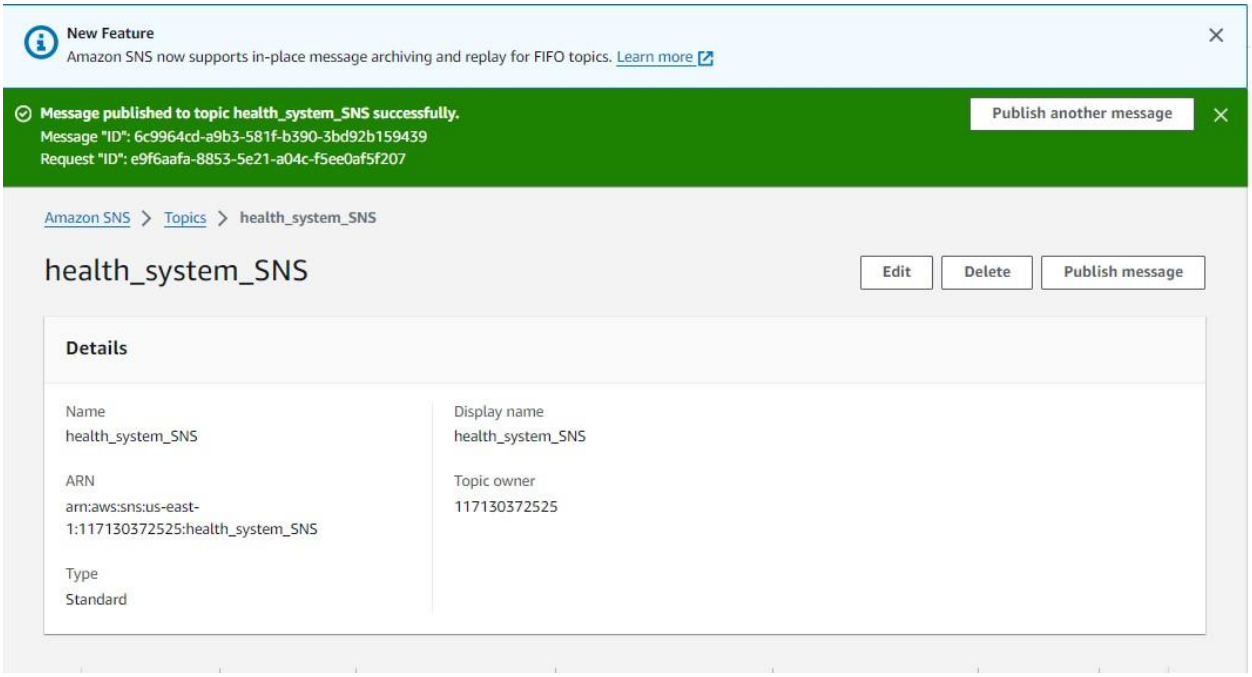
C. After generating the alert from spark streaming job, then this goes to another kafka topic (vital-alerts) from that topic a kafka consumer aplication (**kafka_consume_alerts.py**) takes the data and send to send SNS notification to the registered email address by using AWS SNS.

Both of the files i.e kafka_consume_alerts.py and sns.pdf are present in attachment

To run the the python file :

Python **kafka_consume_alerts.py** (Use this command)

D. At the end we create SNS topic with name health_system_sns and subscribe our email to this topic for receiving the notification of alerts.



New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics.

Message published to topic health_system_SNS successfully.
Message "ID": 6c9964cd-a9b3-581f-b390-3bd92b159439
Request "ID": e9f6aafa-8853-5e21-a04c-f5ee0af5f207

health_system_SNS

Details

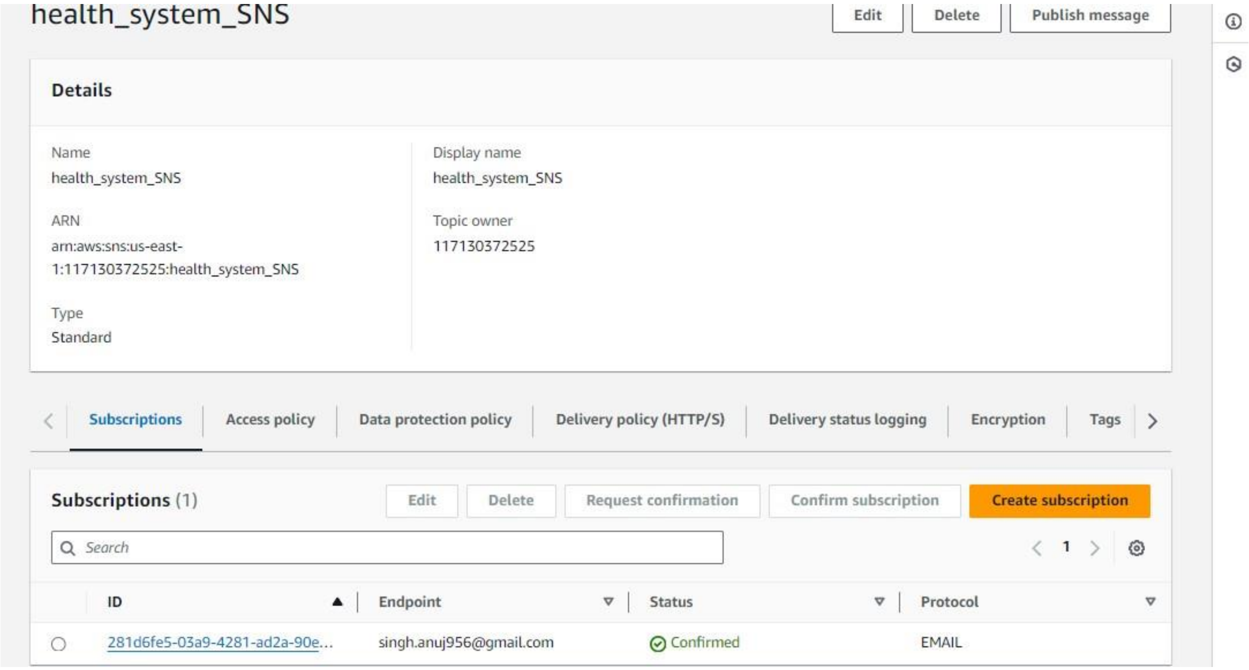
Name
health_system_SNS

ARN
arn:aws:sns:us-east-1:117130372525:health_system_SNS

Type
Standard

Display name
health_system_SNS

Topic owner
117130372525



health_system_SNS

Details

Name
health_system_SNS

ARN
arn:aws:sns:us-east-1:117130372525:health_system_SNS

Type
Standard

Display name
health_system_SNS

Topic owner
117130372525

Subscriptions (1)

	ID	Endpoint	Status	Protocol
○	281d6fe5-03a9-4281-ad2a-90e...	singh.anuj956@gmail.com	✓ Confirmed	EMAIL

ZIP FOLDER HAVE THESE FILES :

1. hbase.pdf
2. hive2.pdf

3. sqoop.pdf
4. kafka_spark_generate_alerts.py
5. kafka_consume_alerts.py
6. sns.pdf
7. code_logic.pdf

All these files have necessary code and screen shots which generated during project .