# **API Gateway Setup and Integration for Event Announcement System**

In this phase of my project, I designed and implemented a REST API using AWS API Gateway to securely expose backend Lambda functions. These endpoints handle user email subscriptions and event creation, enabling seamless interaction between the frontend and AWS services.

## **Step 1: Creating the REST API**

I started by creating a new REST API in the API Gateway console, naming it EventManagementAPI. I selected a Regional endpoint type to optimise latency for users in my region.

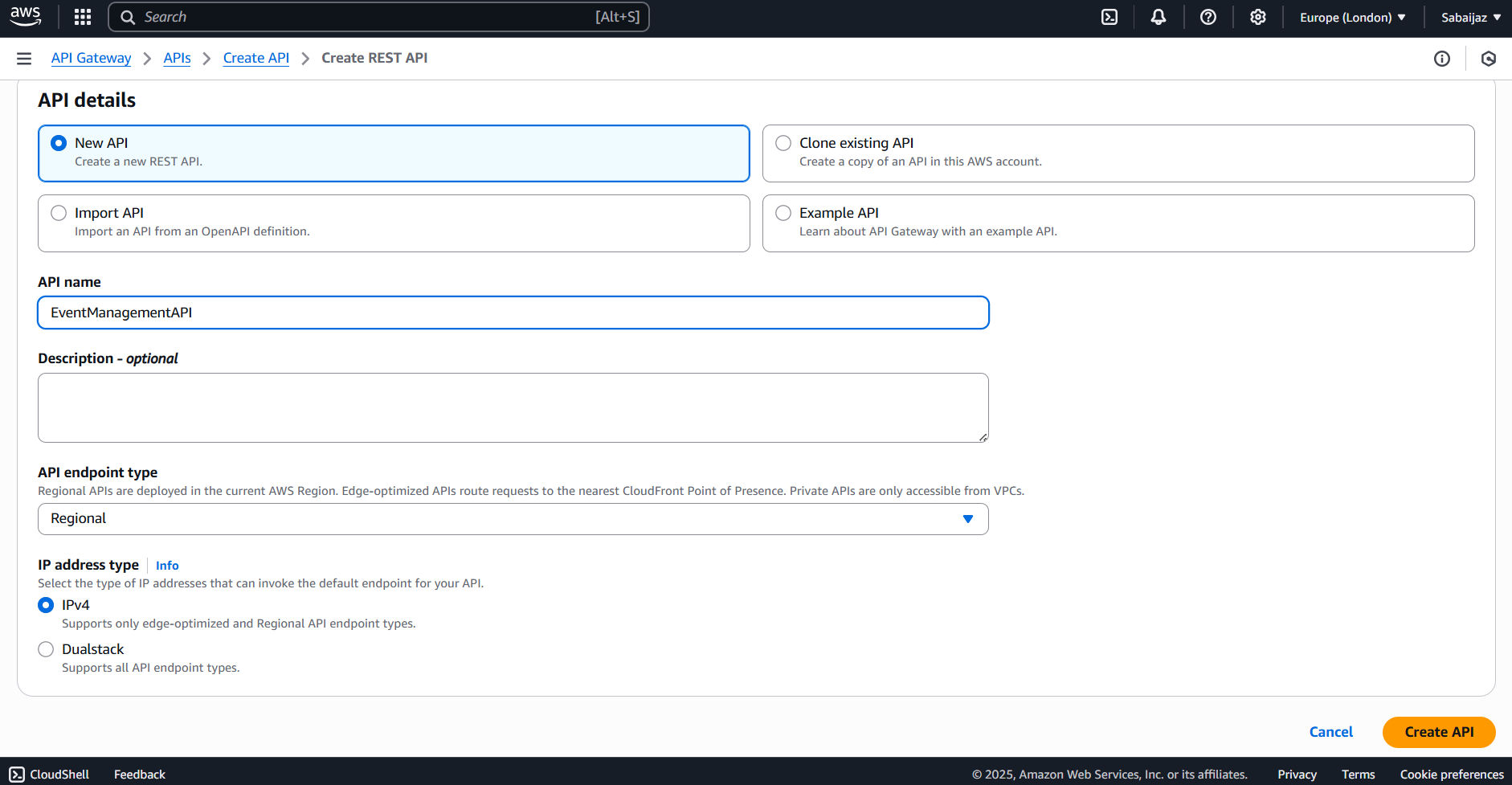


Figure 1: AWS API Gateway Console – Creating New REST API

## **Step 2: Creating and Testing the /subscribe Endpoint**

To allow users to subscribe to event notifications, I created a /subscribe resource connected to the subscribeToSNSFunction Lambda.

* Enabled CORS on the endpoint to support frontend requests from my static website.
* Configured the POST method integration with the Lambda function.
* Added a mapping template to wrap incoming JSON requests inside a body field, matching the Lambda function’s expected input format.
* Tested the endpoint from the API Gateway console, confirming successful Lambda invocation and email subscription flow.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: Creating /subscribe resource setup

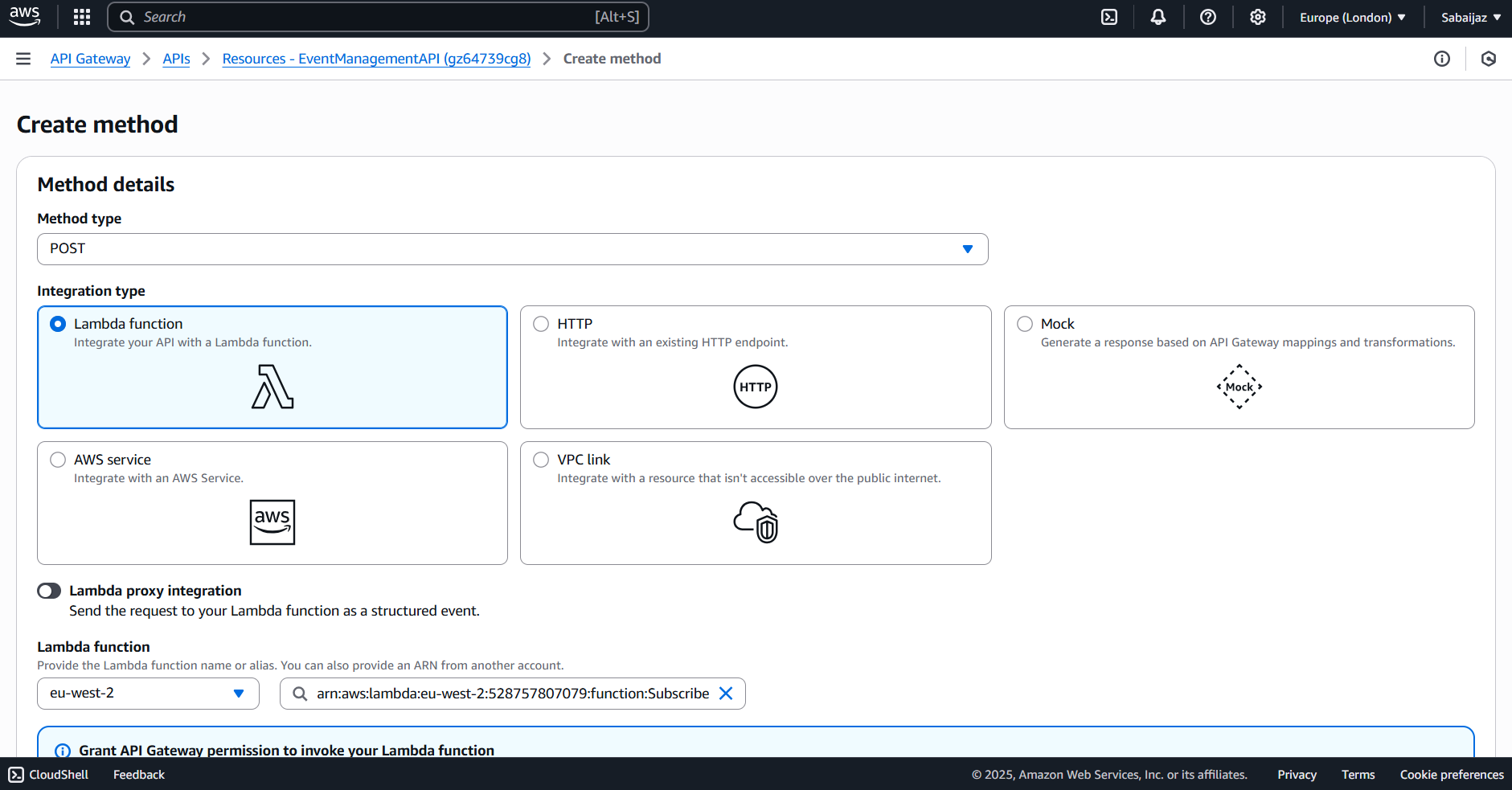


Figure 3: Enabling post method and lambda proxy integration

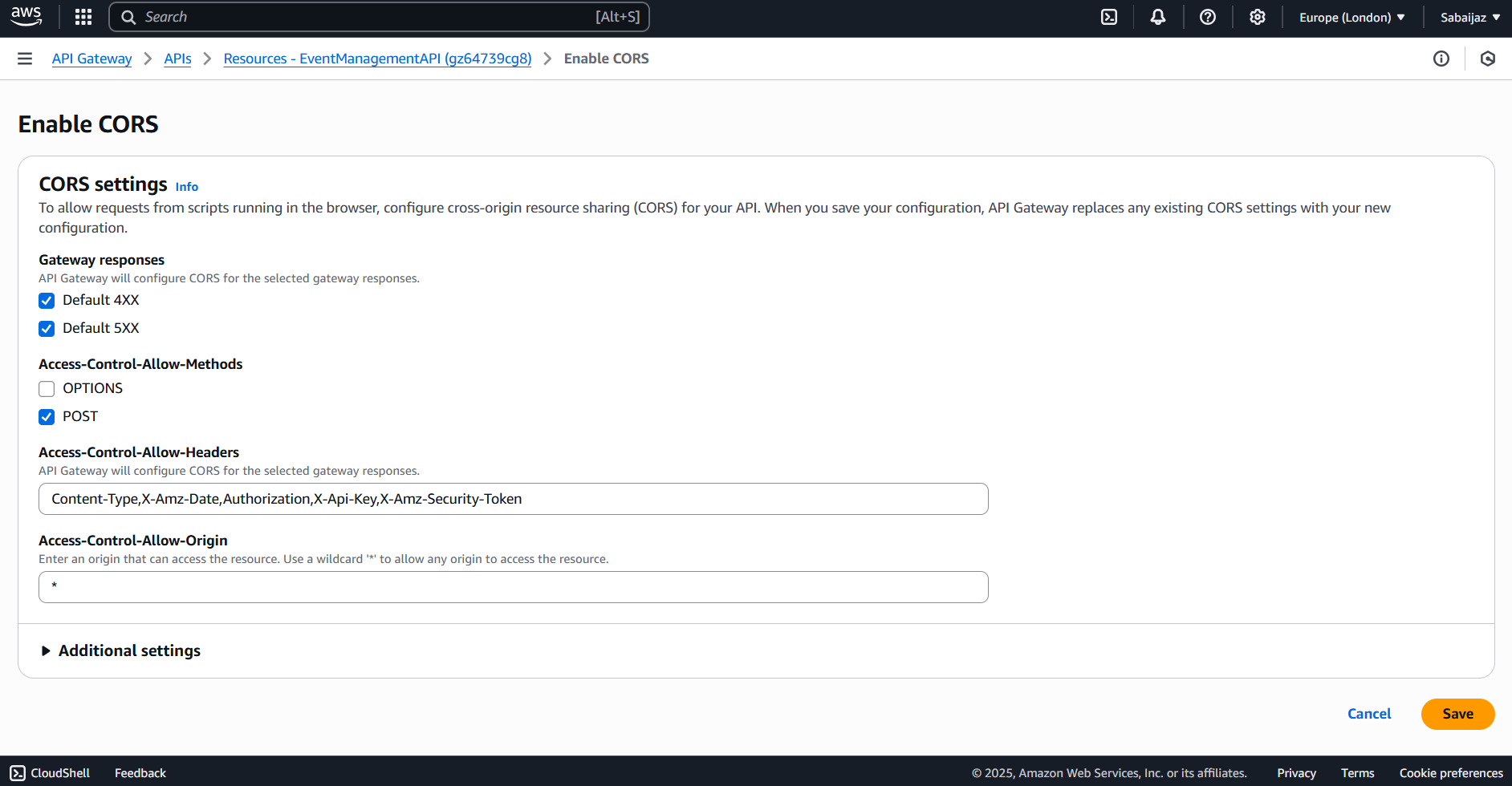


Figure 4: Configuring CORS settings for /subscribe endpoint

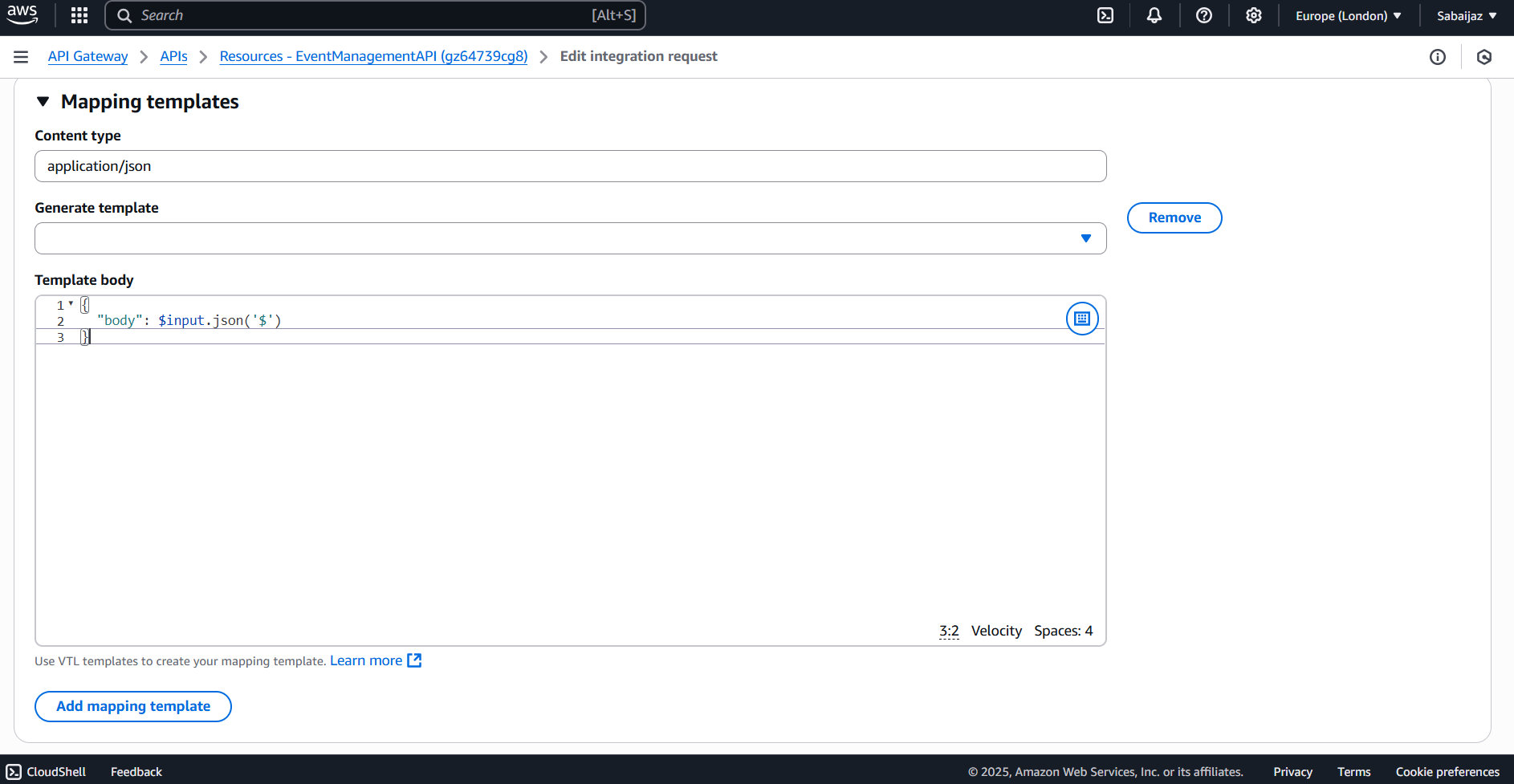


Figure 5: Adding mapping template for JSON requests

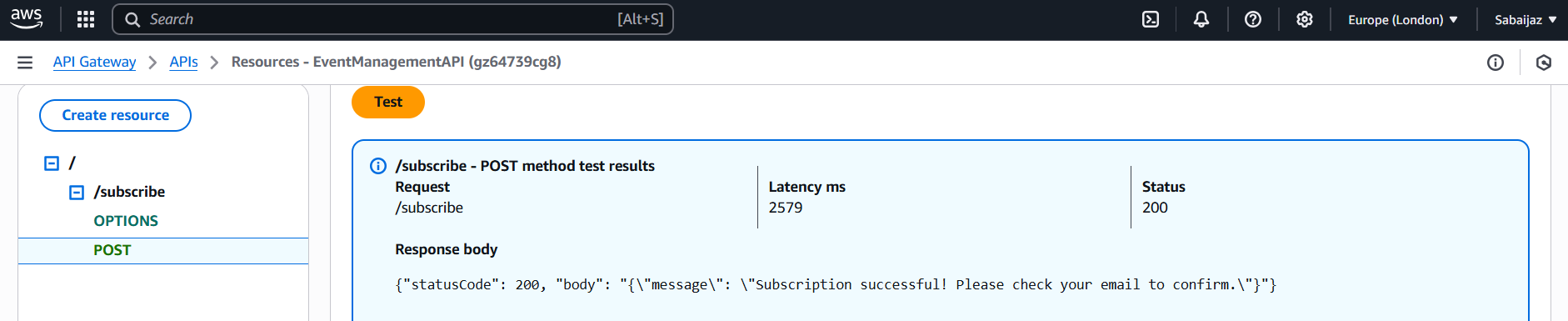


Figure 6: Successful test invocation of /subscribe POST method

## **Step 3: Creating and Testing the /create-event Endpoint**

Next, I set up the /create-event resource and connected it to the createEventFunction Lambda responsible for creating new events.

* Enabled Lambda Proxy integration for passing the entire request directly to the Lambda.
* Enabled CORS for smooth frontend-backend communication.
* Tested the endpoint by sending sample event data via the API Gateway console.
* Verified that the new event was appended to the events.json file in S3 and that all SNS subscribers received notification emails.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: Creating /create-event resource setup

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: Enabling post method Lambda Proxy integration

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Enabling CORS for /create-event

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: Testing /create-event endpoint with sample payload

A close-up of a computer screen

AI-generated content may be incorrect.

Figure 11: Confirmation email received by SNS subscribers

## **Step 4: Deploying the API**

To make the API publicly accessible, I deployed it to a new stage called dev.

* Noted the base Invoke URL provided by API Gateway.
* Confirmed the full endpoint URLs for both /subscribe and /create-event, which are used in my frontend JavaScript to connect to backend Lambda functions.

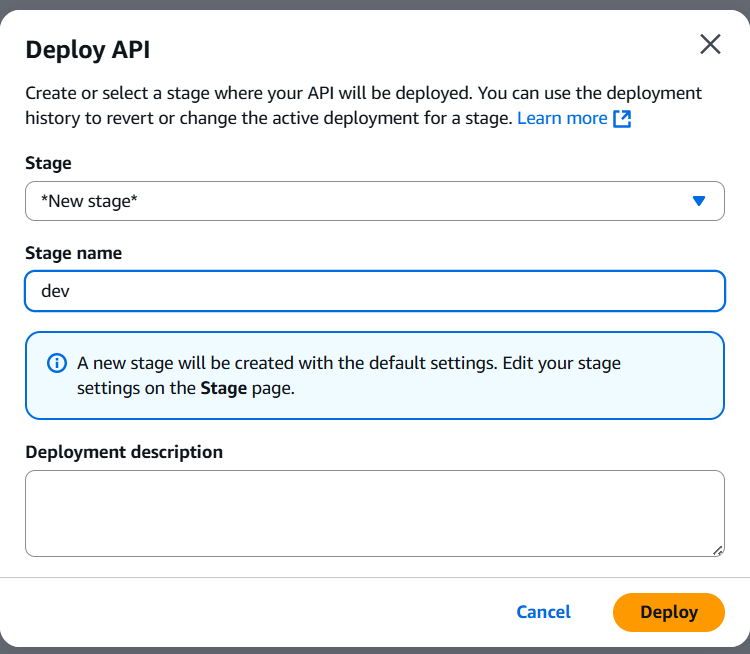


Figure 12: Deploy API dialogue and stage creation

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: API Invoke URL displayed in the console

This setup ensured secure, scalable, and maintainable communication between my frontend and backend services, enabling a fully serverless Event Announcement System with dynamic event updates and real-time notifications.