Cloud-Based Event Booking System using AWS Event Bridge

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

K. VARSHINIKA

2210030196

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 **"Cloud Based Event Booking System using Event-Bridge"** which is a Experimental work carried out by **P.Varshinika(2210030196)**, 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

The paradigm of event-driven architecture (EDA) has gained significant traction in building modern, scalable, and resilient cloud applications [6]. By focusing on the flow of events between decoupled services, EDA allows for greater agility and independent scaling of individual components. In the context of event management, a serverless approach leveraging AWS services offers a compelling solution for handling varying loads and intricate workflows [2]. This project explores the development of a cloud-based event booking system, strategically utilizing AWS EventBridge as its central nervous system.

- AWS EventBridge acts as a fully managed, serverless event bus that enables the creation of event-driven applications at scale [1].

- Leveraging a serverless framework with AWS Lambda functions as event consumers allows for cost-effective and scalable processing of booking-related events [3].

- By centralizing event routing, EventBridge eliminates the need for direct integrations between services, fostering a more maintainable and extensible architecture.[6]

- Furthermore, AWS Identity and Access Management (IAM) plays a crucial role in securing such a system by defining permissions and controlling access to AWS resources [4].

2. AWS Services Used as part of the project

The project utilizes several AWS services to build a serverless architecture:

1. AWS Lambda:

- Provides serverless compute power to run code in response to events, such as processing booking requests or sending notifications.

- Scales automatically based on the number of incoming requests, making it suitable for handling fluctuating booking volumes.

- Integrates seamlessly with other AWS services like EventBridge, DynamoDB, and SNS [8].

2. Amazon DynamoDB:

- Offers a fully managed NoSQL database for storing event details, user information, booking records, and other application data.

- Provides high performance and scalability, capable of handling the data requirements of a busy event booking platform.[2]

3. Amazon SNS :

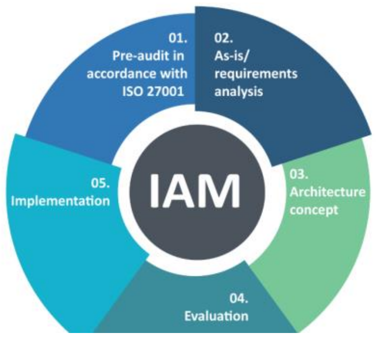
- SQS could be used for asynchronous processing and decoupling of services that don't need immediate responses.

- These services help ensure reliable and scalable handling of events within the system. [5]

amazon
SNS

4. AWS Identity and Access Management (IAM) :

- AWS Identity and Access Management (IAM) is a fundamental service for securing your AWS environment, playing a crucial role in event-driven architectures and serverless applications
- It enables you to define and manage permissions, controlling who (users, groups, roles, services) can access which AWS resources and what actions they can perform [4].

- 
IAM
01. Pre-audit in accordance with ISO 27001
02. As-is/ requirements analysis
03. Architecture concept
04. Evaluation
05. Implementation

5. Amazon CloudWatch :

- AWS CloudWatch is a monitoring and observability service that provides you with data and actionable insights to monitor your applications, infrastructure, and services running on AWS and on-premises [9].

Amazon CloudWatch

3. Steps Involved in Solving the Project Problem Statement

The project problem statement was to build a serverless inventory management system that allows users to know the event is booked successfully. The following steps were taken to achieve this:

I. Setting up the DynamoDB Table:[3]

- Created a DynamoDB table named Events with eventID as the partition key.
- (AWS Serverless for Event-Driven Architecture) likely highlights DynamoDB as a scalable and serverless database suitable for such architectures. DynamoDB is a fundamental component in many serverless applications.

II. Lambda function:[8]

- Developed a Lambda function named CRMFunction to handle API routes (/, /add-customer, /get-customer, /get-all-customers, /update-customer, /delete-customer).

- Lambda confirms that AWS Lambda is the core serverless compute service used to run code without managing servers. This step inherently involves creating a Lambda function to manage the application's backend logic.

III. Creating EventBus using EventBridge:[1]

- EventBridge rules to route specific events to target AWS services for processing is the

primary source for understanding how to create and configure rules within EventBridge to filter and route events to various target services.

- A rule matching BookingCreated events and triggering a Lambda function (likely using SES for email).

IV. Setting up the Policies using IAM:[3]

- IAM Policies: You define these permissions using IAM policies, which are JSON documents that specify the actions allowed and the resources they apply to.
- dynamodb:PutItem,dynamodb:GetItem,dynamodb:Scan,dynamodb:UpdateItem,dynamodb:DeleteItem permissions on your CustomerTable

V. Checking the working using Cloud watch:[8]

- CloudWatch Logs automatically collects logs generated by your Lambda functions. This is invaluable for debugging, troubleshooting errors, and understanding the execution flow of your code [Implicitly needed for any Lambda function.

VI. Debugging:[5]

- You can directly invoke your CRMFuntion (or any other Lambda acting as an event producer) using the aws lambda invoke command. This lets you test if the function executes correctly and produces the intended event.

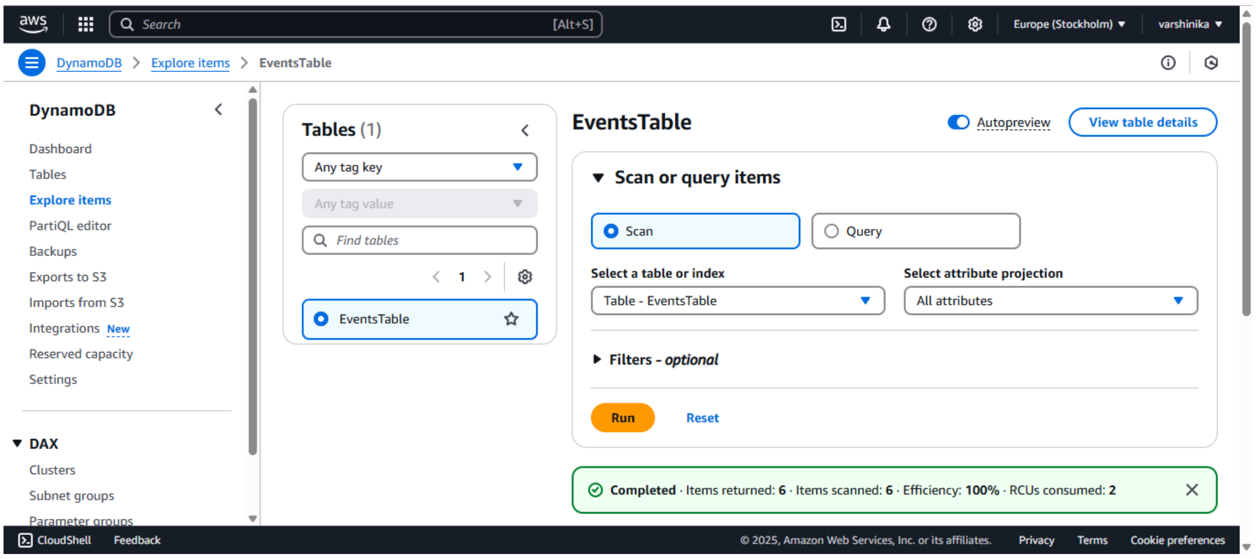
VII. Testing the application:[6]

- Monitor CloudWatch logs for the event-producing Lambda to ensure the put-events call was successful.
- Check CloudWatch logs of the Lambda function that should be triggered by the "BookingCreated" rule (e.g., the email sending Lambda).

Verify if the expected outcome occurred

4. Stepwise Screenshots with Brief Description

Step 1: DynamoDB Table with Items



EventsTable

Select a table or index: Table - EventsTable

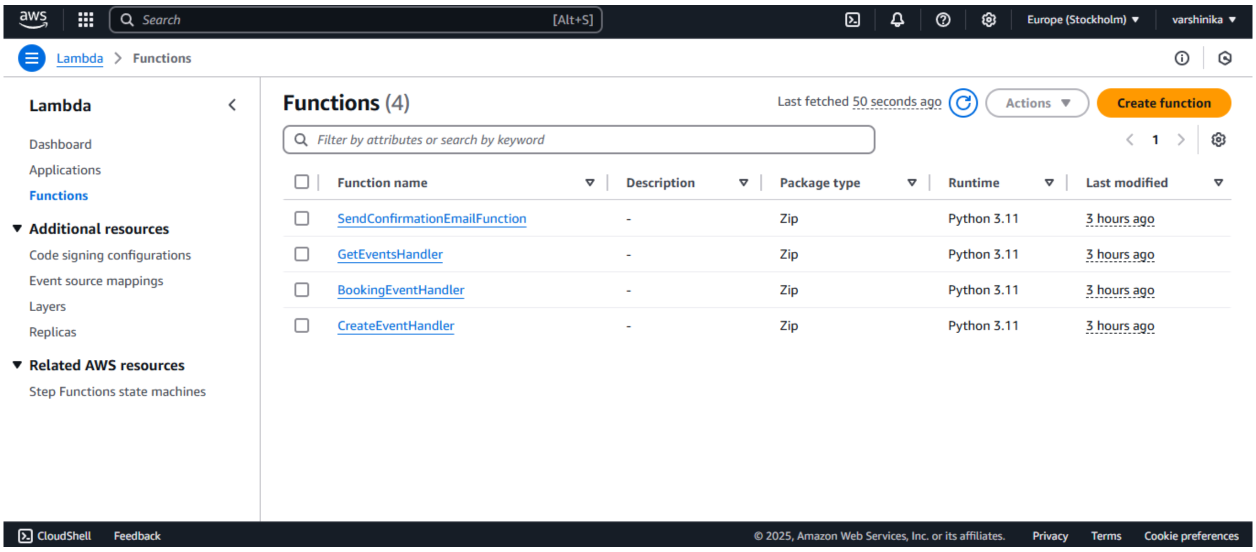
Select attribute projection: All attributes

Completed · Items returned: 6 · Items scanned: 6 · Efficiency: 100% · RCUs consumed: 2

Fig: 4.1: DynamoDB table Setup

The screenshot displays the AWS DynamoDB console, specifically showing the details of a table named "EventsTable". The user is currently in the "Explore items" section and has selected the "Scan" operation to retrieve data from the table. The table "EventsTable" is selected, and the console is set to retrieve "All attributes" of the items.

Step 2: Lambda Function Creation

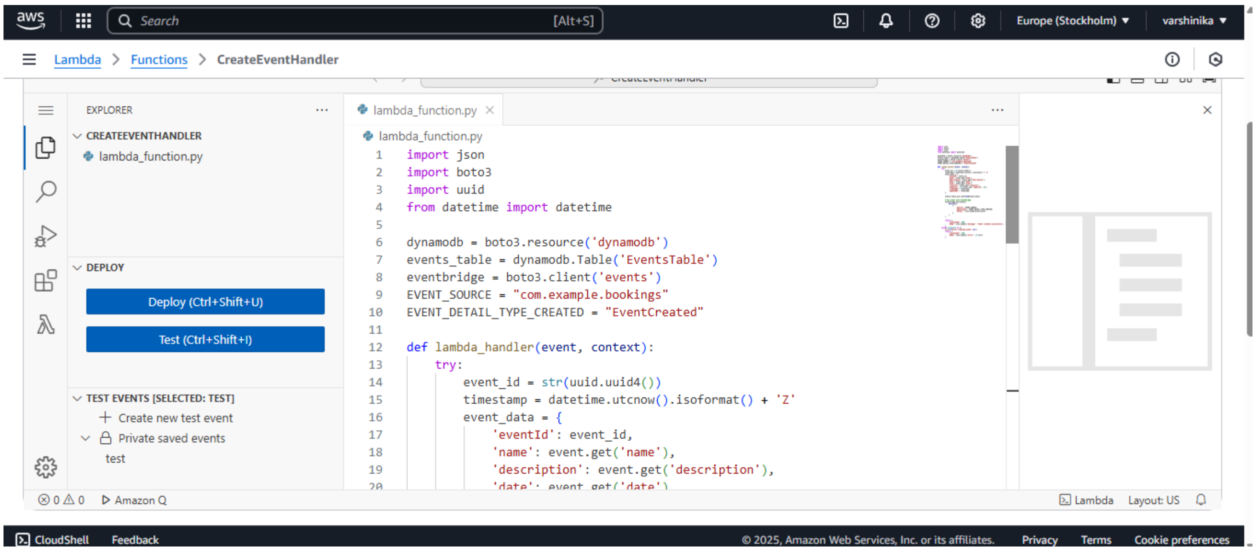


Functions (4)

Function name	Description	Package type	Runtime	Last modified
SendConfirmationEmailFunction	-	Zip	Python 3.11	3 hours ago
GetEventsHandler	-	Zip	Python 3.11	3 hours ago
BookingEventHandler	-	Zip	Python 3.11	3 hours ago
CreateEventHandler	-	Zip	Python 3.11	3 hours ago

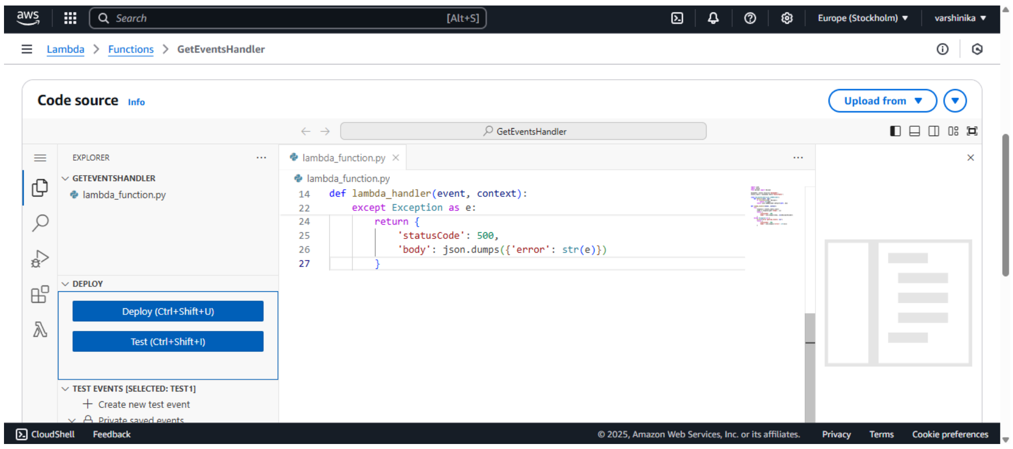
Fig: 4.2.1: Lambda function’s creation

The screenshot shows the AWS Lambda console, displaying a list of four Lambda functions. These functions are named "SendConfirmationEmailFunction", "GetEventsHandler", "BookingEventHandler", and "CreateEventHandler". All functions have a package type of "Zip" and run on the Python 3.11 runtime.



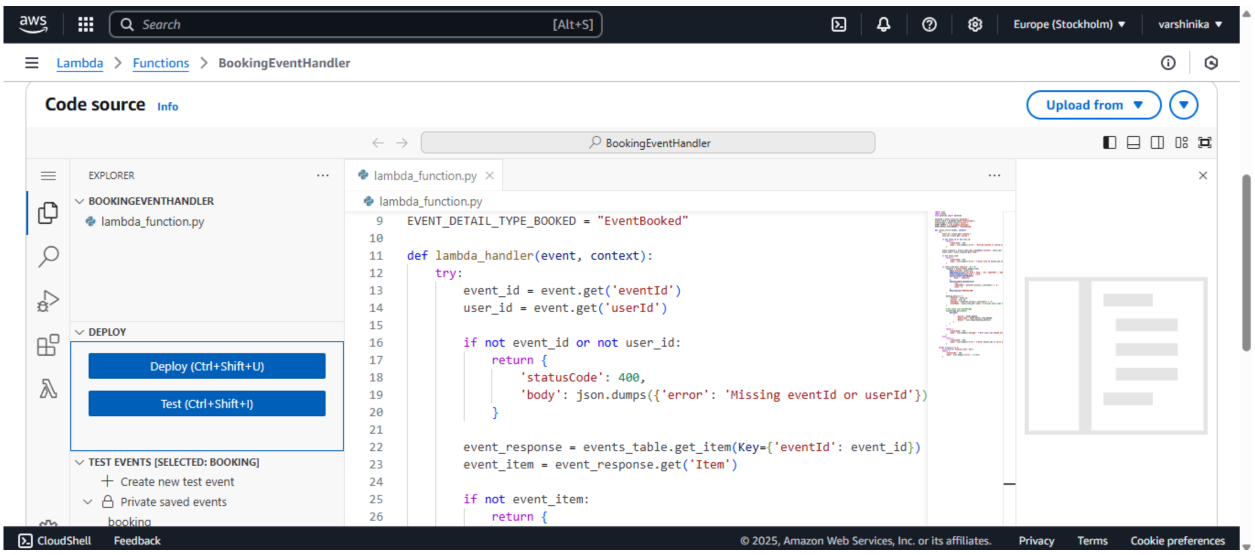
```python
import json
import boto3
import uuid
from datetime import datetime

dynamodb = boto3.resource('dynamodb')
events_table = dynamodb.Table('EventsTable')
eventbridge = boto3.client('events')
EVENT_SOURCE = "com.example.bookings"
EVENT_DETAIL_TYPE_CREATED = "EventCreated"

def lambda_handler(event, context):
    try:
        event_id = str(uuid.uuid4())
        timestamp = datetime.utcnow().isoformat() + 'Z'
        event_data = {
            'eventId': event_id,
            'name': event.get('name'),
            'description': event.get('description'),
            'date': event.get('date')
```

Fig: 4.2.2: CreateEventHandler Lambda setup



```python
def lambda_handler(event, context):
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

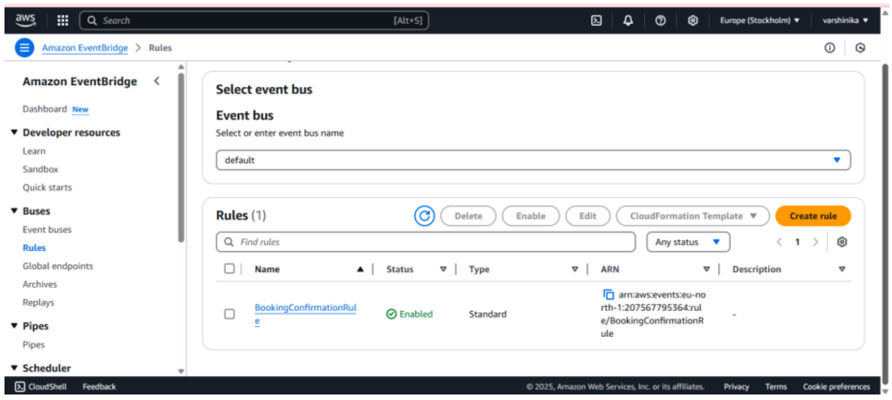
Fig: 4.2.3: GetEventHandler Lambda setup



```python
EVENT_DETAIL_TYPE_BOOKED = "EventBooked"

def lambda_handler(event, context):
    try:
        event_id = event.get('eventId')
        user_id = event.get('userId')

        if not event_id or not user_id:
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'Missing eventId or userId'})
            }

        event_response = events_table.get_item(Key={'eventId': event_id})
        event_item = event_response.get('Item')

        if not event_item:
            return {
```

Fig: 4.2.4: BookingEventHandler Lambda setup

Step 3: Event bridge


Amazon EventBridge > Rules

Select event bus

Event bus

Select or enter event bus name

default

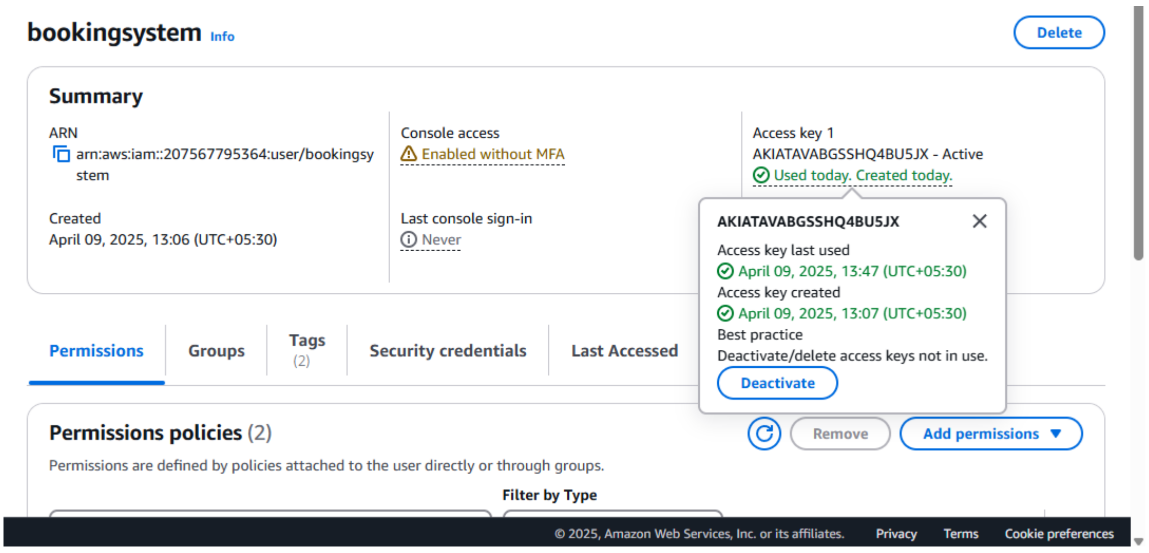
Rules (1)

	Name	Status	Type	ARN	Description
☐	BookingConfirmationRule	✓ Enabled	Standard	arn:aws:events:eu-north-1:207567795364:rule/BookingConfirmationRule	-

Fig: 4.3.1: EventBus Creation

The screenshot shows the AWS EventBridge console, specifically the "Rules" section under the default event bus. There is one rule listed, named "BookingConfirmationRule", which is currently enabled and of the "Standard" type. The ARN (Amazon Resource Name) for this rule is also displayed. The user has options to create a new rule, delete, enable, edit, or create a CloudFormation template for the existing rule.

Step 4: IAM roles


bookingsystem

Summary

ARN
arn:aws:iam::207567795364:user/bookingsystem

Created
April 09, 2025, 13:06 (UTC+05:30)

Console access
Enabled without MFA

Last console sign-in
Never

Access key 1
AKIATAVABGSSHQ4BU5JX - Active
Used today. Created today.

AKIATAVABGSSHQ4BU5JX
Access key last used
April 09, 2025, 13:47 (UTC+05:30)
Access key created
April 09, 2025, 13:07 (UTC+05:30)
Best practice
Deactivate/delete access keys not in use.

Permissions policies (2)

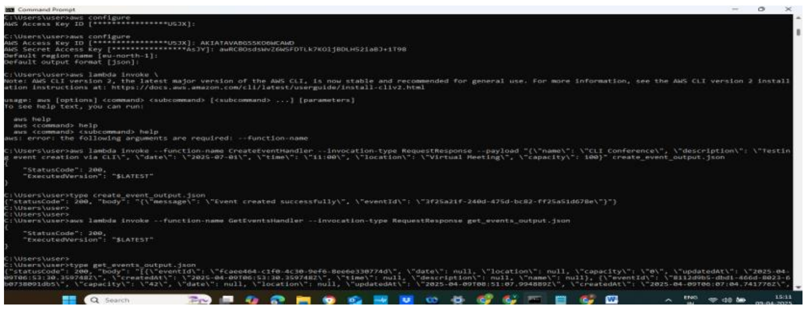
Permissions are defined by policies attached to the user directly or through groups.

Filter by Type

Fig: 4.4: Attaching DynamoDB Permissions to the IAM Role

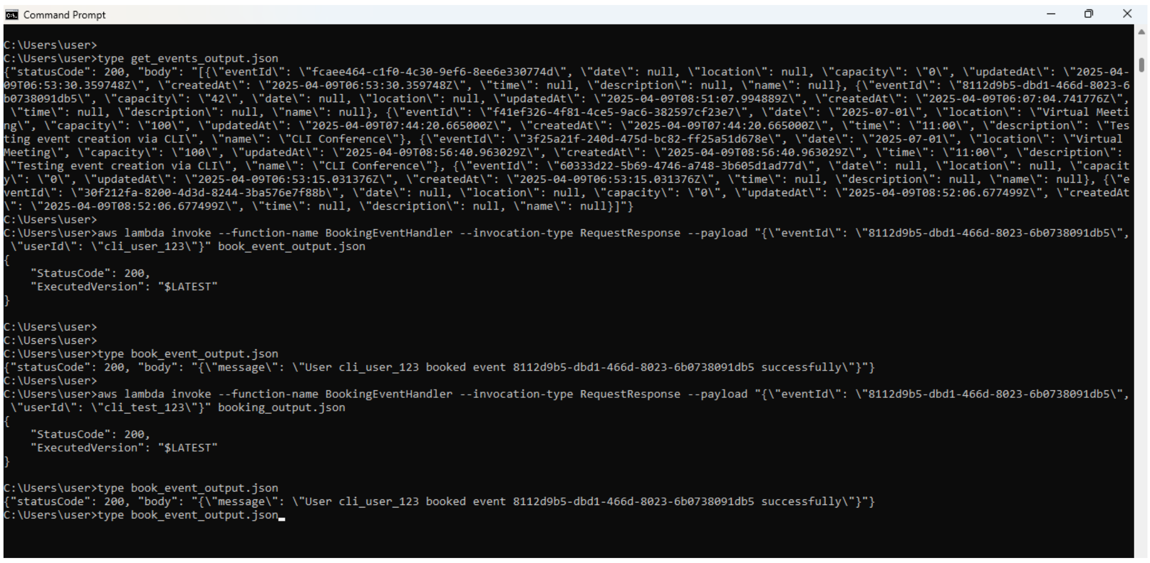
The screenshot shows the summary information for an AWS IAM user named "bookingsystem". Console access is enabled but without MFA. An active access key (AKIATAVABGSSHQ4BU5JX). A pop-up details the access key's last used time and creation date, along with a "Best practice" recommendation to deactivate/delete unused access keys and a "Deactivate" button.

Step 5: Implementing aws command line interface



```
C:\Users\user>aws configure
AWS Access Key ID [****************U5JX]:

C:\Users\user>aws configure
AWS Access Key ID [****************U5JX]: AKIATAVABGSSKD6WCAWD
AWS Secret Access Key [****************AsJY]: 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
Default region name [eu-north-1]:
Default output format [json]:

C:\Users\user>aws lambda invoke \
Note: AWS CLI version 2, the latest major version of the AWS CLI, is now stable and recommended for general use. For more information, see the AWS CLI version 2 install
ation instructions at: https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

  aws help
  aws <command> help
  aws <command> <subcommand> help
aws: error: the following arguments are required: --function-name

C:\Users\user>aws lambda invoke --function-name CreateEventHandler --invocation-type RequestResponse --payload "{\"name\": \"CLI Conference\", \"description\": \"Testin
g event creation via CLI\", \"date\": \"2025-07-01\", \"time\": \"11:00\", \"location\": \"Virtual Meeting\", \"capacity\": 100}" create_event_output.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}

C:\Users\user>type create_event_output.json
{"statusCode": 200, "body": "{\"message\": \"Event created successfully\", \"eventId\": \"3f25a21f-240d-475d-bc82-ff25a51d678e\"}"}
C:\Users\user>
C:\Users\user>
C:\Users\user>aws lambda invoke --function-name GetEventsHandler --invocation-type RequestResponse get_events_output.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}

C:\Users\user>
C:\Users\user>type get_events_output.json
{"statusCode": 200, "body": "[{\"eventId\": \"fcaee464-c1f0-4c30-9ef6-8ee6e330774d\", \"date\": null, \"location\": null, \"capacity\": \"0\", \"updatedAt\": \"2025-04-
09T06:53:30.359748Z\", \"createdAt\": \"2025-04-09T06:53:30.359748Z\", \"time\": null, \"description\": null, \"name\": null}, {\"eventId\": \"8112d9b5-dbd1-466d-8023-6
b0738091db5\", \"capacity\": \"42\", \"date\": null, \"location\": null, \"updatedAt\": \"2025-04-09T08:51:07.994889Z\", \"createdAt\": \"2025-04-09T06:07:04.741776Z\",
```

Fig: 4.5.1: Aws cli

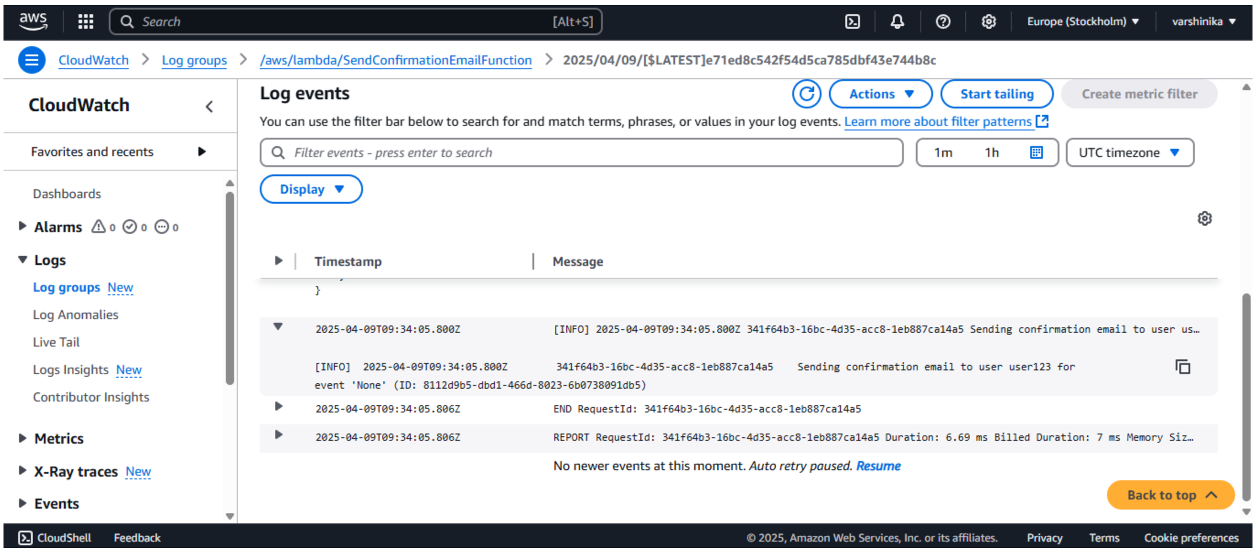


```
C:\Users\user>
C:\Users\user>type get_events_output.json
{"statusCode": 200, "body": "[{\"eventId\": \"fcaee464-c1f0-4c30-9ef6-8ee6e330774d\", \"date\": null, \"location\": null, \"capacity\": \"0\", \"updatedAt\": \"2025-04-
09T06:53:30.359748Z\", \"createdAt\": \"2025-04-09T06:53:30.359748Z\", \"time\": null, \"description\": null, \"name\": null}, {\"eventId\": \"8112d9b5-dbd1-466d-8023-6
b0738091db5\", \"capacity\": \"42\", \"date\": null, \"location\": null, \"updatedAt\": \"2025-04-09T08:51:07.994889Z\", \"createdAt\": \"2025-04-09T06:07:04.741776Z\",
 \"time\": null, \"description\": null, \"name\": null}, {\"eventId\": \"f41ef326-4f81-4ce5-9ac6-382597cf23e7\", \"date\": \"2025-07-01\", \"location\": \"Virtual Meeti
ng\", \"capacity\": \"100\", \"updatedAt\": \"2025-04-09T07:44:20.665000Z\", \"createdAt\": \"2025-04-09T07:44:20.665000Z\", \"time\": \"11:00\", \"description\": \"Tes
ting event creation via CLI\", \"name\": \"CLI Conference\"}, {\"eventId\": \"3f25a21f-240d-475d-bc82-ff25a51d678e\", \"date\": \"2025-07-01\", \"location\": \"Virtual
Meeting\", \"capacity\": \"100\", \"updatedAt\": \"2025-04-09T08:56:40.963029Z\", \"createdAt\": \"2025-04-09T08:56:40.963029Z\", \"time\": \"11:00\", \"description\":
\"Testing event creation via CLI\", \"name\": \"CLI Conference\"}, {\"eventId\": \"60333d22-5b69-4746-a748-3b605d1ad77d\", \"date\": null, \"location\": null, \"capacit
y\": \"0\", \"updatedAt\": \"2025-04-09T06:53:15.031376Z\", \"createdAt\": \"2025-04-09T06:53:15.031376Z\", \"time\": null, \"description\": null, \"name\": null}, {\"e
ventId\": \"30f212fa-8200-4d3d-8244-3ba576e7f88b\", \"date\": null, \"location\": null, \"capacity\": \"0\", \"updatedAt\": \"2025-04-09T08:52:06.677499Z\", \"createdAt
\": \"2025-04-09T08:52:06.677499Z\", \"time\": null, \"description\": null, \"name\": null}]"}
C:\Users\user>
C:\Users\user>aws lambda invoke --function-name BookingEventHandler --invocation-type RequestResponse --payload "{\"eventId\": \"8112d9b5-dbd1-466d-8023-6b0738091db5\",
 \"userId\": \"cli_user_123\"}" book_event_output.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}

C:\Users\user>
C:\Users\user>
C:\Users\user>type book_event_output.json
{"statusCode": 200, "body": "{\"message\": \"User cli_user_123 booked event 8112d9b5-dbd1-466d-8023-6b0738091db5 successfully\"}"}
C:\Users\user>
C:\Users\user>aws lambda invoke --function-name BookingEventHandler --invocation-type RequestResponse --payload "{\"eventId\": \"8112d9b5-dbd1-466d-8023-6b0738091db5\",
 \"userId\": \"cli_test_123\"}" booking_output.json
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}

C:\Users\user>type book_event_output.json
{"statusCode": 200, "body": "{\"message\": \"User cli_user_123 booked event 8112d9b5-dbd1-466d-8023-6b0738091db5 successfully\"}"}
C:\Users\user>type book_event_output.json_
```

Fig: 4.5.2: AWS CLI output

The command prompt shows the output of AWS CLI commands interacting with a Lambda function named "BookingEventHandler". The first command retrieved event data, outputting a JSON structure. Subsequent commands invoked "BookingEventHandler" with a specific eventId and userId, receiving a successful response (StatusCode: 200) and a message confirming the booking. The output is also saved in "book_event_output.json".

Step 6: Cloud Watch



CloudWatch > Log groups > /aws/lambda/SendConfirmationEmailFunction > 2025/04/09/[$LATEST]e71ed8c542f54d5ca785dbf43e744b8c

Log events

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

Timestamp	Message
2025-04-09T09:34:05.800Z	[INFO] 2025-04-09T09:34:05.800Z 341f64b3-16bc-4d35-acc8-1eb887ca14a5 Sending confirmation email to user us…
2025-04-09T09:34:05.806Z	END RequestId: 341f64b3-16bc-4d35-acc8-1eb887ca14a5
2025-04-09T09:34:05.806Z	REPORT RequestId: 341f64b3-16bc-4d35-acc8-1eb887ca14a5 Duration: 6.69 ms Billed Duration: 7 ms Memory Siz…

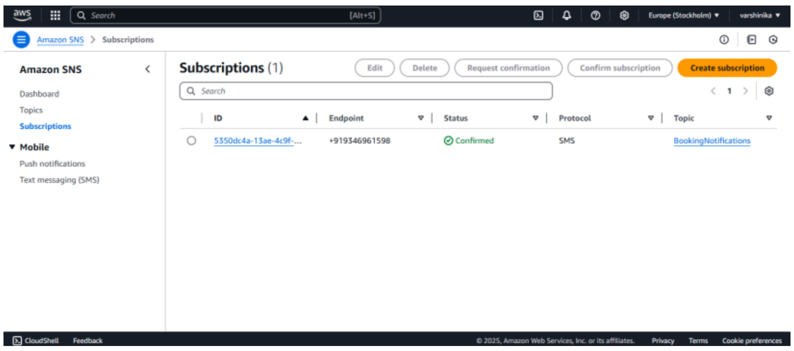
```
[INFO]  2025-04-09T09:34:05.800Z    341f64b3-16bc-4d35-acc8-1eb887ca14a5    Sending confirmation email to user user123 for
event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091db5)
```

No newer events at this moment. *Auto retry paused.* ***Resume***

Fig: 4.6: Cloud Log

The screenshot shows AWS CloudWatch Logs for the Lambda function "SendConfirmationEmailFunction". It displays log events indicating the function successfully sent a confirmation email to user "us..." and then to "user123". The log includes the request ID, duration (6.69 ms), and memory usage. There are no newer events at this moment.

Step 7: SNS



Amazon SNS > Subscriptions

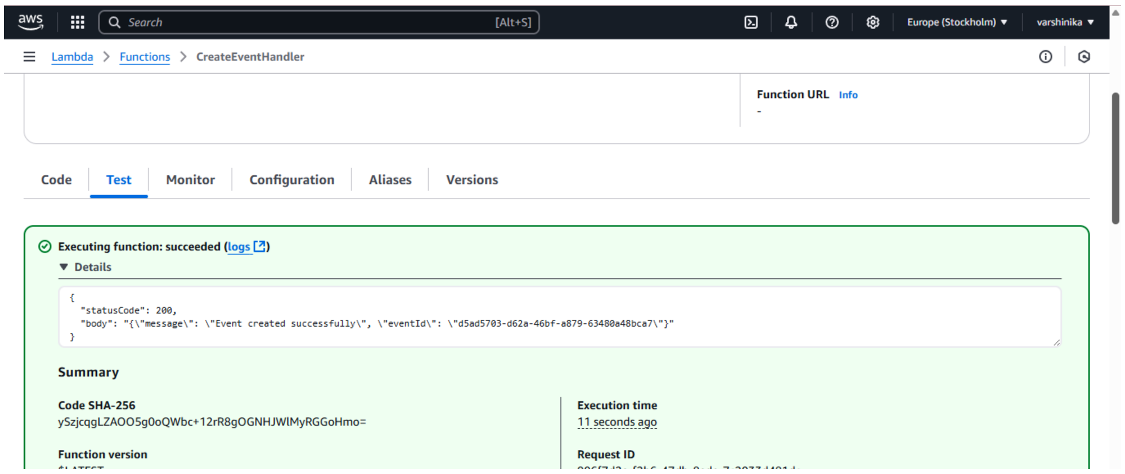
Subscriptions (1)

ID	Endpoint	Status	Protocol	Topic
5350dc4a-13ae-4c9f-...	+919346961598	Confirmed	SMS	BookingNotifications

Fig: 4.7: SNS

The screenshot shows the Amazon SNS (Simple Notification Service) console, specifically the "Subscriptions" section. It displays one subscription with the ID "5350dc4a-13ae-4c9f-...", an endpoint of "+919346961598", a "Confirmed" status, using the "SMS" protocol, and subscribed to the "BookingNotifications" topic. The user has options to edit, delete, request confirmation, confirm subscription, or create a new subscription.

Step 8: Final output



Lambda > Functions > CreateEventHandler

Function URL Info
-

Code | Test | Monitor | Configuration | Aliases | Versions

Executing function: succeeded (logs)

Details

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Event created successfully\", \"eventId\": \"d5ad5703-d62a-46bf-a879-63480a48bca7\"}"
}
```

Summary

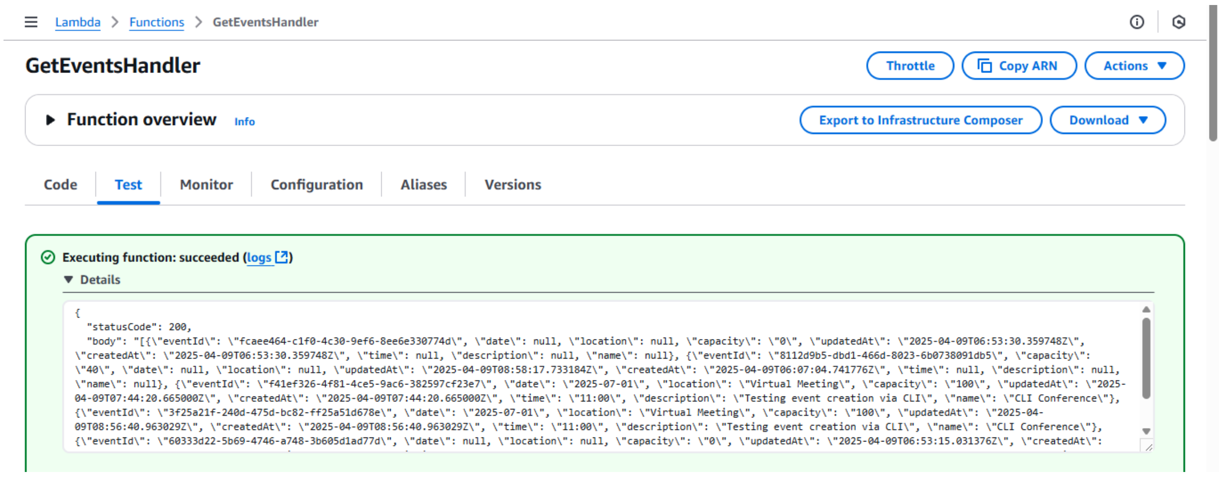
Code SHA-256
ySzjcqgLZAOO5g0oQWbc+12rR8gOGNHJWlMyRGGoHmo=

Execution time
11 seconds ago

Function version

Request ID

Fig: 4.8.1:event creation successfully



Lambda > Functions > GetEventsHandler

GetEventsHandler

Throttle | Copy ARN | Actions

Function overview Info

Export to Infrastructure Composer | Download

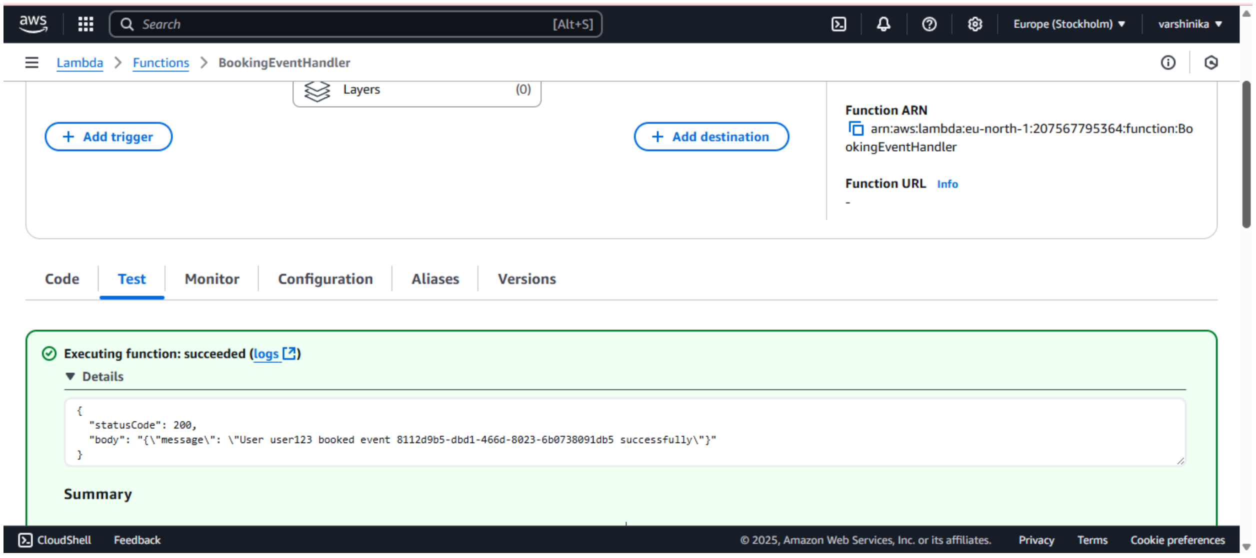
Code | Test | Monitor | Configuration | Aliases | Versions

Executing function: succeeded (logs)

Details

```
{
  "statusCode": 200,
  "body": "[{\"eventId\": \"fcaee464-c1f0-4c30-9ef6-8ee6e330774d\", \"date\": null, \"location\": null, \"capacity\": \"0\", \"updatedAt\": \"2025-04-09T06:53:30.359748Z\",
\"createdAt\": \"2025-04-09T06:53:30.359748Z\", \"time\": null, \"description\": null, \"name\": null}, {\"eventId\": \"8112d9b5-dbd1-466d-8023-6b0738091db5\", \"capacity\":
\"40\", \"date\": null, \"location\": null, \"updatedAt\": \"2025-04-09T08:58:17.733184Z\", \"createdAt\": \"2025-04-09T06:07:04.741776Z\", \"time\": null, \"description\": null,
\"name\": null}, {\"eventId\": \"f41ef326-4f81-4ce5-9ac6-382597cf23e7\", \"date\": \"2025-07-01\", \"location\": \"Virtual Meeting\", \"capacity\": \"100\", \"updatedAt\": \"2025-
04-09T07:44:20.665000Z\", \"createdAt\": \"2025-04-09T07:44:20.665000Z\", \"time\": \"11:00\", \"description\": \"Testing event creation via CLI\", \"name\": \"CLI Conference\"},
{\"eventId\": \"3f25a21f-240d-475d-bc82-ff25a51d678e\", \"date\": \"2025-07-01\", \"location\": \"Virtual Meeting\", \"capacity\": \"100\", \"updatedAt\": \"2025-04-
09T08:56:40.963029Z\", \"createdAt\": \"2025-04-09T08:56:40.963029Z\", \"time\": \"11:00\", \"description\": \"Testing event creation via CLI\", \"name\": \"CLI Conference\"},
{\"eventId\": \"60333d22-5b69-4746-a748-3b605d1ad77d\", \"date\": null, \"location\": null, \"capacity\": \"0\", \"updatedAt\": \"2025-04-09T06:53:15.031376Z\", \"createdAt\":
```

Fig: 4.8.2:List of event attributes



Lambda > Functions > BookingEventHandler

Layers (0)

Add trigger | Add destination

Function ARN
arn:aws:lambda:eu-north-1:207567795364:function:BookingEventHandler

Function URL Info
-

Code | Test | Monitor | Configuration | Aliases | Versions

Executing function: succeeded (logs)

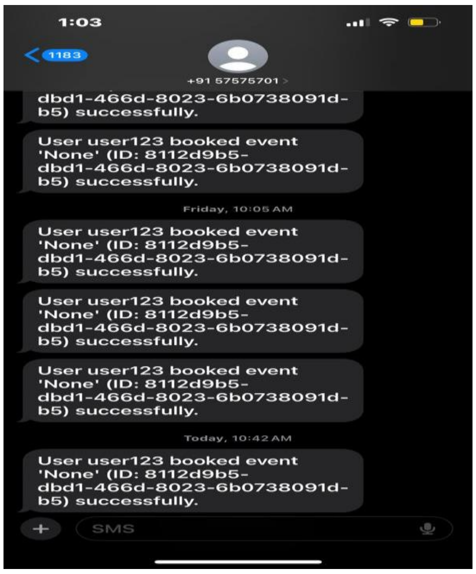
Details

```
{
  "statusCode": 200,
  "body": "{\"message\": \"User user123 booked event 8112d9b5-dbd1-466d-8023-6b0738091db5 successfully\"}"
}
```

Summary

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Fig: 4.8.3: User has booked the event.



1:03

+91 57575701

dbd1-466d-8023-6b0738091d-b5) successfully.

User user123 booked event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091d-b5) successfully.

Friday, 10:05 AM

User user123 booked event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091d-b5) successfully.

User user123 booked event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091d-b5) successfully.

User user123 booked event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091d-b5) successfully.

Today, 10:42 AM

User user123 booked event 'None' (ID: 8112d9b5-dbd1-466d-8023-6b0738091d-b5) successfully.

Fig: 4.8.3: Notification has been sent

5. Learning Outcomes

1. Understanding Serverless Architecture: Learn the principles and benefits of building applications without managing servers, utilizing services like AWS Lambda and API Gateway for increased agility and reduced operational overhead [3].

2. Developing Event-Driven Systems: Grasp the concepts of event-driven architecture (EDA) and how services like Amazon EventBridge enable decoupled communication between different components of the application, leading to better scalability and resilience [7].

3. Implementing Event Management with EventBridge: Learn how to use Amazon EventBridge to create event buses, define rules to route events from various sources (including custom applications and AWS services), and connect them to target services for processing [1].

4. Securing AWS Resources with IAM: Understand how to use AWS Identity and Access Management (IAM) to create users, define permissions, and control access to AWS resources, ensuring the security of the booking system [4].

5. Monitoring and Logging with CloudWatch: Learn how to use AWS CloudWatch to monitor the performance and health of the application, collect and analyze logs from Lambda functions and other services, and set up alarms for proactive issue detection [5].

6. Interacting with AWS Services via CLI: Gain proficiency in using the AWS Command Line Interface (CLI) to interact with and manage various AWS services, including deploying Lambda functions and testing event flows [8].

7. Managing Data with DynamoDB: Understand how to design and interact with NoSQL databases like Amazon DynamoDB for storing application data, leveraging its scalability and performance for the booking system [3].

8. Debugging and Troubleshooting: Develop skills in identifying and resolving issues within a serverless and event-driven architecture using tools like CloudWatch Logs and by understanding event flow and service interactions

6. Conclusion

The Serverless Inventory Management System project successfully demonstrated the power of serverless architecture in building scalable, cost-effective applications by leveraging AWS Lambda, API Gateway, DynamoDB, IAM, and CloudWatch to provide a fully functional inventory management solution with a web-based interface for performing CRUD (Create, Read, Update, Delete) operations. The project effectively addressed challenges such as CORS configuration issues, DynamoDB Decimal serialization problems, and initial 500 Internal Server Errors through thorough debugging and the implementation of helper functions, resulting in a robust application that benefits from automatic scaling, cost efficiency, and reduced operational overhead. This development process underscored the importance of secure IAM role configuration, effective error handling, and the use of CloudWatch logs, validating the system's readiness for practical use with its deployment to the live API endpoint (https://mo1eyckg8c.execute-api.us-east-1.amazonaws.com/dev). Looking ahead, future enhancements such as user authentication with AWS Cognito, advanced search with DynamoDB indexes, integration with AWS S3 for file storage, and CSV export functionality could further elevate its utility, making it a versatile tool for small businesses or educational purposes while encouraging further exploration and iteration to meet evolving needs..

7. References

[1]**AWS.(n.d.).What is Amazon EventBridge? AWS Documentation**.
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html

[2] **Using AWS Serverless to Power Event Management Applications | AWS Architecture Blog:**
https://aws.amazon.com/blogs/architecture/using-aws-serverless-to-power-event-management-applications/

[3] **AWS Serverless for Event-Driven Architecture:**
https://aws.amazon.com/awstv/watch/a1142798bf0/ (This is a link to an AWS TV video/blog post)

[4] **AWS IAM Documentation:**
https://docs.aws.amazon.com/apigateway/

[5] **Building a Serverless Event-Driven Retail Order Management System | AWS for Industries:**
https://aws.amazon.com/blogs/industries/building-a-serverless-event-driven-retail-order-management-system/

[6] **What is EDA? - Event Driven Architecture Explained – AWS:**
https://aws.amazon.com/what-is/eda/

[7] **Building a Serverless Event-Driven Retail Order Management System | AWS for Industries:**
https://aws.amazon.com/blogs/industries/building-a-serverless-event-driven-retail-order-management-system/

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