**Dataset**

In the previous segment, you learnt about the Problem statement that you will solve in this project. Now in this segment, you will learn more in detail about the different datasets and that you will be working within this project.

Now in the next video, we will take a look at the schema of the data sets available for this project.

Play Video

3916596

As discussed in the video, a **centralised RDS** has been set up where the data will be hosted. You will be required to import the data from this RDS and then use this data to create a Kafka topic for the Patient's vital information. We will also provide the dataset for Patient’s Contact information and for the Reference Threshold values. You will have to use Sqoop to consume the data from the contact information table whereas, for the latter, you will have to create the table yourself.

**Hostname**: [upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com](http://upgradtest1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/)  
**username**: student  
**password**: STUDENT123  
**dbname**: testdatabase  
**table-name**: patients\_vital\_info

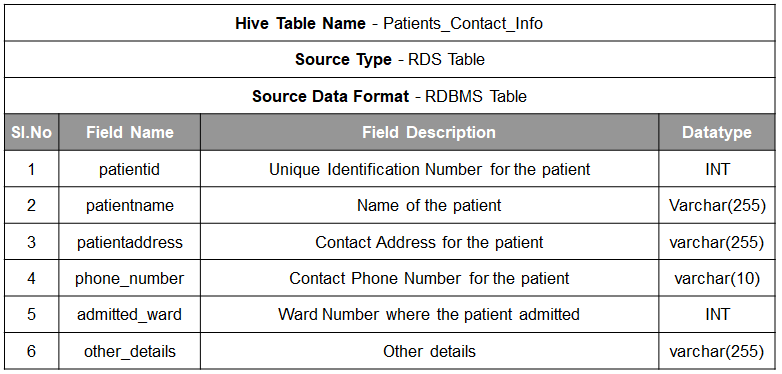
The data is based on daily health monitoring mechanisms which take the heart rate and blood pressure of the patient into consideration to check for any abnormalities.

The Datasets that you will be working with, in this Capstone Project are as follows -

Patient’s Contact Info Dataset

This data set will be used to access the contact information of the patient so that it can be used in case of any emergencies.

The schema for this dataset is given below



Patient's Contact Info Dataset Schema

Here is a snippet of how the Patient's Contact Info dataset looks like:

| **patientid** | **patientname** | **patientaddress** | **phone\_number** | **admitted\_ward** | **age** | **other\_details** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alex S | XDC test Address | 8982739282 | 1 | 23 | NULL |
| 2 | Sammy A | New Building Address | 2382739282 | 2 | 45 | NULL |
| 3 | Karan C | Aws Address | 8923739282 | 3 | 56 | NULL |
| 4 | Dara M | India Address | 2182739282 | 4 | 67 | NULL |
| 5 | Pam | ABC test Address | 4982739282 | 5 | 72 | NULL |

The details for this table in RDS are as follows:

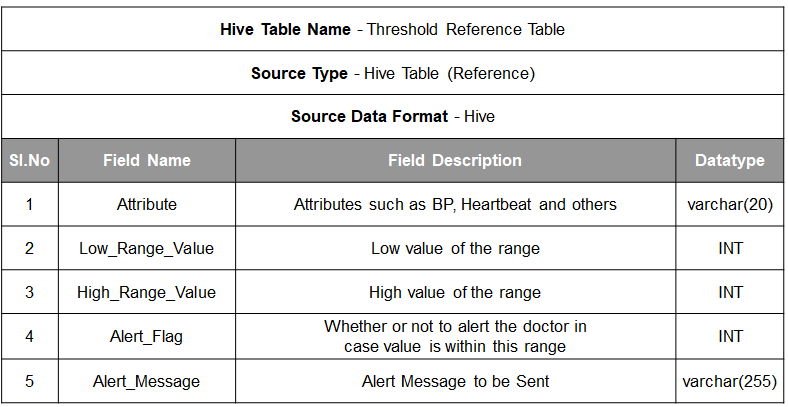
**Hostname**: [upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com](http://upgradtest1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/)  
**username**: student  
**password**: STUDENT123  
**dbname**: testdatabase  
**table-name**: patients\_information

Threshold Reference Table

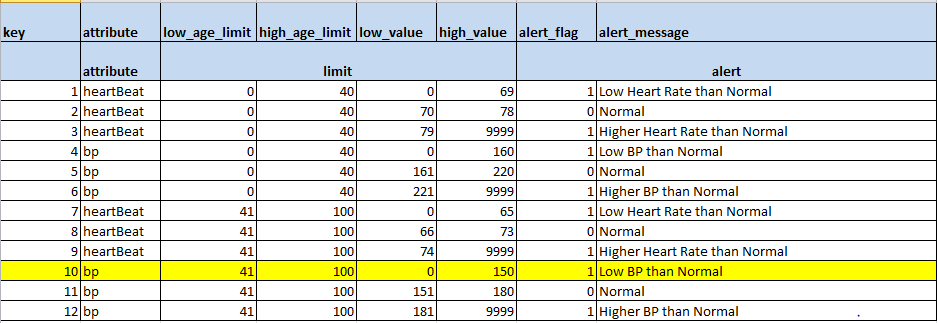
This data set will be used for comparing the vital incoming data from the patient with standard threshold information to check for anomalies.

The schema for this dataset is given below

**Note**: There are 2 columns which are not mentioned in the schema - **low\_age\_limit and high\_age\_limit**, both with the data type INT which are also supposed to be there in the table as mentioned in the Threshold Reference Table Image.



Threshold Reference Table Schema

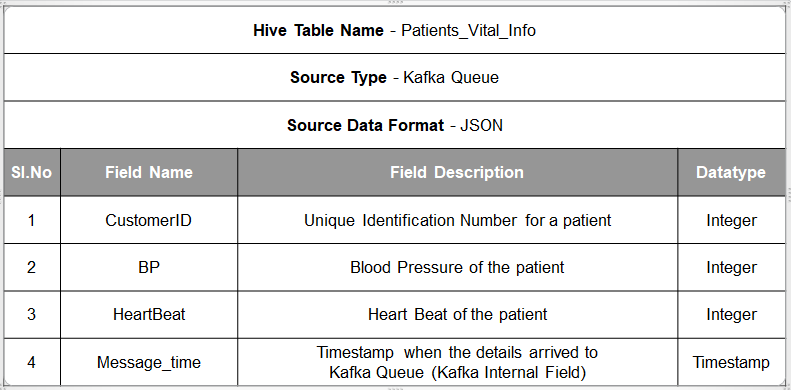


Threshold Reference Table

Patient’s Vital Information Dataset

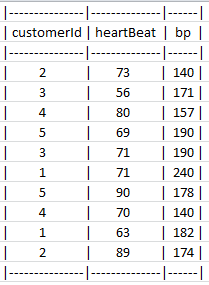
This data set will be used to monitor the patient on a real-time basis and check their overall well-being.

The schema for this dataset is given below:



Patient's Vital Info Dataset Schema

The vital information snapshot for 10 seconds would like:



10 sec Vitals Snapshot

**Output:**

Once it is decided that the alert has to be sent, the notification message will have both, the vital information and the contact information of the concerned patient which will be sent in JSON format. Here is a sample output format of how the alert message will be sent to the registered email address:

{"patientname":"Sammy A",

"age":**45**,

"patientaddress":"New Building Address",

"phone\_number":"2382739282",

"admitted\_ward":**2**,

"bp":**140**,

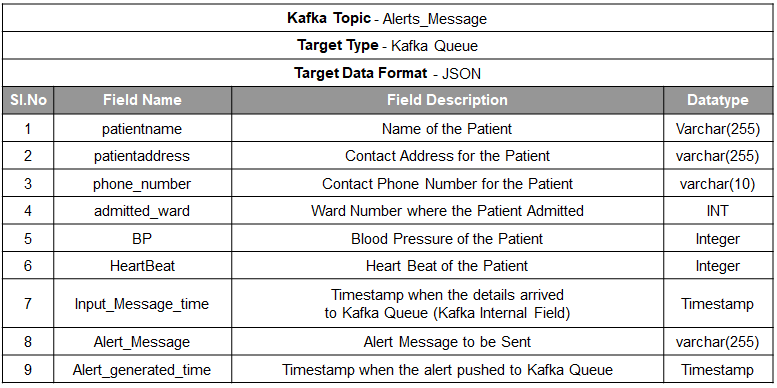
"heartBeat":**73**,

"input\_message\_time":"2020-11-02T05:52:01.885Z",

"alert\_generated\_time":"2020-11-02T07:00:20.942Z",

"alert\_message":"Low BP than Normal"}

The schema for the output will look like this:



Output Schema

Now let’s move on to the next segment where we will see the tasks to be performed in order to complete

**Solution Approach And Tasks**

Now, in this segment, you will go through the solution approach that you will be following in this project. You will also learn about the tasks in detail that you will be working on in this project.

In the next video, let’s look at the solution approach for this problem statement.

Play Video

3916596

In the video, we saw the following components in our pipeline:

* **Producer app**: It simulates IoT devices to push data through to the Kafka queue and will generate vital information and push it in the JSON format into the Kafka Queue. Eg: {"heartBeat": 73, "bp": 140, "customerId": 2}
* **Kafka Queue (Patient’s)**: It stores the data pushed into it by the producer in a pipeline to send it further to the next component.
* **Spark Streaming Job 1**: To convert data into the desired form (parquet) and store it into the HDFS table.
* **Hive Table (Patient’s Info)**: Query the table data and do analysis.
* **Spark Streaming Job 2**: Monitoring and analysing the data collected using already present data (Reference Table and Contact Info) and look for any anomalies to be reported.
* **Hive Table (Reference)**- Store and query threshold data to check whether incoming data have any anomalies.
* **RDS + Batch Sqoop Job**: Push the contact data of the patients from RDS  into the Hive table.
* **Hive Table (Contact Info)**: Store and query contact and other information to check whether incoming data have any anomalies.
* **Kafka Queue (Doctor’s)**-Collect data from Spark application to put in a queue and feed to Consumer application
* **Consumer Application**- Send SNS push notification to the subscribed email-id to report any anomalies.

In the next video, our expert will walk you through the entire solution approach for this project.

Play Video

3916596

In our system, we have three sources: The first source is the producer application, which pushes vital info into the queue. The second source is the reference data to decide on anomalies. The final source is contact information. All these three would be used in conjunction to monitor and analyse whether or not an alert has to be sent.

The first task is to bring the stream of data by the producer application into HDFS. So, you need to build the application to simulate the working of an IoT device which would send real-time data.

The second task is to make sure that the data is stored in HDFS, which would be done through the Spark application.

To monitor this incoming data, you need:

* **Reference data**, which needs to be loaded into HBase and to build a Hive on top of it so that the data can be accessed and queried from the Spark application; and
* **Patient contact information**, which comprises the rest of the data that can be used together with the vital information.

So, the reference data, contact information and the vital information will be used together by a Spark application to check for anomalies and decide whether or not to send an alert.

Once the Spark application has completed its job, if the vital information is out of the threshold limits, an SNS notification is sent to the registered email ID.

Now, in the next video, you will learn about the tasks that the students are expected to carry out to complete this project.

Play Video

3916596

In this video, you learnt about the following tasks expected from you:

1. To push vital data from RDS into a Kafka topic, you need to build a producer application. The 1,800 messages need to be pushed into the Kafka topic, one message per second. This will simulate the IoT device.

So the two subtasks for this are:

* 1. Copy data from RDS
  2. Push it in JSON format into the Kafka Queue

1. To collect a stream of vital records from patients (producer application), you need to build a Kafka topic. The topic will store the information in a queue and release it to the next component every 1 second.
2. To collect messages from the Kafka queue into the  HDFS and write it in the parquet format, you need to build a Spark application.

So the three broad subtasks for this are:

* 1. Build a schema to accommodate the incoming information
  2. Add a new field named “timestamp”
  3. Push it in parquet format

1. Create an Hbase table and insert reference data. This contains the upper and lower limits for the vital information coming in every second.
2. Build a Sqoop application to extract patients’ data from the RDS instance into a Hive table.
3. Build a Spark application to compare reference data with input data and send a notification. This will run every time a new record gets added into the HDFS table.

So broadly speaking the subtasks for this are:

      a) Create a schema with attributes “heart rate”, “bp”, “customerId”, “message time”

      b) Read the vital information table

      c) Read the threshold information table

      d) Compare and pick out any anomalies

      e) Send a notification, if required into the Kafka Queue

      8. Build a consumer application to send SNS notification to the registered email address.

Please refer to the additional resources segment where you can learn more about the new tools and techniques that you will have to work within this project.

**Note**: The **Amazon SNS** service is a **costly service** and we highly recommend that you **delete** the **SNS topic** after you are done with the Capstone project.

**NOTE- This is the mid submission module which has to be completed by the end of three weeks after starting the project. The final submission module has to be completed by the end of six weeks after starting the project.**

In the next segment let’s see the overall code flow of our project

**Additional Resources**

Welcome to the segment where you will be learning about a couple of additional resources required for this project. You will find the steps to be followed in this segment along with a document with screenshots attached for your reference.

First, we start with SNS which will be used to send a real-time notification to the registered email address about the complete details of the concerned patient.

Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication.

The A2A pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications. Using Amazon SNS topics, your publisher systems can fanout messages to a large number of subscriber systems including Amazon SQS queues, AWS Lambda functions and HTTPS endpoints, for parallel processing. The A2P functionality enables you to send messages to users at scale via SMS, mobile push, and email.

**To create an SNS topic**

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. On the Amazon SNS dashboard, choose Create Topic.  (type = standard )
3. In the Create new topic dialog box, for Topic name, enter a name for the topic.
4. Choose Create topic.
5. Copy the Topic ARN for the next task (for example, arn:aws:sns:us-east-1:111122223333:my-topic).

**To subscribe to an SNS topic**

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose Subscriptions, Create subscription.
3. In the Create subscription dialog box, for Topic ARN, paste the topic ARN that you created in the previous task.
4. For Protocol, choose Email.
5. For Endpoint, enter an email address that you can use to receive the notification, and then choose to Create subscription.
6. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon SNS.

**To publish a test message to an SNS topic**

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose Topics.
3. On the Topics page, select a topic and choose to Publish to the topic.
4. In the Publish a message page, for Subject, enter a subject line for your message, and for Message, enter a brief message.
5. Choose Publish Message.
6. Check your email to confirm that you received the message.

Now that you have set up your SNS topic, we take a look at how to connect to the centralised RDS containing the datasets required for the project.

**Note**: The **Amazon SNS** service is a **costly service** and we highly recommend that you **delete** the **SNS topic** after you are done with the Capstone project.

**Steps to connect to RDS from python**

To access the two databases stored in RDS, you will be using "mysql.connector". The database name is "**testdatabase**" and the two tables in it are: "**patients\_vital\_info**" and "**patients\_information**". The vital information will be consumed by the python producer application while the contact information will be consumed using Sqoop.

The connection string for both the tables would be:  jdbc:[mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase](mysql://upgradtest1.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase)

import mysql.connector

connection = mysql.connector.**connect**(host='http://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/', **database**='testdatabase',user='student', password='STUDENT123')

To access all the records you will use the cursor() method of the connection object. Following this, you will make use of the execute() and fetchall() methods. Please refer to the official documentation attached below for your reference for more on how to use these methods.

**AWS SNS Setup**

**Download**

**Note**: Please refer to the Kafka Integration segment in the Industry demo session in the Spark Streaming module to see how to read and write to Kafka with the help of Spark Streaming.

Additional Resources

[AWS SNS DOCUMENTATION](https://docs.aws.amazon.com/sns/?id=docs_gateway) - The official AWS SNS documentation.

[MySQL DOCUMENTATION](https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html) - The official MySQL documentation.

**Grading Rubrics**

| **SI NO** | **TASKS** | **WEIGHTAGE** | **EXPECTATIONS** |
| --- | --- | --- | --- |
| 1 | Run/start Kafka server, create a topic | 5% | Successfully create the topic |
| 2 | Create a producer application to read from RDS and push the message into the topic  And list the messages in the topic | 15% | Successfully create the application and read data from RDS and push it into the topic |
| 3 | Pyspark application to read all messages from Kafka topic into HDFS file in parquet format in\_df2.writeStream.format(). | 20% | Successfully create the application and read data and store in parquet format |
| 4 | Build an external hive table on HDFS and view data | 5% | Successfully build the table |
| 5 | Create hbase table with 3 families (attr, limit, msg) -- insert 12 records into hbase table | 10% | Successfully create the table and populate it |
| 6 | Create an external hive table on hbase and view data | 5% | Successfully build the table |
| 7 | Extract patient info using sqoop into hive table. | 5% | Successfully bring data from RDS to Hive table |
| 8 | Spark streaming application to read data from HDFS, compare with hbase | 20% | Successfully read from HDFS and compare it with threshold data |
| 9 | SNS configuration to the assessment email ID | 5% | Successfully configure SNS |
| 10 | Consumer application to send email | 10% | Successfully create an application and send emails to the registered email address |

**Submission Guidelines**

**Note** - **The first four points(below) of submission will be considered for the mid submission three weeks after the start of the project.**

**Submissions Required**:

Submit a zipped file containing:

1. A script to run/start Kafka server, create a topic(**kafka.pdf**)
2. A script to create producer application to read from RDS and push the message into the topic in below format  and list the messages in the topic (**kafka\_produce\_patient\_vitals.py**)
3. A script of Pyspark application to read all messages from Kafka topic into HDFS file in parquet format in\_df2.writeStream.format().. (**kafka\_spark\_patient\_vitals.py**)
4. A script to build an external hive table for the threshold data and view threshold data(**hive1.pdf**)
5. A script to create hbase table with 3 families (attr, limit, msg) -- insert 12 records into hbase table(**hbase.pdf**)
6. A script to create an external hive table for patients vital information and view data(**hive2.pdf**)
7. A script to extract patient info using sqoop into hive table(**sqoop.pdf**)
8. A script of Spark streaming application to read data from HDFS compare with hbase (**kafka\_spark\_generate\_alerts.py**)
9. A script of consumer application to send an email (**kafka\_consume\_alerts.py**)
10. A screenshot of successful SNS configuration(**sns.pdf**)
11. A document comprising your overall code logic. This document should also have the commands needed to run all the scripts mentioned above. (**code\_logic.pdf)**