IoT-Based Smart Home Automation using AWS IoT Core

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

Sanda Manojkumar

2210030429

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **"IoT-Based Smart Home Automation using AWS IoT Core"** which is a experimental &/ theoretical &/ Simulation&/ hardware work carried out by Sanda Manojkumar (2210030429), in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi

Course Coordinator

Dr. Arpita Gupta

Head of the Department

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

1. INTRODUCTION

In today's digital era, Smart Home Automation is revolutionizing the way we interact with and manage our living spaces. By leveraging the Internet of Things (IoT), smart home systems connect various household devices—such as lights, fans, locks, and air conditioners—to a central control system, enabling users to operate them remotely and intelligently. This not only enhances user convenience and comfort but also improves security, reduces energy consumption, and contributes to sustainable living practices.

Traditionally, smart home systems required complex hardware setups and local automation hubs. However, with the advancement of cloud computing and IoT platforms like Amazon Web Services (AWS), it is now possible to build powerful, scalable, and cost-effective smart home systems entirely in the cloud—without the need for physical hardware.

This project focuses on developing a cloud-based Smart Home Automation system using AWS IoT Core, which acts as a central broker to receive and distribute control messages via the MQTT protocol. The system simulates control of multiple smart devices—such as smart lights, fans, locks, and air conditioners—entirely through the AWS Console. The core functionality is built using various AWS services:

- AWS IoT Core enables secure communication between the devices and the cloud.
- AWS Lambda serves as the backend compute service to process incoming device messages.
- Amazon DynamoDB is used to persist and track device states and actions.
- Amazon SNS (Simple Notification Service) provides real-time alerts via SMS or email whenever a device action is performed.

The project does not use physical sensors or microcontrollers, making it an ideal prototype for understanding cloud-based IoT workflows, device communication protocols, and serverless architecture. It demonstrates how cloud technologies can be leveraged to simulate real-world IoT applications in a scalable, reliable, and easy-to-manage way.

By the end of this project, users will gain a comprehensive understanding of how to design and implement a complete IoT-based smart home system using only cloud-native tools and services offered by AWS.

2. AWS SERVICES USED AS PART OF THE PROJECT

To build a fully cloud-based Smart Home Automation system, a combination of powerful and scalable AWS services was used. Each service played a specific role in ensuring smooth communication, secure data handling, real-time processing, and user notifications.

AWS IoT Core served as the central communication hub of the project. It allowed simulated IoT devices to securely publish and receive messages using the MQTT protocol. Acting as a fully managed MQTT broker, AWS IoT Core ensured that device commands such as turning on a light or locking a door could be sent to the cloud, where further processing would occur.

AWS Lambda was used as the serverless compute engine in the project. Whenever a device action message was received by AWS IoT Core, it triggered a Lambda function. This function parsed the message payload, handled the business logic, stored the device action in the database, and published notifications through SNS. Lambda made the system event-driven and scalable without the need for server management.

Amazon DynamoDB acted as the persistent data store for smart device actions and states. As a fast and flexible NoSQL database, DynamoDB enabled efficient storage and retrieval of device activity logs. It recorded important information such as device IDs, action types, timestamps, and additional metadata, ensuring that the system maintained a real-time overview of the smart home environment.

Amazon SNS (Simple Notification Service) was responsible for sending real-time alerts to users. Whenever a device action was processed by Lambda, SNS sent out an SMS or email notification to inform users of the event. This added a layer of interactivity and ensured that users were instantly updated on the status of their smart devices.

These services, when combined, created a seamless and fully cloud-native solution for smart home automation without requiring any physical hardware or local infrastructure.

3. STEPS INVOLVED IN SOLVING PROJECT PROBLEM STATEMENT

To build a fully functional, cloud-based Smart Home Automation system using AWS services without any physical devices, a structured and step-by-step approach was followed. Each step focused on a specific component of the system—from device creation to communication and alert mechanisms.

Step 1: Registering IoT Things

The first step involved registering four virtual IoT devices using AWS IoT Core. These devices represent real-world smart appliances commonly found in modern homes: a smart light, smart fan, smart door lock, and smart air conditioner. In the AWS IoT Core console, each device was created as an individual "Thing" with a unique name:

- smart_light_01
- smart_fan_01
- smart_lock_01
- smart_ac_01

This process laid the foundation for establishing a virtual identity for each device, enabling them to securely communicate with the AWS cloud. Although no physical hardware was used, this simulation effectively mimicked real-world IoT device behavior.[1]

Step 2: Setting Up MQTT Communication

Next, MQTT (Message Queuing Telemetry Transport)—a lightweight messaging protocol widely used in IoT—was configured using the built-in MQTT test client in AWS IoT Core. A topic named smartHome/actions was created to serve as the communication channel between the user and the devices. The test client subscribed to this topic to monitor incoming messages and simulate control actions such as turning devices on or off.

This MQTT setup allowed developers to simulate sending commands to multiple devices, replicating how a mobile app or voice assistant might communicate with smart devices in a real-world smart home setup.[5]

Step 3: Create and Configure Lambda Function

A Lambda function named ProcessSmartHomeActions was developed to serve as the backend processor. This function was triggered automatically when a new message was published to the MQTT topic. The function's logic was designed to:

- Parse the incoming JSON payload
- Loop through multiple device actions
- Log or update the device state in DynamoDB
- Send notifications using Amazon SNS

The use of AWS Lambda ensured that the entire system remained serverless, cost-effective, and scalable, as the function only ran when needed and did not require manual provisioning of servers.[2]

Step 4: Attach Permissions Using IAM

To enable secure interaction between AWS services, an IAM (Identity and Access Management) role was created and assigned to the Lambda function. This role was granted the following permissions:

- Publish and subscribe to AWS IoT Core
- Read and write data in Amazon DynamoDB
- Publish messages to Amazon SNS

Proper IAM configuration ensured that the Lambda function could interact seamlessly with all required services while adhering to strict access control policies to maintain system security.[6]

Step 5: Set Up DynamoDB

A DynamoDB table named SmartHomeData was created to store the state and history of actions performed by the devices. The table used device_id as the partition key to uniquely identify each device. Other attributes included action, timestamp, and status.

This NoSQL data store played a critical role in enabling real-time updates and persistent storage of device activity, which could be later used for reporting, monitoring, or analytics.[4]

Step 6: Create SNS Topics and Subscriptions

To ensure users were notified whenever a device action was performed, an Amazon SNS topic named SmartHomeAlerts was created. Subscriptions were added to this topic in the form of:

- A valid email address
- A verified mobile number

Once configured, this setup allowed the system to send immediate SMS or email alerts whenever the Lambda function detected an action, such as turning on a light or locking the door. This enhanced user awareness and system interactivity.[3]

Step 7: Publish MQTT Messages to Simulate Device Actions

In the final step, MQTT messages were published to the smartHome/actions topic using the MQTT test client. The messages were structured in JSON format and included multiple devices and their corresponding actions. For example:

```
{
  "devices": [
    { "device_id": "smart_light_01", "action": "TURN_ON" },
    { "device_id": "smart_fan_01", "action": "TURN_OFF" },
    { "device_id": "smart_lock_01", "action": "LOCK" },
    { "device_id": "smart_ac_01", "action": "TURN_ON" }
  ]
}
```

Fig3.1 : Things in Json format

Upon publishing, the message triggered the Lambda function, which updated DynamoDB and sent SMS/email alerts via SNS. The results were validated by checking the DynamoDB entries and confirming alert delivery to the user.[5]

4. STEPWISE SCREENSHOTS WITH BRIEF DESCRIPTION

This section provides a visual walkthrough of the major steps involved in setting up the IoT-based Smart Home Automation system on AWS. Each step is accompanied by a brief description of the action performed and its significance in the project workflow.

Step 1: Creating IoT Things

In this step, four virtual IoT devices were registered in the AWS IoT Core service. These devices represented common smart appliances:

- smart_light_01 (Smart Light)
- smart_fan_01 (Smart Fan)
- smart_lock_01 (Smart Lock)
- smart_ac_01 (Smart Air Conditioner)

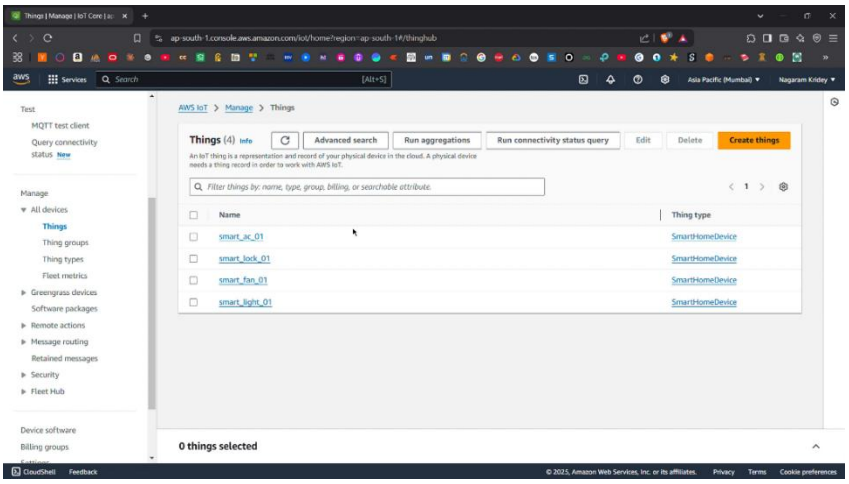

Fig 4.1 : Creating IoT Things

Each device was created under the "Things" section, establishing a unique identity in the cloud. These Things act as endpoints for MQTT communication and simulate real-world IoT devices in a smart home environment.[1]

Step 2: MQTT Subscription

The MQTT test client was used to establish a communication channel between the user and the virtual devices. By subscribing to the topic smartHome/actions, the system could listen for incoming control messages in real-time.

This topic acted as the central medium where commands (such as TURN_ON, TURN_OFF, LOCK, UNLOCK, etc.) were sent to control smart devices. Subscribing to this topic was essential for monitoring the flow of actions and verifying message delivery.[5]

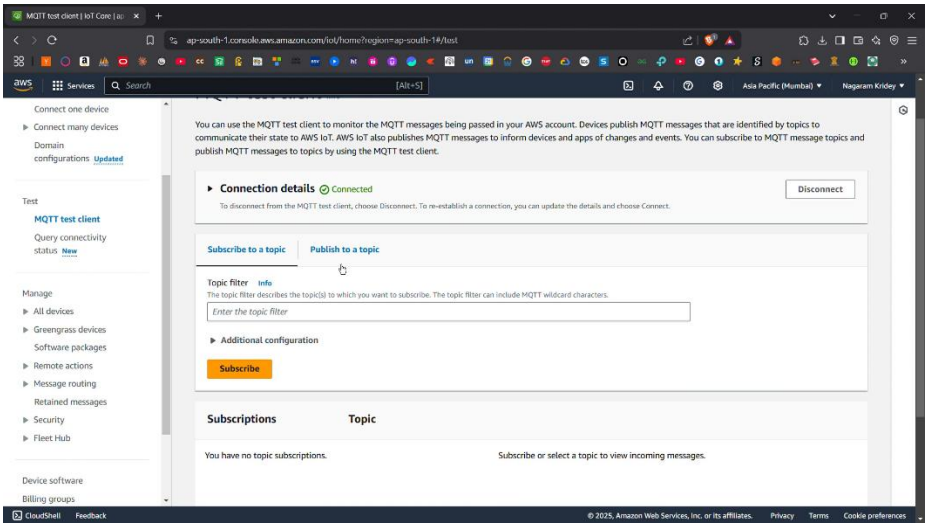

Fig 4.2 : : MQTT Subscription.

Step 3: Lambda Function Configuration

A Lambda function named ProcessSmartHomeActions was created to process the incoming MQTT messages. The function's code was designed to:

- Parse a JSON payload containing a list of device commands
- Loop through each device
- Store the command in DynamoDB
- Trigger SNS alerts

The Lambda function was event-driven, meaning it only ran when new messages were published to the MQTT topic. This serverless architecture ensured minimal cost and high efficiency.[2]

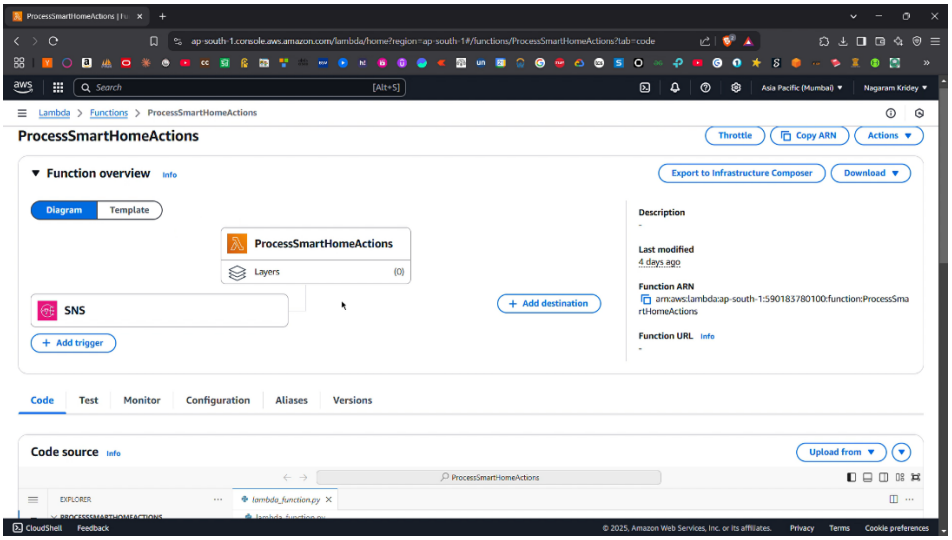

Fig 4.3 : Lambda Function Configuration.

```
import json
import boto3

dynamodb = boto3.resource('dynamodb')
sns = boto3.client('sns')

SNS_TOPIC_ARN = "arn:aws:sns:your-region:your-account-id:SmartHomeAlerts"

table = dynamodb.Table('SmartHomeData')

def lambda_handler(event, context):
    print("Received event: ", event)
    if "devices" in event:
        for device in event["devices"]:
            store_device_action(device)
            send_notification(device)
    else:
        store_device_action(event)
        send_notification(event)

    return {
        'statusCode': 200,
        'body': json.dumps('Actions processed and notifications sent
successfully!')
    }

def store_device_action(device):
    """ Stores device action in DynamoDB """
    device_id = device.get("device_id", "unknown")
    device_type = device.get("device_type", "unknown")
    action = device.get("action", "unknown")
    table.put_item(
        Item={
```

```
            'device_id': device_id,
            'device_type': device_type,
            'action': action
        }
    )
    print(f"Stored: {device_id} - {action}")

def send_notification(device):
    """ Sends an SMS or email notification via SNS """
    device_id = device.get("device_id", "unknown")
    action = device.get("action", "unknown")

    message = f" Smart Home Alert: {device_id} performed action {action}."

    response = sns.publish(
        TopicArn=SNS_TOPIC_ARN,
        Message=message
    )

    print(f"Notification sent for {device_id} - {action}")
```

Fig 4.4 : lamda_function.py.

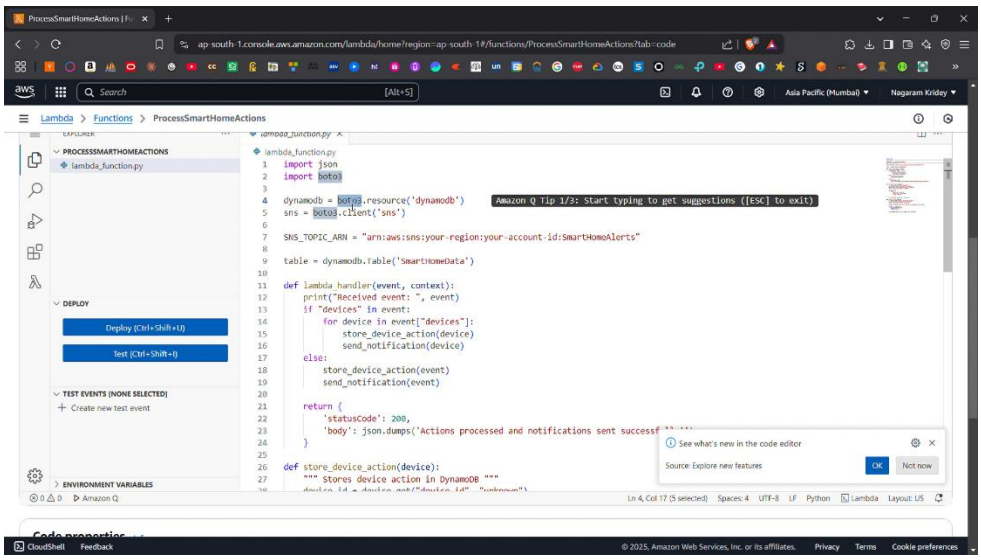

Fig 4.5 : lamda_function.py in aws console.

Step 4: IAM Permissions Setup

To enable the Lambda function to securely interact with other AWS services, an IAM Role was configured and attached. The following policies were added to the role:

- AWSIoTFullAccess: Grants full access to IoT Core for subscribing to and receiving messages.

- AmazonDynamoDBFullAccess: Enables reading from and writing to the DynamoDB table.
- AmazonSNSFullAccess: Allows publishing notifications via SNS.

These permissions were crucial to ensure that each service could perform its assigned role while maintaining a secure system architecture.[6]

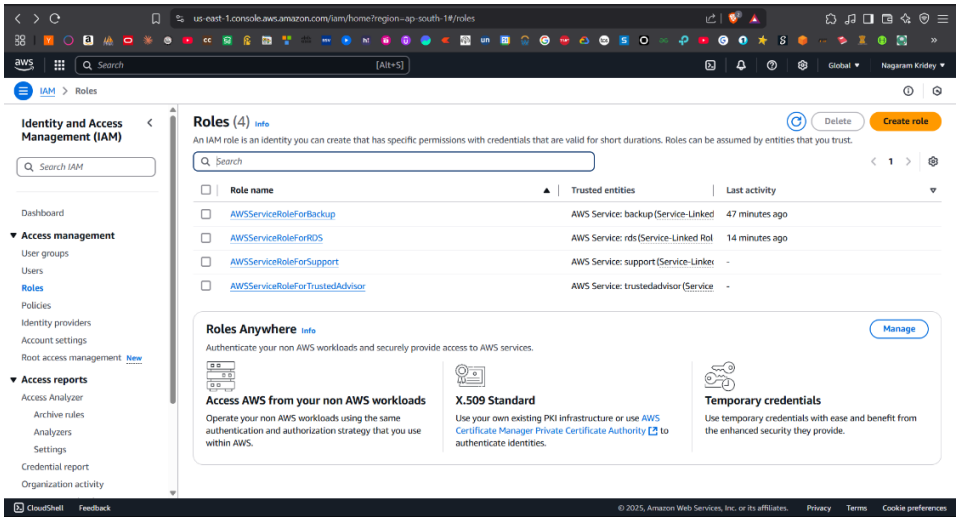

Fig 4.6 : IAM Permissions Setup

Step 5: DynamoDB Table Configuration

A DynamoDB table named SmartHomeData was created to persist the state and activity of all registered devices. Each record contained the following fields:

- device_id (Partition key): Uniquely identifies the smart device
- action: The command executed (e.g., TURN_ON, LOCK)
- timestamp: The exact time the action was triggered

This NoSQL database provided high-speed, low-latency storage for the device actions, allowing easy retrieval and analysis of smart home activity over time.

Overall, the SmartHomeData table in DynamoDB plays a critical role in the backend infrastructure, enabling persistent, scalable, and intelligent data management. It enhances the system's reliability, supports decision-making through data insights, and contributes to the overall goal of building a smart, connected, and context-aware home automation solution.[4]

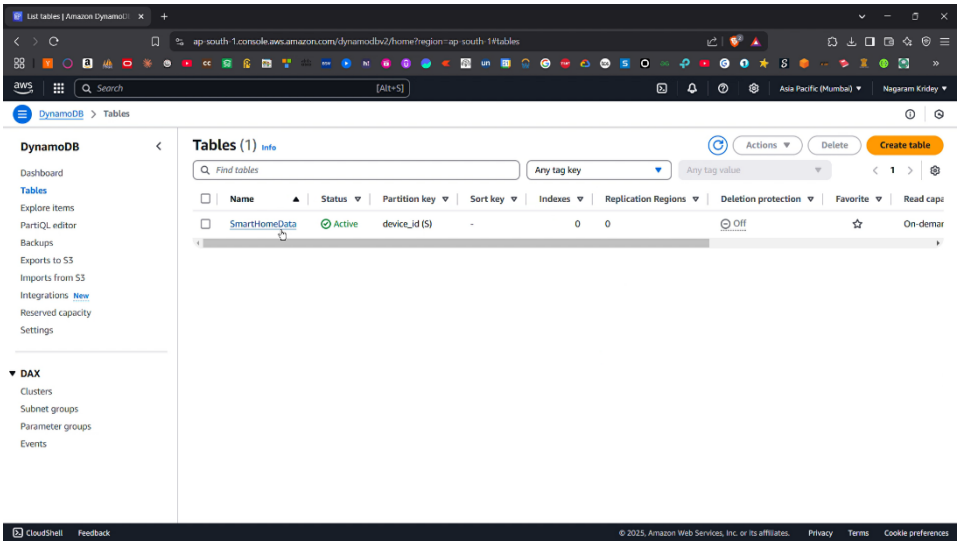

Fig 4.7 : DynamoDB Table Configuration

Step 6: SNS Topic and Subscriptions

An Amazon SNS (Simple Notification Service) topic named SmartHomeAlerts was created to handle user notifications for smart device activities. This topic acts as a central hub for broadcasting messages to multiple subscribers. Two types of subscriptions were configured for this topic:

- A verified email address, which receives notifications via email.
- A verified mobile number, which receives notifications via SMS.

The alerts sent via SNS are automatically triggered by AWS Lambda functions, which act as the logic layer in the system. These Lambda functions are invoked in response to MQTT messages published by the simulated IoT devices through AWS IoT Core. Each device publishes status updates or command acknowledgments to specific MQTT topics, which are monitored by predefined rules or triggers that activate the corresponding Lambda functions. Once invoked, the Lambda function processes the incoming message, extracts relevant information (such as the device ID, device type, event type, and timestamp), and formats it into a meaningful notification. This message is then published to the SmartHomeAlerts SNS topic, instantly notifying all subscribed endpoints.[3]

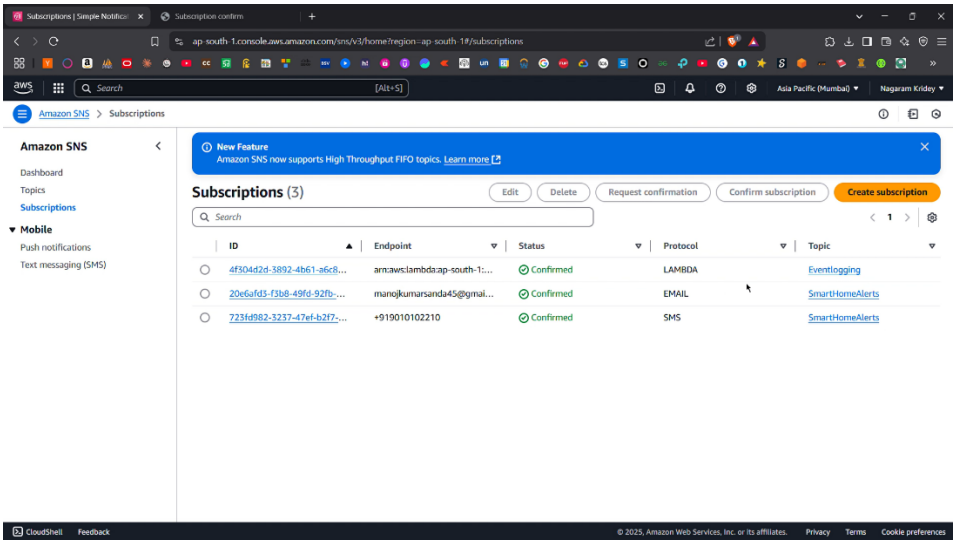

Fig 4.8 : SNS Topic and Subscriptions

Step 7: MQTT Publish for Device Actions

Finally, MQTT messages simulating device commands were published to the topic smartHome/actions. The JSON payload included multiple devices and their respective actions.After publishing, the system workflow was triggered: Lambda processed the message, updated the database, and alerts were instantly sent via SNS. This validated the successful integration of all components and completed the smart home automation cycle.

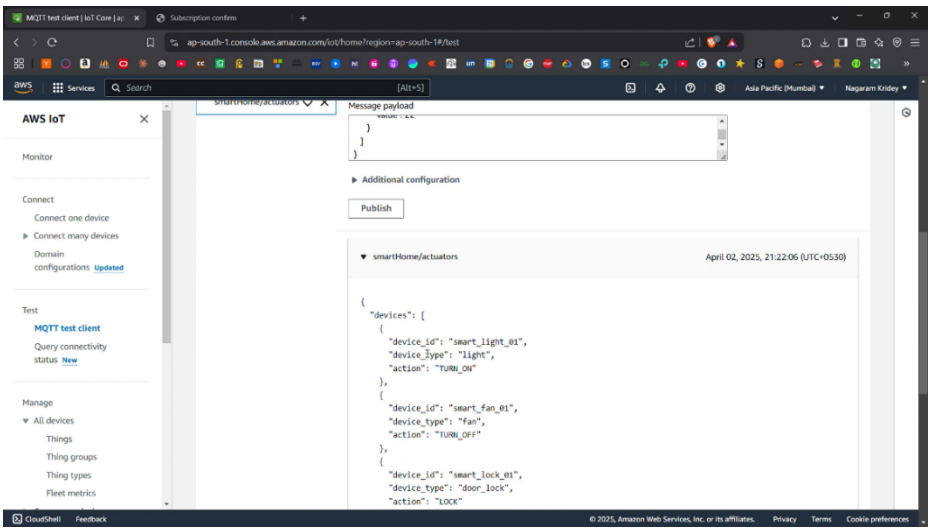

Fig 4.9: MQTT Publish for Device Actions

5. LEARNING OUTCOMES

This project offered valuable insights into cloud-based IoT development and helped build foundational skills in working with a suite of AWS services. The following are the key learning outcomes derived from the successful implementation of the IoT-Based Smart Home Automation system:

Hands-on Experience with AWS IoT Core and MQTT Protocol : Gained practical knowledge of how AWS IoT Core functions as a cloud platform to connect, manage, and communicate with IoT devices. Learned how the MQTT protocol works as a lightweight, publish-subscribe model to exchange real-time messages between devices and cloud applications. Setting up MQTT topics and using the AWS test client provided a solid understanding of how IoT messaging protocols operate in real-world environments.

Integration of Serverless Computing Using AWS Lambda : Explored the power and flexibility of AWS Lambda to run code in response to events without managing any servers. By configuring the Lambda function to trigger upon MQTT message receipt, the project demonstrated how to handle device control logic and backend automation in a scalable and efficient way.

Data Persistence with Amazon DynamoDB : Understood how to store and manage IoT-generated data using DynamoDB, AWS's fast and flexible NoSQL database. Learned how to create tables, define primary keys, and structure records to store information like device states, actions, and timestamps. This knowledge is crucial for implementing state tracking, usage analytics, and history logs in real-world applications.

Real-Time Notification Automation with Amazon SNS : Learned to set up an Amazon SNS topic and configure it to send email and SMS notifications to users in real-time. This showcased the ability to automate alerts and enhance user engagement by providing immediate feedback on device activity. Subscribing to and managing different notification channels helped understand event-driven communication in cloud environments.

Cloud-Native Smart Home System Without Physical Devices : Successfully simulated a complete Smart Home Automation system purely using cloud-based components—eliminating the need for physical hardware. This reinforced the concept that powerful IoT applications can be designed, tested, and even deployed virtually, using services like AWS IoT Core, Lambda, DynamoDB, and SNS.

Improved Cloud Architecture Design Skills : Built a deeper understanding of how multiple AWS services can be orchestrated together to create scalable and secure solutions. This project highlighted key architectural best practices, such as using IAM roles for permission control, serverless logic for processing, and asynchronous communication models for real-time interaction.

Problem Solving and Debugging in a Cloud Environment : Faced and resolved several challenges such as connectivity issues with MQTT, IAM permission errors, and SNS subscription confirmations. These experiences helped develop strong troubleshooting and problem-solving skills in working with cloud platforms and real-time systems.

6. CONCLUSION

This project demonstrates the practical potential of building a scalable, secure, and intelligent Smart Home Automation system using AWS cloud-native services. By simulating devices and leveraging tools like AWS IoT Core, Lambda, DynamoDB, and SNS, it successfully showcases seamless communication, real-time processing, efficient data storage, and proactive user notifications—all without physical hardware. The architecture's modularity and serverless design offer a flexible foundation for future enhancements, making it an ideal model for modern IoT applications.AWS Lambda is employed to handle real-time data processing in a serverless manner. Lambda functions are triggered by device messages, enabling logic execution such as state analysis, rule enforcement, and triggering downstream services. This approach not only reduces operational overhead but also scales automatically based on demand, ensuring high availability and cost-efficiency.

7. REFERENCES

[1] AWS IoT Core Documentation: https://docs.aws.amazon.com/iot

[2] AWS Lambda Documentation: https://docs.aws.amazon.com/lambda

[3] AWS SNS Documentation: https://docs.aws.amazon.com/sns

[4] Amazon DynamoDB Documentation: https://docs.aws.amazon.com/dynamodb

[5] AWS MQTT Test

Client: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html

[6] AWS IAM Roles: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html