VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Class Based Project Report

BCSE355L- Cloud Architecture Design

Faculty: Dr. Mohanraj Gopa

Title:

Cloud-Based Health Monitoring System

Reg. No. & Names:

23BIT0151: Veda Vishal Chandergi

23BIT0193: Kashish Singh

23BIT0189: Peehu Saxena

TABLE OF CONTENTS

CHAPTER 1

Problem Definition and Objectives

1.1 Introduction

In today's world, healthcare systems increasingly rely on technology to ensure the safety and well-being of patients. Monitoring patient vitals such as heart rate, temperature, and oxygen saturation is critical for detecting health risks early. However, traditional monitoring systems depend on dedicated devices, local servers, and complex infrastructure, making them costly and inaccessible to smaller medical facilities.

1.2 Statement of the Problem

Many healthcare centers face challenges such as:

- Limited access to real-time monitoring systems.
- High setup costs for IoT-based hardware and on-premise servers.
- No central cloud system for storing and analyzing patient data.
- Lack of instant alerts for abnormal health conditions.

To overcome these, a cloud-based solution is proposed using Amazon Web Services (AWS), where data from virtual or real sensors can be processed, stored, analyzed, and visualized seamlessly.

1.3 Objectives of the Project

- To design a real-time health monitoring system using AWS cloud infrastructure.
- To implement a serverless architecture using Lambda and API Gateway.
- To store the vitals and patient info using AWS DynamoDB.
- To analyze and alert on abnormal readings using AWS SNS.
- To create a user-friendly web dashboard hosted on AWS S3 and CloudFront.
- To use Python-simulated vitals for testing, representing IoT sensor data.

1.4 Scope of the Project

This project demonstrates how cloud computing can revolutionize healthcare monitoring systems. It can be used for:

- Remote patient supervision.
- Early detection of anomalies.
- Hospital dashboards for doctors and nurses.
- Integration with wearable devices in future iterations.

CHAPTER 2

System Components and AWS Services Used

2.1 AWS Services Used

AWS Service	Description	Purpose in Project
Amazon API Gateway	Manages and exposes REST API endpoints.	Receives and sends patient data (GET/POST).
AWS Lambda	Executes functions without managing servers.	Processes incoming data and performs analysis.
Amazon DynamoDB	NoSQL database optimized for scalability and speed.	Stores patient vitals with timestamps.
Amazon SNS (Simple Notification Service)	Sends messages, emails, or alerts.	Notifies users when abnormal readings occur.
Amazon CloudWatch	Monitors AWS resources and logs.	Used to debug Lambda and track anomalies.
Amazon S3	Object storage service with website hosting.	Hosts the web dashboard frontend.
Amazon CloudFront	CDN for secure HTTPS access.	Delivers the S3-hosted website globally with faster performance.

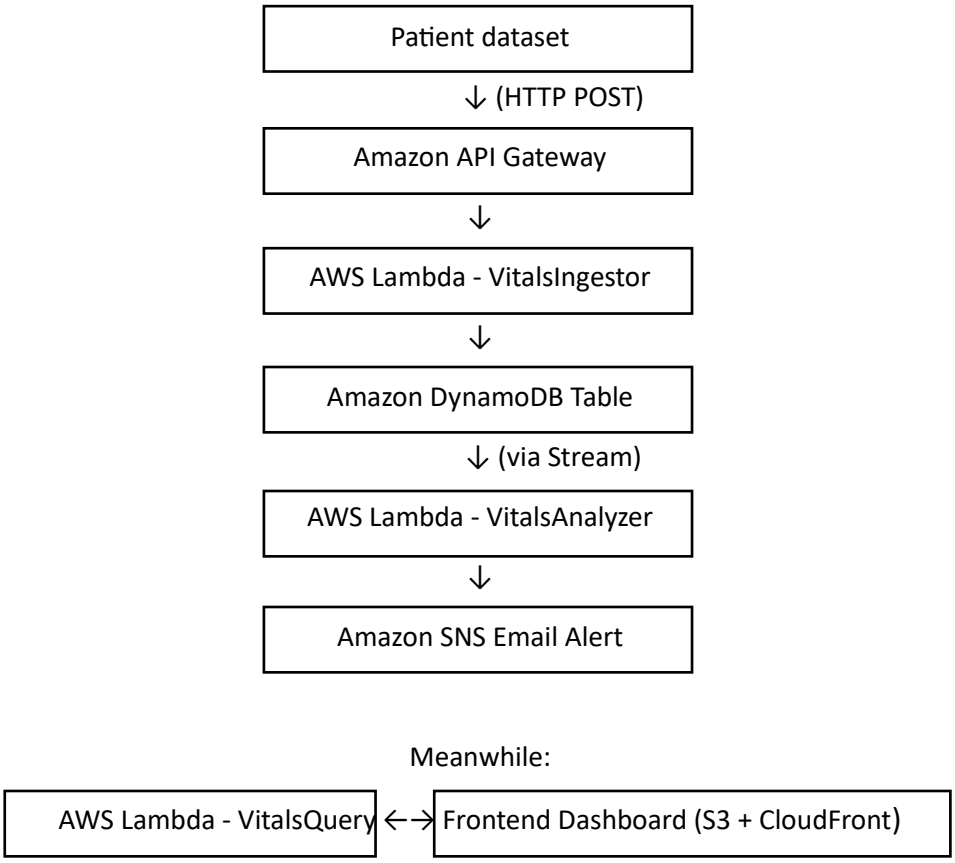
2.2 Why These Services Were Chosen

- **Scalability:** Each service scales automatically with no manual intervention.
- **Cost-effectiveness:** All components operate within AWS's Free Tier.
- **Serverless Architecture:** No server management; AWS handles provisioning and scaling.
- **Integration:** All chosen services work seamlessly with each other.

CHAPTER 3

System Architecture and Workflow Explanation

3.1 System Architecture



3.2 Workflow Explanation

1. **Data Simulation:**
   The Python script loads the Patient vitals such as Heart Rate, Temperature, and SpO_2 and patient data such as department and patient id. These are sent as POST requests to the API Gateway endpoint.

2. **Data Ingestion:**
   API Gateway triggers the VitalsIngestor Lambda, which validates the incoming data and stores it in the DynamoDB table.

3. **Stream Processing:**
   Each new data entry in DynamoDB activates the VitalsAnalyzer Lambda through DynamoDB Streams. This function calculates trends and z-score anomalies to identify health risks.

4. **Notification System:**
   If any anomaly is detected, the analyzer function publishes a message to SNS, which instantly sends an email alert to subscribed users.

5. **Data Retrieval:**
   The VitalsQuery Lambda fetches the latest vitals for a given patient when the frontend makes a GET request.

6. **Visualization:**
   The frontend website, hosted on S3 and CloudFront, displays real-time data using charts and alert indicators.

CHAPTER 4

Implementation and Development Process

4.1 Backend Development

The backend is built entirely on AWS using a **serverless model**:

- **Lambda Functions:**
  - VitalsIngestor: Accepts and stores data.
  - VitalsAnalyzer: Detects anomalies and triggers alerts.
  - VitalsQuery: Retrieves data for the frontend.
- **DynamoDB Table:**
  Table Name: Vitals
  - Partition Key: patientId
  - Sort Key: ts (timestamp)
  - Stream: Enabled (New Image).
- **SNS Topic:**
  HealthAlerts for alert distribution via email.
- **API Gateway Resource:**
  /vitals with two methods:
  - POST - Data ingestion
  - GET - Data query
- **CloudWatch:**
  Used for debugging, monitoring, and execution logging.

4.2 Frontend Development

- **Tools:** HTML, CSS, JavaScript, and Chart.js.
- **Features:**
  - Dropdowns for department and patient selection.
  - Real-time display of vitals (heart rate, SpO_2, temperature).
  - Alert indicators showing whether patient condition is “Under Control” or “Critical.”
  - Dynamic chart updates every 5 seconds using API data.
- **Hosting:**
  - Uploaded to **Amazon S3** bucket.
  - Static website hosting enabled.
  - Linked to **CloudFront** for HTTPS and fast delivery.

4.3 Data Simulation

Since actual IoT hardware was not available, the project includes a **Python-based data simulator**. This simulator:

- Gets the data from the provided Patient vitals dataset.
- Sends them via POST request to the deployed API Gateway endpoint.
- Provides consistent and realistic data flow for development and testing.
  This allows us to validate the backend and frontend workflow exactly as it would behave with real IoT devices.

CHAPTER 5

System Testing and Output

1. DynamoDB Table

Displays continuous insertion of simulated patient vitals.

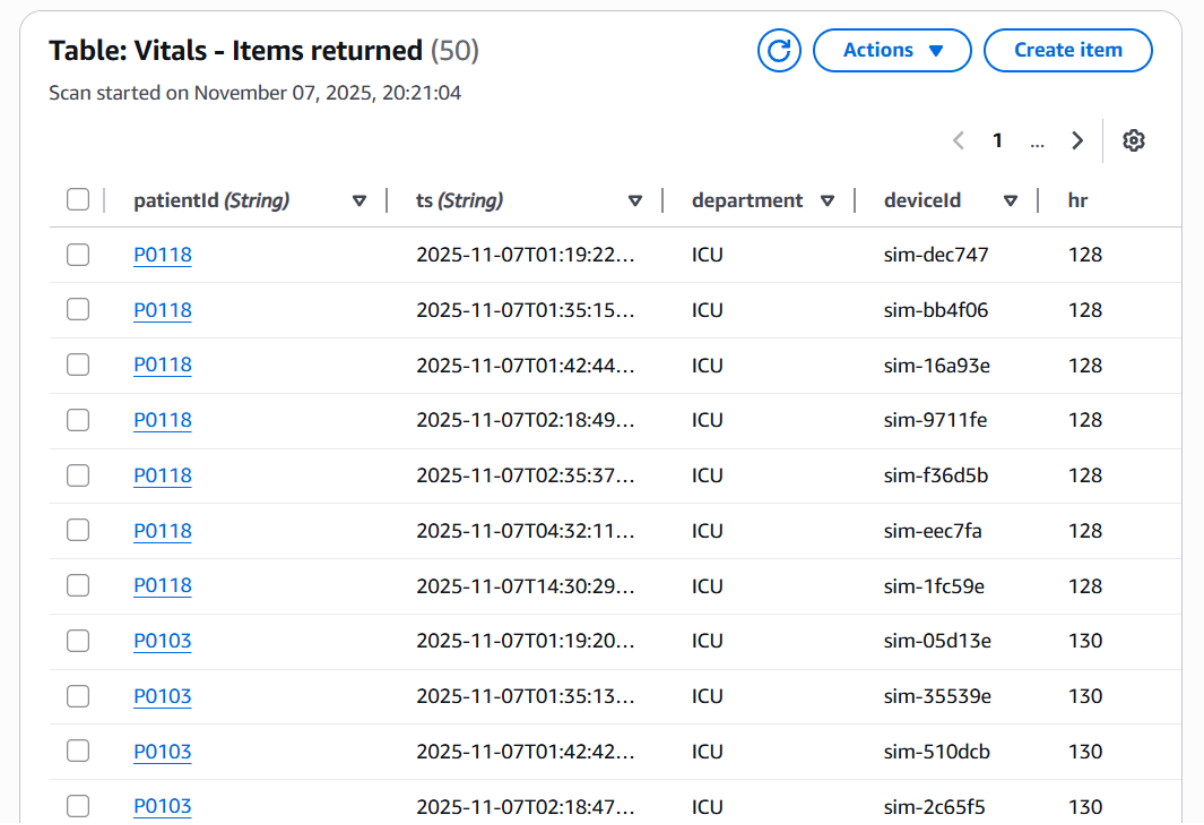


Table: Vitals - Items returned (50)

Actions | Create item

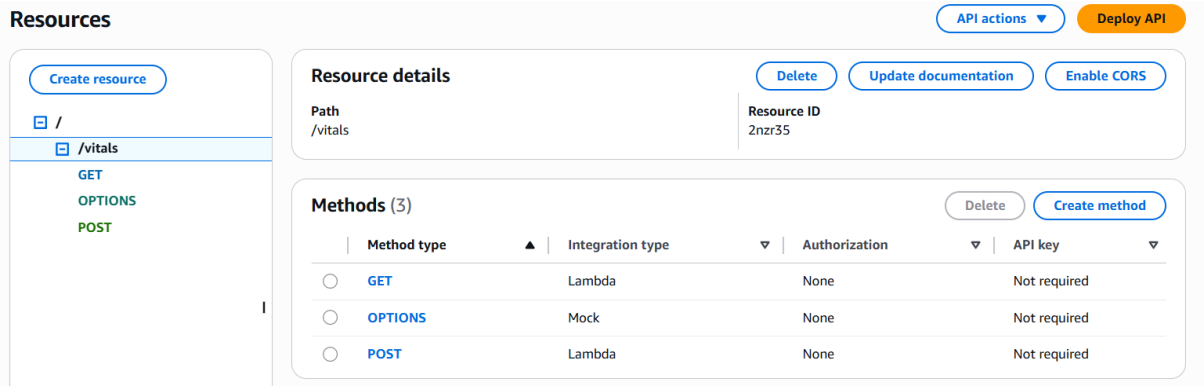
Scan started on November 07, 2025, 20:21:04

1 ...

patientId *(String)*	ts *(String)*	department	deviceId	hr
P0118	2025-11-07T01:19:22...	ICU	sim-dec747	128
P0118	2025-11-07T01:35:15...	ICU	sim-bb4f06	128
P0118	2025-11-07T01:42:44...	ICU	sim-16a93e	128
P0118	2025-11-07T02:18:49...	ICU	sim-9711fe	128
P0118	2025-11-07T02:35:37...	ICU	sim-f36d5b	128
P0118	2025-11-07T04:32:11...	ICU	sim-eec7fa	128
P0118	2025-11-07T14:30:29...	ICU	sim-1fc59e	128
P0103	2025-11-07T01:19:20...	ICU	sim-05d13e	130
P0103	2025-11-07T01:35:13...	ICU	sim-35539e	130
P0103	2025-11-07T01:42:42...	ICU	sim-510dcb	130
P0103	2025-11-07T02:18:47...	ICU	sim-2c65f5	130

2. API Gateway Test

Successful POST and GET



Resources

API actions | Deploy API

Create resource

/

/vitals

GET

OPTIONS

POST

Resource details

Delete | Update documentation | Enable CORS

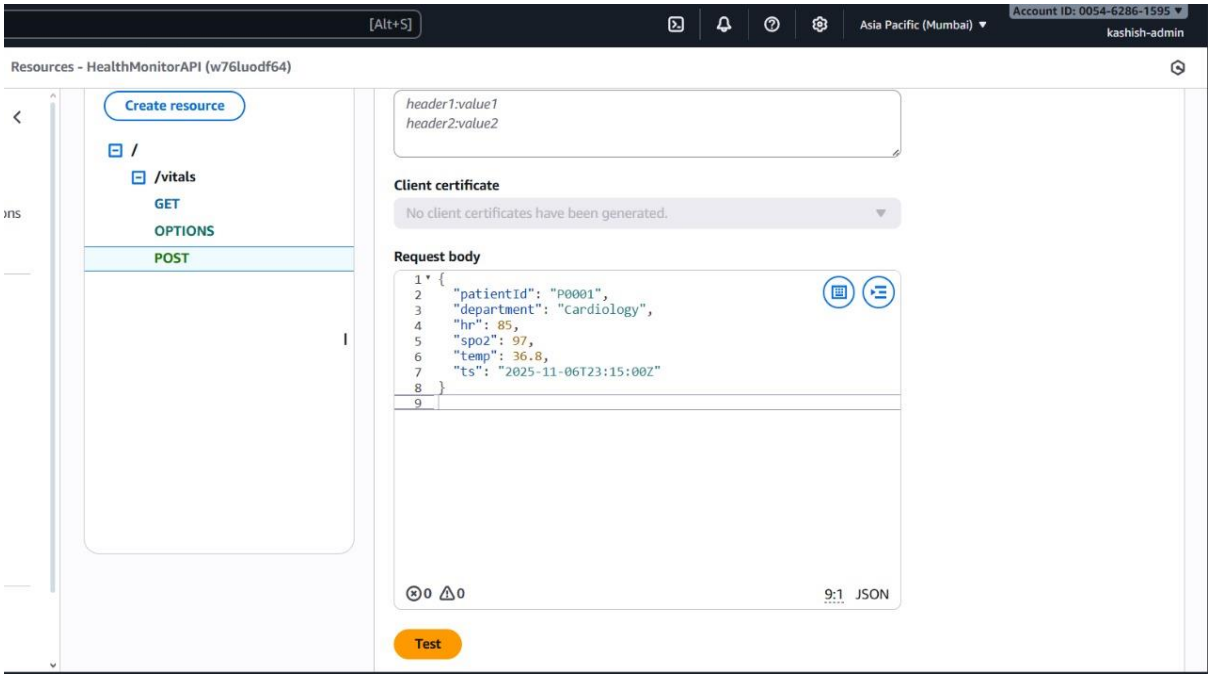
Path
/vitals

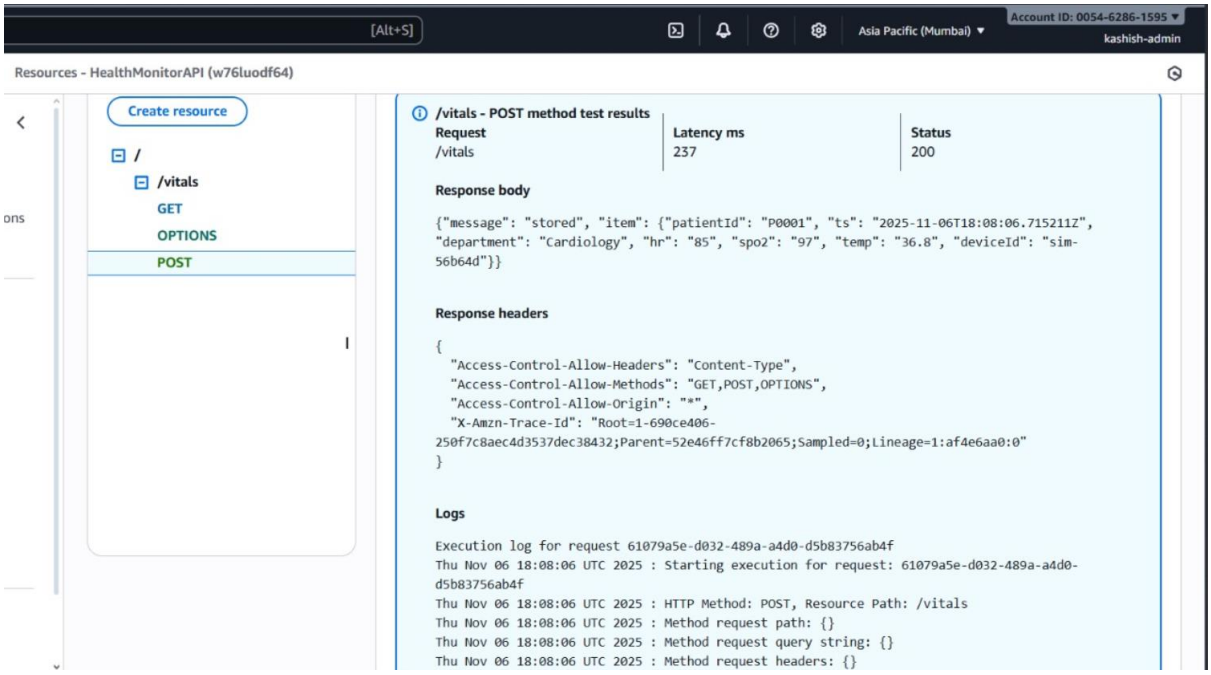
Resource ID
2nzr35

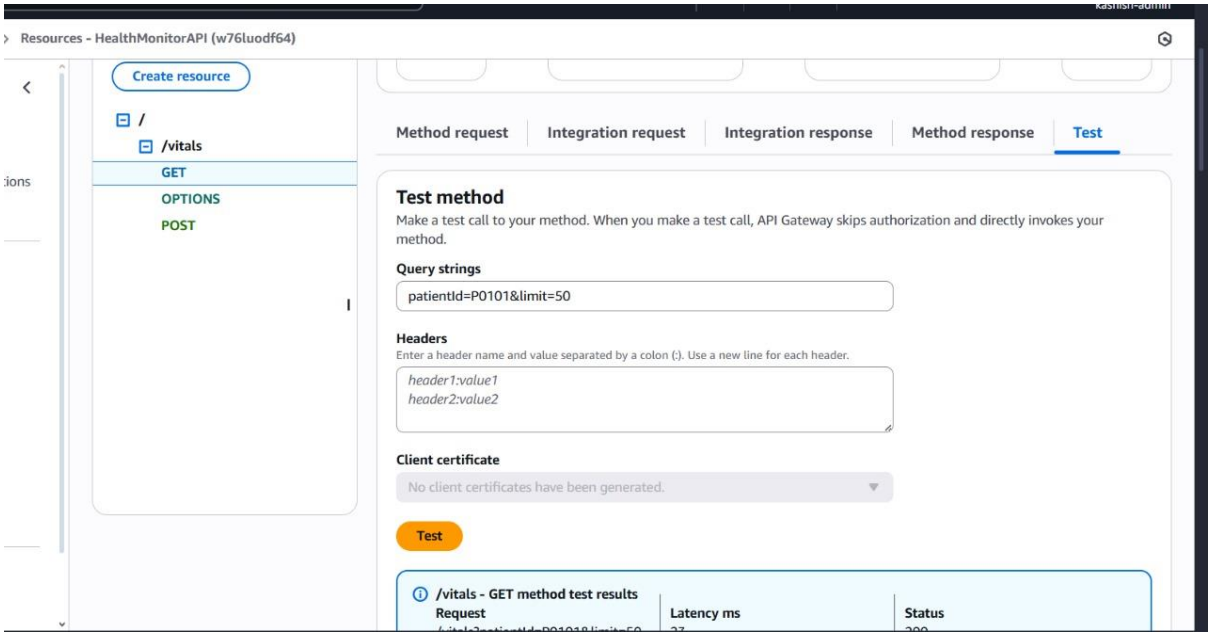
Methods (3)

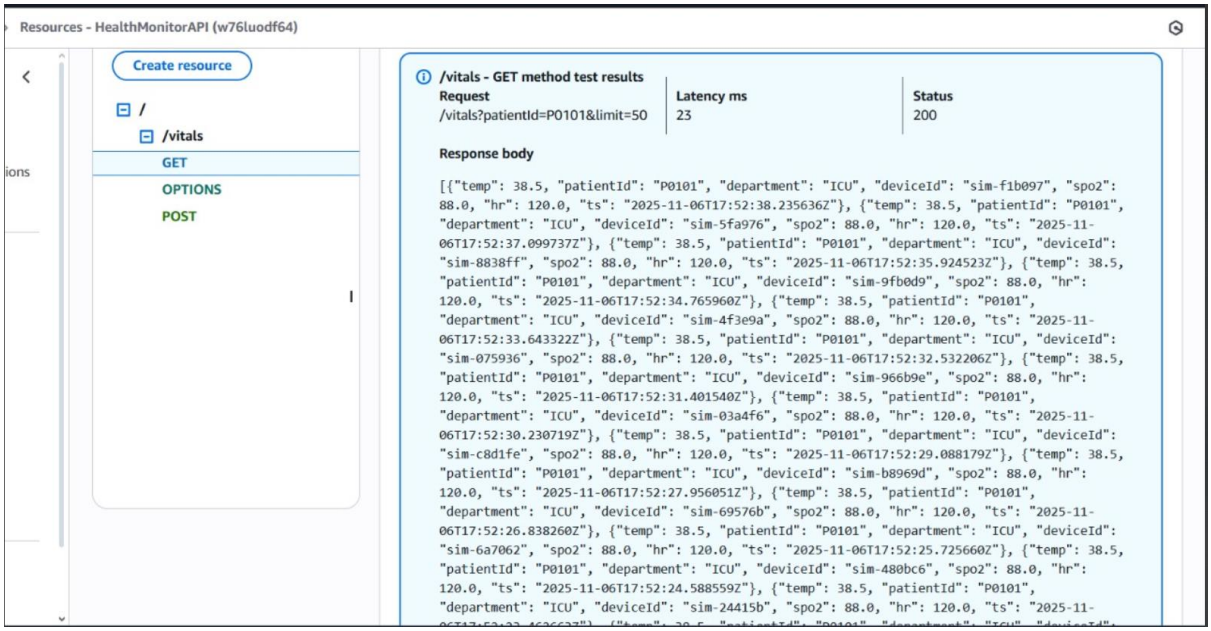
Delete | Create method

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required


[Alt+S]
Asia Pacific (Mumbai)
Account ID: 0054-6286-1595
kashish-admin
Resources - HealthMonitorAPI (w76luodf64)
Create resource
/
/vitals
GET
OPTIONS
POST
header1:value1
header2:value2
Client certificate
No client certificates have been generated.
Request body
{
"patientId": "P0001",
"department": "Cardiology",
"hr": 85,
"spo2": 97,
"temp": 36.8,
"ts": "2025-11-06T23:15:00Z"
}
0 0
9:1 JSON
Test

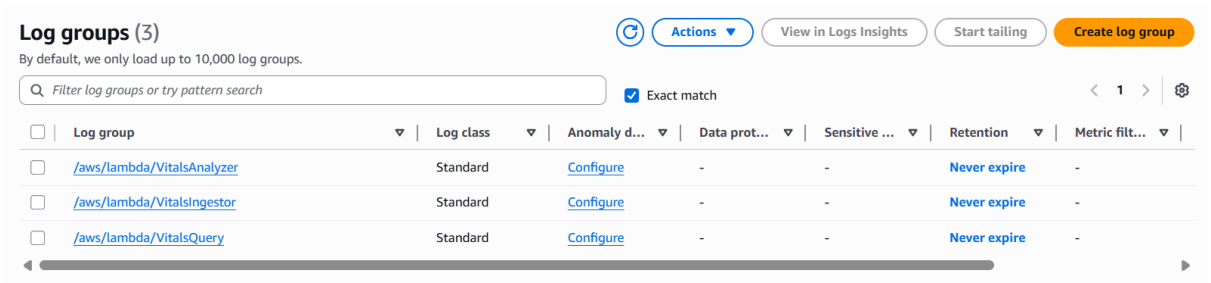

[Alt+S]
Asia Pacific (Mumbai)
Account ID: 0054-6286-1595
kashish-admin
Resources - HealthMonitorAPI (w76luodf64)
Create resource
/
/vitals
GET
OPTIONS
POST
/vitals - POST method test results
Request
/vitals
Latency ms
237
Status
200
Response body
{"message": "stored", "item": {"patientId": "P0001", "ts": "2025-11-06T18:08:06.715211Z", "department": "Cardiology", "hr": "85", "spo2": "97", "temp": "36.8", "deviceId": "sim-56b64d"}}
Response headers
{
"Access-Control-Allow-Headers": "Content-Type",
"Access-Control-Allow-Methods": "GET,POST,OPTIONS",
"Access-Control-Allow-Origin": "*",
"X-Amzn-Trace-Id": "Root=1-690ce406-250f7c8aec4d3537dec38432;Parent=52e46ff7cf8b2065;Sampled=0;Lineage=1:af4e6aa0:0"
}
Logs
Execution log for request 61079a5e-d032-489a-a4d0-d5b83756ab4f
Thu Nov 06 18:08:06 UTC 2025 : Starting execution for request: 61079a5e-d032-489a-a4d0-d5b83756ab4f
Thu Nov 06 18:08:06 UTC 2025 : HTTP Method: POST, Resource Path: /vitals
Thu Nov 06 18:08:06 UTC 2025 : Method request path: {}
Thu Nov 06 18:08:06 UTC 2025 : Method request query string: {}
Thu Nov 06 18:08:06 UTC 2025 : Method request headers: {}





3. CloudWatch Logs

Lambda functions execute and log outputs.



/aws/lambda/VitalsIngestor > 2025/11/06/[$LATEST]01197ff69dc54bbeb9a52548506fba5b

Log events

Actions | Start tailing | Create metric filter

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

Filter events - press enter to search | Clear | 1m | 30m | 1h | 12h | Custom | UTC timezone

Display

Timestamp	Message
2025-11-06T17:52:37.730Z	REPORT RequestId: 839f3379-ff78-4a0d-ae7b-fa3124c42c12 Duration: 35.86 ms Billed Duration: 36 ms Memory Size: 128 MB Max Memory Used: 89 MB
2025-11-06T17:52:37.872Z	START RequestId: 08960cca-a286-4320-bedb-a0d3955fe197 Version: $LATEST
2025-11-06T17:52:37.890Z	END RequestId: 08960cca-a286-4320-bedb-a0d3955fe197
2025-11-06T17:52:37.890Z	REPORT RequestId: 08960cca-a286-4320-bedb-a0d3955fe197 Duration: 16.77 ms Billed Duration: 17 ms Memory Size: 128 MB Max Memory Used: 89 MB
2025-11-06T17:52:37.937Z	START RequestId: 138e571e-a349-4416-9396-7df1e94bf73a Version: $LATEST
2025-11-06T17:52:37.968Z	END RequestId: 138e571e-a349-4416-9396-7df1e94bf73a
2025-11-06T17:52:37.968Z	REPORT RequestId: 138e571e-a349-4416-9396-7df1e94bf73a Duration: 30.99 ms Billed Duration: 31 ms Memory Size: 128 MB Max Memory Used: 89 MB
2025-11-06T17:52:38.143Z	START RequestId: 7d67e0aa-3928-40ee-a595-62e6fa4cd517 Version: $LATEST

4. SNS Email Alert – Real-Time Notification

Email alert is received when abnormal data is detected.

Health Alert - P0111 (ICU) Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 23:12 (17 minutes ago)

to me

{"patientId": "P0111", "department": "ICU", "timestamp": "2025-11-06T17:42:31.397360Z", "measure": "spo2", "value": 87, "reason": "Low SpO2 threshold", "context": {"recent": [87]}}

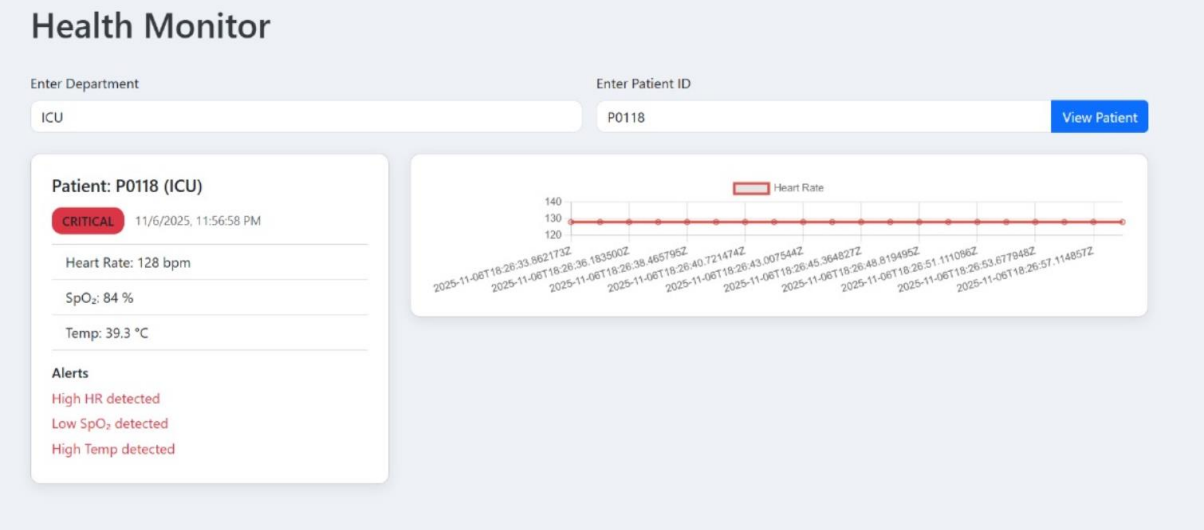
--

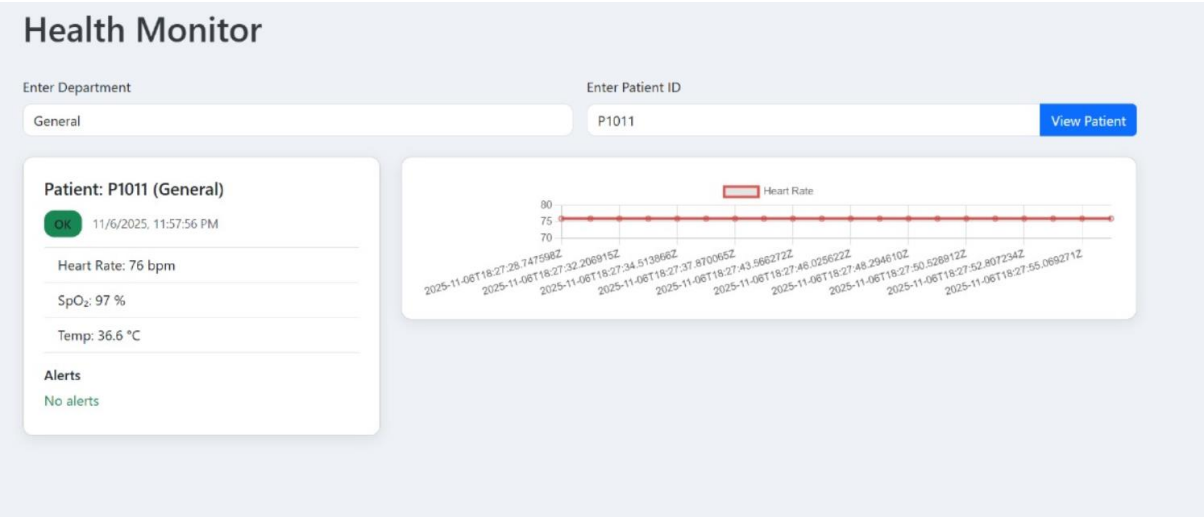
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:005462861595:HealthAlerts:57c4e975-e004-4c9b-81eb-9b9b180fc132&Endpoint=vedavc123@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support

5. Frontend Dashboard – Visualization

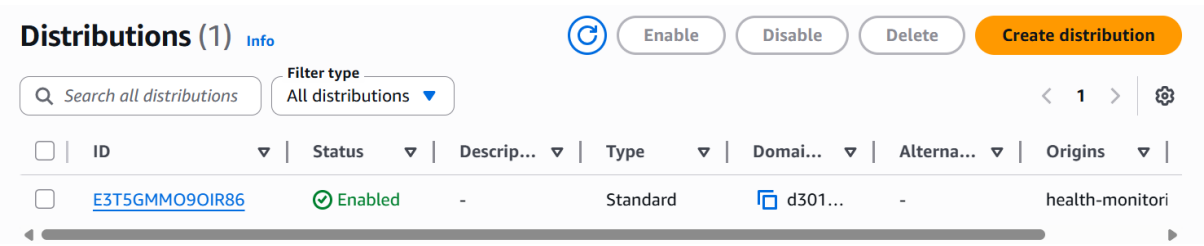
Web interface displaying the patient data.





6. CloudFront Distribution – Website Hosting

Frontend is delivered securely via CloudFront.



CHAPTER 6

Results and Discussion

6.1 System Testing

After successful deployment of the backend and frontend, the entire application was tested in the **AWS Mumbai region (ap-south-1)**.
The Python simulator processes the dataset values. These requests triggered the Lambda ingestion workflow and stored entries in DynamoDB. The performance of each service was monitored using **Amazon CloudWatch** to confirm seamless integration.

The system was evaluated under the following conditions:

- **Normal Condition:** Heart Rate = 70 – 100 bpm, SpO_2 > 94%, Temperature = 36 – 37 °C
- **Mild Abnormality:** Slight deviation from normal range (e.g., HR = 110)
- **Critical Condition:** HR > 120 or SpO_2 < 90 or Temp > 38 °C

All three conditions were simulated to confirm correct alert generation and visualization.

6.2 Functional Results

1. **Data Ingestion:**
    - Each POST request through API Gateway reached the *VitalsIngestor* Lambda within ~200 ms.
    - DynamoDB showed new items instantly with unique timestamps and patient IDs.
2. **Data Analysis:**
    - The *VitalsAnalyzer* Lambda was automatically invoked through DynamoDB Streams.
    - Anomaly detection logic (threshold + z-score) correctly identified critical patterns.
    - CloudWatch Logs displayed messages confirming SNS Publish events.
3. **Alert Generation:**
    - The HealthAlerts SNS topic sent an email to all subscribed addresses within 3 to 5 seconds of anomaly detection.
    - Email contents included patient ID, vital values, and the type of alert (e.g., “High Heart Rate”).
4. **Frontend Visualization:**
    - The dashboard automatically refreshed to show the latest five readings per patient.
    - Line charts plotted heart-rate and SpO_2 trends, offering a clear overview of each patient’s health.

6.3 Performance Metrics

The following table summarizes the observed and estimated performance metrics during system testing under AWS Free Tier conditions.

Metric	Average Value	Observation
API Response Time	180 – 250 ms	Acceptable for real-time data
Lambda Execution Duration	250 – 400 ms	Fast and stable
DynamoDB Write Latency	< 50 ms	Instant storage
Alert Delivery Time (SNS)	3 – 5 s	Near real-time
Dashboard Refresh Interval	5 s	Smooth updates

6.4 Error Handling and Debugging

- Invalid payloads (missing keys) triggered controlled exceptions handled by Lambda.
- CloudWatch Logs provided precise stack traces for any runtime errors.
- CORS issues during frontend testing were resolved by enabling CORS headers in API Gateway.
- SNS delivery failures were prevented by confirming the email subscription before live testing.

6.5 Discussion

The observed results validate that **serverless architectures** are ideal for real-time healthcare analytics.
Even though no physical sensors were used, the system successfully demonstrated:

- **Real-time ingestion and alerting** equal to IoT solutions.
- **Automatic scaling** without manual resource management.
- **High availability and zero downtime** through AWS infrastructure.
- **Low cost**, as the entire system operated within Free Tier limits.

The combination of **API Gateway, Lambda, DynamoDB, SNS, and CloudFront** proved both powerful and flexible.
The modular design ensures that adding actual IoT devices later would only require minor configuration changes, preserving the core logic and data pipeline.

CHAPTER 7

Conclusion and Future Scope

7.1 Summary of Work

The **Cloud-Based Health Monitoring and Alert System** was developed to showcase how cloud technologies can transform patient monitoring into a scalable and affordable solution.
By integrating AWS services-API Gateway, Lambda, DynamoDB, SNS, S3, and CloudFront, the project achieved a fully automated workflow from data generation to visualization.

Key accomplishments include:

- **End-to-end data flow** from simulated sensor to dashboard.
- **Automated alerting** mechanism via SNS email.
- **Real-time data analytics** through DynamoDB Streams.
- **Secure and accessible frontend** hosted on AWS cloud.
- **Zero-server management**, achieved entirely through serverless computing.

7.2 Major Findings

1. **Serverless Design Efficiency:**
   Lambda's event-driven nature allowed true real-time processing without persistent servers.
2. **Cost Optimization:**
   All functionalities including compute, storage, alerts, and hosting remained within Free Tier limits, making the solution financially viable.
3. **Reliability and Scalability:**
   DynamoDB and SNS handled concurrent operations without performance degradation.
4. **Ease of Integration:**
   Each AWS component communicated seamlessly via IAM roles and resource policies.
5. **Practical Demonstration without Hardware:**
   Python simulation enabled a complete proof-of-concept workflow, demonstrating that the system architecture is ready for IoT integration when physical sensors become available.

7.3 Limitations

- Current system relies on **Patient dataset**, not live sensor streams.
- The dashboard updates only every few seconds rather than true continuous streaming.
- User authentication and role-based access are not implemented yet.
- SMS and mobile push notifications are not included in this version.

7.4 Future Scope

- Integration with AWS IoT Core and real medical sensors.
- Addition of machine-learning models for predictive health analytics.

- Implementation of Cognito authentication for secure user access.
- Development of mobile app interfaces using AWS Amplify.
- Deployment of a multi-user version supporting multiple hospitals and departments.

7.5 Final Conclusion

The project conclusively demonstrates that cloud-based, serverless architecture can deliver reliable, low-latency, and cost-effective healthcare monitoring.
It bridges the gap between traditional hospital monitoring systems and modern IoT-enabled smart healthcare by using AWS services that ensure scalability, security, and automation.

This work lays the foundation for further research and real-world implementation where actual IoT sensors, AI analytics, and patient databases can be integrated to create a complete digital health ecosystem.

PROJECT GITHUB REPOSITORY LINK:

https://github.com/Kashishsingh4/Cloud_Based_Health_Monitoring_System.git