# Healthcare IoT Processing in Cloud

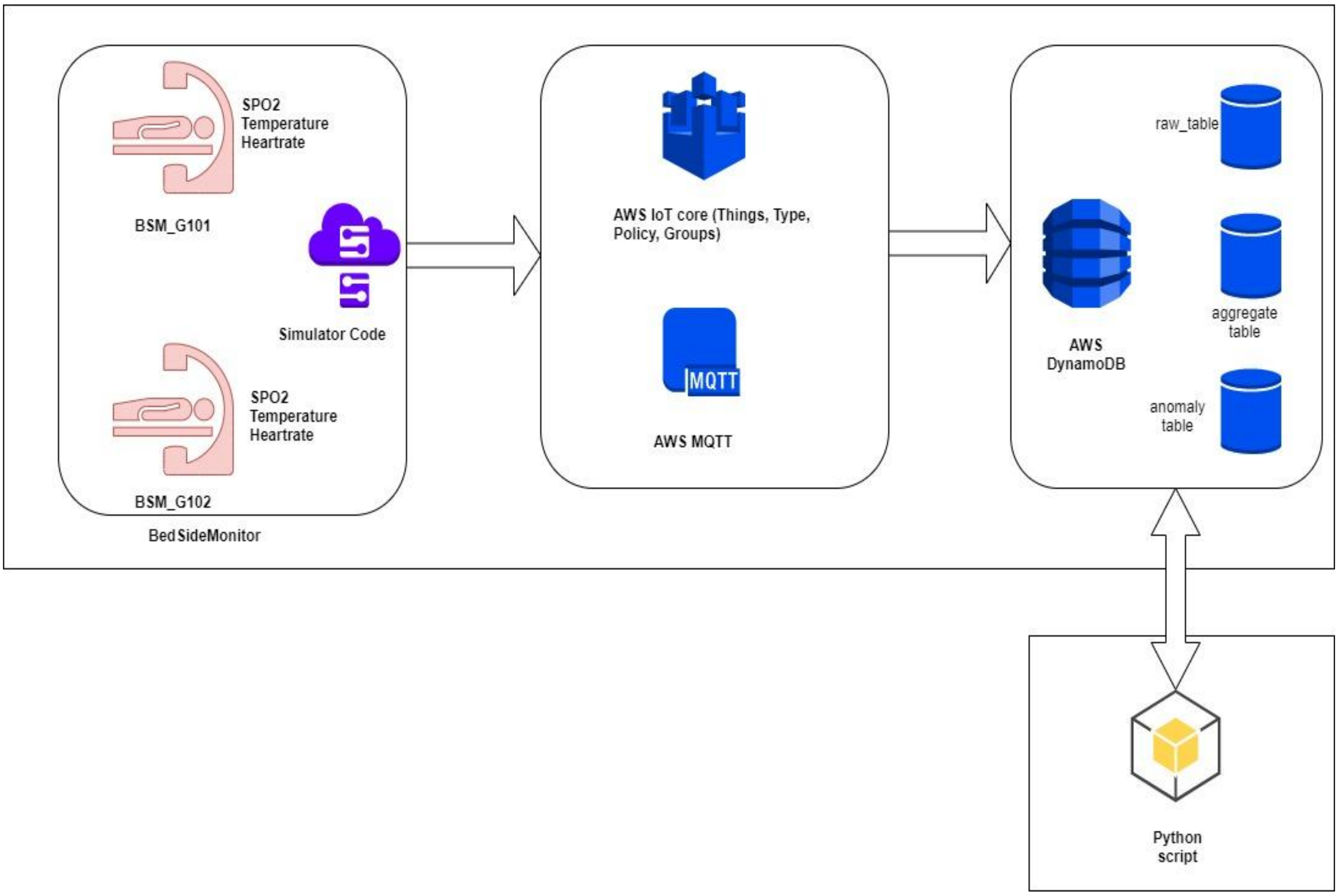
## Introduction

In this project, we'll be building up on the IoT Core and DynamoDB lab. The device simulator will publish Heart Rate, SpO2 (Oxygen Saturation) and Body Temperature values periodically. You'll need to channel that to the cloud (Iot Core → DynamoDB), and then aggregate data and detect anomalies.

The initial part of the project is similar to the IoT Core lab.

## Housekeeping points

- This is a simple example and may not follow some standard practices.
- The focus is on the main flow, and not as much on error handling.
- There will be some differences from standard flow since we are mimicking 'continuous monitoring by an always-on service' with a manually triggered program.



## Program Organization

1. BedSideMonitor.py - This is quite similar to the device simulator in IoT Core and DynamoDB lab.
2. The complete post-processing code needs to be built up. The basics will be similar to the model and database class layers from one of the previous projects.

## Problem Statement

There are three top-level tasks in this project. The device simulator code can push data to an IoT core endpoint. Please demonstrate functioning of tasks 2 and 3 with example calls in your main code file. You need to do the following:

1. **(Easy)** Similar to the lab, create Things, Certificates, Policy, DynamoDB table (bsm_data) and IoT Core Rule to push the data created to the DynamoDB table. The simulator code is similar to the lab's simulator.
   You should create at least two devices/things. You can modify the simulator code to take deviceid as another parameter or run two copies in parallel with different hard-coded device ids. Please run them for at least an hour continuously to generate enough data. The data is being generated at 1, 10, and 15 second frequencies for the three different sensor types.
   Please provide the screenshots from your Nuvepro account as configuration validation. Look at the Workbook named as **C04P01 - IoT - HealthCare - Cloud-Workbook. This has all the placeholders to attach the screenshot relevant to all the tasks.**

2. **(Medium)** Aggregate the data in another table
   a. Create a new table (bsm_agg_data) using code or manually via the UI. Create model and database classes to access DynamoDB, similar to the previous project.
   b. Provide functionality to aggregate the data from the three sensors individually, at the minute level. You should calculate and store the average, minimum and maximum values taking all values within a minute in account. Since we are only mimicking a continuous monitoring system, please provide the aggregator method call to be for a specific time range and aggregate data per minute within that range. This should be stored programmatically in the aggregator table.
   For example, when called with a time range of '2021-02-17 13:00:00 to 2021-02-17 14:00:00', it should create 60 rows per sensor type in the aggregated table, each being the aggregation of all values within a minute for that sensor data type. Entries can be skipped if there is no relevant data.
3. **(Hard)** Detect anomalies based on rules and store them

a. Create a rules config file using any format that you see fit, including json. This should have the ability to define combination alert rules. One should be able to define an OR rule with min and max **average** values for any of the three data types. In addition, one should be able to define a trigger count so that the values should be breached continuously that many times, before an alert should be raised.
   For example, a rule can be {'type': 'HeartRate', 'avg_min': 55, 'avg_max': 105, 'trigger_count': 5} This should trigger an alert if five continuous aggregated (at the minute level) average values are outside the min/max range.
   Please write an appropriate parser for these rules in the code. There should be at least two rules specified in the config file for demonstration purposes.

b. Create a new table (bsm_alerts) using code or manually via the UI. Create the appropriate model to access it.
   Provide functionality to run the rule check for a specified time range. It should check all rules on all devices individually within that time period and raise alerts as appropriate.
   These alerts should be printed on the console, and saved to the bsm_alerts table with an appropriate format, specifying deviceid, data type, the timestamp of the first instance of breach, and the rule that was breached.

## Evaluation Rubric

### Total Project Points: **240**

- Basic compilation without errors (**10%**) : **24 Points**
- Correctness:
  Correctness of implementation
    - Problem statement - point 1 (40%) : **96 Points**
    - Problem statement - point 2.a (**10%**) : **24 Points**
    - Problem statement - point 2.b (**30%**) : **72 Points**
    - Problem statement - point 3.a (**5%**) : **12 Points**
    - Problem statement - point 3.b (**5%**) : **12 Points**

## Note

- **Minimum Requirements**: The final submission that you upload needs to have a successful compilation, at the least.
    - The basic code that will mimic the BedsideMonitor data is already provided in the attached folder.
    - You are supposed to write and submit code that should perform aggregation and anomaly on the raw data that was generated. As part of the submission you are also supposed to write certain JSON based rules that will determine if the data should be considered as anomaly/alert or not.

- Submitted files should have implementation pertaining to aggregation and anomaly detection on the recorded raw_data. Ideally you should create different python files for different tasks.

- **Expected Submission Files:**
  - **RawDataModel.py:** This file will process the raw data that was created and pushed in the bsm_raw_data table by BedSideMonitor.py.
  - **AggregateModel.py:** This file will have implementation related to the aggregation on the raw data
  - **Database.py:** This is a class that will invoke and call the various databases and methods to fetch and store the data in the bsm_agg_data table.
  - **AlertDataModel.py:** This is a file that will access the created rules and will detect the anomaly values based on the parsed rule. The detected output will then be pushed in the bsm_alerts
  - **Main.py:** This will be the driver code that should be invoked to perform the aggregation operation and alert operation on the inserted data that is available in the raw form in one of the tables.
  - **BedSideMonitor.py:** The modified or newly created data publisher should also be present in the zipped folder along with other files.
  - **C04P01-IoT-HealthCare-Cloud-Workbook.pdf:** Attach all the screenshots at the mentioned placeholders. Follow the instructions mentioned in the workbook while attaching the screenshot. You should submit this file in PDF version only.

## Program Instructions

1. Download the zipped folder named **C04P01-Project-HealthCare-IoT-Cloud.zip**, unzip it on your local machine, and save it. Go into the directory named **C04P01-Project-HealthCare-IoT-Cloud.**

2. Make sure you have Python 3.6 or higher installed. At your command prompt, run:

   ```
   $ python --version
   Python 3.7.3
   ```
   If not installed, install the latest available version of Python 3.

3. Ensure that you have the boto3 module installed to interact with DynamoDB apis.

4. After the appropriate settings in IoT Core and DynamoDB, run the simulator code for at least an hour to create enough data in DynamoDB.
5. Please follow the account creation doc to get hold of the session key and secret key to perform the insert and fetch operation in DynamoDB using python program.

## Resources

1. Boto3:
   https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
2. DynamoDB access from python:
   https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html
   https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html#DynamoDB.Client.list_tables

3. DynamoDB UI (Please watch the DynamoDB videos from the course):
   https://aws.amazon.com/getting-started/hands-on/create-nosql-table/
4. IoT Core (Please review notebooks related to IoT Core):
   https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html