BotReminder

A cloud based reminder for regular users.

Project By [GROUP 4]:-

Udatya Deb

Prakash Pratap Singh

Manan Mal

Project Report

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN COMPUTER SCIENCE ENGINEERING

Under the guidance of MR. SHOUVIK SARKAR

ARDENT COMPUTECH PVT LTD (AN ISO 9001:2015 CERTIFIED)

SDF Bldg, Module 132, GP Block, Sector V, Bidhanagar, Kolkata- 700091

NETAJI SUBHASH ENGINEERING COLLEGE, KOLKATA

September– December 2021

- Title of the Project: BotReminder (A cloud based reminder for regular users.)
- Project Members
  - UDATYA DEB
  - PRAKASH PRATAP SINGH
  - MANAN MAL

Table of Contents

Table of Content

Acknowledgement

Success of any project depends largely on the encouragement and guidelines of many others. We take this sincere opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project work. Our heartfelt thanks to Mr. Sanjoy Banerjee , for providing us the opportunity to develop the project at Ardent Computech Pvt. Ltd

We would like to show our greatest appreciation to Mr. Shouvik Sarkar , Project Manager at Ardent , Kolkata. We always feel motivated and encouraged every time by his valuable advice and constant inspiration ; without his encouragement and guidance , this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees , project assistants and other members at Ardent Computech Pvt. Ltd for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project , was vital for the success of this project.

Objective of the project

The purpose of this project is to create a chatbot that can organize an individual's schedule for a day and can notify him at the time when a task is scheduled.

Overall Plan (System Design)

Data given by user

Calls the lambda Function

reminderBOTFunction

Stores Data in DynamoDB

AWS Eventbridge

Calls Lambda Function every 1min

sendEMAIL

Sends the required notification at specified time

Sends stored data to lambda function

Amazon AWS Dynamodb

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets offload the administrative burdens of operating and scaling a distributed database so that a user don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

With DynamoDB, a user can create database tables that can store and retrieve any amount of data and serve any level of request traffic. He can scale up or scale down the tables' throughput capacity without downtime or performance degradation. He can use the AWS Management Console to monitor resource utilization and performance metrics.

DynamoDB provides on-demand backup capability. It allows a user to create full backups of your tables for long-term retention and archival for regulatory compliance needs.

Amazon Lex

Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text. With Amazon Lex, the same conversational engine that powers Amazon Alexa is now available to any developer, enabling any user to build sophisticated, natural language chatbots into any new and existing applications. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) so a person can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.

Amazon Lex provides pre-built integration with AWS Lambda, and can easily be integrated with many other services on the AWS platform, including Amazon Cognito, AWS Mobile Hub, Amazon CloudWatch, and Amazon DynamoDB. Integration with Lambda provides bots access to pre-built serverless enterprise connectors to link to data in SaaS applications, such as Salesforce, HubSpot, or Marketo.

AWS Lambda

Lambda is a compute service that's used to run a code without provisioning or managing servers. Lambda runs the code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. With Lambda, a code can be run for virtually any type of application or backend service.

A developer organizes the code into Lambda functions. Lambda runs the function only when needed and scales automatically, from a few requests per day to thousands per second.

Lambda functions can be invoked using the Lambda API, or Lambda can run the functions in response to events from other AWS services. For example, we can use Lambda to:

- Build data-processing triggers for AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.
- Process streaming data stored in Amazon Kinesis.
- Create your own backend that operates at AWS scale, performance, and security.

Amazon CloudWatch Events

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Using simple rules that can be quickly set up, a user can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. CloudWatch Events responds to these operational changes and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information. We can also use CloudWatch Events to schedule automated actions that self-trigger at certain times using cron or rate expressions.

A developer can configure the following AWS services as targets for CloudWatch Events:

- Amazon EC2 instances
- AWS Lambda functions
- Delivery streams in Amazon Kinesis Data Firehose
- Log groups in Amazon CloudWatch Logs
- Amazon ECS tasks
- Systems Manager Run Command
- Systems Manager Automation
- AWS Batch jobs

Basic codes (Python)

(For Storing information into the dynamodb from lex)

```
# Code For Our Chatbot to Store Informations From User

# import json
# import datetime
# import os
# import time
# import dateutil.parser
# import logging
import boto3
import uuid

# logger = logging.getLogger()
# logger.setLevel(logging.DEBUG)

dyn_client = boto3.client('dynamodb')
TABLE_NAME = "tableReminder"
```

```
# def close(fulfillment_state, message):
# # def close(fulfillment_state, message,slots,intent_name):
#     response = {
# #         'sessionAttributes': session_attributes,
#         'dialogAction': {
#             'type': 'Close',
#             'fulfillmentState': fulfillment_state,
#             'message': message
#         }
#         # 'intentName': intent_name,
#         # "slots": {
#         #     "PhoneNo": slots['PhoneNo'],
#         #     "TimeReminder": slots['TimeReminder'],
#         #     "UserName": ["UserName"],
#         #     "topic": ['topic']
#         # }
#     }
#     return response
```

```
# # .....DynamDB save.....

def save(user_name, email, topic, timeReminder):
    id = str(uuid.uuid4())
    data = dyn_client.put_item(
        TableName=TABLE_NAME,
        Item={
            'id': {
                'S': id
            },
            'user_name': {
                'S': user_name
            },
            'timeReminder': {
                'S': timeReminder
            },
            'email': {
                'S': email
            },
            'topic': {
                'S': topic
            }
        }
    )
```

```
# def take(intent_request):
#     slots = intent_request['currentIntent']['slots']
#     user_name = slots['UserName']
#     phone = slots['PhoneNo']
#     topic = slots['topic']
#     timeReminder = slots['TimeReminder']



#     # .....Validation function.....

#     # logger.debug(intent_request['invocationSource'])

#     # if intent_request['invocationSource'] == 'DialogCodeHook':
#     #     if len(phone)!=13:
#     #         return "Invalid phone number"

#     save(user_name, phone, topic, timeReminder)
```

```
#     return close(
#         # session_attributes,
#         'Fulfilled',
#         {
#             'contentType': 'Plaintext',
#             'content': 'Thanks {}, we have recorded your
survey.'.format(user_name)
#         },
#         # slots,
#         # intent_name
#     )


# def dispatch(intent_request):
#     logger.debug('dispatch userId={},
intentName={}'.format(intent_request['userId'],
intent_request['currentIntent']['name']))
#     # # logger.debug('dispatch
intentName={}'.format(intent_request['intentName']))
#     # intent_name = intent_request['currentIntent']['name']
#     # # intent_name = intent_request['intentName']
#     # if intent_name == 'botReminder':
#     return take(intent_request)
```

```
def dispatch(event):
   # logger.debug(event)
   # logger.debug(event['currentIntent']
   # logger.debug(event['currentIntent']['slots'])
   slots = event['currentIntent']['slots']
   user_name = slots['UserName']
   email = slots['Email']
   topic = slots['topic']
   timeReminder = slots['TimeReminder']
   save(user_name, email, topic, timeReminder)
   return {
      "dialogAction": {
         "type": 'Close',
         "fulfillmentState":'Fulfilled',
         "message": {
            "contentType":"PlainText",
            "content":"Your reminder is succesfully set. Have a good
day!"
         }
        #  responseCard
      }
   }
   # return response
```

```
# def lambda_handler(event, context):
#     os.environ['TZ'] = 'Asia/Singapore'
#     time.tzset()
#     # logger.debug('event={}'.format('event'))
#     logger.debug('event.bot.name={}'.format(event['bot']['name']))
#     # logger.debug(event['invocationSource'])
#     return dispatch(event)

def lambda_handler(event,context):
    # logger.debug('event={}'.format(event))
    return dispatch(event)
```

For accessing data from dynamodb and send the reminder mail


```
import boto3
import smtplib
from datetime import datetime
from zoneinfo import ZoneInfo
# import smtplib
from email.message import EmailMessage

def lambda_handler(event,context):
    client = boto3.resource("dynamodb")
    table = client.Table("tableReminder")
    tableReminder = table.scan()['Items']
    tableReminder_list = []
    i=0
    for x in tableReminder:
        tableReminder_list.insert(i,x)
        i+=1
    now = datetime.now(tz=ZoneInfo('Asia/Kolkata'))
    print(tableReminder_list)
```

```
for i in tableReminder_list:
    dt_string = now.strftime("%H:%M")
    a = i['timeReminder']
    # a = '14:12'
    if(dt_string == a):
        msg = EmailMessage()
        msg['Subject'] = 'Your Reminder'
        msg['From'] = 'BotReminder'
        msg['To'] = i['email']
        msg.set_content(f"Topic: {i['topic']} at {a}")

        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.login('botreminderaws@gmail.com', 'Pass@123')
        server.send_message(msg)
        server.quit
        # response = table.delete_item(Key={"id":i['id']})
        table.delete_item(Key={"id":i['id']})
```

IMPLEMENTATION AND TESTING

- A Software System Test Plan is a document that describes the objectives , scope, approach and focus of software testing effort.
- The process of preparing a test plan is a usual way to think the efforts needed to validate the software and ask ourselves, why will it be accepted.
- The complete document will help people outside the test group, understand why should they use the software and how will they be able to use it. Also they will get to know the process of the software.
- It should be through enough to be useful. Also everyone outside the test group will be able to read and understand it.

INRODUCTION

- Testing is the process of running a system with the intention of finding errors.
- Testing enhances the integrity of a system by detecting deviations in design and errors in the system.
- Testing aims at detecting error-prone areas. This helps in the prevention of the errors in a system.
- Testing also adds value to the product by conforming to the user's requirements.
- The main purpose of Testing is to detect errors and error-prone areas in a system.
- Testing must be thorough and well planned. A partially tested system is as bad as an untested system. And the price of an untested and under-tested system is high.
- The implementation is the final and important phase. It involves usertraining, system testing in order to ensure successful running of the proposed system.
- The user tests the system and changes are made according to their needs.
- The Testing involves the Testing of the developed system using various kinds of data. While testing, errors are noted and correctness is the mode.

STEPS FOR SYSTEM TESTING :-

1.Test Environment Setup :- Create testing environment for the better quality of testing.

2.Create Test Case :- Generate test case for the testing process.

3.Create Test Data

4.Execute Test Case

5.Defect Reporting

6.Regression Testing

7.Log Defects

8.Retest

OBJECTIVES OF TESTING

The objective of our test plan is to find and report as many bugs as possible to improve the integrity of our program. Although exhaustive testing is not possible, we will exercise a broad range of tests to achieve our goal. Our user interface to utilize these these functions, is designed to be user-friendly and provide an easy manipulation of the tree. The application will only be used as a demonstration tool, but we would like to ensure that it could be run from a variety of platforms with little impact on performance and usability.

PROCESS OVERVIEW :-

The following represents the overall flow of the Testing Process :-

- Identify the requirements to be tested. All Test Cases shall be derived using the current Program Specifications.
- Identify which particular Test(s) will be used to test each module.

- Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
- Identify the expected results for each test.
- Document the test case configuration , test data , and expected results.
- Perform the test(s).
- Document the test data , test cases , and test configuration used during the testing process. The information shall be submitted via the Unit/System Test Report (STR).

- Successful unit testing is required before the unit is eligible for component integration/system testing.
-
- Unsuccessful testing requires a Bug Report Form to be generated. This document shall describe the test case , the problem encountered , its possible cause , and the sequence of events that led to the problem. It shall be used as a basis for the later technical analysis.

- Test Documents and reports shall be submitted. Any specifications to be reviewed , revised , or updated shall be handled immediately.

TEST CASES

A Test case is a document that describes an input , action , or event and expected response , to determine if a feature of an application is working correctly. A Test case should contain particular such as test case identifier , test condition , test condition , input data.
Requirement expected results. The process of developing test cases can help find problems in the requirement or design of an application , since it requires completely thinking through the operation of the application.

TESTING STEPS :-

Unit testing :-

Unit Testing focuses efforts on the smallest unit of software design. This is known as Module Testing. The modules are tested separately. The Test is carried out during programming stage itself. In this step , each module is found to be working satisfactorily as regards to the expected output from the module.

INTEGRATION TESTING :-

Data can be lost across an interface. One module can have an adverse effect on another, sub functions, when combined, may not be linked in desired manner in major functions. Integration Testing is a systematic approach for constructing the program structure, while at the same time conducting test to uncover errors associated within the interface. The objective is to take unit tested modules and builds program structure. All the modules are combined and tested as a whole.

VALIDATION :-

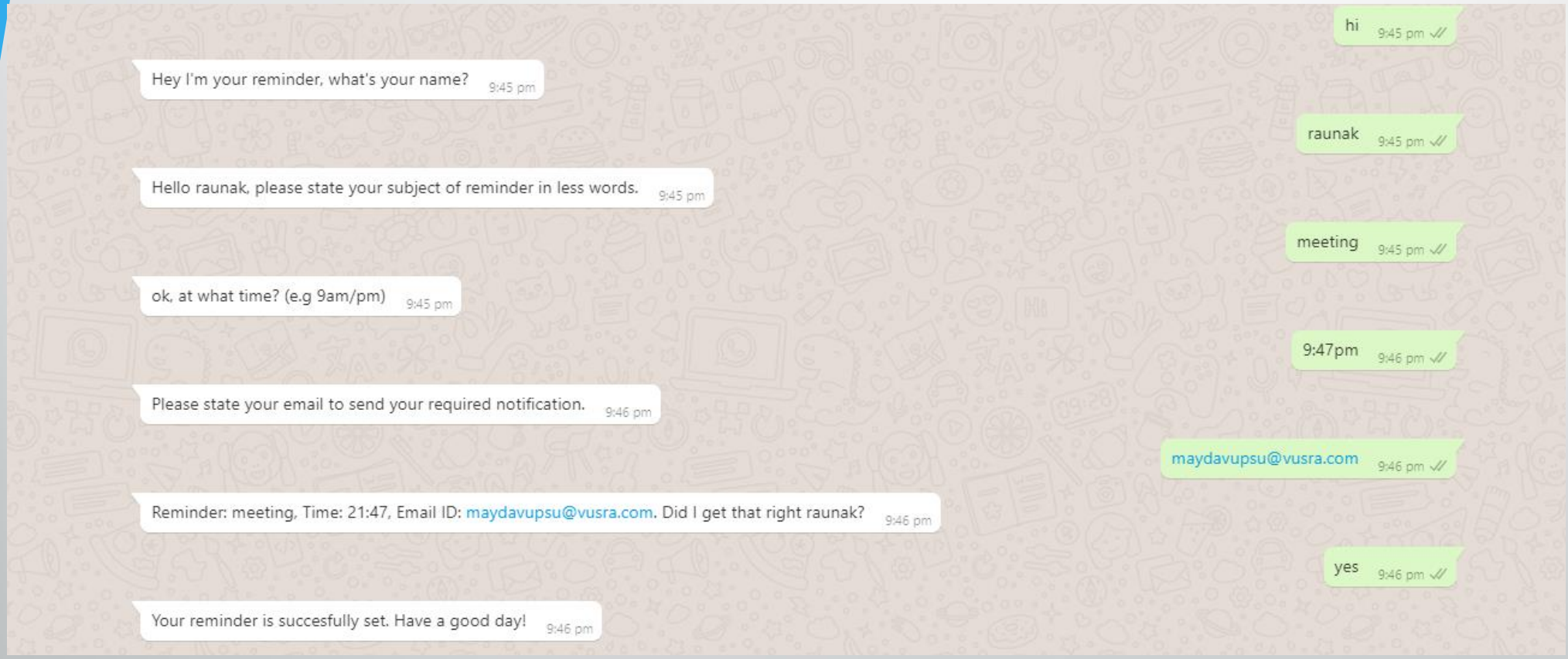
At the culmination of the integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test begins in Validation Testing. Validation Testing can be defined in many ways. But a simple definition is that the validation succeeds when the software functions in a manner that is expected by the customer. After validation test has been conducted, one of the three possible conditions exists.

- The function or performance characteristics confirms to specification and are accepted.
- A deviation from specification is uncovered and a deficiency list is created.
- Proposed system under consideration has been tested by using validation test and found to be working satisfactorily.

Snapshots

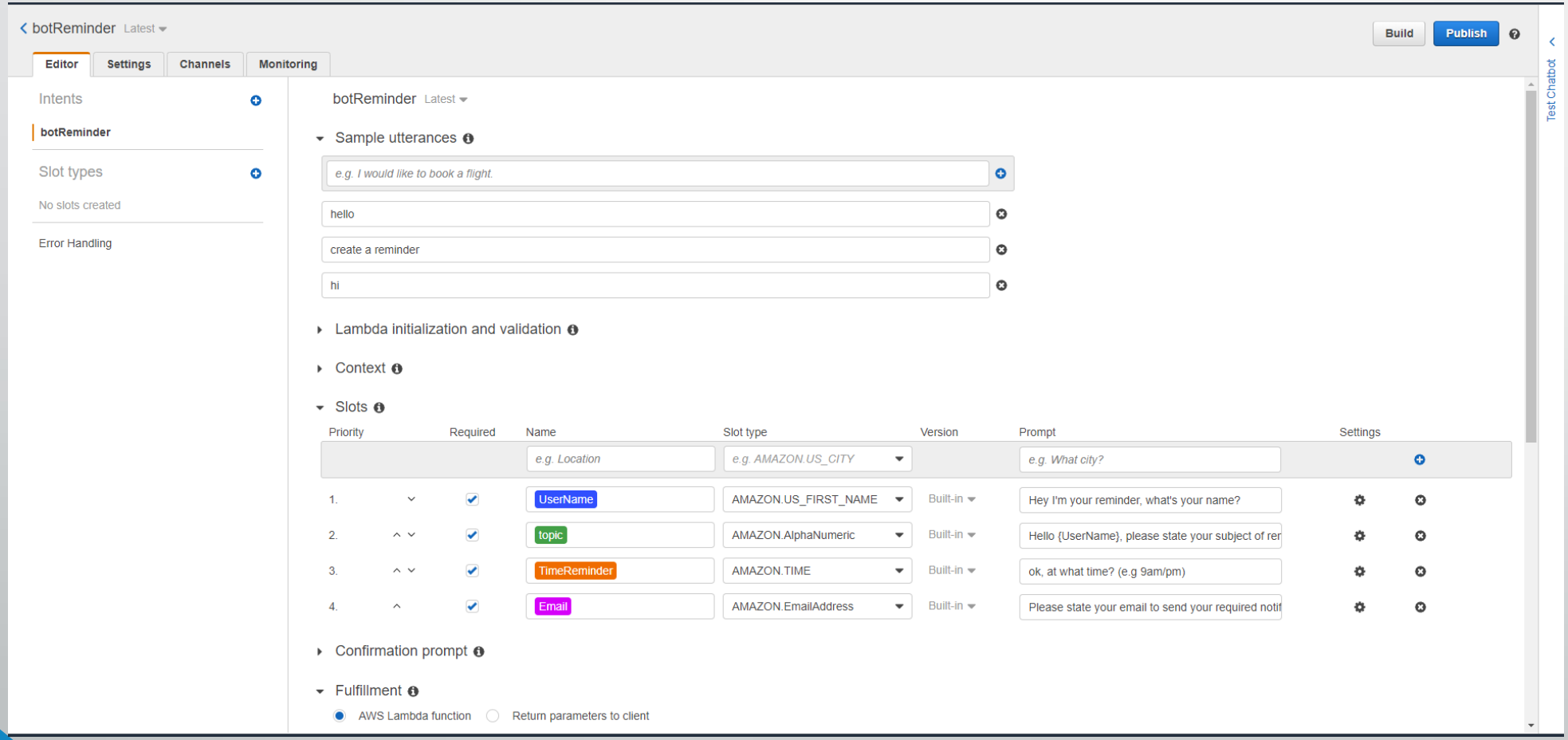
Snapshots

- User with BOT conversation:-



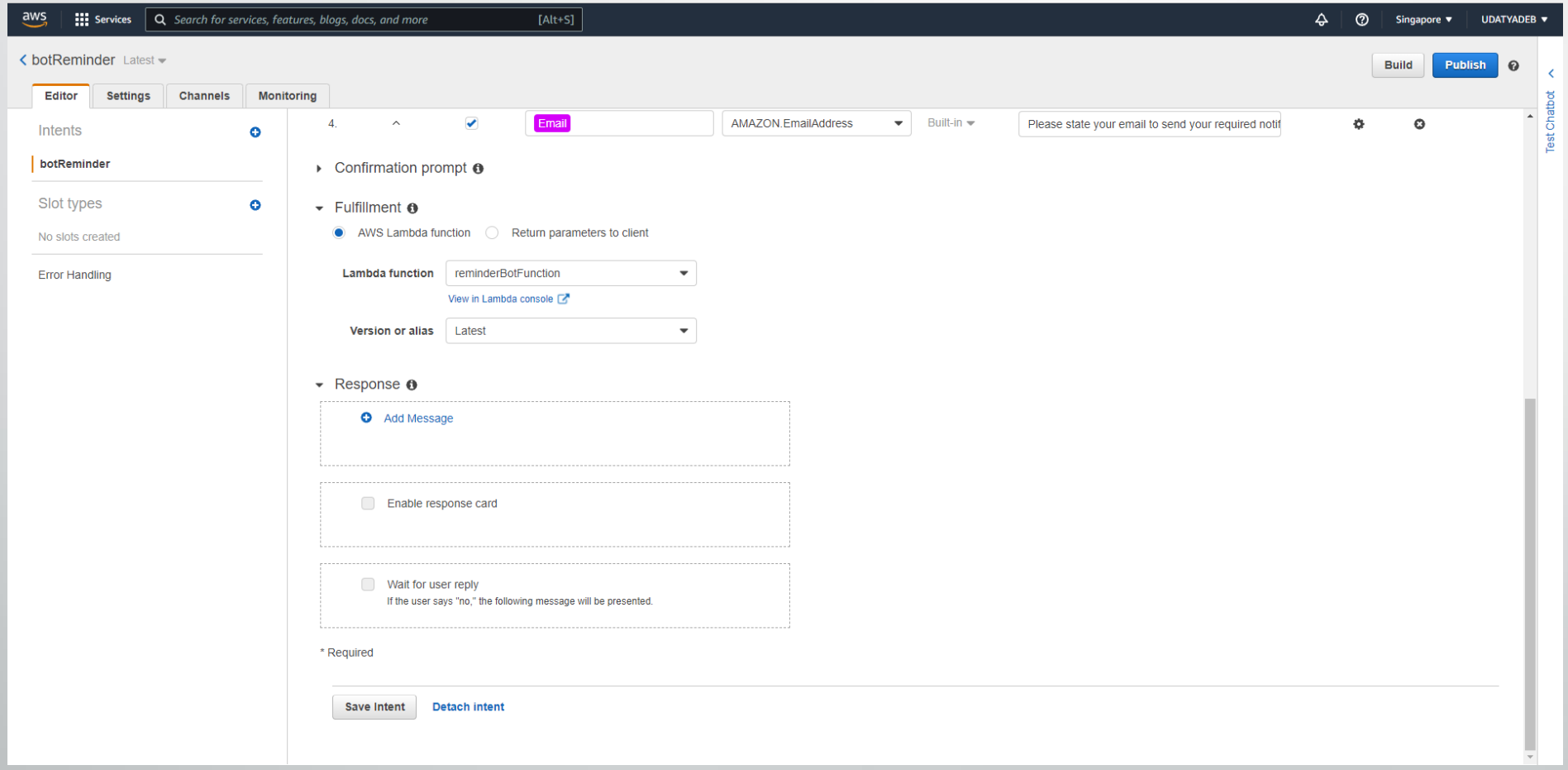
Snapshots

- Amazon LEX Bot console :- (PART I)



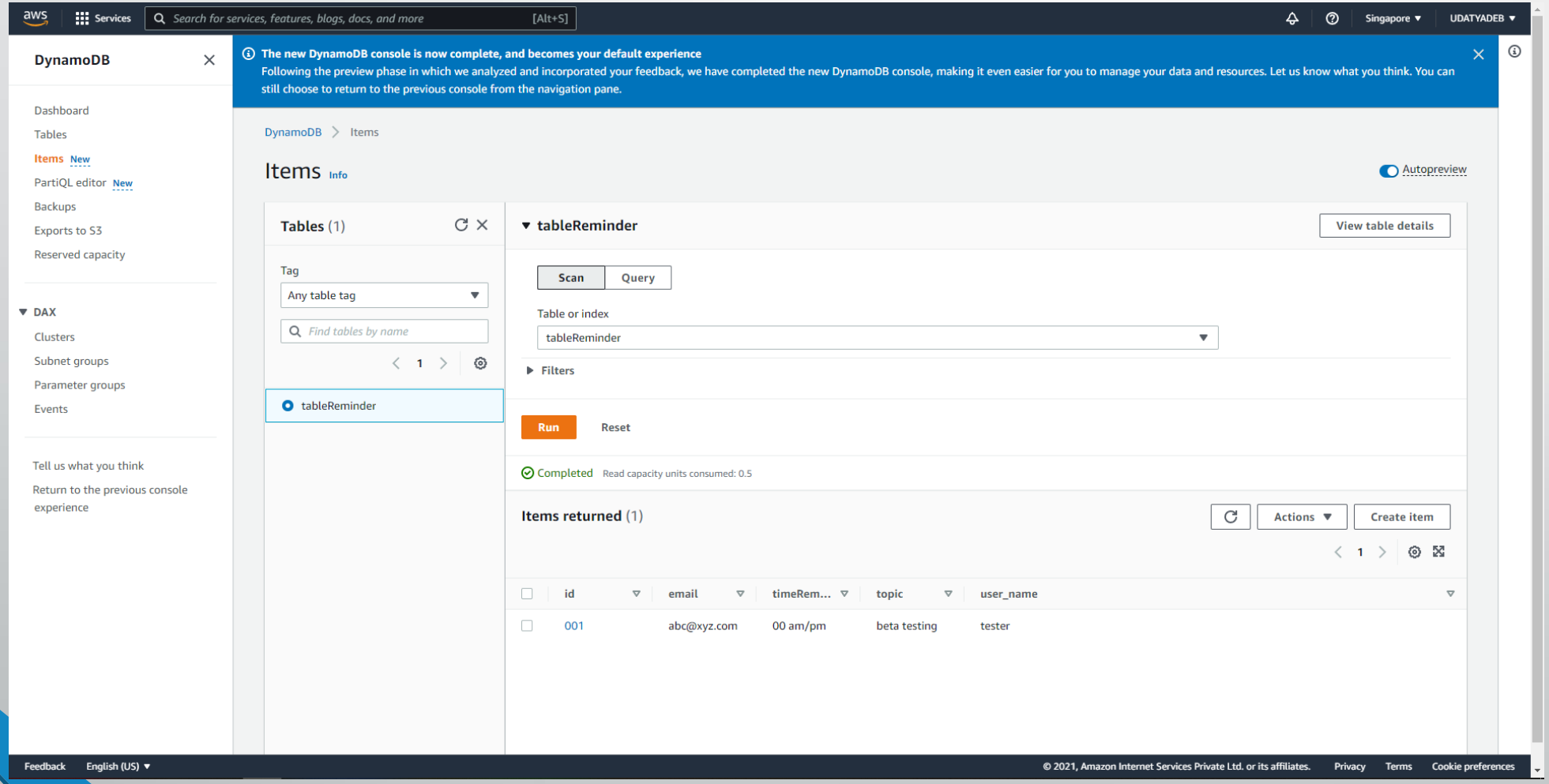
Snapshots

- Amazon LEX Bot console :- (PART II)



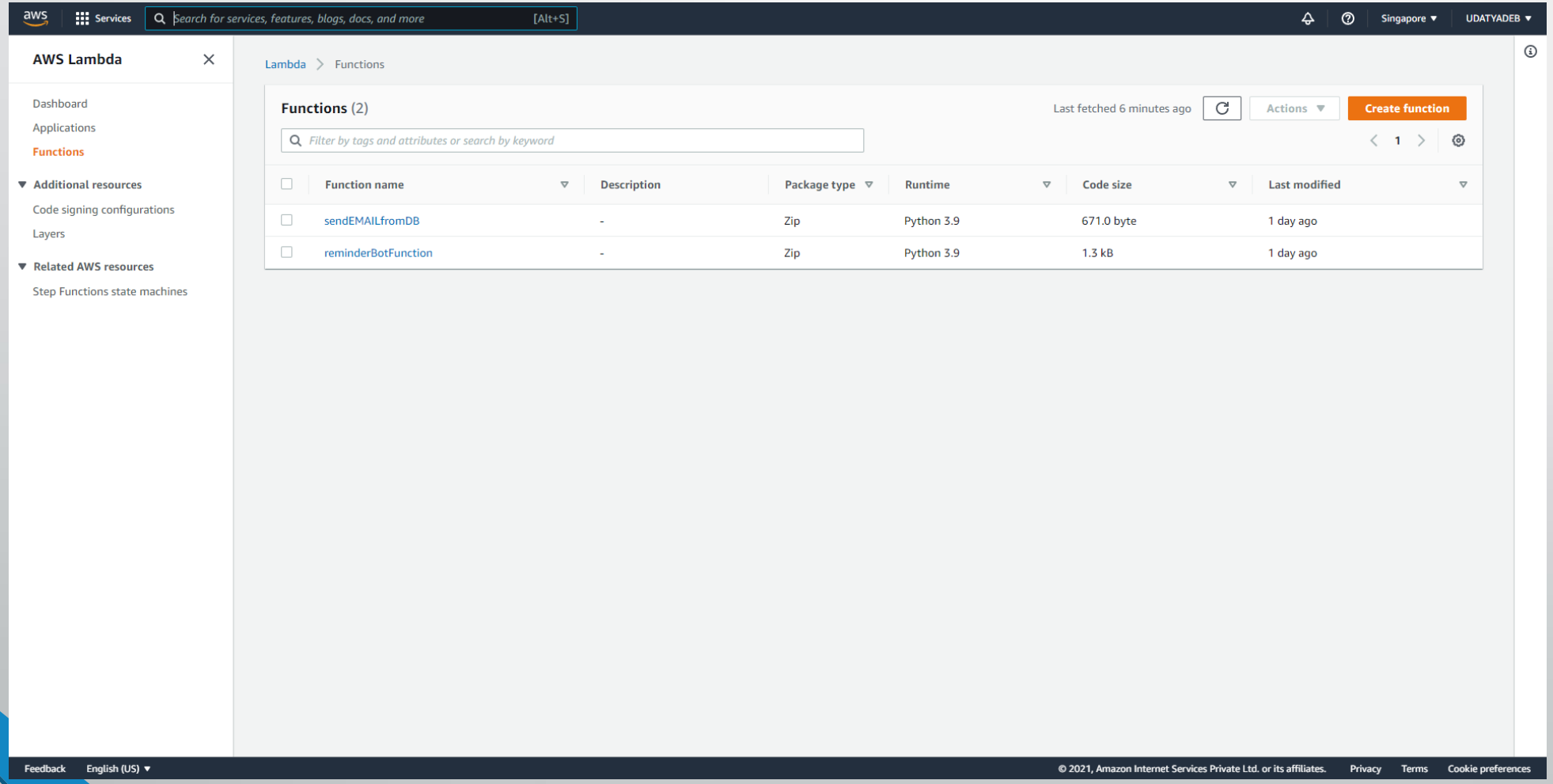
Snapshots

- Amazon DynamoDB:-



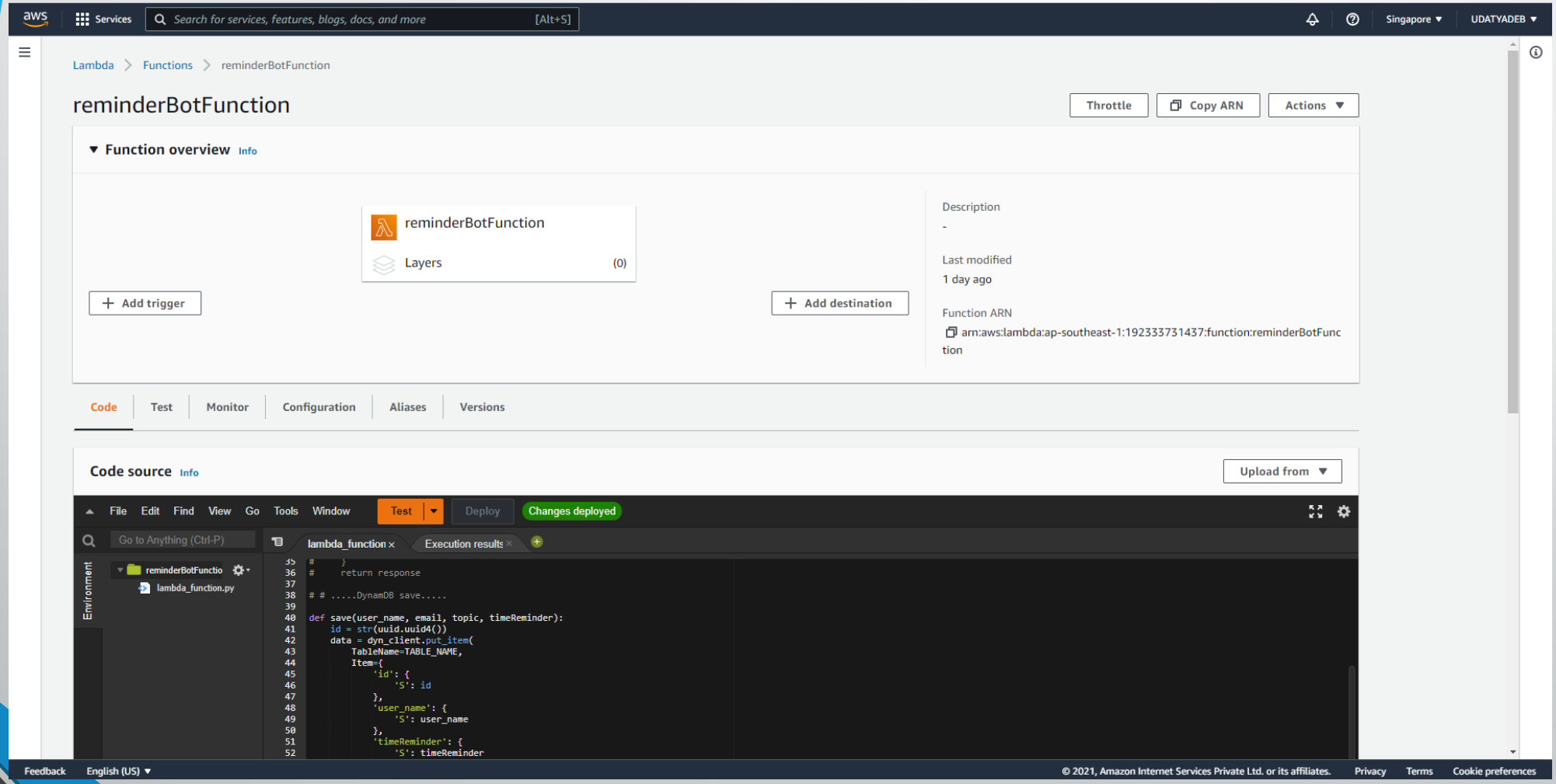
Snapshots

- Amazon Lambda Functions:-



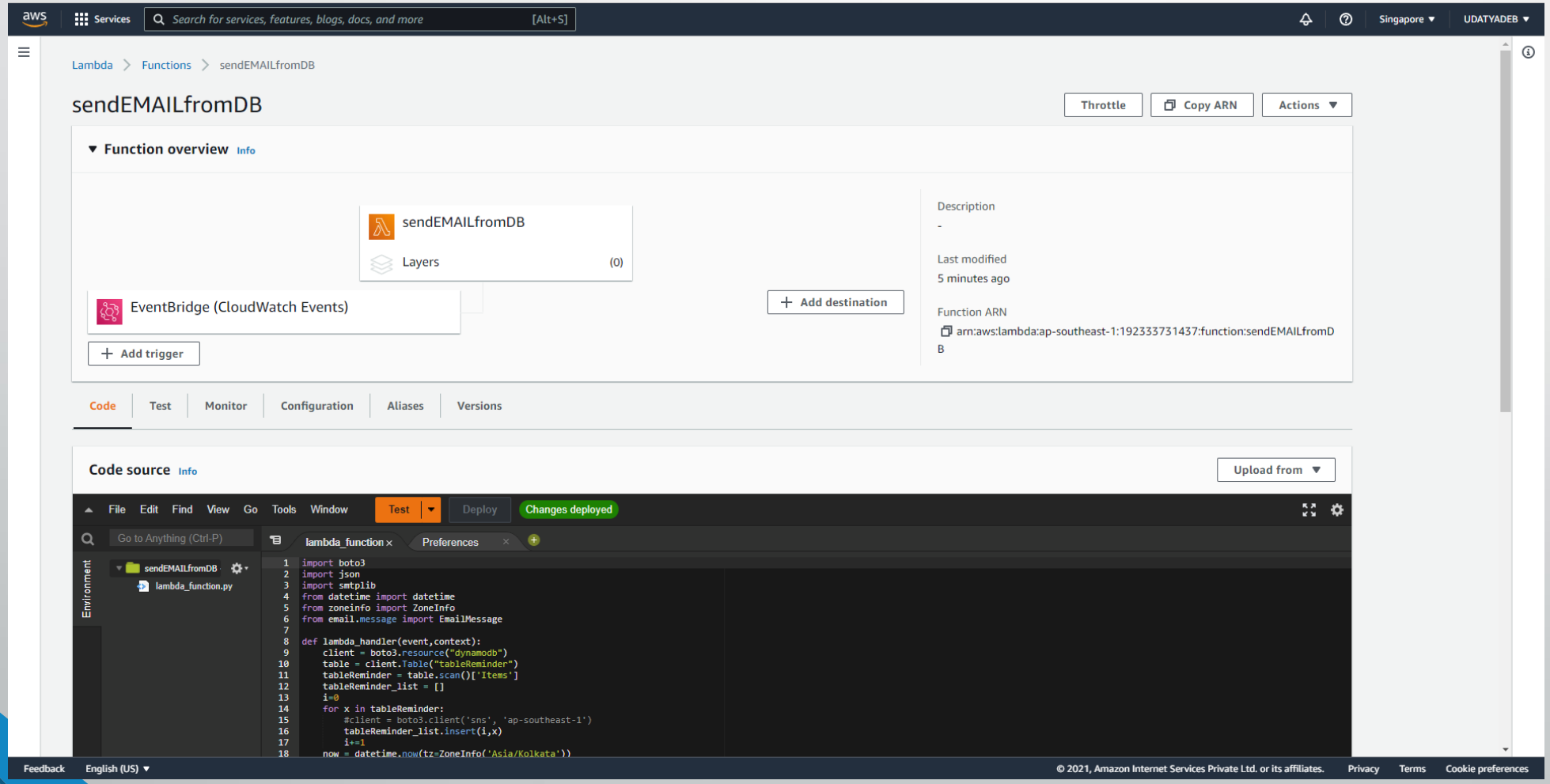
Snapshots

- Amazon Lambda Function:- (reminderBotFunction)



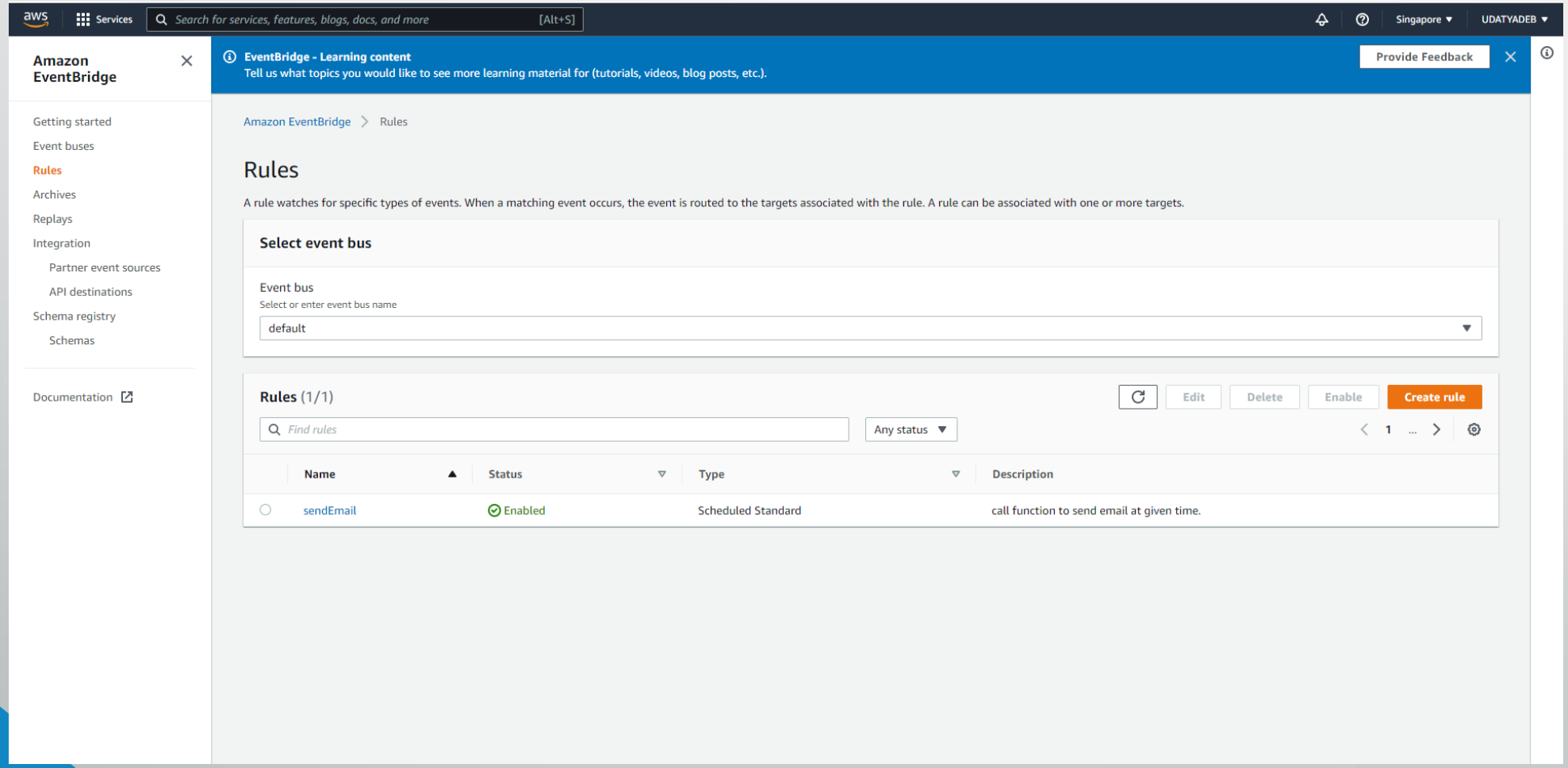
Snapshots

- Amazon Lambda Function:- (sendEMAILfromDB)



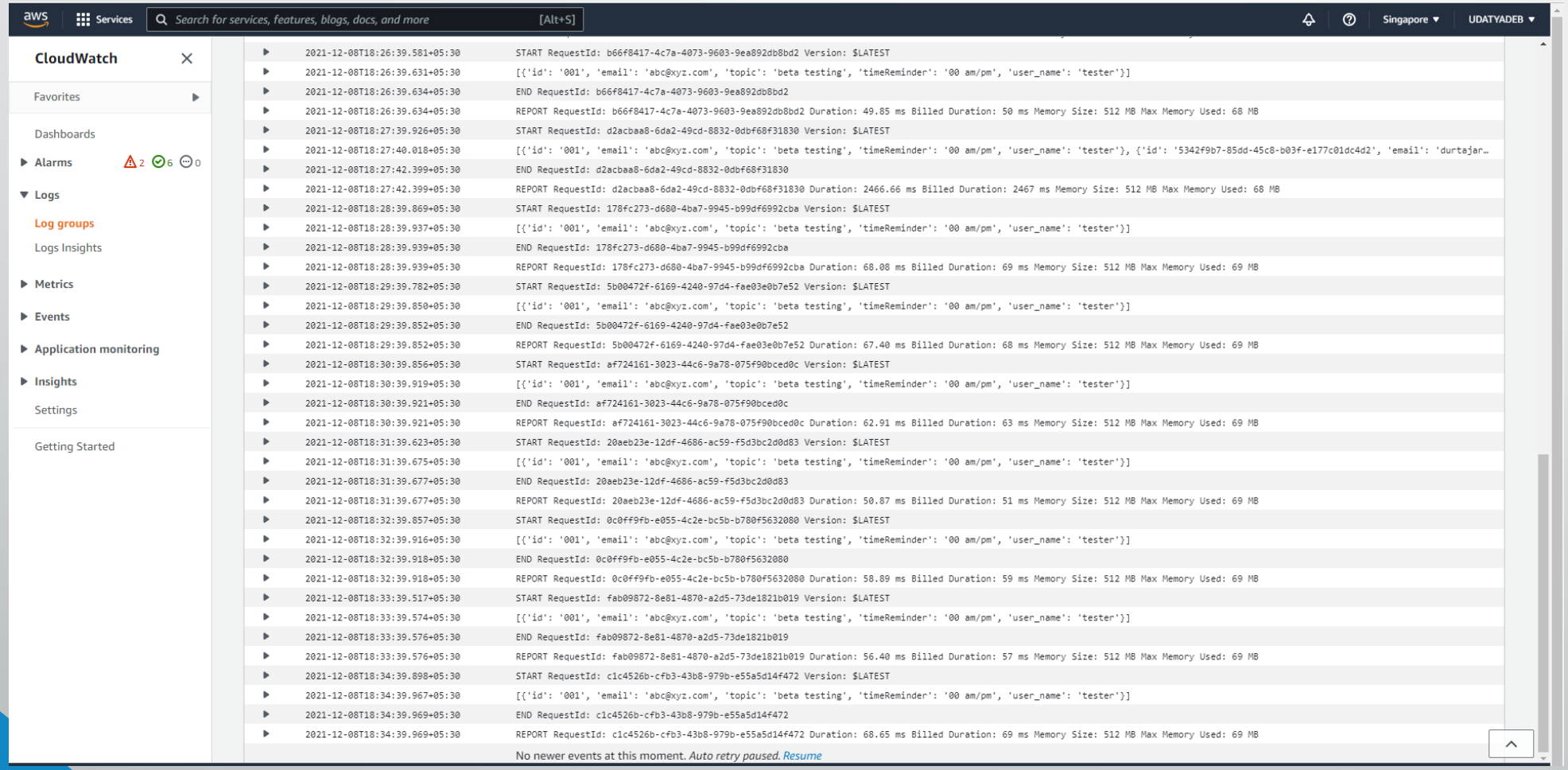
Snapshots

- Amazon EventBridge:- (PART I)



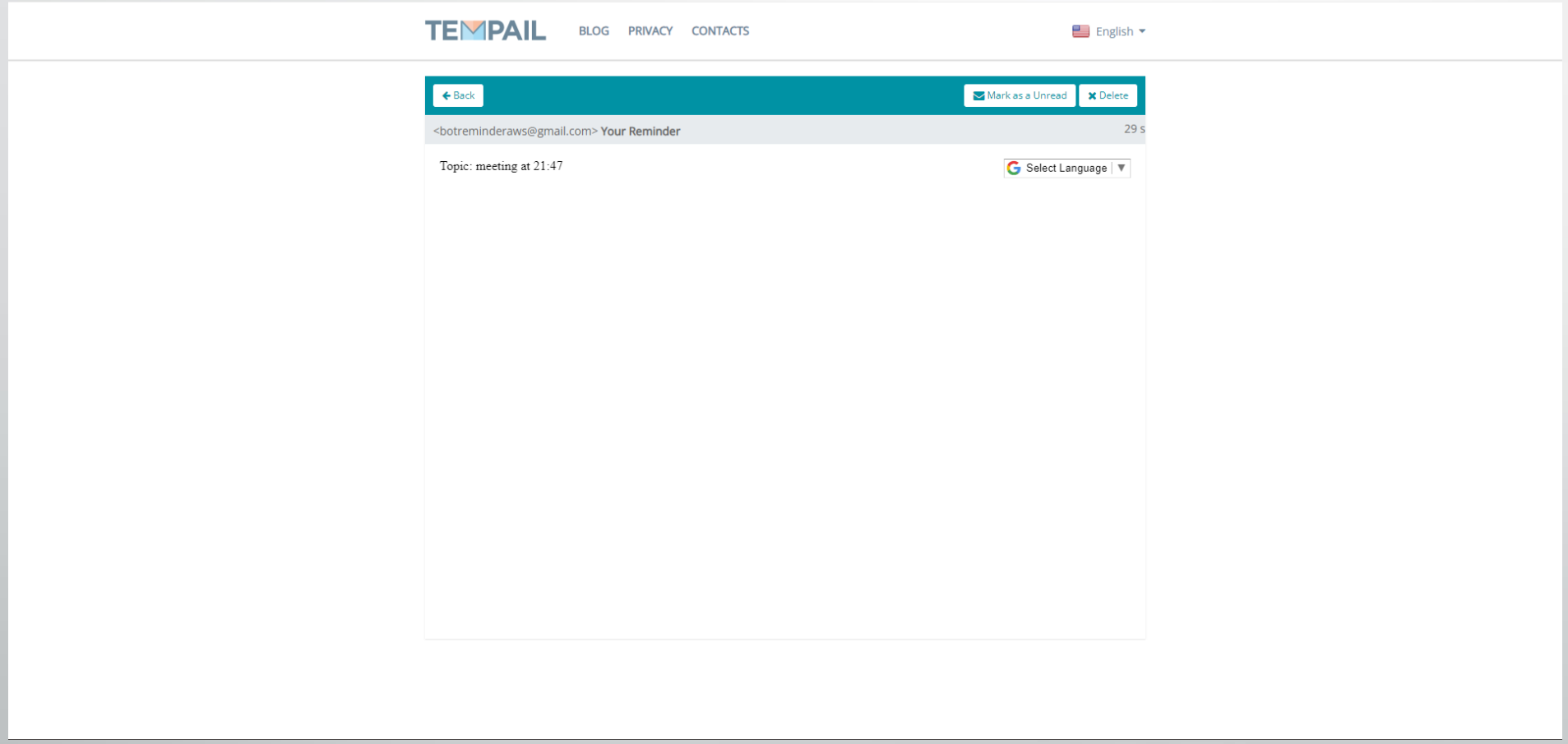
Snapshots

- Amazon EventBridge:- (PART II) Cloudwatch Logs



Snapshots

- EMAIL NOTIFICATION RECEIVED :-



Conclusion

The project has been appreciated by all the users in the organization. It is easy to use , since it uses the AWS provided in the user dialog. User friendly screens are provided. The usage of software increases the efficiency , decreases the effort. It has been thoroughly tested and implemented.

The project "BotReminder" is the ideal place for every person who wants to set a reminder , or create a schedule to get notified at specified time , just with a click sitting in his/her room.

The software collects all the required information from the user to provide a smooth experience. The software provides a reliable platform as it as it does NOT RETAIN any given data as soon as the user is notified through E-Mail. Though our project's topic was simple, but the implementation of various AWS micro-services in our project has further enhanced our knowledge regarding the use of AWS platform, which I am sure will be useful in the future.

Bibliography

Deleting Data from a DynamoDB Table using AWS Lambda – YouTube

How To Send Email In Python via smtplib – YouTube

AWS Lambda Tutorial | Use AWS CloudWatch Events to schedule Lambda function invocation – YouTube

Writing data to DynamoDB (Getting started with AWS Lambda, part 7) – YouTube

How to Integrate AWS lex bot with whatsapp – YouTube

Create Your Own Amazon Lex Chatbot - Full tutorial - YouTube

Thank You.