# Serverless Labs

## Building a Serverless Web Application

### Objective:

Create a serverless web application using AWS Lambda, API Gateway, S3, and DynamoDB.

### Approach:

**Set Up Backend:** Create Lambda functions to handle backend logic. These functions will interact with a DynamoDB table for data storage.

**API Gateway**: Set up API Gateway to create RESTful endpoints that trigger the Lambda functions.

**Frontend Hosting**: Host a static website on S3 that interacts with the backend via API Gateway.

**Integration**: Ensure that the frontend can successfully send requests to the backend and display responses.

### Goal:

Understand the basics of building and connecting serverless backend services with a static frontend, enabling a fully serverless web application.

## 1. Creating table in DynamoDB

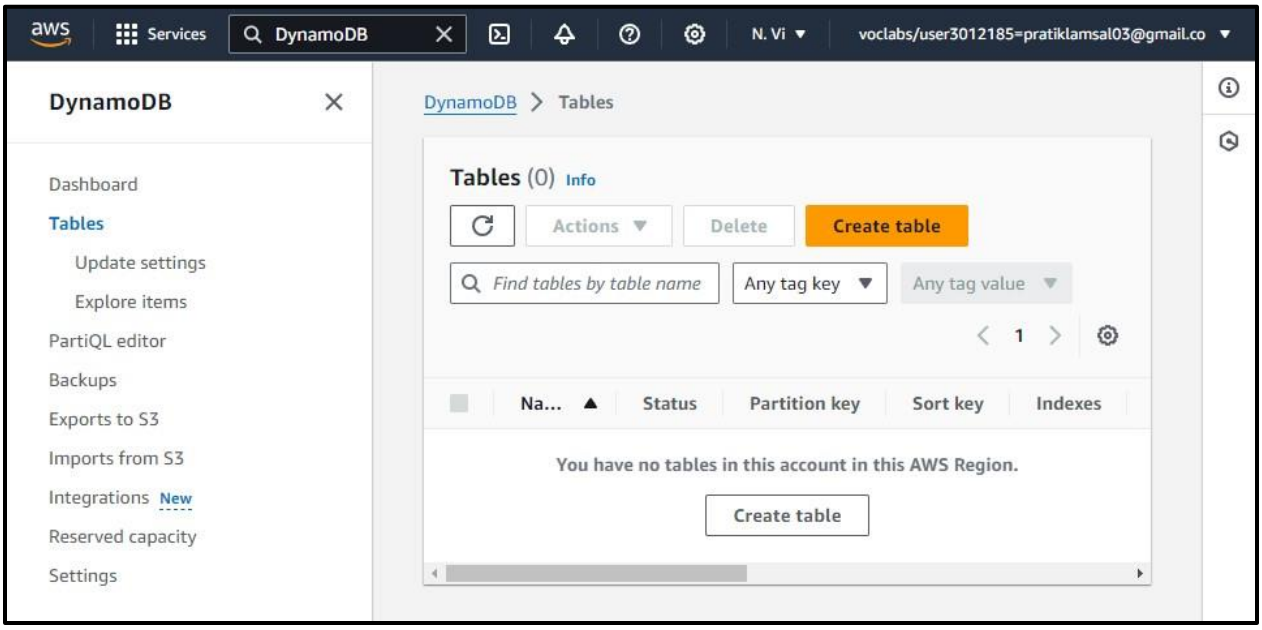


*Figure 1 Table Creation*

## 2. Table Details

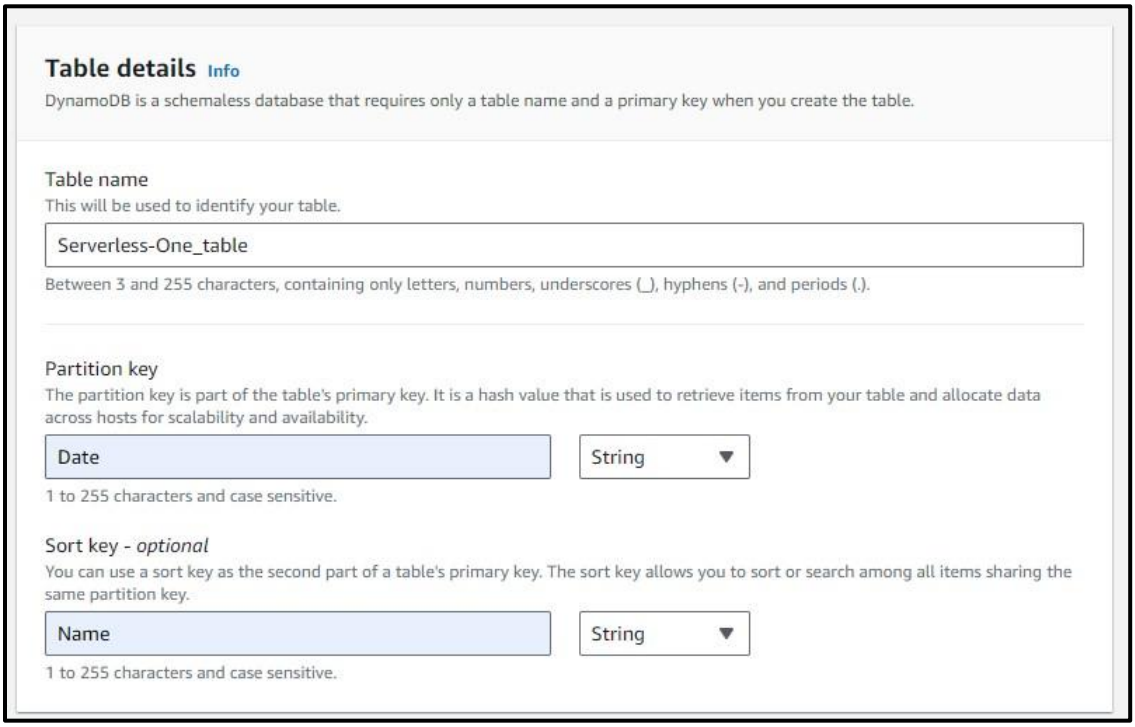


*Figure 2 Table Details*

## 3. Table Created Successfully

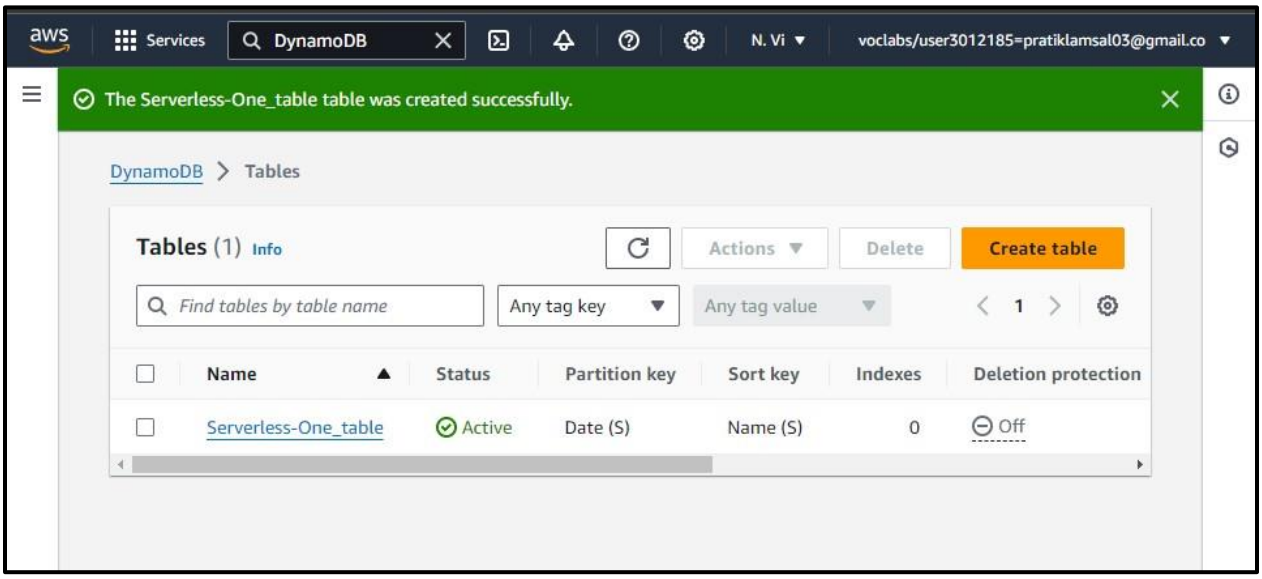


*Figure 3 Successful Table Creation*

## 4. Lambda Function Creation

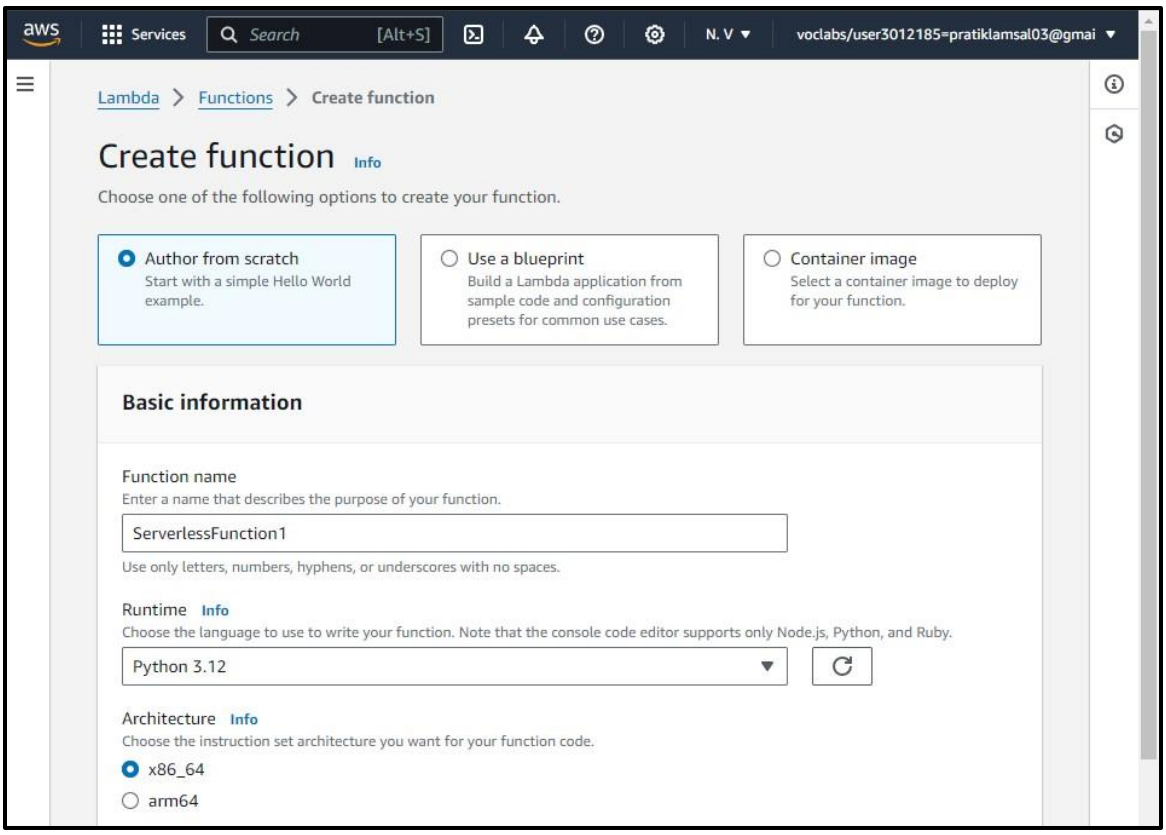


*Figure 4 Lambda Function Creation*

## 5. Permission Assignment

An existing role (LabRole) is assigned to the Function

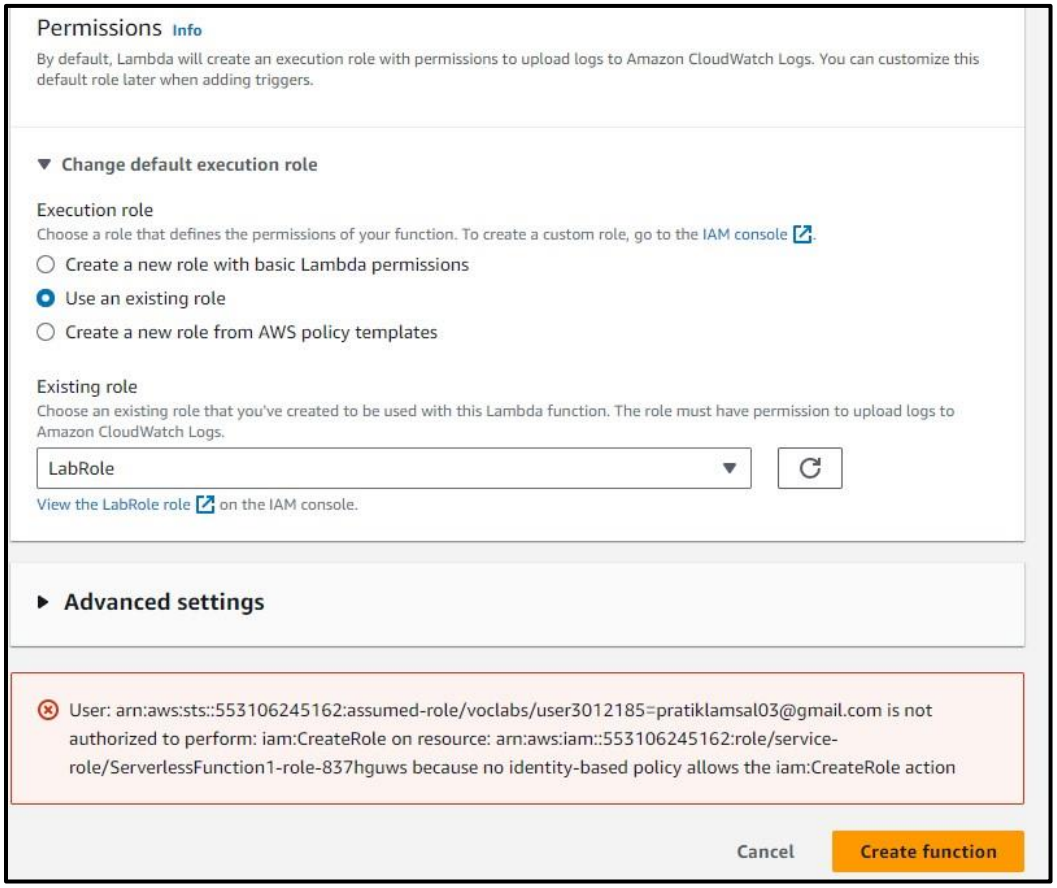


*Figure 5 Role Assignment*

## 6. Successful Function Creation

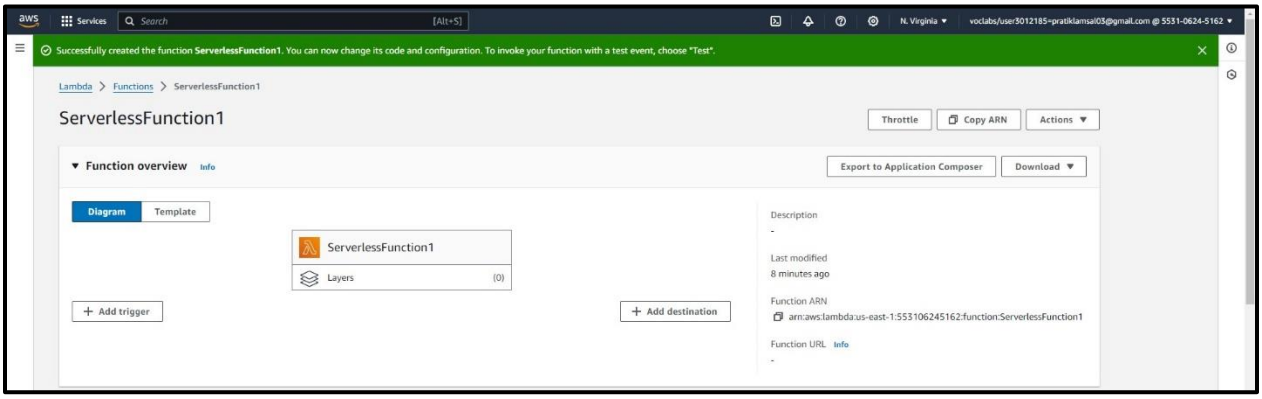


*Figure 6 Function Created Successfully*

## 7. Lambda Function Code Source

A Code Source is Added to the Lambda Function and deployed.

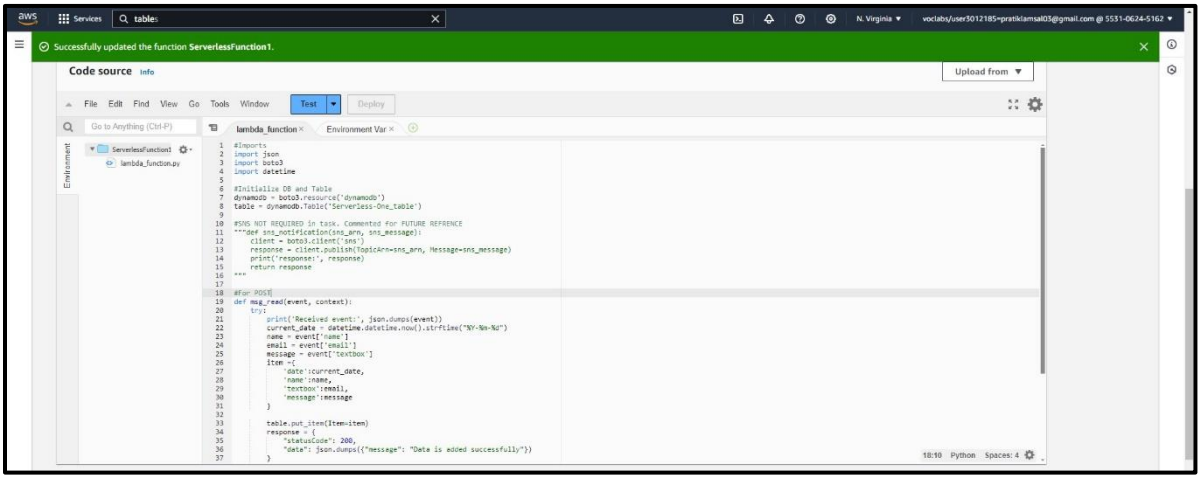


*Figure 7 Lambda Function Code Source*

## 8. REST API Creation

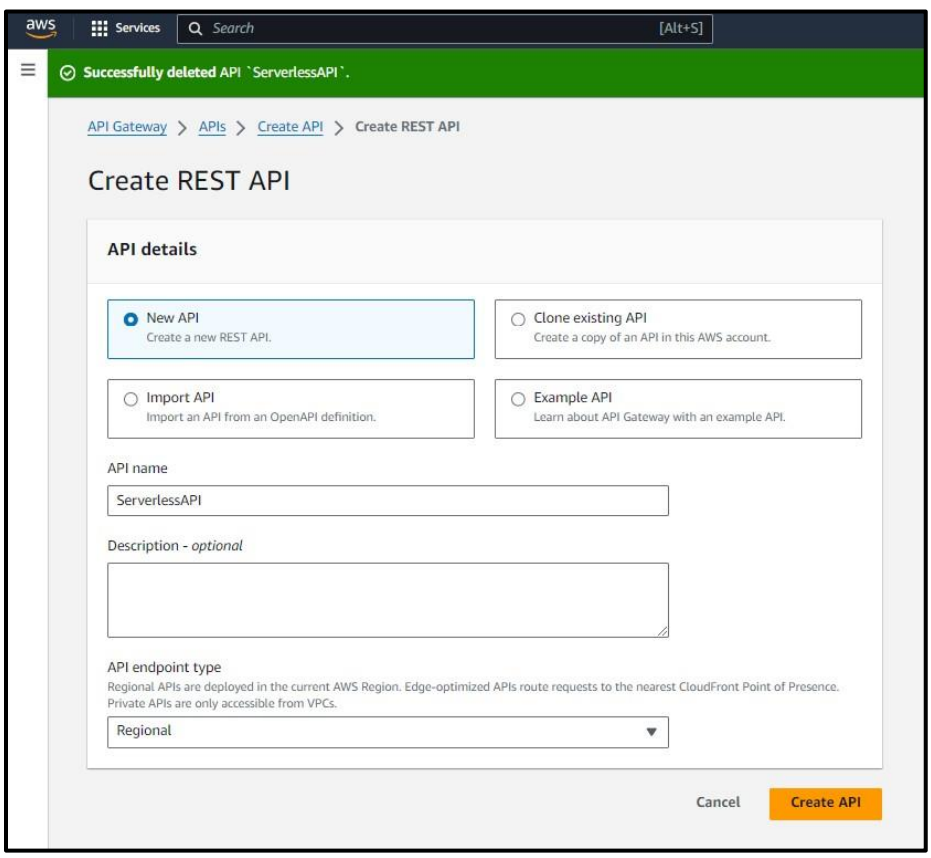


*Figure 8 REST API Creation*

## 9. Resource Creation

REST API is created successfully. Now, resources required are to be created.

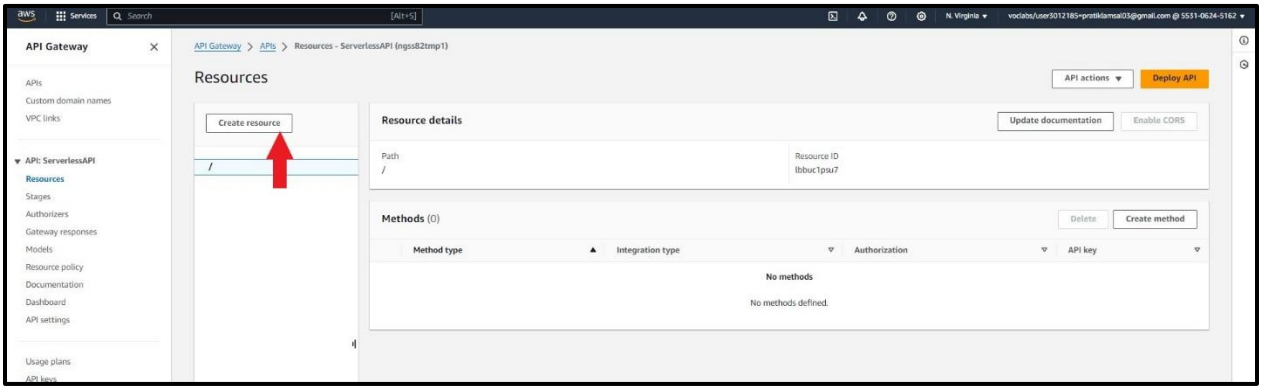


*Figure 9 Resource Creation*

## 10. Resource Details

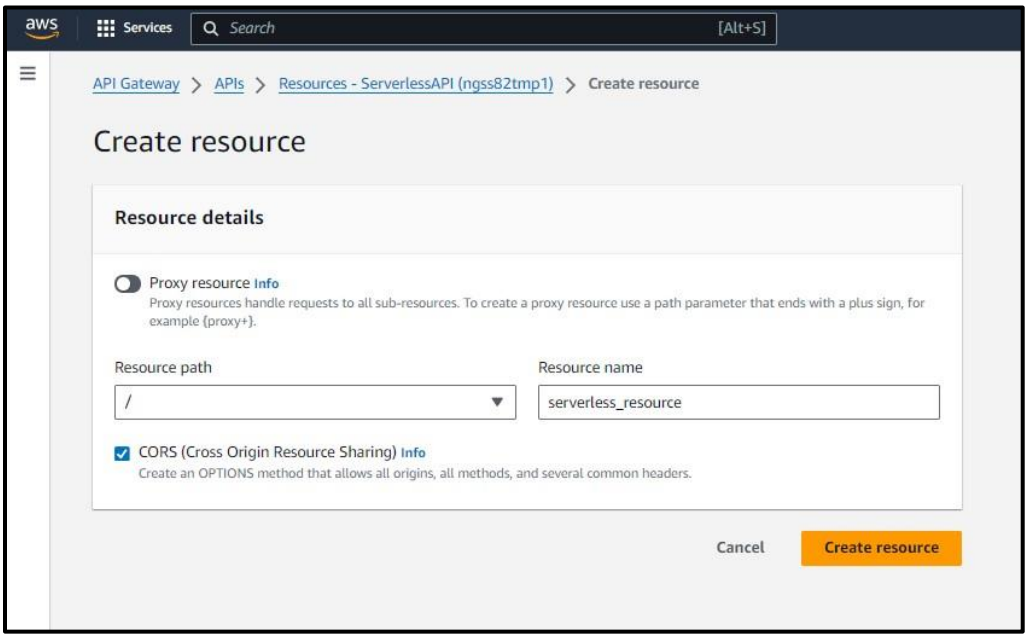


*Figure 10 Resource Details*

## 11. Resource Created Successfully

Resource is Created Successfully. Now, methods are to be created.

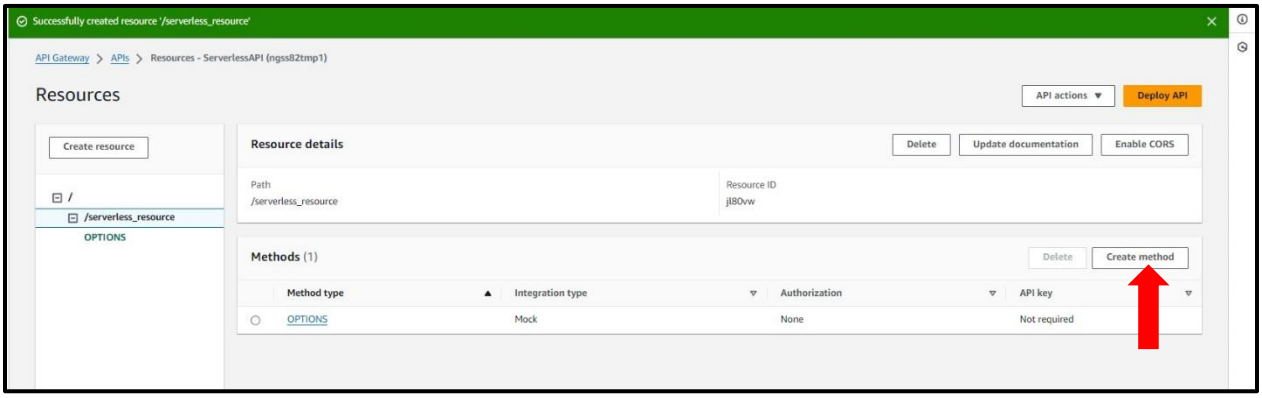


*Figure 11 Successful Resource Creation*

## 12. POST Method Creation

POST method is created and Lambda Function Created earlier is selected.

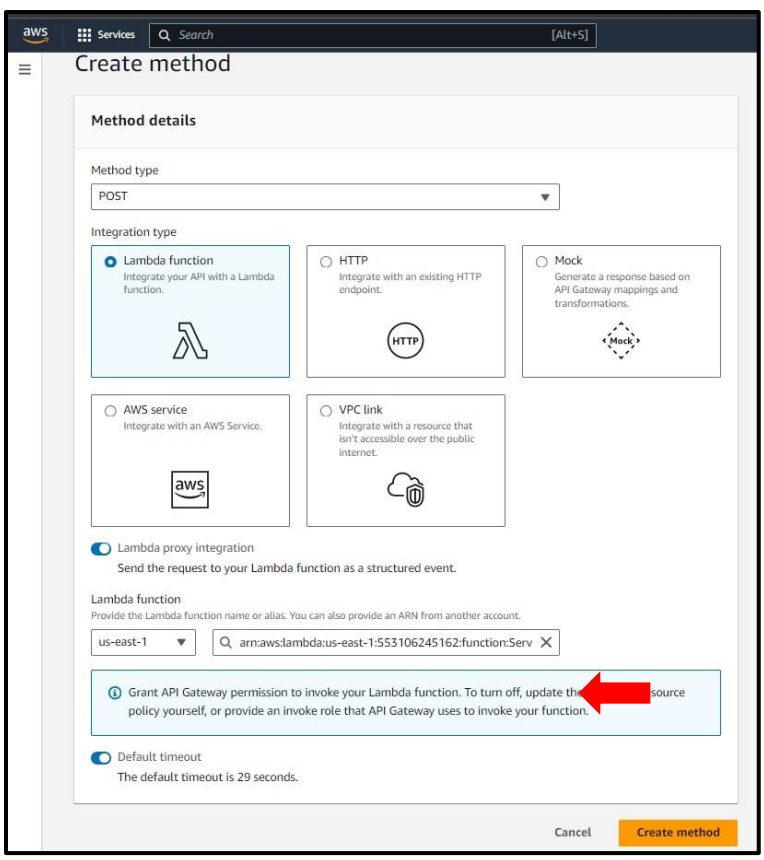


*Figure 12 POST Method*

## 13. Successful Method Creation

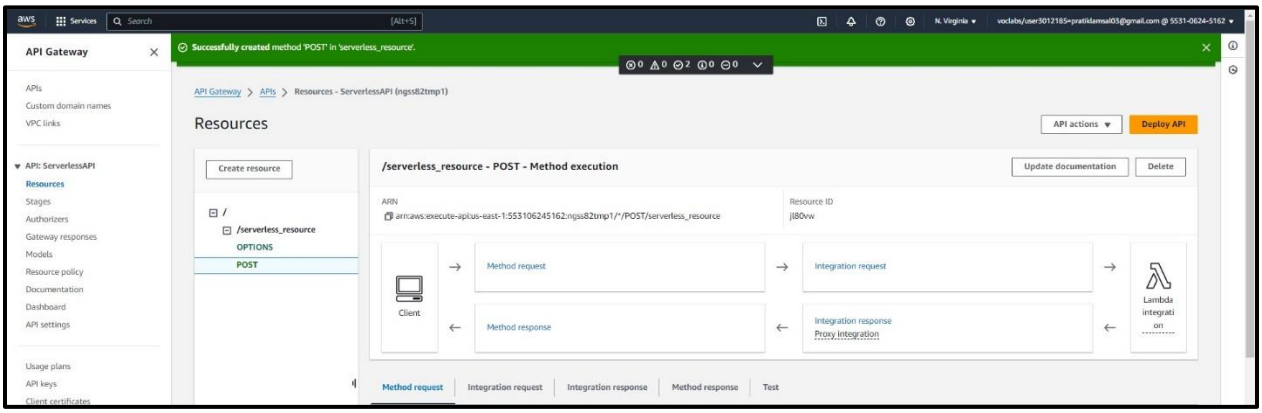


*Figure 13 Successful Method Creation*

## 14. Enabling CORS

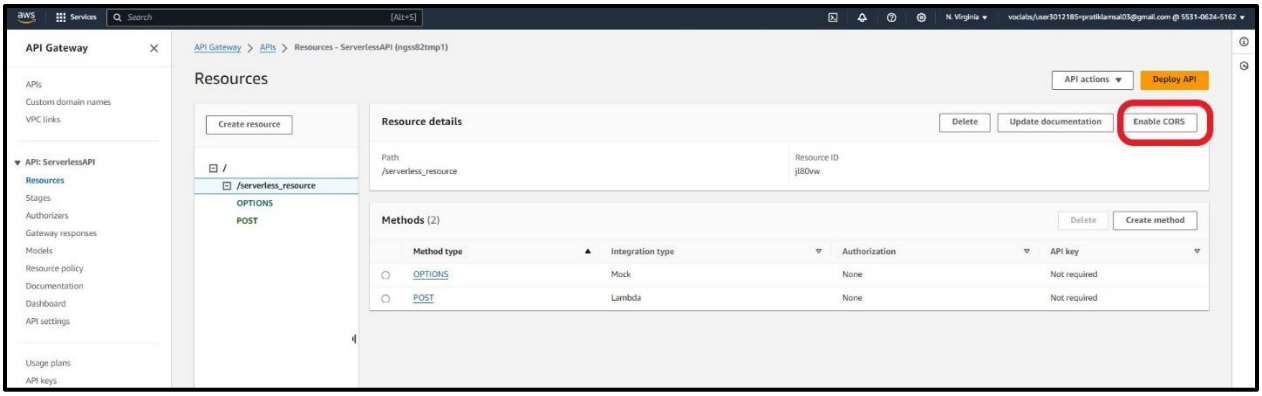


*Figure 14 Enabling CORS*

## 15. CORS Settings

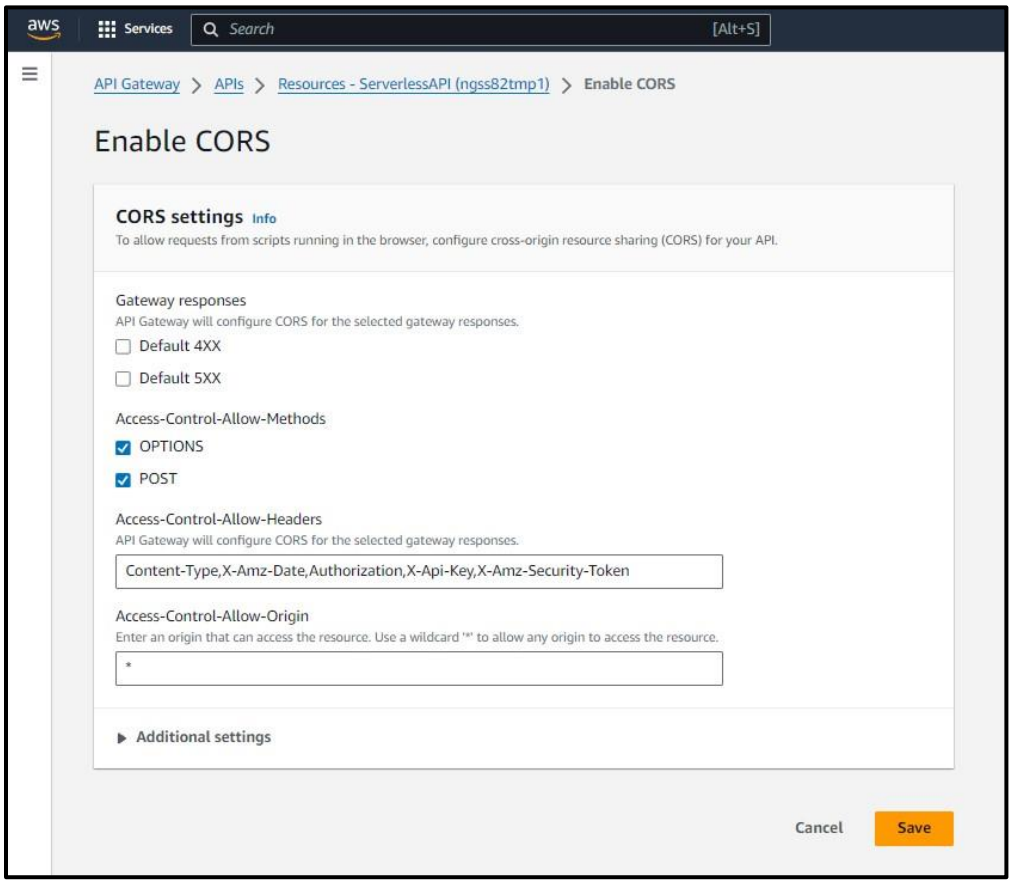


*Figure 15 CORS Settings*

## 16. Successful CORS Enabling

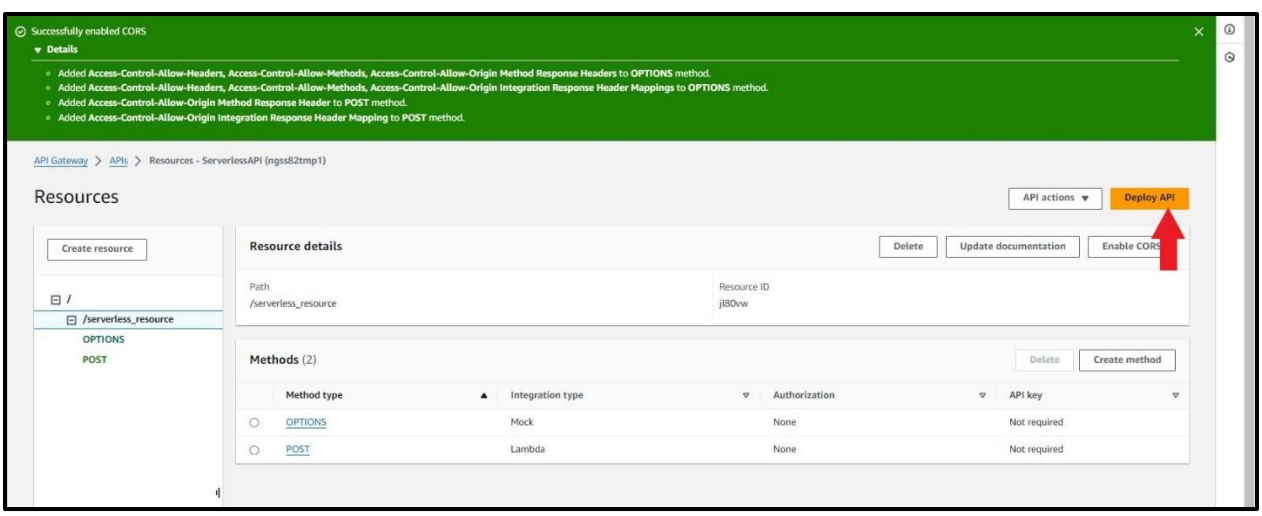


*Figure 16 CORS Enabled Successfully*

## 17. API Gateway for Function

API Gateway is added automatically.

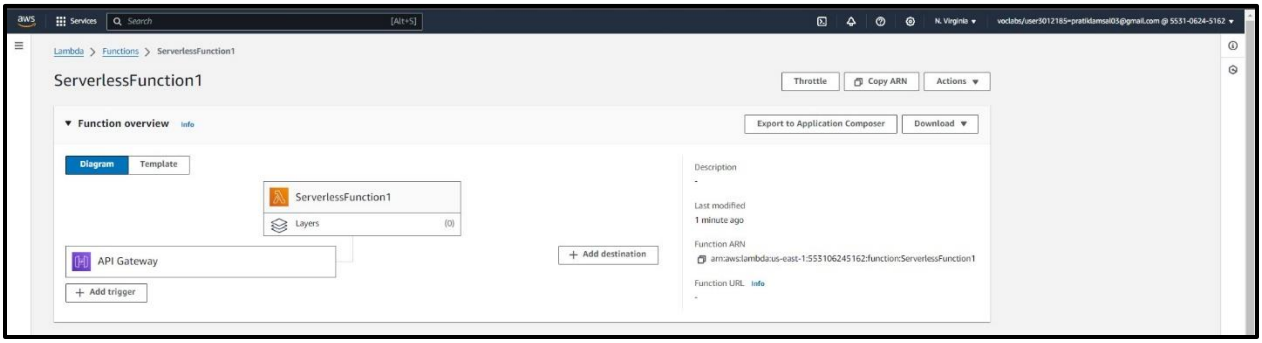


*Figure 17 API Gateway Addition*

## 18. Bucket Creation

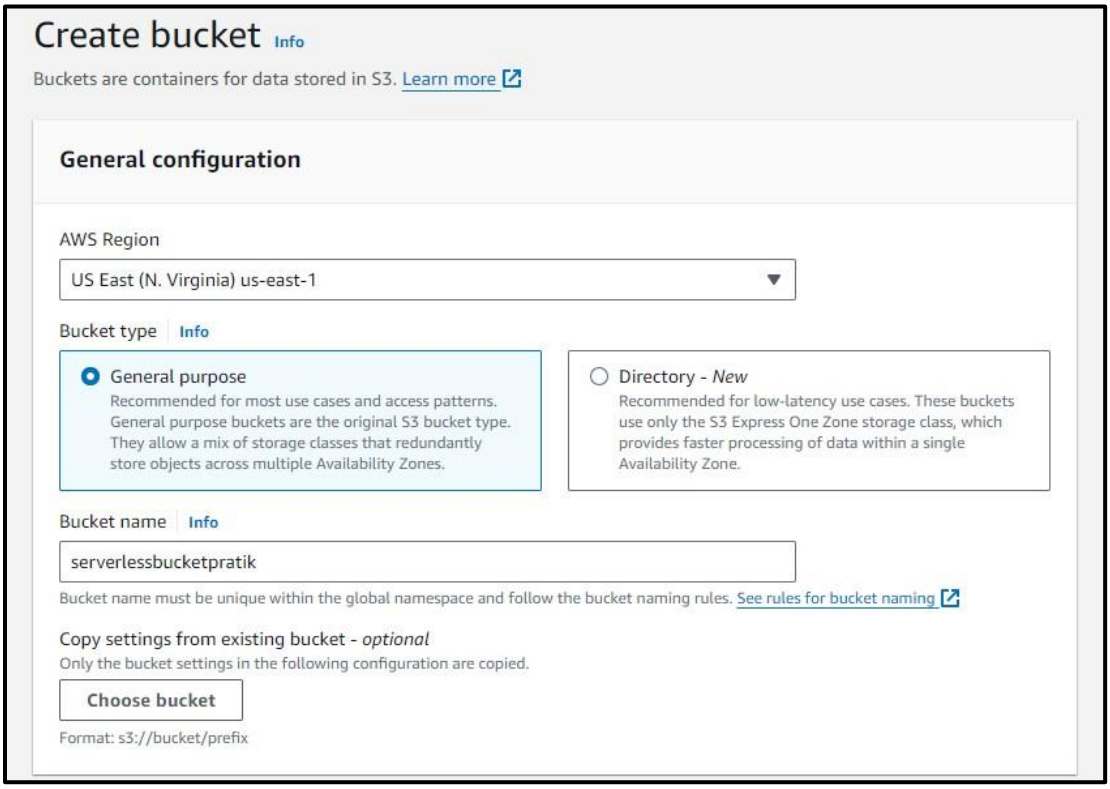


*Figure 18 Bucket Creation*

# 19. ACL Settings

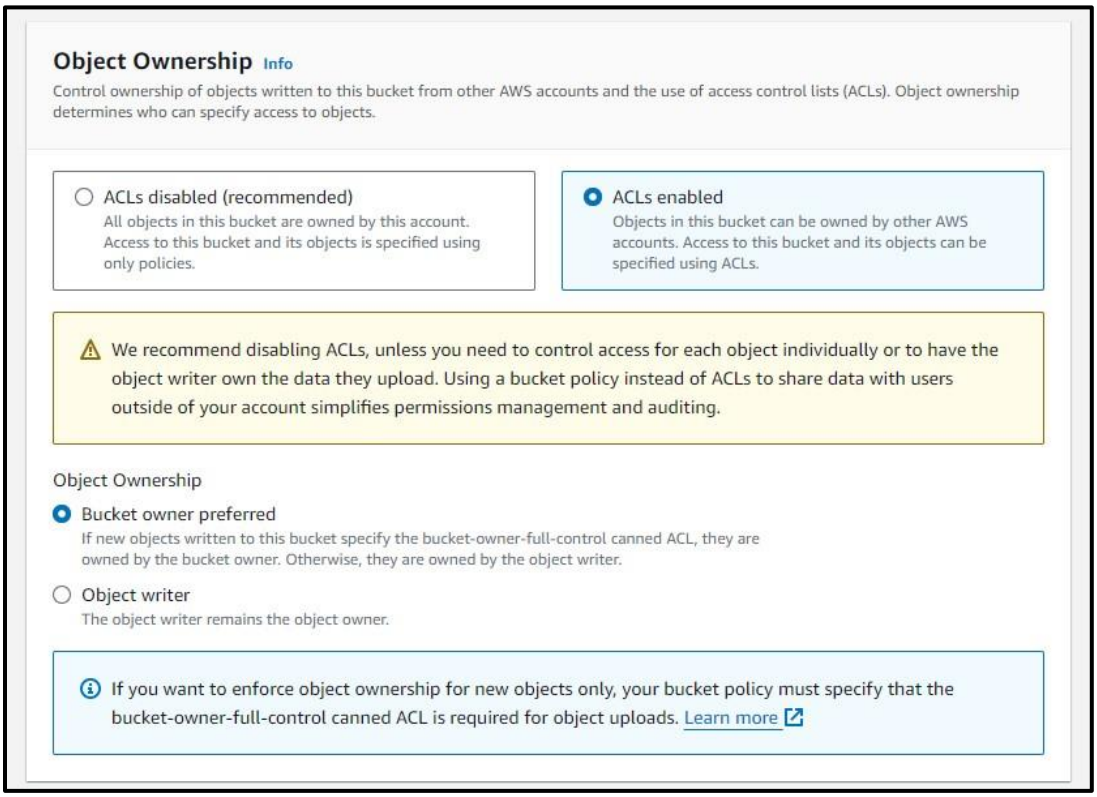


*Figure 19 Bucket ACL Settings*

## 20. Public Access Settings

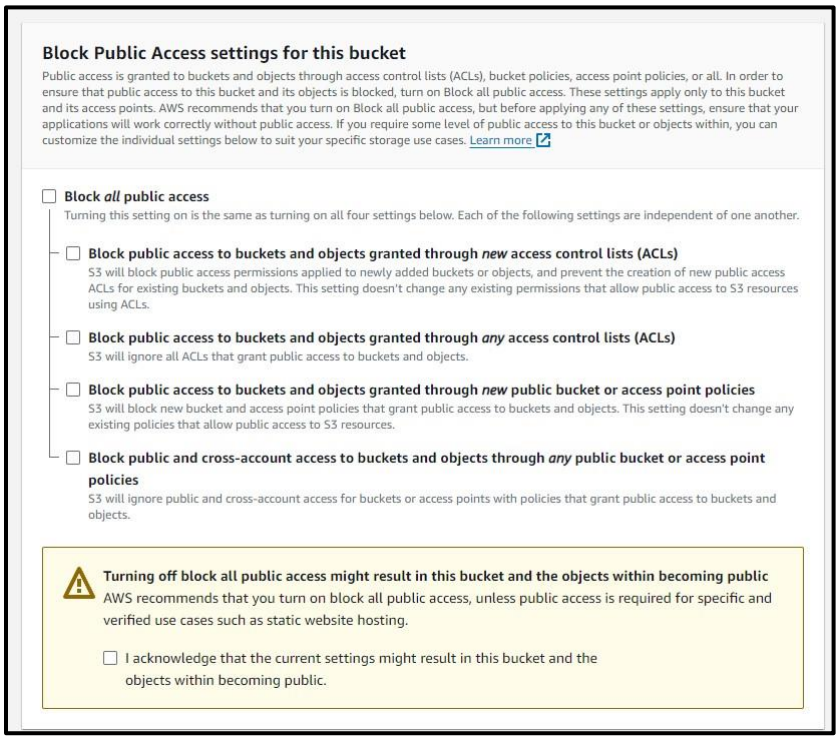


*Figure 20 Bucket Public Access Settings*

## 21. Successful Bucket Creation

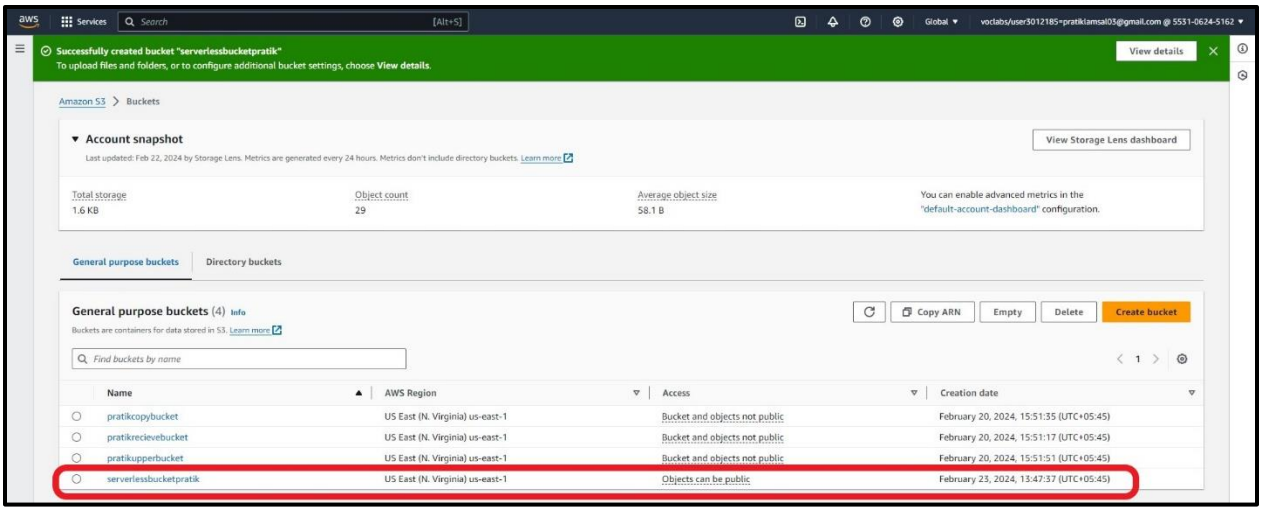


*Figure 21 Successful Bucket Creation*

## 22. API Deployment

Forgot to Deploy API in previous steps as errors occurred and new API was created again. Now Deploying by creating a new stage.

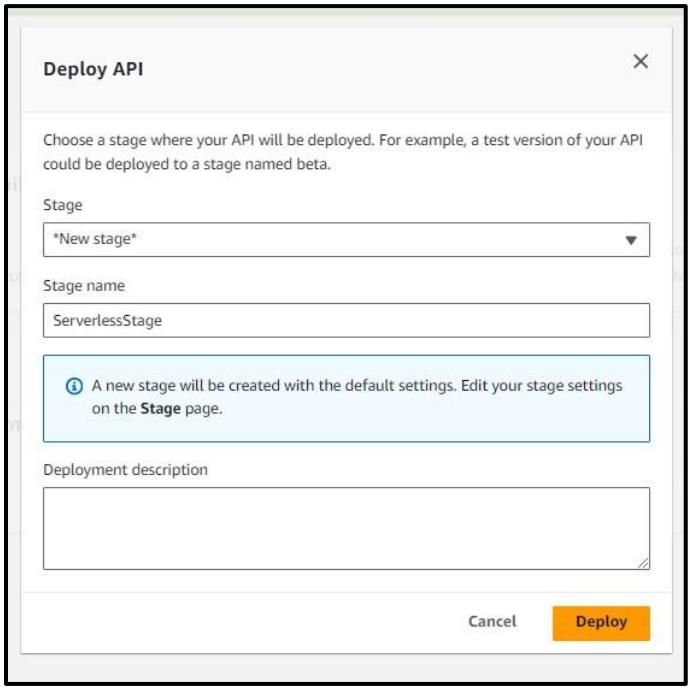


*Figure 22 API Deployment*

## 23. Stage Created

Copied for use in the code mentioned above.

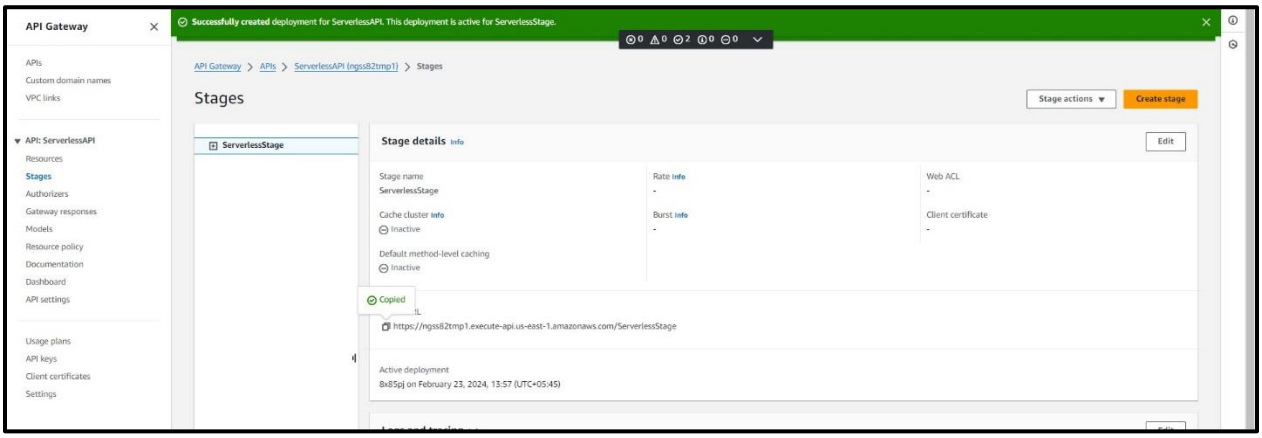


*Figure 23 Stage Created Successfully*

## 24. File Upload

HTML file uploaded to bucket.

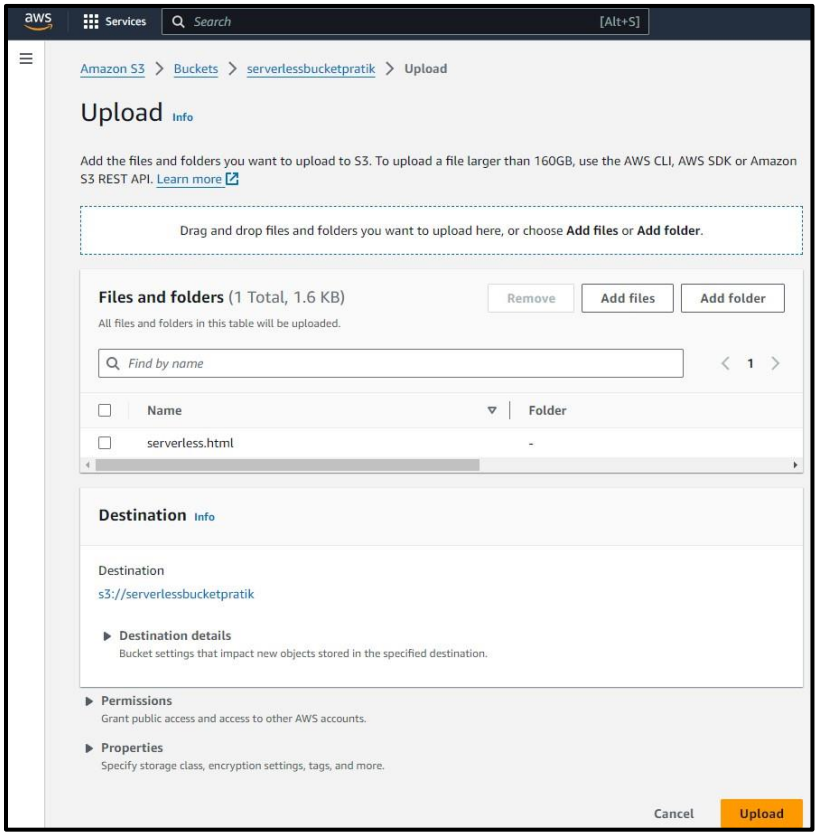


*Figure 24 HTML File Upload*

## 25. File Uploaded Successfully

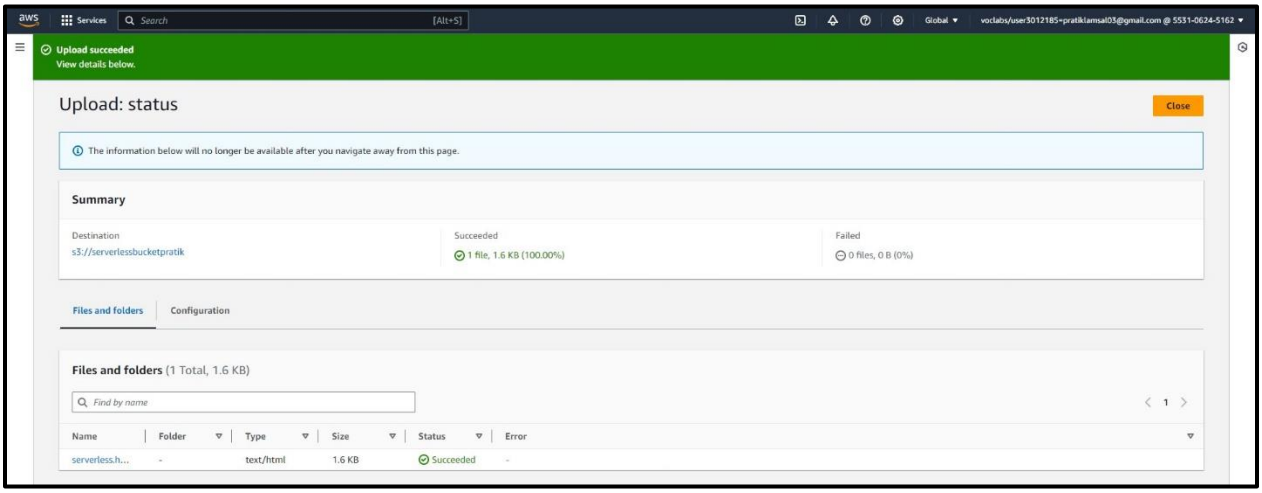


*Figure 25 Successful File Upload*

## 26. Object URL

Object URL is copied. This can be done by clicking on the object and clicking on the url.

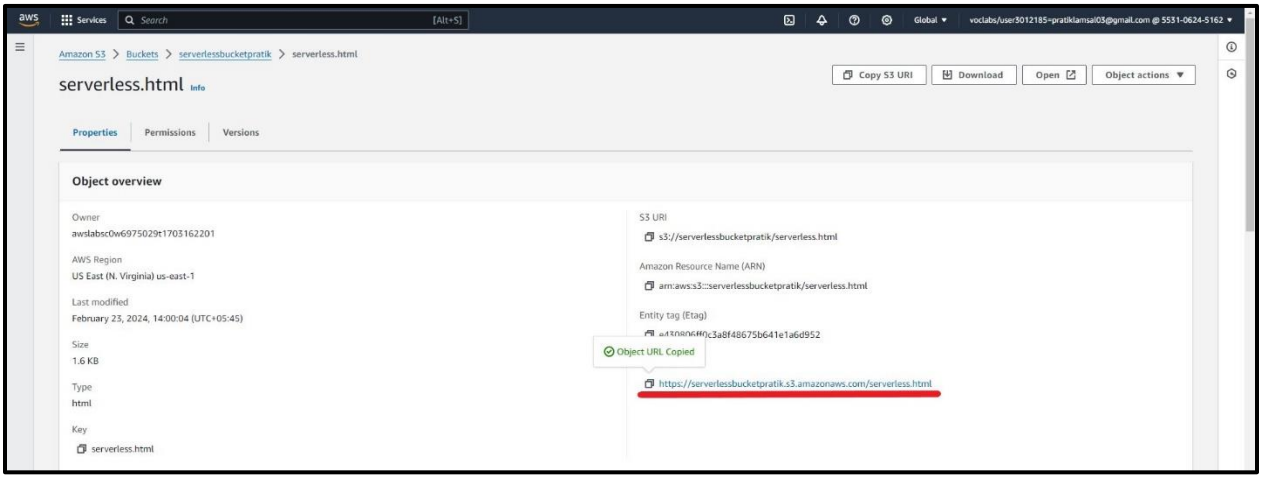


*Figure 26 Object URL*

## 27. Unsuccessful Attempt

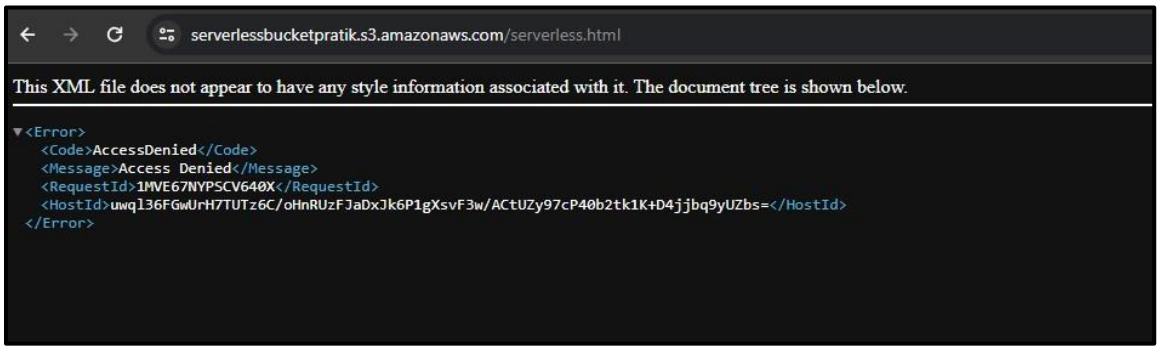


*Figure 27 Unsuccessful Attempt*

## 28. Bucket ACL

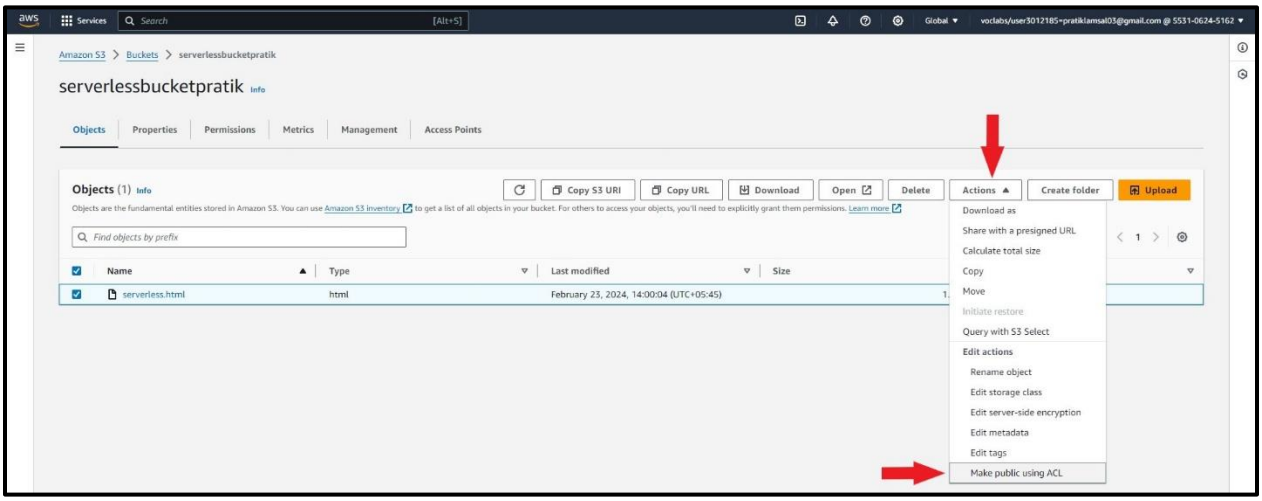


*Figure 28 Changing Bucket ACL*

## 29. Public Bucket

Bucket is made public.

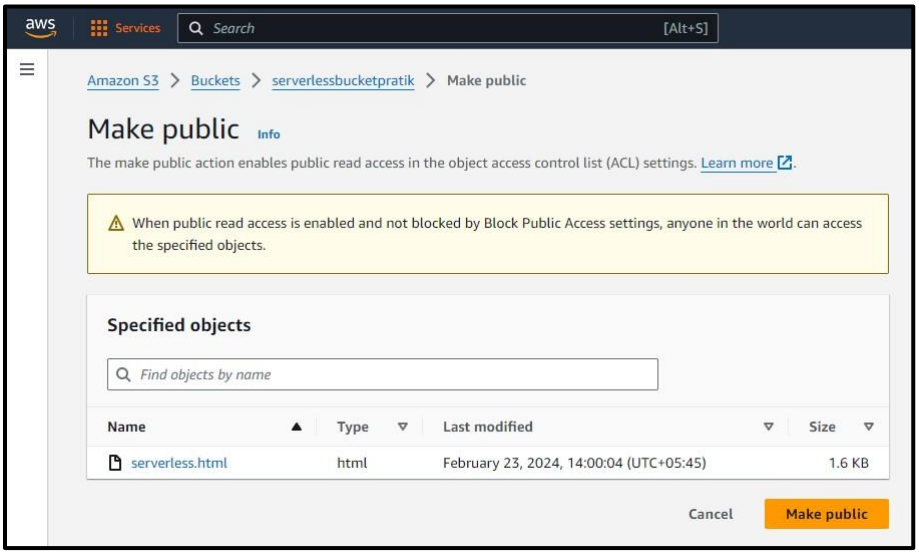


*Figure 29 Public Bucket*

## 30. Made Public Successfully

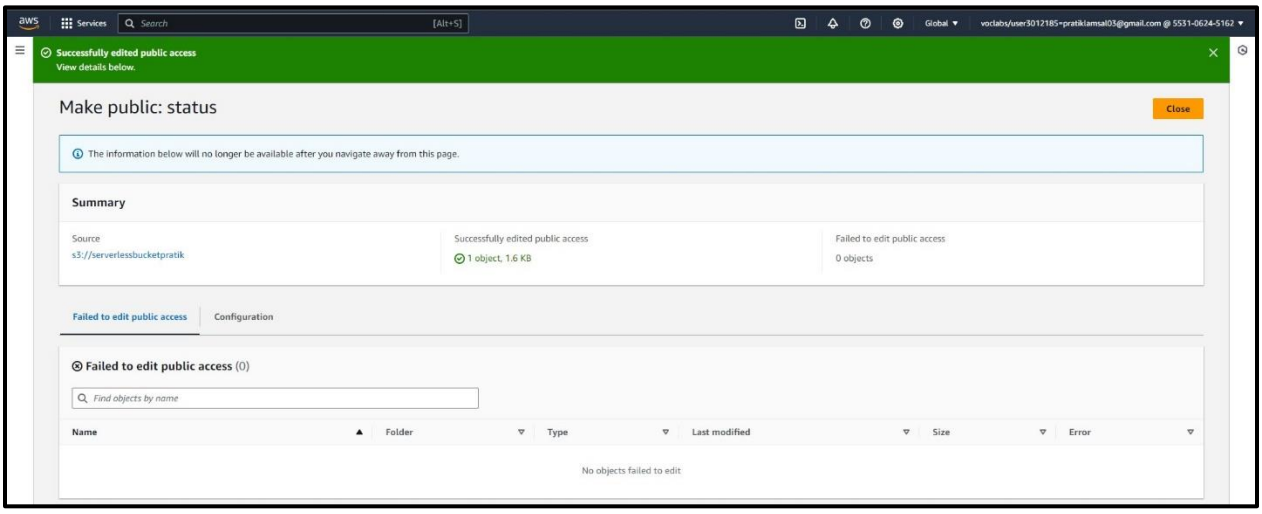


*Figure 30 Successfully Made Public*

## 31. Form Open Successful

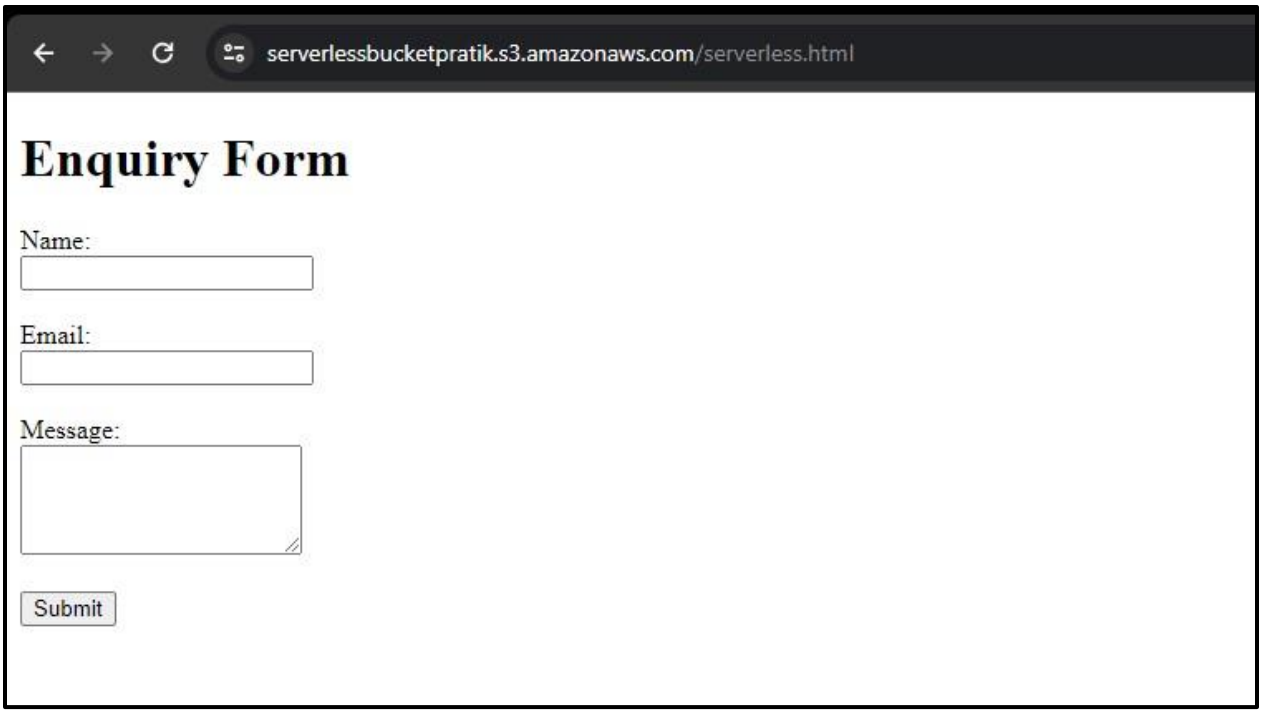


*Figure 31 Form Opens Successful*

## 32. Static Website Hosting

Further error occurred. Changing setting by going into bucket settings and scrolling a little down. Edit static website hosting is there. Click the EDIT button and change settings.

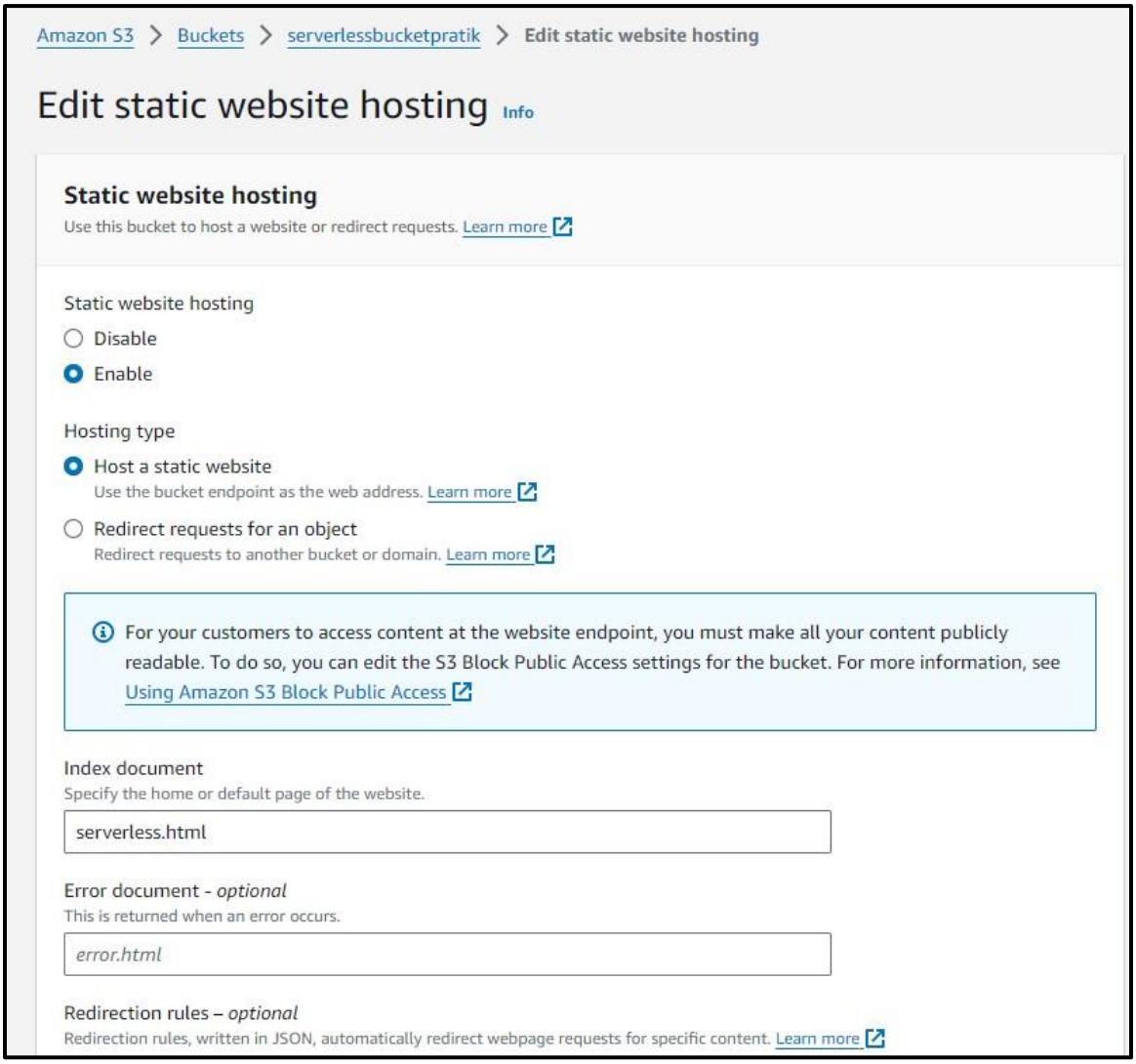


*Figure 32 Edit Static Website Hosting*

## 33. Log Events

Data entered in the page. On checking logs in CloudWatch, no errors are seen and payload is received.

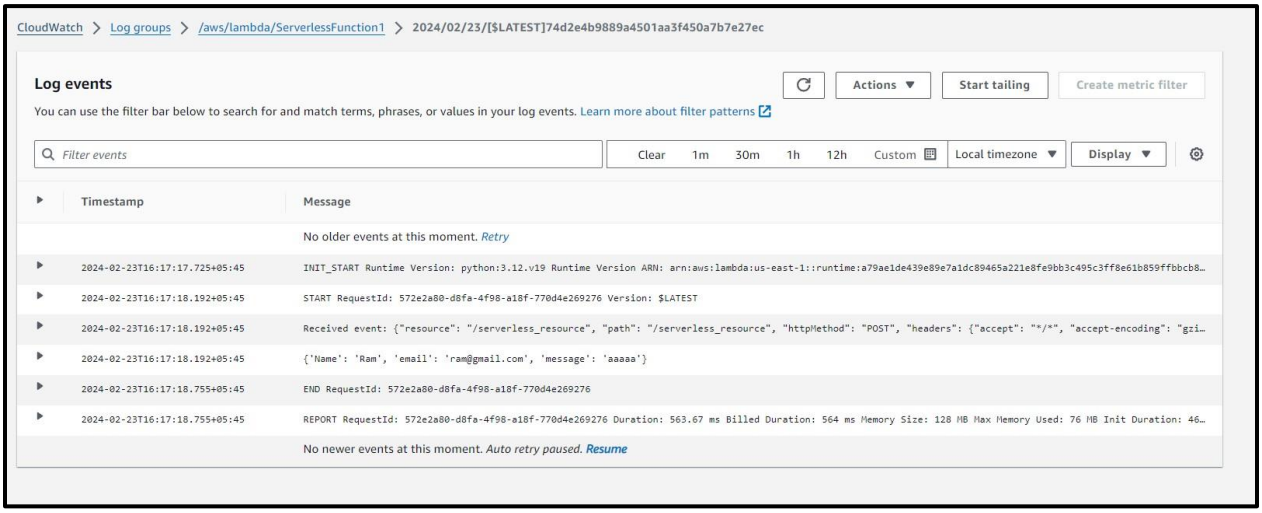


*Figure 33 Log Events*

## 34. Inspecting Network in Browser

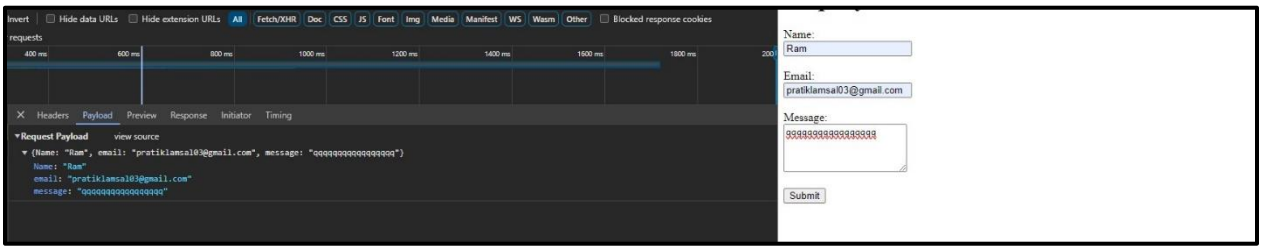


*Figure 34 Payload while inspecting in browser*

## 35. Data in Table

Data is stored in the table successfully.

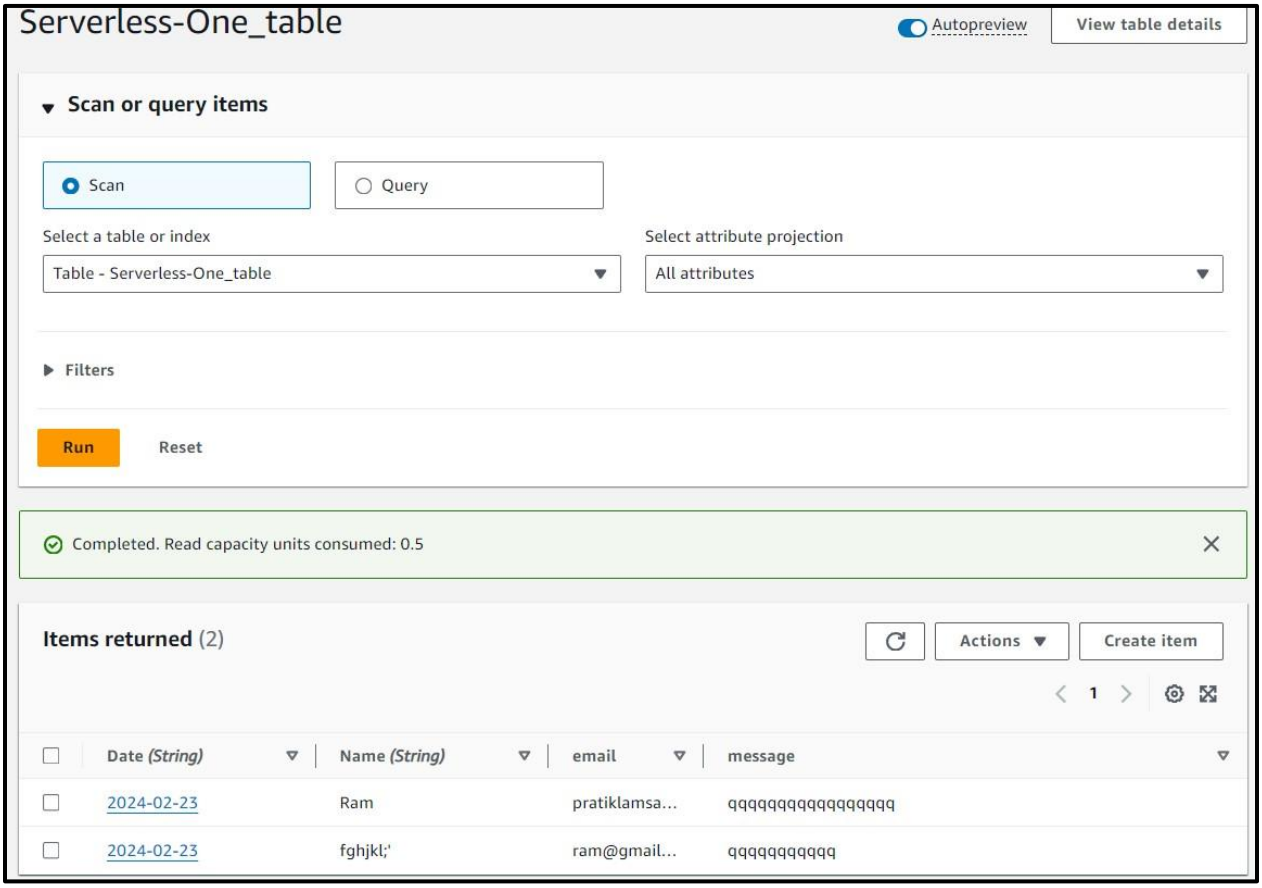


*Figure 35 Data in Table*

## 36. TASK COMPLETED