# Lab 4 Report: AWS Service Setup

Course: IoMT-Based Stress Monitoring System  
Student: Miltone  
Institution: Michigan Technological University  
Instructor: Dr. Ronghua Xu  
Duration: Week 4  
Lab Title: AWS Service Setup for IoMT Application Backend  
Objective: To configure AWS cloud services including DynamoDB, Lambda, and API Gateway, creating the foundation for serverless backend integration with the Flutter application.

## 1. Introduction

This lab marks the transition of the IoMT-Based Stress Monitoring System from a purely frontend application to a cloud-integrated ecosystem. The goal of Lab 4 was to establish the AWS backend infrastructure that would later support data exchange between the mobile Flutter client and wearable IoMT devices like Fitbit. By setting up DynamoDB, Lambda, and API Gateway, this lab provided the fundamental building blocks for a scalable, secure, and serverless data architecture.

## 2. Objectives

By the end of this lab, the following objectives were achieved:

* Create and configure an AWS account using the free tier.
* Set up DynamoDB to manage patient and Fitbit data storage.
* Develop and test Lambda functions to process backend operations.
* Integrate Lambda with API Gateway to expose REST API endpoints.
* Validate all services through AWS Console and Postman.

## 3. Overview of AWS Services Used

The following AWS services were utilized to establish the backend for the IoMT application:

|  |  |
| --- | --- |
| AWS Service | Function in Project |
| AWS Lambda | Provides serverless compute power for executing backend code without managing servers. Used for API data processing and Fitbit integration. |
| Amazon DynamoDB | NoSQL database for storing Fitbit access tokens, user data, and daily stress readings. Enables fast and reliable key-value access. |
| Amazon API Gateway | Acts as the entry point for mobile apps to communicate securely with Lambda functions. |
| AWS Identity and Access Management (IAM) | Manages permissions and roles for secure Lambda and DynamoDB interactions. |
| AWS CloudWatch | Monitors logs and performance metrics for Lambda executions and API calls. |

## 4. Setup and Configuration Process

1. Stage 1: AWS Account and Access Setup

Created an AWS account using the free tier plan. Enabled required services such as DynamoDB, Lambda, and API Gateway. Verified region configuration (US East 1 - N. Virginia) and set up IAM credentials for administrative access.

1. Stage 2: DynamoDB Configuration

1. Opened DynamoDB console and created a test table named 'PatientData'.  
2. Defined Partition Key as 'patient\_id' (String) and Sort Key as 'collect\_date' (String).  
3. Added sample items manually for CRUD testing.  
4. Enabled default read/write capacity (on-demand mode) for free-tier optimization.  
5. Verified operations through the AWS console and Python script execution using boto3.

1. Stage 3: AWS Lambda Setup

1. Created a new Lambda function named 'hello\_lambda' using Python 3.12 runtime.  
2. Defined a simple test function to return 'Hello World' response.  
3. Configured execution role with AmazonDynamoDBFullAccess to allow database interaction.  
4. Increased function timeout to 30 seconds to handle future data operations.  
5. Deployed and tested Lambda directly from the AWS console using test event JSON.

1. Stage 4: API Gateway Integration

1. Navigated to API Gateway and created a new REST API named 'IoMT\_API'.  
2. Added a new resource '/testAPI' and linked POST method to 'hello\_lambda'.  
3. Deployed the API under a stage named 'dev'.  
4. Retrieved Invoke URL for external testing.  
5. Verified API call via Postman, successfully receiving Lambda response as JSON output.

1. Stage 5: Testing and Validation

Used Postman and AWS console for validation. Verified that Lambda executed correctly and responded to API Gateway triggers. CRUD operations were tested on DynamoDB items, confirming successful read, write, update, and delete operations.

## 5. Test Results and Validation

All configured AWS components were tested successfully. Below are summarized results:

|  |  |
| --- | --- |
| Component | Validation Result |
| Lambda Function Test | Returned correct response: {'statusCode': 200, 'body': 'Hello World'}. |
| API Gateway | POST request successfully triggered Lambda and returned JSON response. |
| DynamoDB CRUD | All operations (Create, Read, Update, Delete) executed successfully via console and boto3 SDK. |
| IAM Role | Lambda function executed with DynamoDB permissions verified via policy simulation. |
| CloudWatch Logs | Recorded successful Lambda invocation events with no runtime errors. |

## 6. Challenges and Resolutions

|  |  |  |
| --- | --- | --- |
| Challenge | Description | Resolution |
| Permission Denied Errors | Lambda failed to access DynamoDB due to missing IAM policy. | Attached AmazonDynamoDBFullAccess policy and redeployed Lambda. |
| API Gateway Timeout | Initial response delay caused HTTP timeout errors. | Increased Lambda timeout and redeployed API. |
| DynamoDB Schema Error | Incorrect partition key type caused write failure. | Updated schema with correct data type and redeployed table. |
| Postman 403 Error | Unauthorized API calls. | Enabled CORS and verified API key authorization. |

## 7. Outcome and Deliverables

* AWS DynamoDB table successfully created and tested for CRUD operations.
* Lambda function deployed and verified using test events.
* API Gateway configured to trigger Lambda through REST API call.
* IAM roles created to secure communication between services.
* Documentation and screenshots of successful API call via Postman.

## 8. Reflection

This lab introduced the cloud computing backbone of the IoMT system. By configuring AWS Lambda and DynamoDB, I gained a practical understanding of serverless computing and cloud-based data architecture. Each AWS service acted as a vital organ in the digital anatomy of the stress monitoring system. The experience reinforced the importance of security, scalability, and modularity in modern healthcare informatics.

“The cloud is more than a network; it’s the invisible hand that holds every heartbeat of modern technology.”

## 9. Forward Outlook

The established AWS services will form the operational backbone for the next phase (Lab 5), where Fitbit API integration will be implemented. The same Lambda and API Gateway configuration will be extended to communicate with Fitbit servers, allowing the system to pull physiological data and store it in DynamoDB for analysis. This marks the beginning of a true Internet of Medical Things (IoMT) pipeline.

✅ Lab 4 Successfully Completed. The AWS backend is operational and ready to support real data integration in the upcoming Fitbit API development stage.