

**Design And Implementation of a Serverless Event Management Platform on AWS**

**V. P. S. Sanjeewa - COADDS24.2F – 011**

**H. Kumareshan – COADDS24.2F – 022**

**D. M. D. Gunawardhana – COADDS24.2F – 024**

**Course : Cloud Computing for Data Science**

**Date of Submission = 10.09.2025**

Table of Contents

[Abstract 3](#_Toc208396158)

[01. Introduction 3](#_Toc208396159)

[1.1 Background : 3](#_Toc208396160)

[1.2 Project Objective : 4](#_Toc208396161)

[Other objectives : 4](#_Toc208396162)

[1.3 Application overview : 5](#_Toc208396163)

[02.Architecture Design 5](#_Toc208396164)

[2.1 Architecture Overview : 5](#_Toc208396165)

[2.2 Justification of AWS Services : 6](#_Toc208396166)

[2.3 Data flow and Architecture Diagram : 7](#_Toc208396167)

[03. Implementation Details 8](#_Toc208396168)

[3.1 Deployment steps : 8](#_Toc208396169)

[3.2 Security and IAM Configuration 10](#_Toc208396170)

[3.3 Key Screenshots 11](#_Toc208396171)

[04. Cost Analysis and Optimization 12](#_Toc208396172)

[4.1 Cost Estimation 12](#_Toc208396173)

[4.2 Cost Saving Strategies 13](#_Toc208396174)

[05. Challenges and Solutions 14](#_Toc208396175)

[Challenge 01 - Cross-Origin Resource Sharing (CORS) Errors 14](#_Toc208396176)

[Challenge 02 - Managing IAM permissions 14](#_Toc208396177)

[Challenge 03- Debugging Lambda Functions 14](#_Toc208396178)

[Challenge 04 - Data Backup and Recovery 14](#_Toc208396179)

[Challenge 05 – Cost Monitoring 15](#_Toc208396180)

[06. Conclusion and Future works 15](#_Toc208396181)

[6.1 Conclusion 15](#_Toc208396182)

[6.2 Key Findings 15](#_Toc208396183)

[6.3 Future Work 15](#_Toc208396184)

Table of Tables

[Table 1 - Cost Estimation 12](#_Toc208396567)

Table of Figures

[Figure 1 8](#_Toc208396636)

[Figure 2 - Architecture Diagram 9](#_Toc208396637)

[Figure 3 - IAM Role Permissions 11](#_Toc208396638)

[Figure 4 12](#_Toc208396639)

[Figure 5 12](#_Toc208396640)

[Figure 6 13](#_Toc208396641)

# Abstract

The following report explains the design and development of a serverless Event Management platform using Amazon Web service (AWS). The main objective of this project is to create a modern and flexible system for discovering and sharing events in sri lanka. For this platform we used several AWS features, like S3 for hosting the website, API Gateway for manage API requests, Lambda for backend, DynamoDB for store event and user details, overall these features provide a fully serverless architecture.

The platform shows how a serverless approach can be both cost effective and scalable, while also keeping things secure without the needs for heavy server management. It also make it easier for different types of users to register, organize and manage events

Overall the project highlights the value of using AWS is serverless tools to build modern applications. It not only works well for e vent event management but also can be taken as example for creating other web platforms that are low-maintenance and future ready.

# 01. Introduction

## 1.1 Background :

Cloud computing is the delivery of IT services, such as servers, storages ect and also cloud computing has revolutionized how applications are build and executed. Now days a big trend is serverless architecture. It is the forefront of the revolution, by handling server management, serverless computing allows developers to work with code alone. Yielding faster development cycles and decreased operational weight. AWS offer big amount of serverless features that scale automatically and have very high availability.

## 1.2 Project Objective :

The main objective of this study was to design and implement a serverless event management platform using AWS service, and specific functional and non functional features like scalability, security, and cost optimization.

### Other objectives :

* To provide a simple and scalable solutions for discovering, sharing and managing events in Sri Lanka.
* To utilize AWS features like S3, API Gateway, Lambda, and DynamoDB for hosting, backend processing and secure data storage.
* To ensure cost-effectiveness, scalability, and security through a fully serverless architecture.
* To demonstrate how serverless technology can reduce maintenance effort while supporting future growth.
* To create a platform that can act as a reference model for building other modern, low maintenance web application**.**

## 1.3 Application overview :

The web application name is Sri Lanka Events blog. It gives users the capability to view events that take place around the country and also gives organizers a chance to add and manage events, ordinary users can view events, but organizers can manage things easily. The idea is to keep it simple, intuitive and functional.

# 02.Architecture Design

## 2.1 Architecture Overview :

This is a serverless project since it offers the proper scale of scalability, reliability and affordability. With serverless scaling up or down depending on traffic is handled automatically by the system, it can handle peak times without we having to do anything. It follows a pay per use policy as well, so the cost is low as I am only paying for what is being used on the platform. Another important feature is high availability, through which the application is accessible to user at any time.

**AWS Services used for this Blog**

* S3
* API Gateway
* Lambda
* DynamoDB
* IAM
* CloudWatch
* CloudTrail
* VPC
* AWS Backup
* Billing and Cost Management

## 2.2 Justification of AWS Services :

Each AWS service was selected based on the specific need of the event management platform. Together, they create a serverless architecture that is cost efficient, scalable and secure.

* S3 for hosting the website.
* API Gateway for backend-frontend communication management.
* Lambda for serverless backend code execution.
* DynamoDB for storing events and event details.
* IAM for access management with security.
* CloudWatch for monitor the system and checking performance.
* CloudTrail for recording and logging API activity for auditing and security.
* VPC for secure networking between services.
* AWS Backup for backing up data and recovery when needed.
* Billing and Cost Management for monitoring usage and keeping the project on budget.

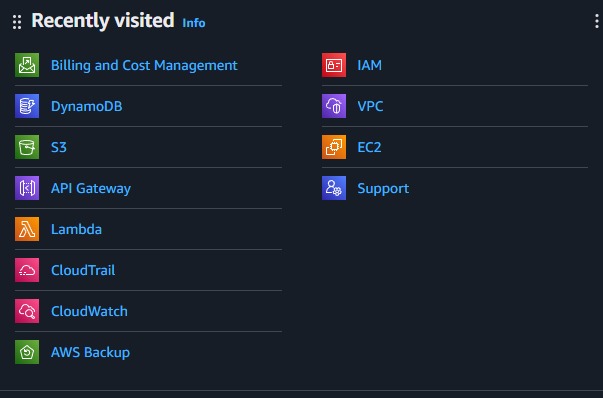


Figure 1

Collectively, these services enable the platform to be lightweight and serverless without sacrificing security, maintainability, and readiness for future scale.

## 2.3 Data flow and Architecture Diagram :

* User access the website through the URL, which it’s serves by Amazon **S3** via **CloudFront**.
* Browser load the files(HTML, CSS and JS) from **S3**.
* When user interact with the Web page **JavaScript** code make a **HTTP POST** request and send to the **API Gateway**.
* **API Gateway** activate the **Lambda** function
* The **lambda** function, get the permission by **IAM** and enter a new event data into **DynamoDB**.
* **CloudWatch** monitor the **Lambda** functions, errors and performance.
* **CloudTrail** record all API and make changes to the AWS for auditing.
* **AWS** **Backup** make sure that **DynamoDB** data and other resources are backed up.
* If there and issue, **support** **tools** notify the admin for troubleshooting.
* **VPC** make sure that the backend services are running perfectly and secure.

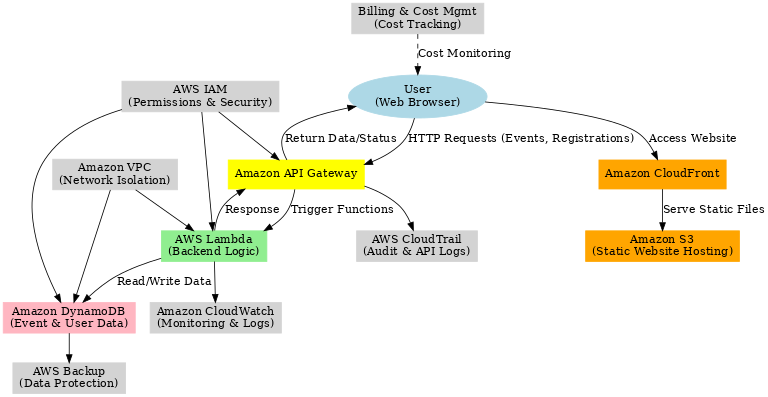


Figure 2 - Architecture Diagram

# 03. Implementation Details

## 3.1 Deployment steps :

The following steps are represent the Deployment methods of Event management blog.

1. VPC setup :

VPC was created by using VPC Wizard. This network environment created for secure communication between Lambda and DynamoDB.

1. S3 Bucket configuration :

S3 Bucket created name as explore-sri-lanka-events for hosting the frontend. After that bucket updated to allow public read access, that makes website accessible from the browser for public.

1. DynamoDB Table creation :

DynamoDB table named Events was set with event\_id as a primary key. This table work as storage for all event data.

1. Lambda Function Deployment :

We created Lambda function by using Node.js to monitor API requests. All this functions work with both GET and POST requests, and also CloudWatch monitor logs and track errors.

1. API Gateway Configuration :

REST API set up with path /events. Both GET and POST methods are going with the lambda function and CORS was allow requests from the S3 website.

1. Frontend Development :

After API update to the S3 bucket all the HTML, CSS, JavaScript files were uploaded.

1. Optional enhancements :

* AWS account activities were monitored by CloudTrail.
* Created Lambda and on-demand backup for protect data and tables.

## 3.2 Security and IAM Configuration

**IAM Role for Lambda :**

IAM role was created for separate the lambda function.

**Custom Policy :**

For the Event table we used read new events method and add new event

Method,

**Monitoring and Logging :**

We used CloudWatch set up with Lambda for secure and loggings, and

also CloudTrail to monitor all API.

Also we created a Billing Alarm for our website, If our predicted cost exceed

We will got a e-mail about that.

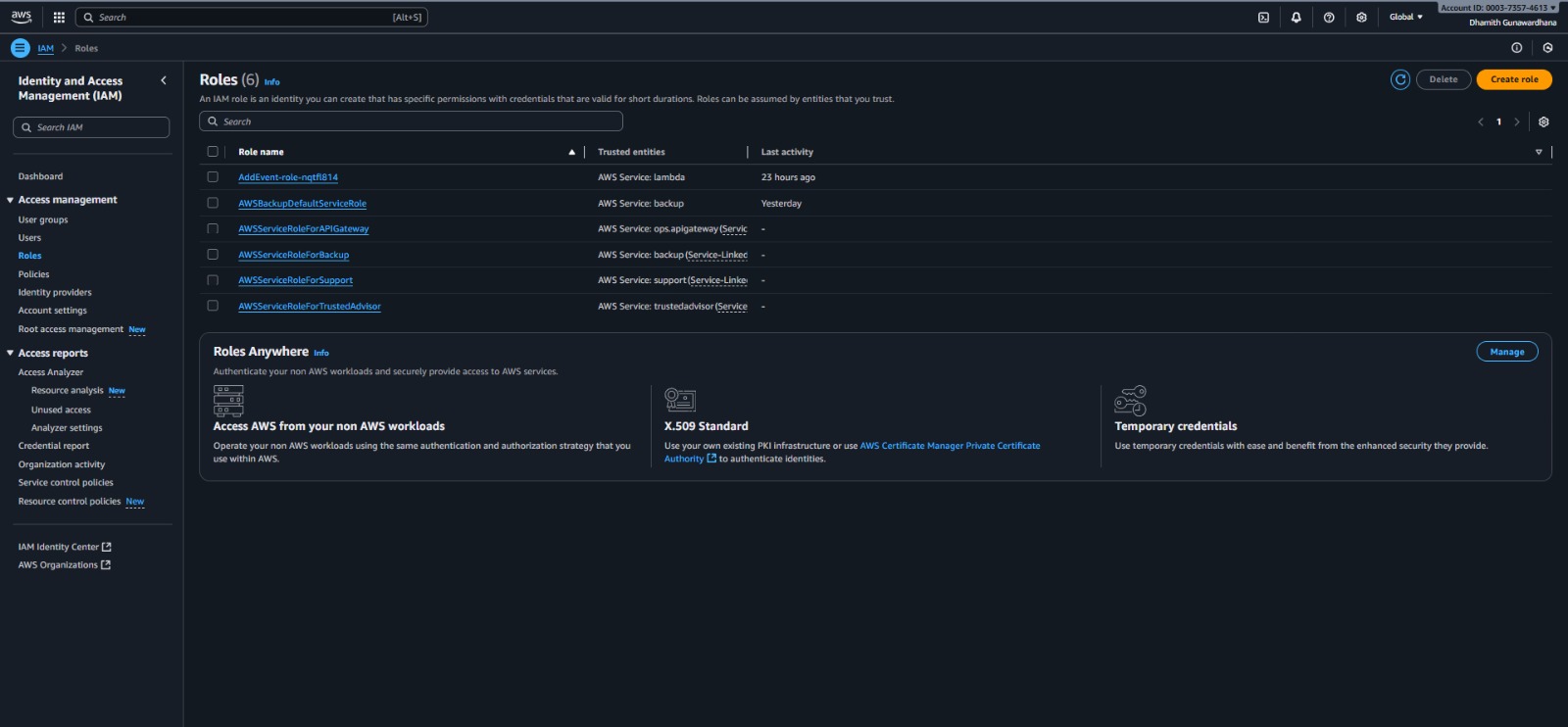


Figure 3 - IAM Role Permissions

## 3.3 Key Screenshots

1. **S3 static website Hosting**

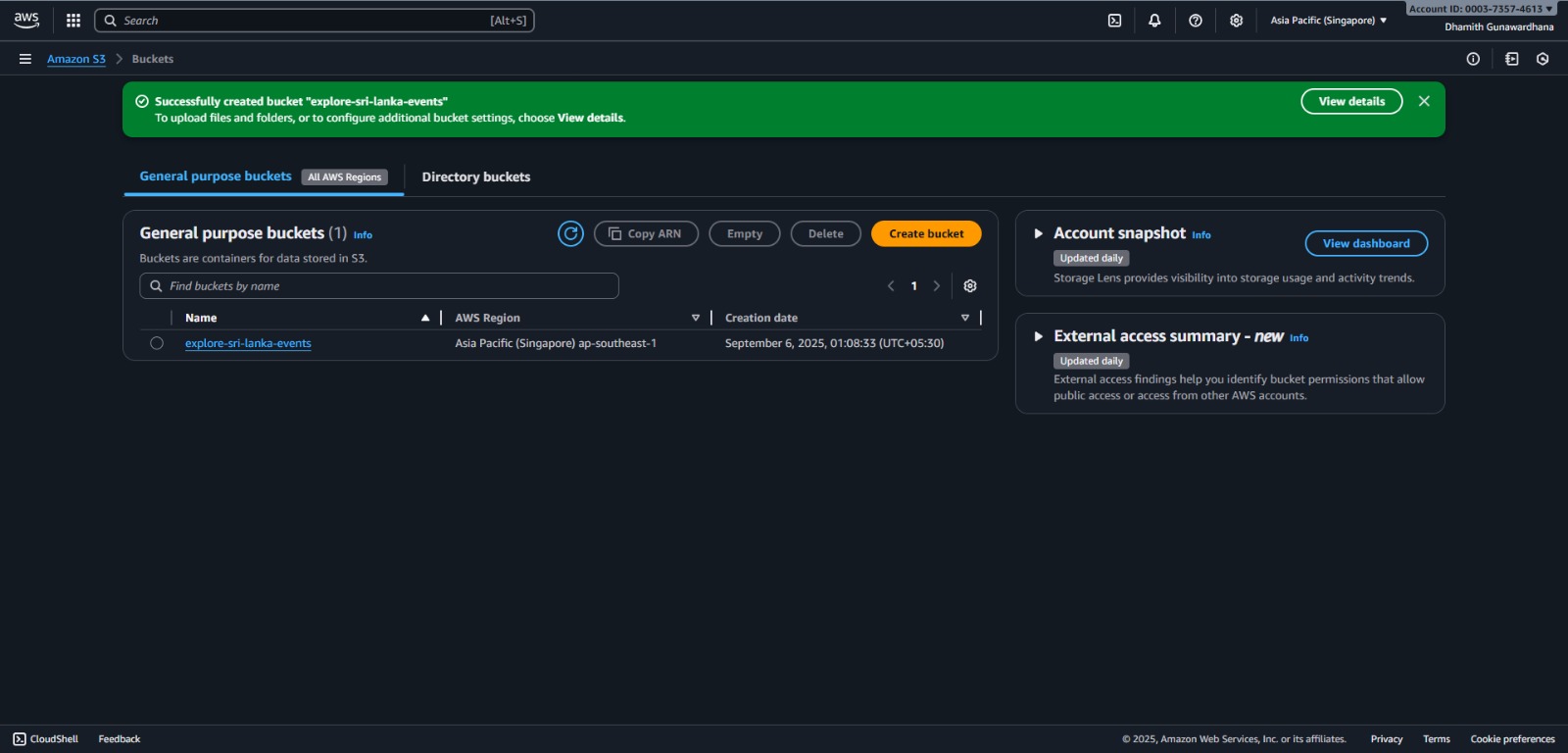


Figure 4

1. **DynamoDB Table Structure**

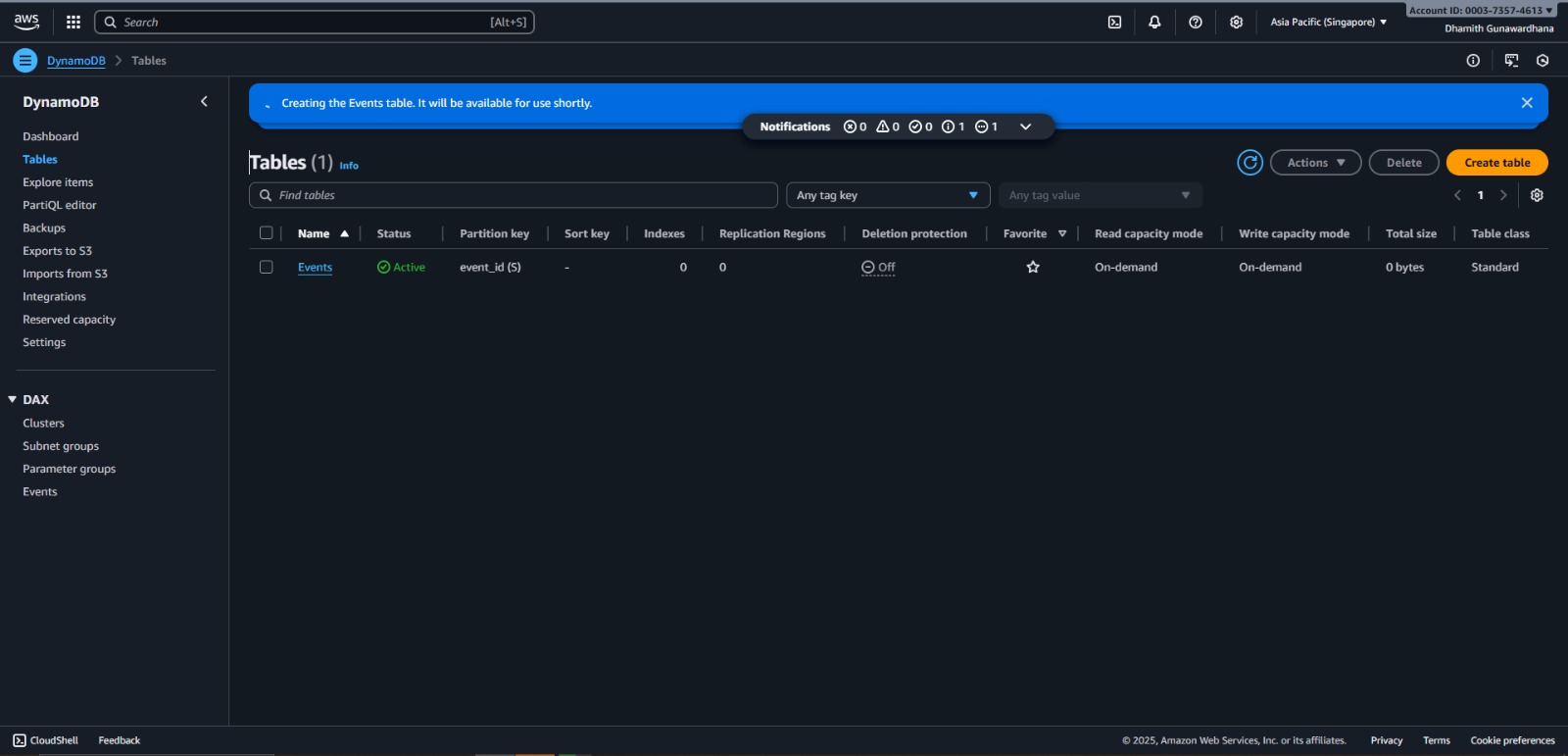


Figure 5

1. **API Gateway Resources**

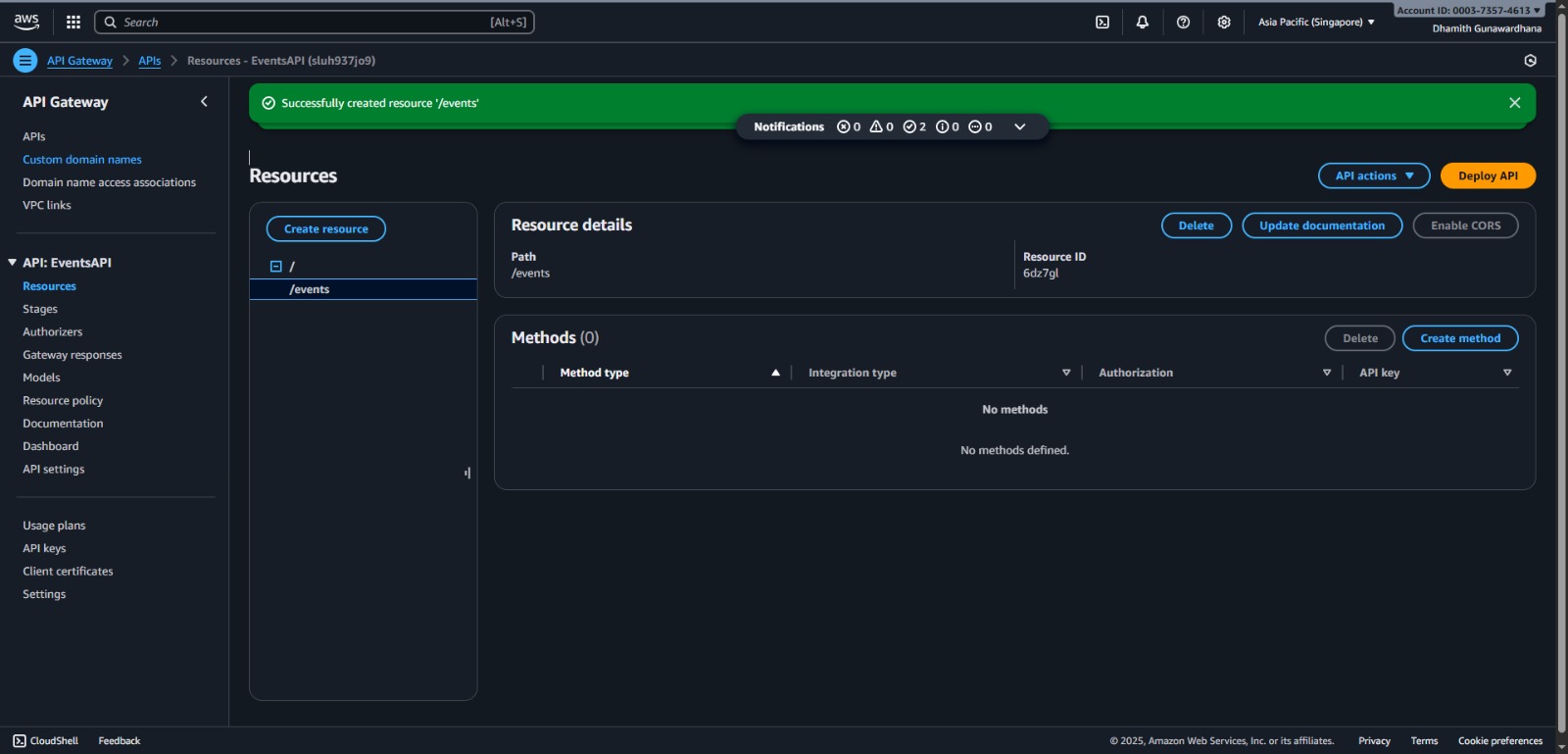


Figure 6

# 04. Cost Analysis and Optimization

## 4.1 Cost Estimation

Use of serverless architecture is big benefit for budget Creators. It is cost effective for low traffic applications. According to AWS pricing calculator, most of the features are free for use.

|  |  |
| --- | --- |
| **Service** | **Estimated Monthly Cost** |
| AWS Lambda | $0.00 (Always Free : 1 M Requests, 400k GB - Seconds) |
| Amazon API Gateway | $0.00 (Always Free : 1 M REST API calls) |
| Amazon DynamoDB | $0.00 (Always Free : 25 GB Storage, 200 M read requests) |
| Amazon S3 | $0.00 (Always Free : 5 GB Storage) |
| **TOTAL** | **$0.00** |

Table 1 - Cost Estimation

## 4.2 Cost Saving Strategies

As earlier mentioned, use of serverless services makes this project highly cost-efficient, based on AWS pricing calculators. Yet to ensure long-term cost efficiency as the system scales several cost saving strategies were adopted:

1. **Maximizing Free Tier** – Keeping storage and request volumes within AWS Free Tier limits wherever possible.
2. **DynamoDB Optimization** – Using on-demand pricing for unpredictable workloads, with the option to switch to provisioned auto-scaling for stable traffic.
3. **Efficient Lambda Design** – Minimizing execution time, avoiding overprovisioned memory, and reusing shared libraries via Lambda Layers.
4. **CloudFront Caching** – Serving static content through CloudFront reduces repeated S3/API calls.
5. **Lifecycle Policies on S3** – Moving outdated logs and files to cheaper storage classes such as S3 Infrequent Access or Glacier.
6. **Billing Alerts** – Setting up cost alarms and resource tagging to detect unnecessary usage.
7. **Backup Optimization** – Scheduling DynamoDB backups efficiently and retaining only critical snapshots.
8. **Future Scalability** – Considering Reserved Capacity and AWS Savings Plans for cost reduction when user traffic grows beyond Free Tier limits.

From these above proposed 08 strategies we can minimize the cost estimation to the maximum level.

# 05. Challenges and Solutions

Throughout the entire development period of the serverless event management platform for a huge higher educational institute like National Institute of Business Management (NIBM) including various branches and faculties, hosting several ice breaking, celebration and extra curricular club related events we faced various Challenges. Here are the challenges we faced during the event management platform and the solutions we propose:

### Challenge 01 - Cross-Origin Resource Sharing (CORS) Errors

*When connecting the frontend hosted on S3 with the backend API Gateway, initial requests were blocked due to missing CORS headers.*

**Solution:** CORS policies were explicitly enabled in API Gateway, allowing requests from the S3-hosted domain.

### Challenge 02 - Managing IAM permissions

*Configuring the right level of permissions for Lambda functions and DynamoDB access was complex.*

**Solution:** Custom IAM roles and policies were created with the principle of least privilege, granting only the required permissions.

### Challenge 03- Debugging Lambda Functions

*Error tracking in serverless functions was initially difficult due to lack of visibility.*

**Solution:** CloudWatch logs were integrated with Lambda to provide real-time monitoring and debugging.

### Challenge 04 - Data Backup and Recovery

*Ensuring that event data was not lost during testing required careful planning.*

**Solution:** Automated DynamoDB backups and AWS Backup service were implemented for reliable recovery.

### Challenge 05 – Cost Monitoring

*Since serverless costs scale with usage, there was concern about unexpected expenses.*

**Solution:** AWS Billing Alarms were set up to send notifications if estimated monthly costs exceed defined thresholds.

# 06. Conclusion and Future works

## 6.1 Conclusion

Using Amazon Web Services, this project was successfully demonstrated the design and implementation of a serverless event management platform by integrating S3, API gateway, Lambda and DynamoDB within a secure and scalable architecture.

This particular platform provides an efficient and cost effective solution for event discovery and management in Sri Lanka.

## 6.2 Key Findings

Referring to this framework, here are the few key findings of this projects which may significantly helps to do more future works based on this frameworks:

1. Serverless architecture significantly reduces infrastructure management overhead.
2. AWS Free Tier makes the platform cost-neutral at low traffic levels.
3. Scalability and high availability are inherent benefits of the chosen architecture.

## 6.3 Future Work

few improvements on this serverless event management platform, can lead to more efficient, powered and user friendly deployments in future works.

These are few implementations and future improvements basically a suggestions for future studies as follows:

1. **Mobile App Support:** Develop exclusive Android and iOS applications for wider accessibility based on those operating systems.
2. **Analytics & Insights:** Use AWS QuickSight to analyze user behavior and event trends, and provide more personalized services.
3. **Personalized Recommendations:** Integrate machine learning (e.g., Amazon Personalize) to suggest relevant events to users.
4. **Multi-Region Deployment:** Deploy in multiple AWS regions to reduce latency and enhance disaster recovery which gives user friendly interface.
5. **Advanced Security:** Incorporate AWS WAF and Shield for protection against malicious attacks.