DATA 228: Big Data Technologies and Applications

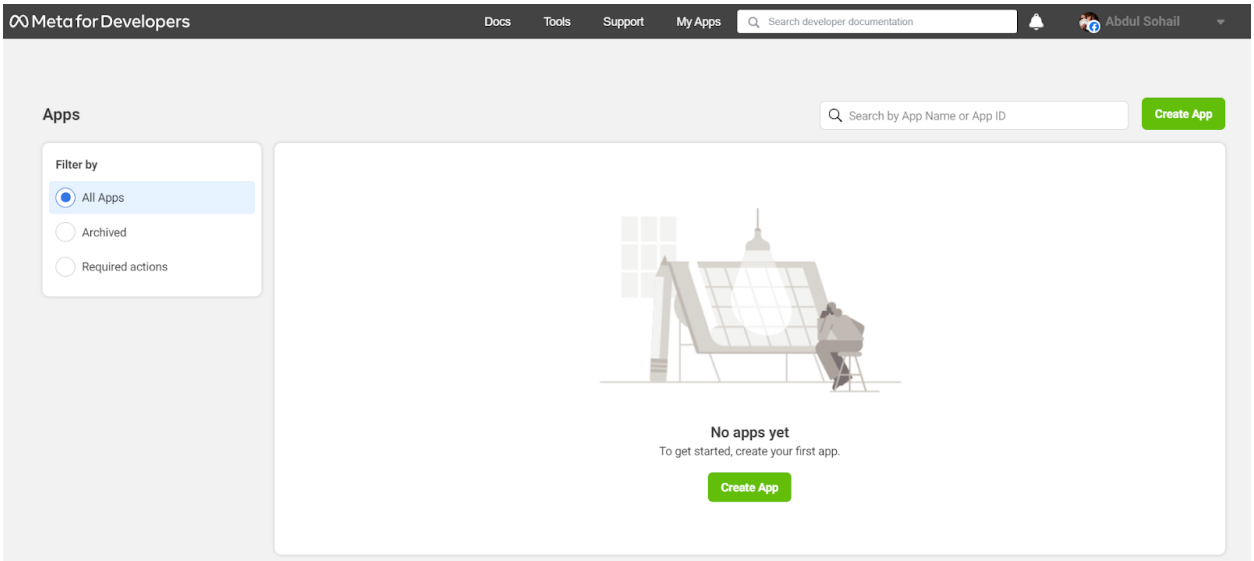
Homework 4: Multi-Region Application

Group Members (Jedi's):

Abdul Sohail Ahmed (016769610), Poojan Gagrani (016795285), Somna Sattoor (014640769), Swetha Neha Kutty Sivakumar (016780712), Kashish Thakur (016919383)

Prerequisites:

Registering as a Meta Developer:



Step 1: Creating AWS Cloud9 Environment:

1. **AWS Cloud 9 Dashboard:**

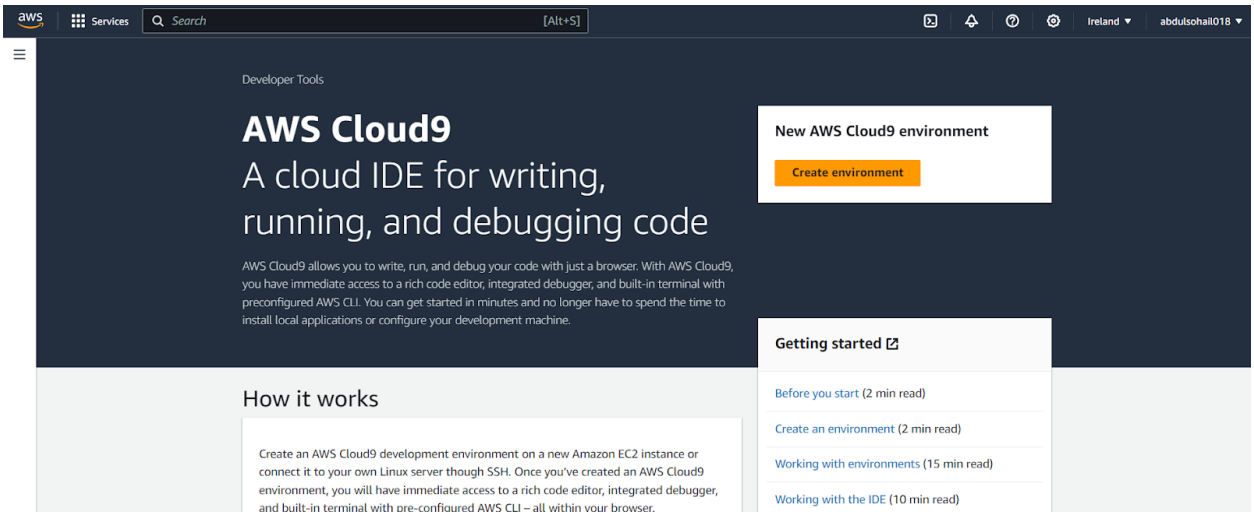


Figure 1: Opening AWS Cloud 9 Dashboard

2. Naming AWS Cloud 9 environment:

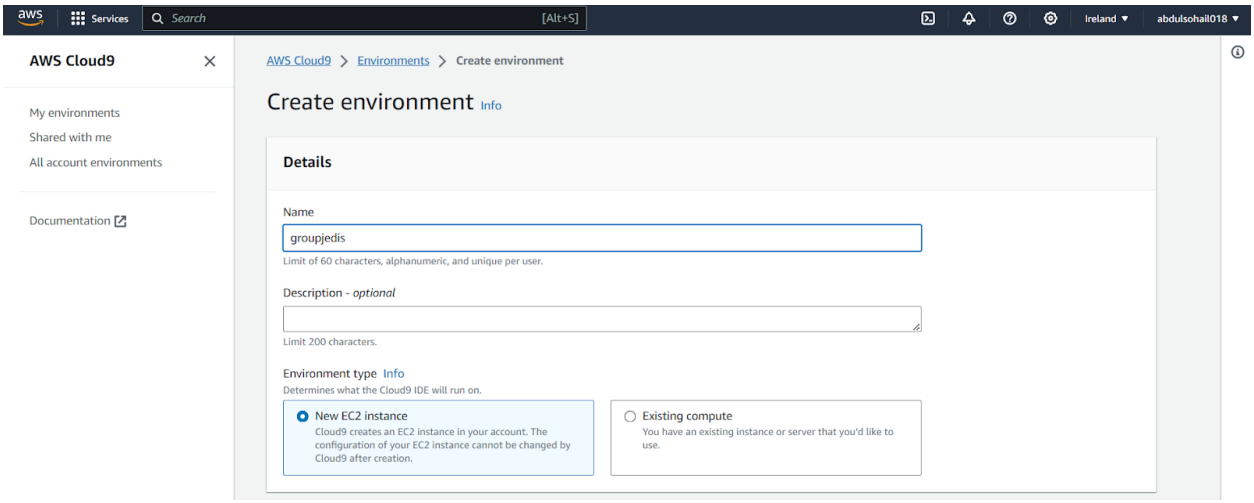


Figure 2: Providing the name of the Cloud9 environment

3. Configuring the settings:

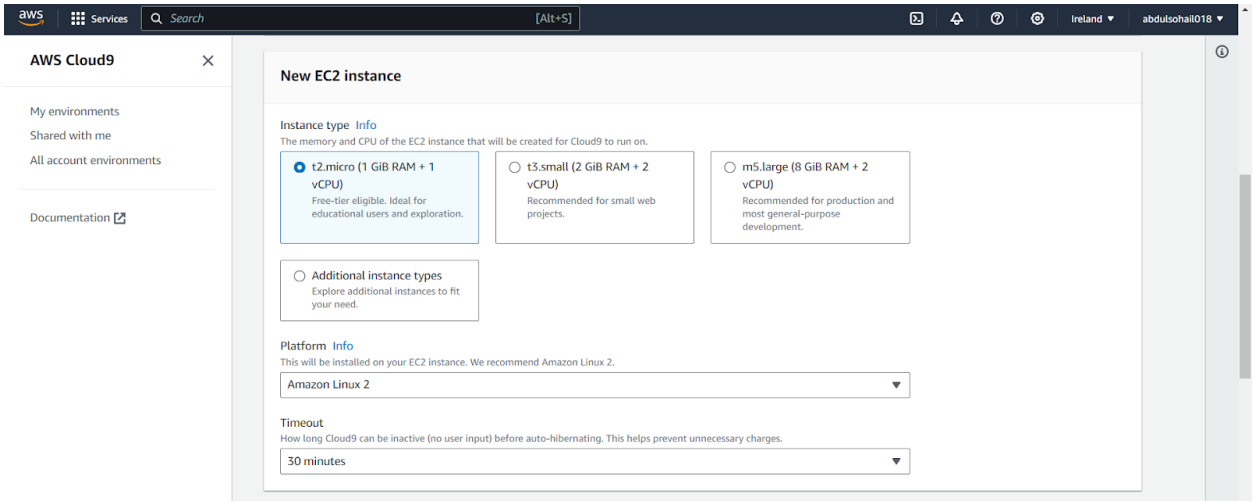


Figure 3: Selecting EC2 instance, platform installed, and timeout

4. Created Cloud9 environment:

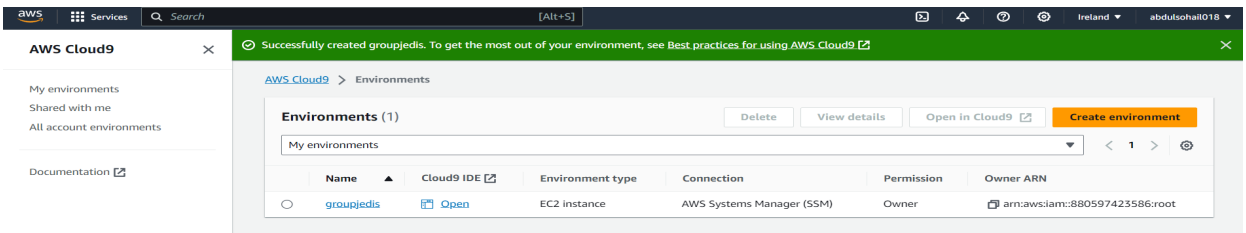


Figure 4: Successfully able to create the Cloud9 environment

5. Installed packages:

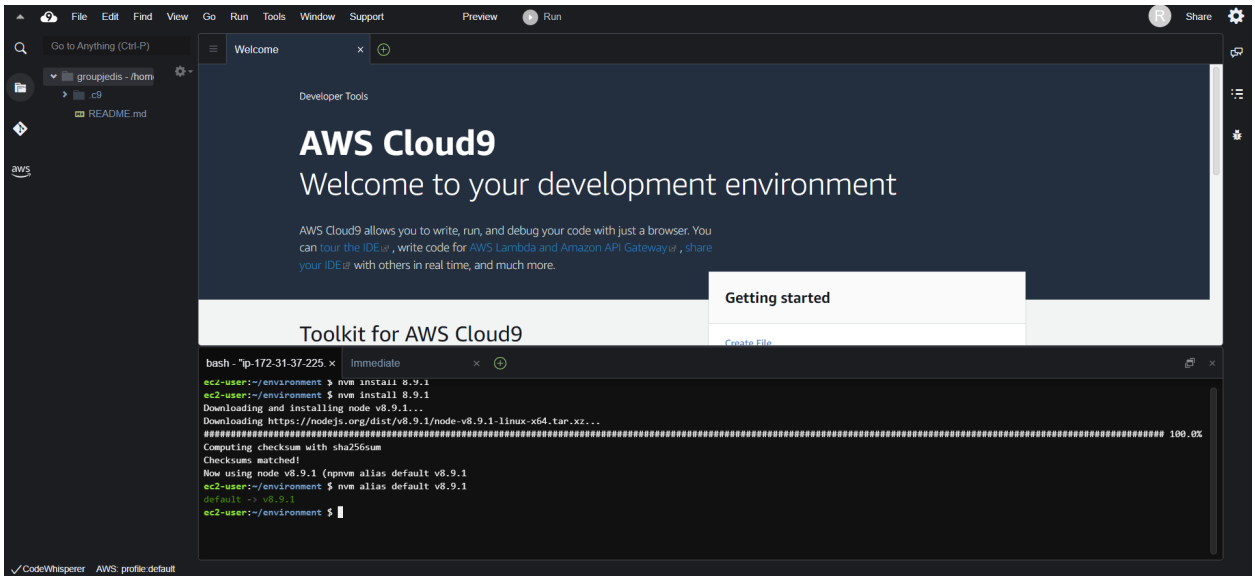


Figure 5: Successfully able to install the necessary packages for AngularJS front end

6. Cloning git repository:

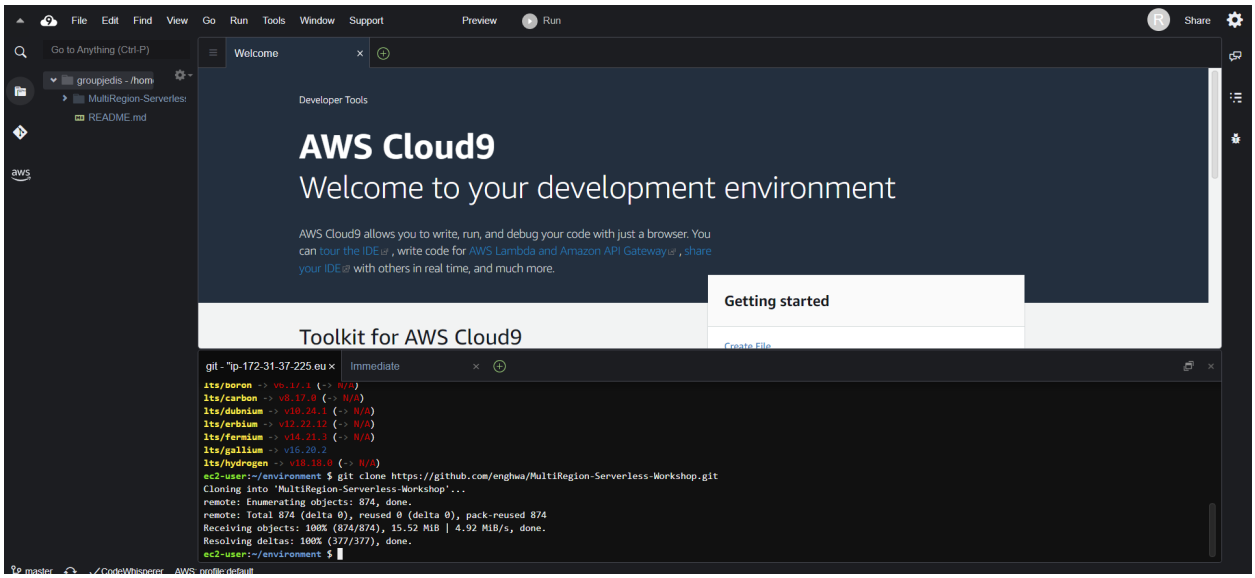


Figure 6: Cloning the MultRegion Serverless Workshop git repository

Step 2: <u>Build an API layer:</u>

1. Creating IAM Policy:

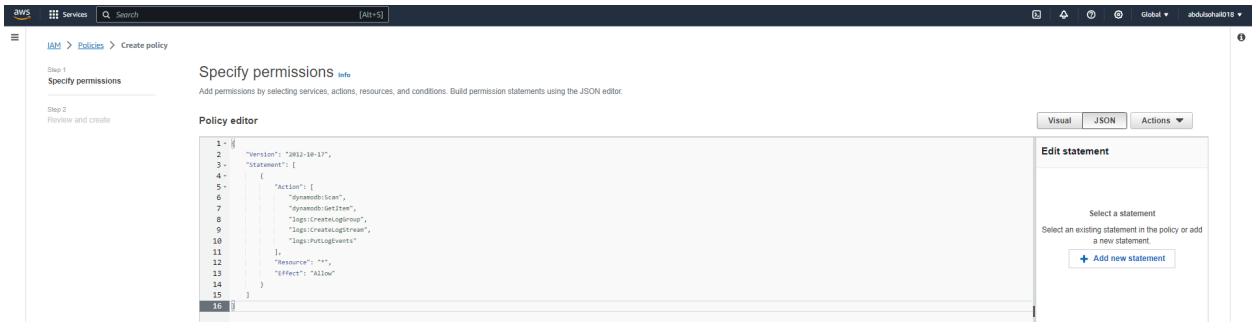


Figure 7: Creating a custom policy in the IAM service

2. Created custom policy named TicketGetPolicy:

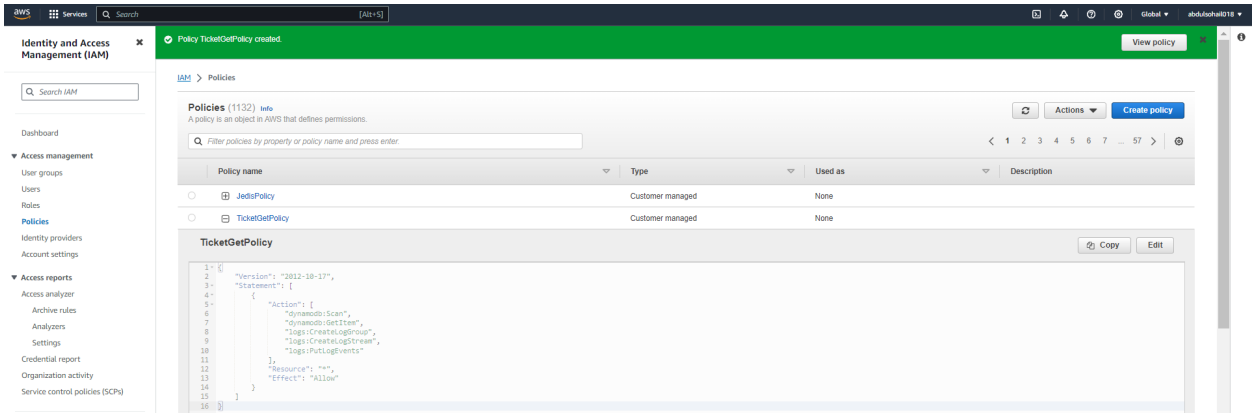


Figure 8: Successfully created custom policy named "TicketGetPolicy"

2.1 Created custom policy named TicketPostPolicy:

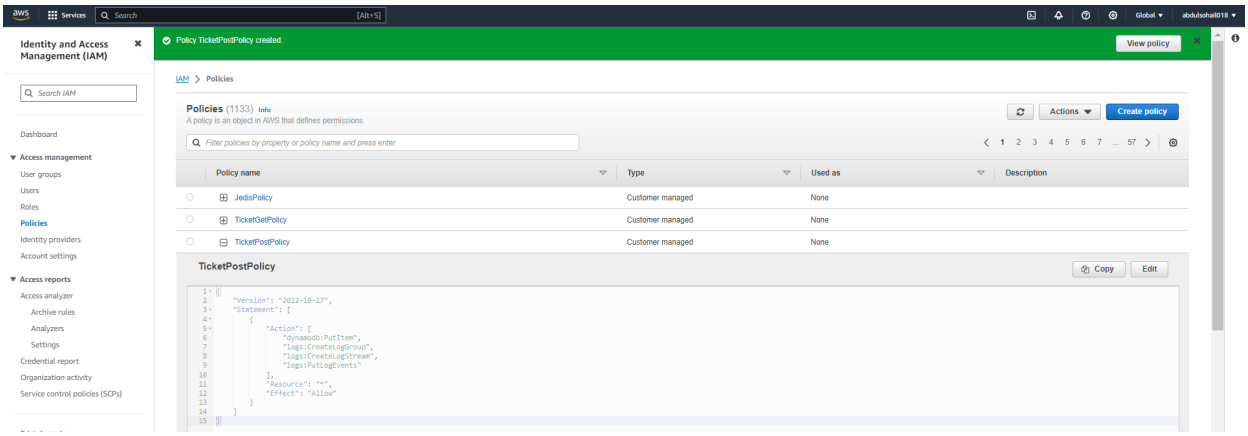


Figure 9: Successfully created custom policy named "TicketPostPolicy"

3. Creating roles for the custom policies:

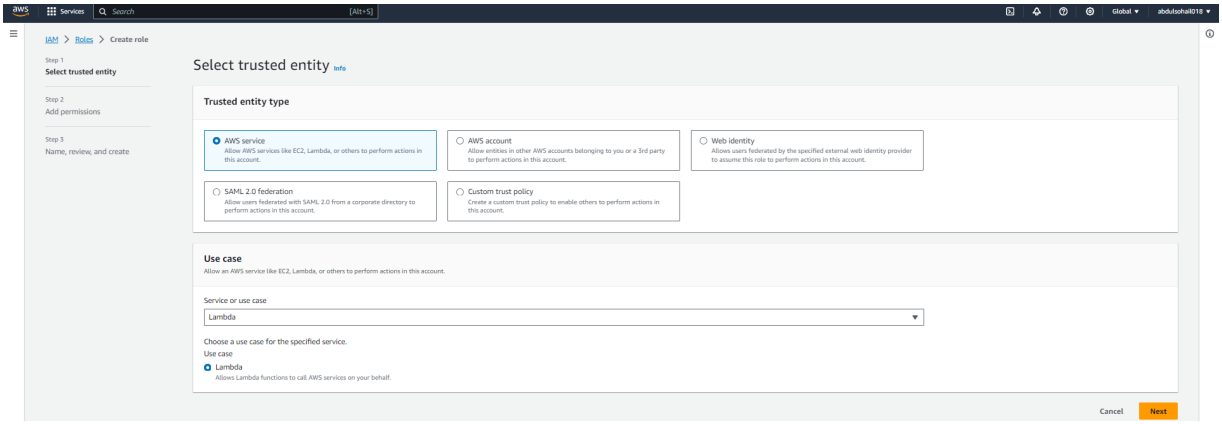


Figure 10: Create an IAM role for the policy "TicketGetPolicy" with the use case "Lambda"

3.1 Creating a custom IAM role with Selection of Trusted Entity and Use Case

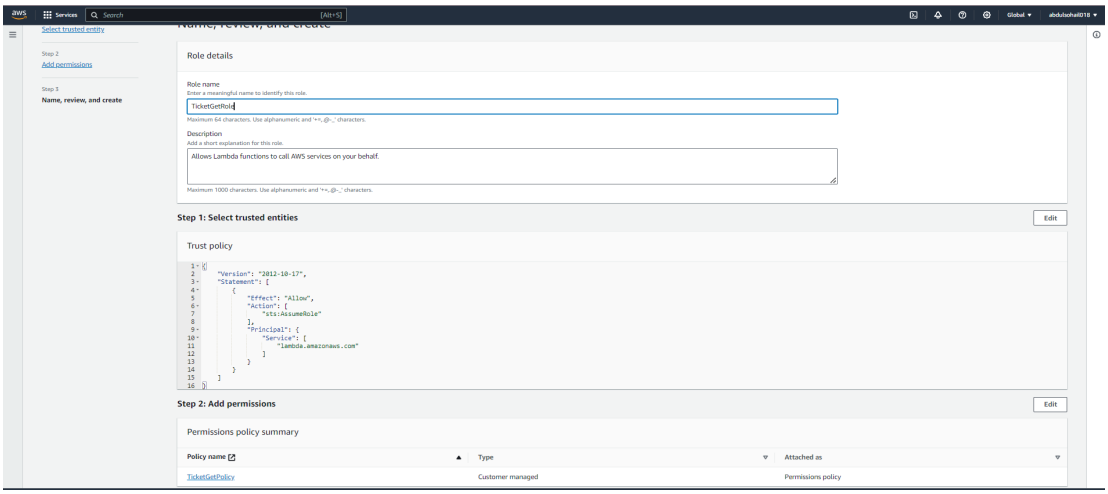


Figure 11: Creating a role named "TicketGetRole"

3.2 Selection of Trusted Entity and Use Case

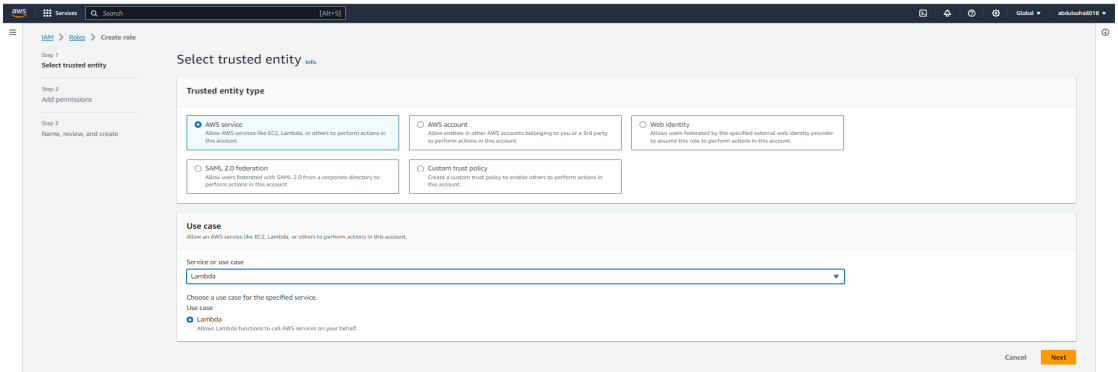


Figure 12: Selecting the trusted entity as AWS Service and use case as Lambda

3.3 Creating a custom IAM role with Selection of Trusted Entity and Use Case

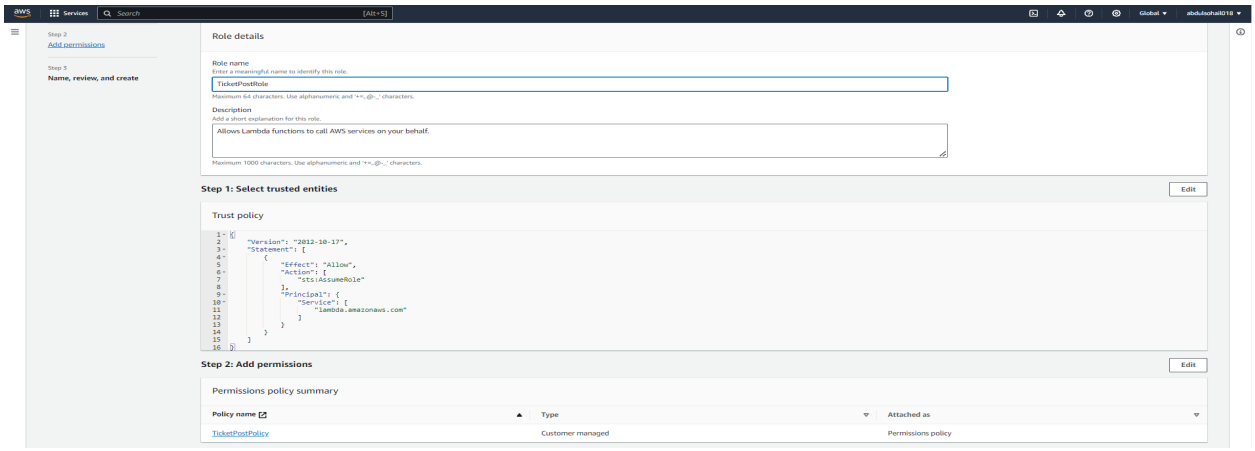


Figure 13: Creating a role named "TicketPostRole"

4. Created IAM roles for the custom policies:

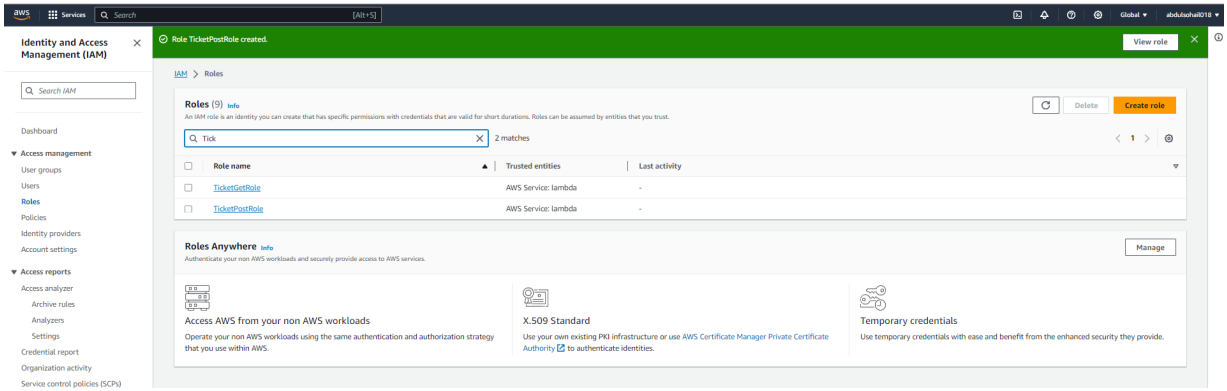


Figure 14: Successfully created both the roles for the custom IAM policies defined before

5. Creating DynamoDB table:

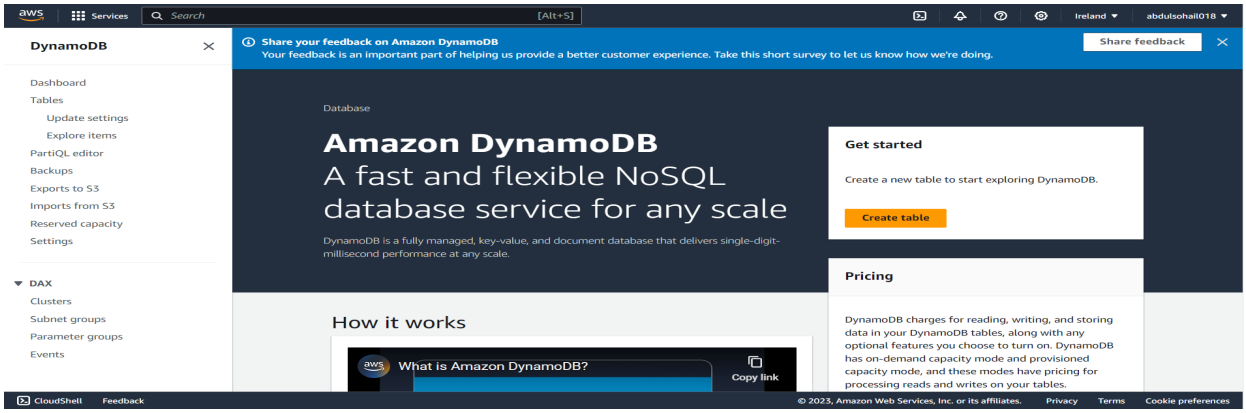


Figure 15: DynamoDB dashboard

5.1 Creation of Table

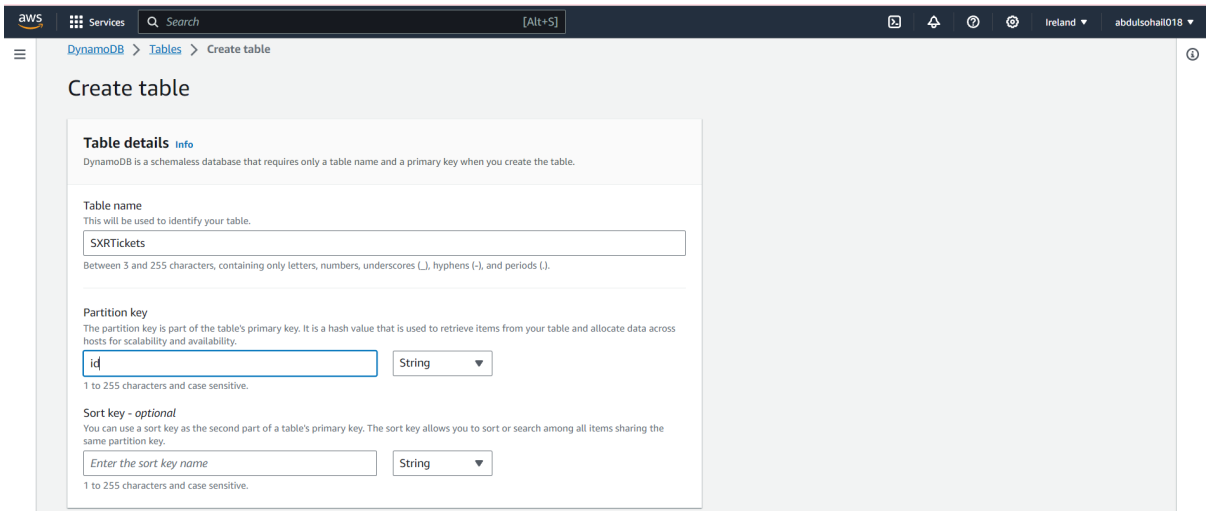


Figure 16: Creating a DynamoDB table named "SXRTickets" and selecting "id" as the primary partition key

5.2 Configuring the Table Settings

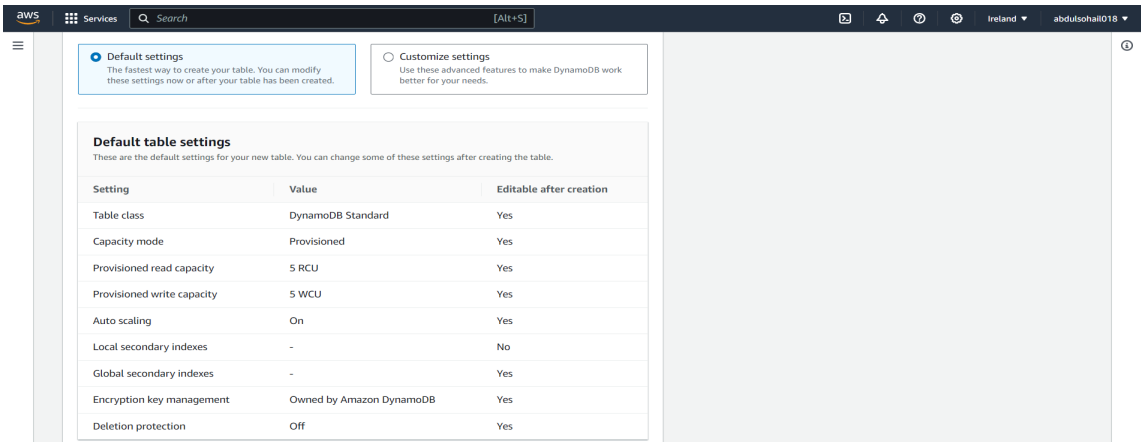


Figure 17: Keeping the table settings as "Default"

6. Created DynamoDB table:

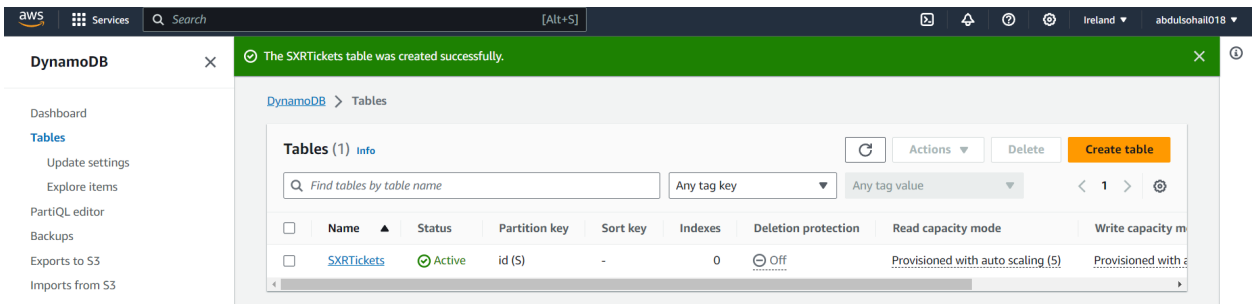


Figure 18: Successfully created "SXRTickets" DynamoDB table

7. Creating a replica of the DynamoDB table:

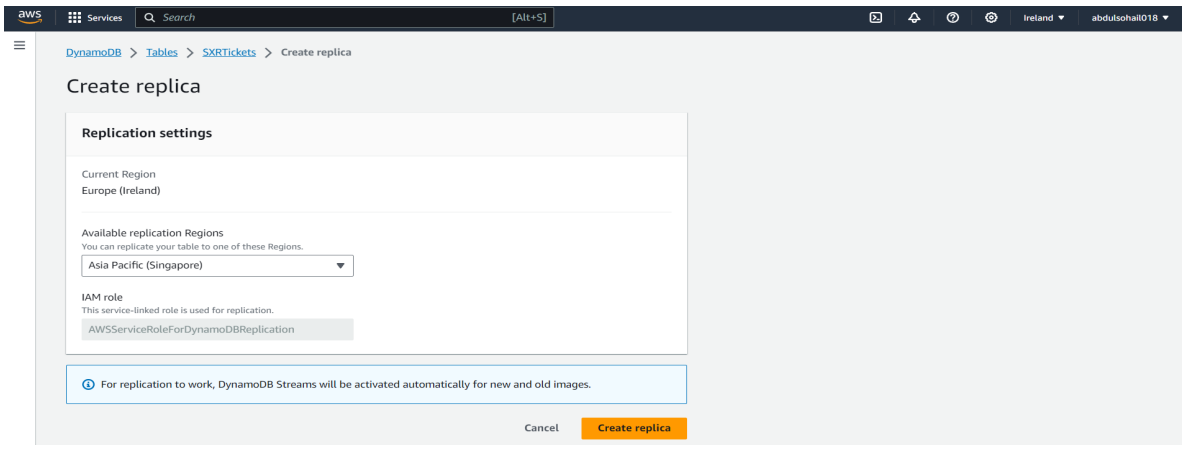


Figure 19: Creating a replica of the original table created in "Europe(Ireland)" in "Asia Pacific (Singapore)"

8. Created a replica of the DynamoDB table:

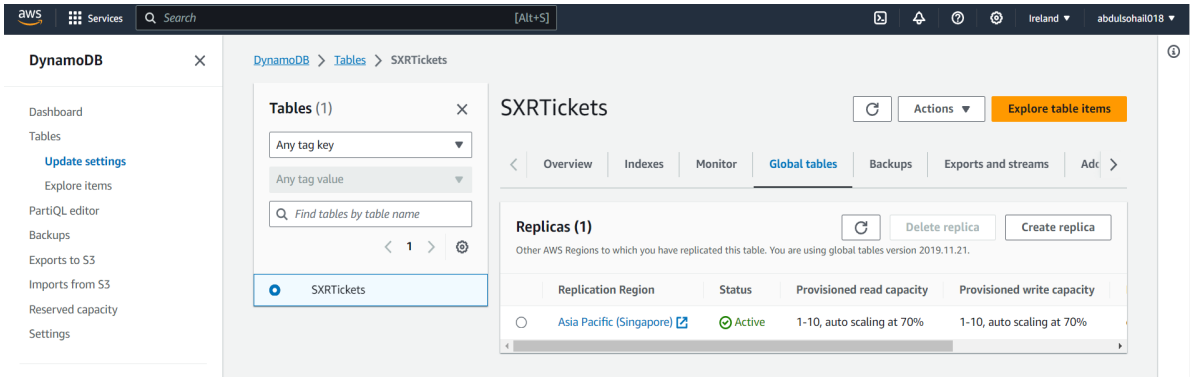


Figure 20: Successfully created the replica of the DynamoDB table in "Asia Pacific (Singapore)"

9. Creating lambda functions:

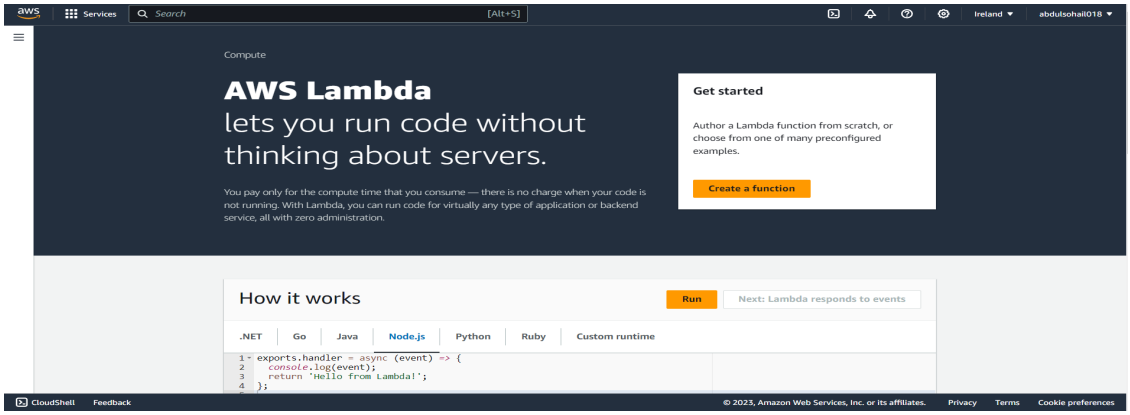


Figure 21: AWS lambda function

9.1 Creation of custom Lambda Function

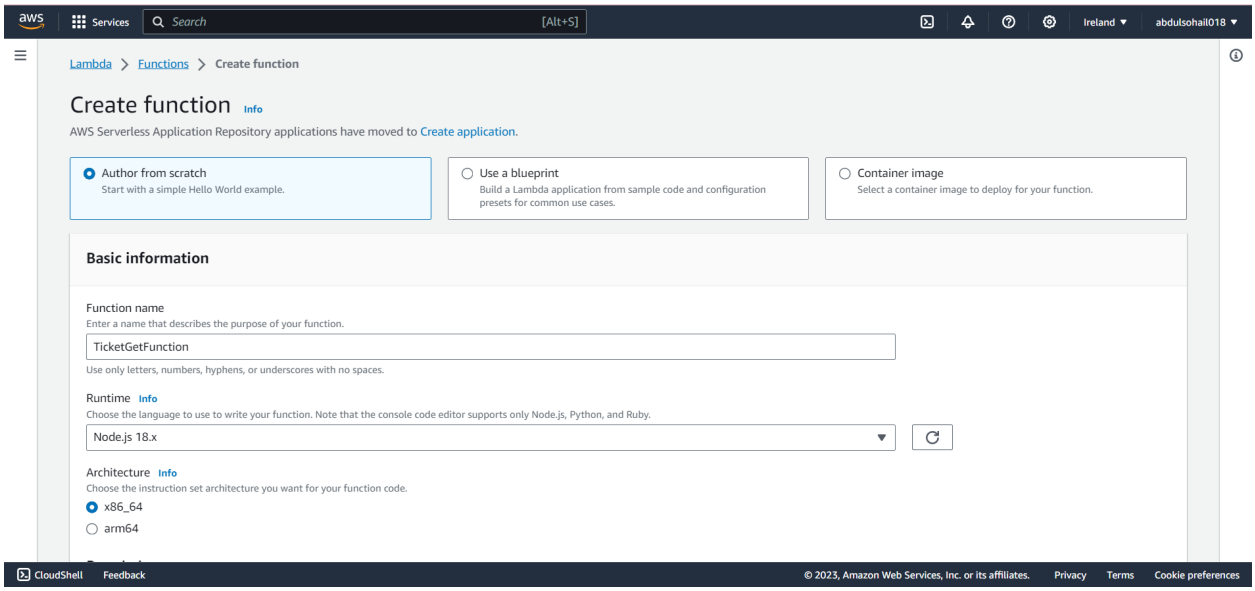


Figure 22: Providing function name "TicketGetFunction" and selecting "Node.js 18.x" to write functions

9.2 Configuring the Function Settings and Completion of Creation

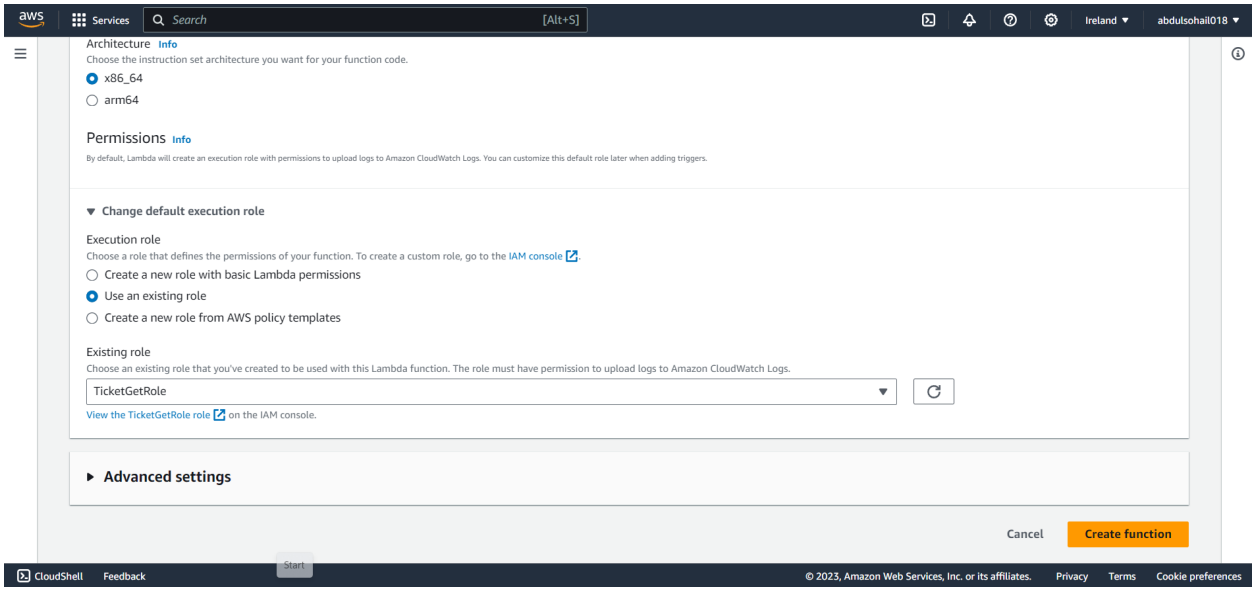


Figure 23: Using the existing role "TicketGetRole" created before

10. Modify lambda function:

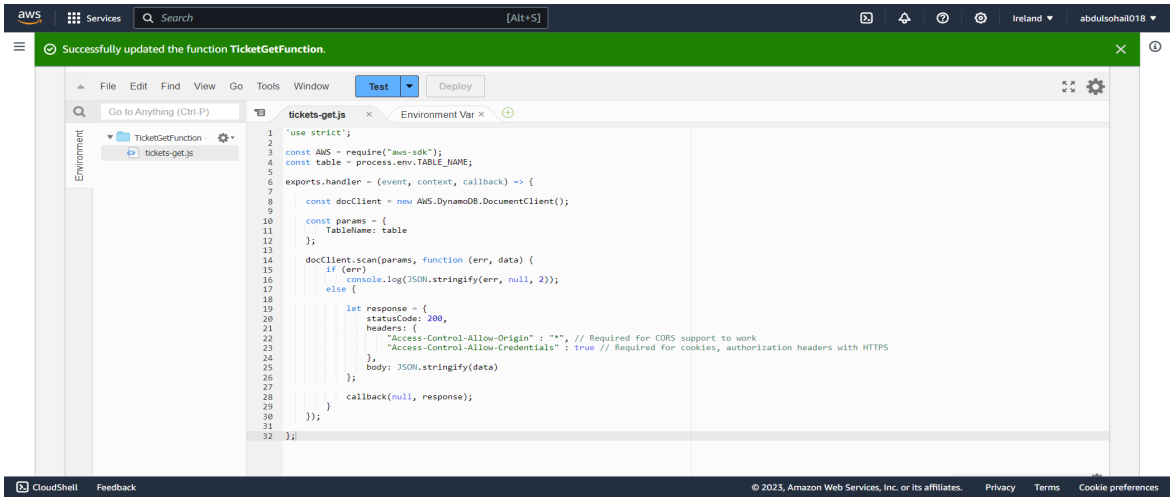


Figure 24: Modifying the "TicketGetFunction" function

10.1 Configuring the Runtime Setting

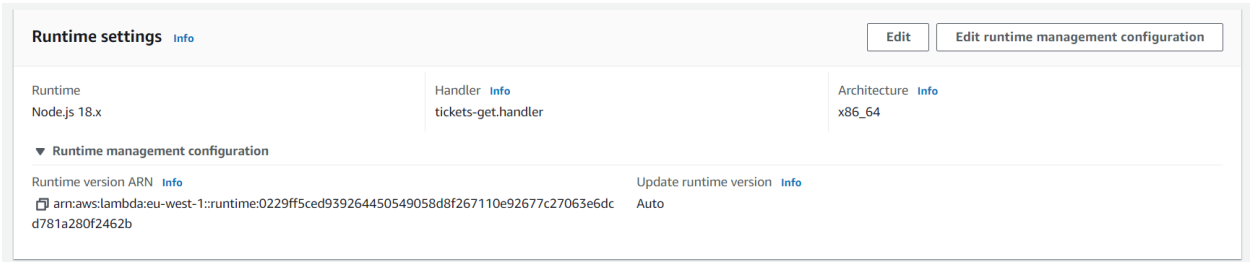


Figure 25: Changed handler to "ticket-get.handler"

10.2 Editing the Environment Variables

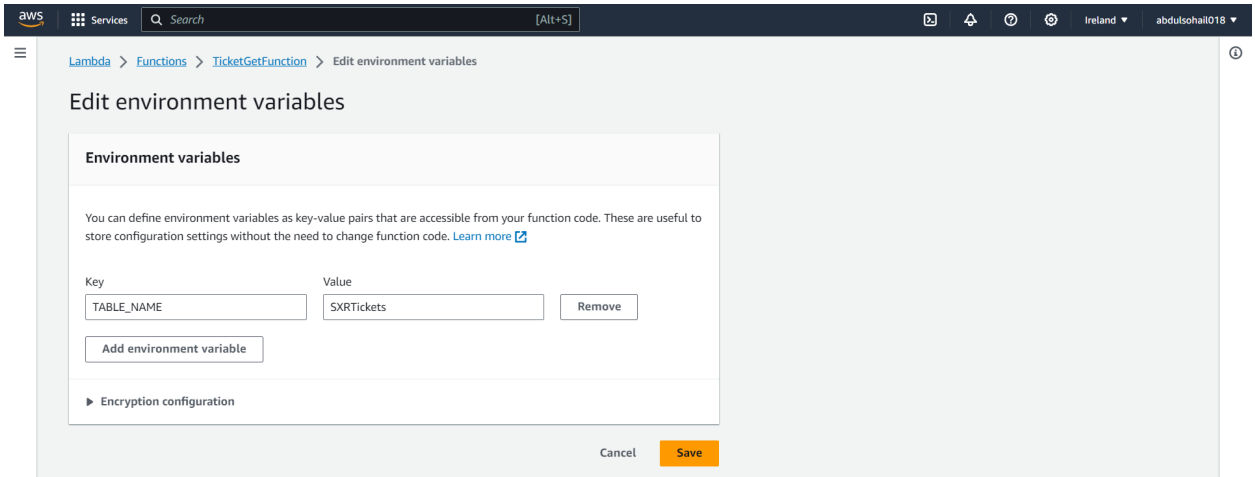


Figure 26: Adding environment variables with key as "TABLE_NAME" and value as "SXRTickets"

10.3 Updation of the Environment Variables

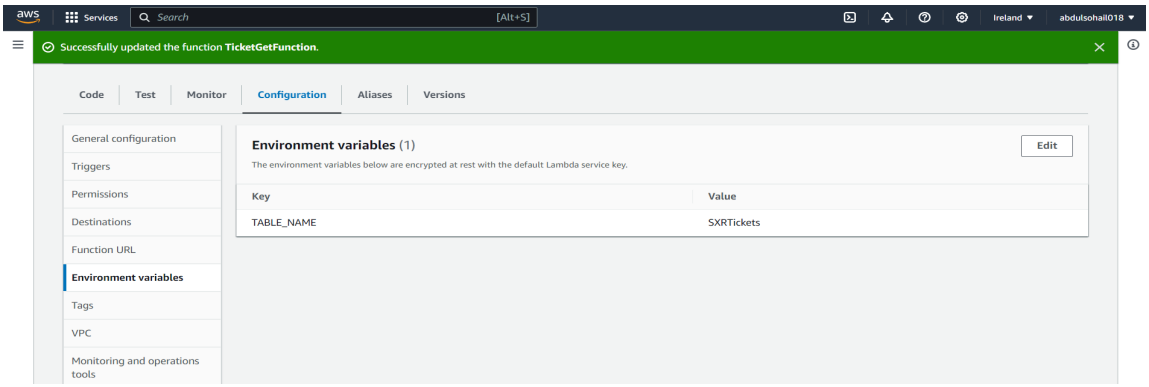


Figure 27: Successfully added environment variables

11. Created lambda functions:

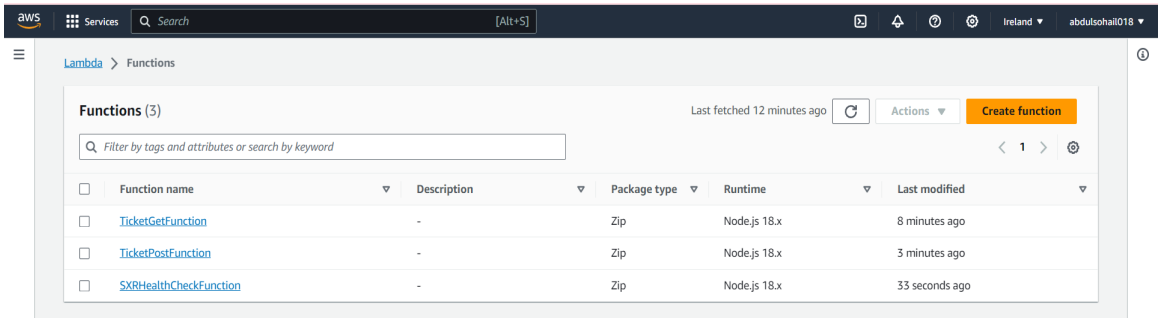


Figure 28: Successfully created all three lambda functions

12. Creating API Gateway Endpoint:

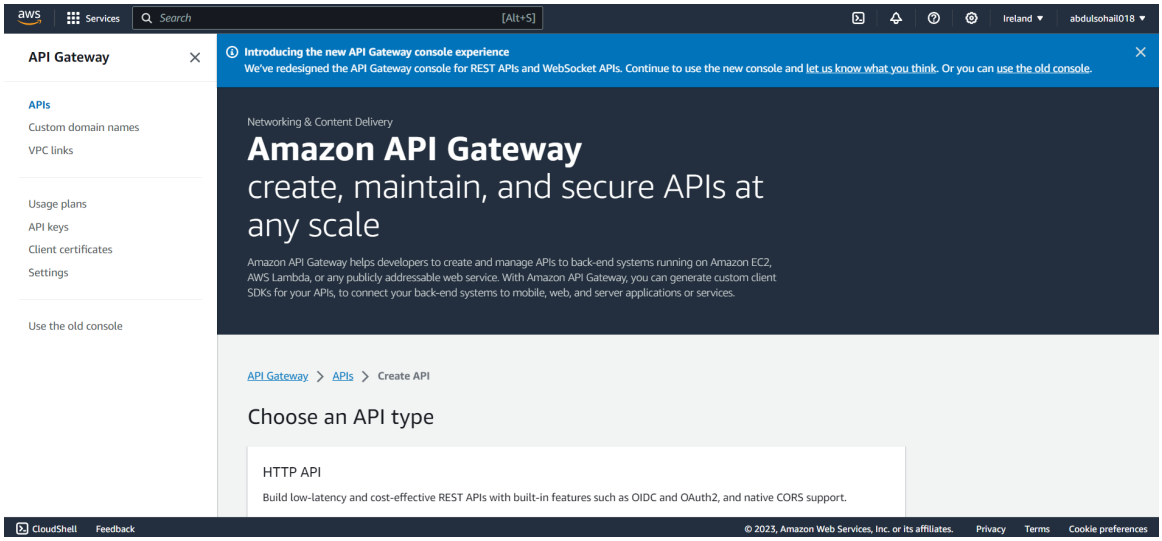


Figure 29: API Gateway Endpoint dashboard

12.1 Creation of the new API

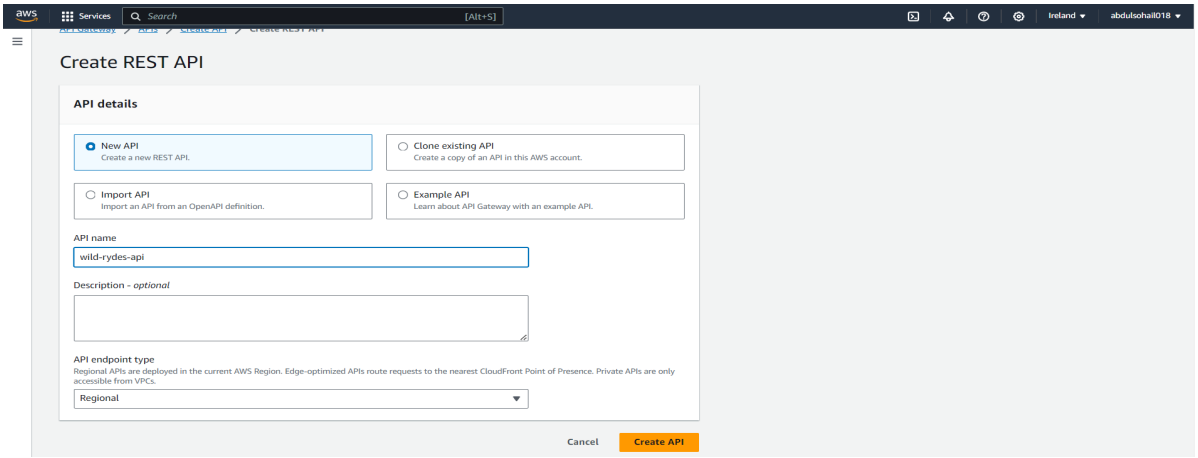


Figure 30: Providing the name of the API as "wild-rydes-api" & selecting the endpoint type as "Regional"

13. Created API Gateway Endpoint:

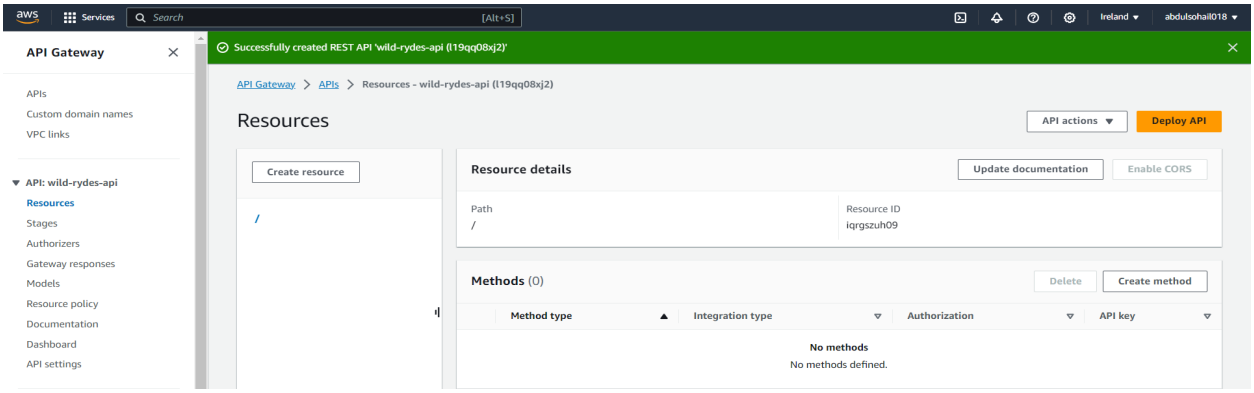


Figure 31: Successfully able to create the API Gateway Endpoint

14. Creating resource:

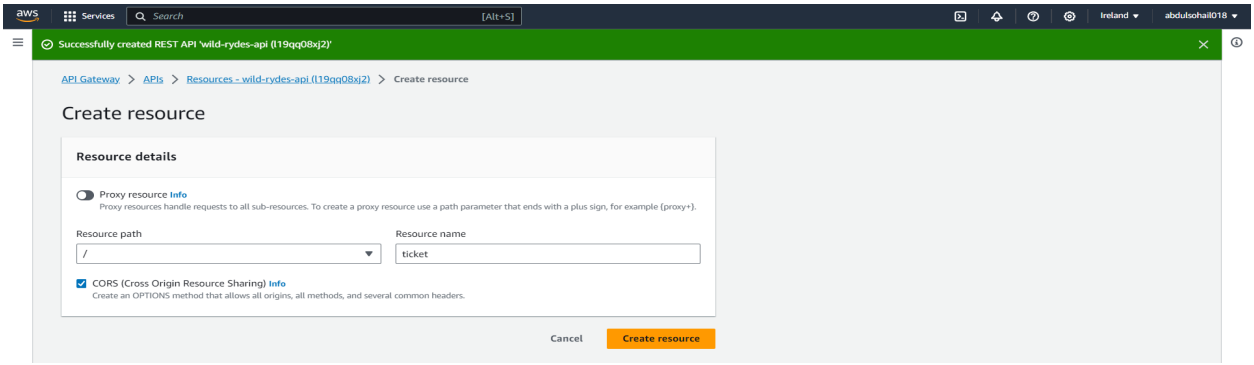


Figure 32: Providing resource name as "ticket" and enabling API Gateways CORS

14.1 Providing the Resource Details

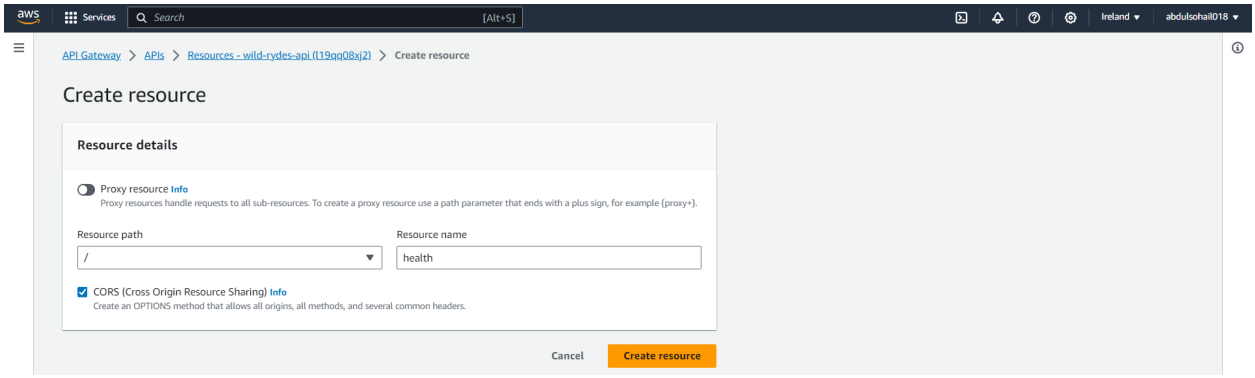


Figure 33: Providing resource name as "health" and enabling API Gateways CORS

15. Created resources:

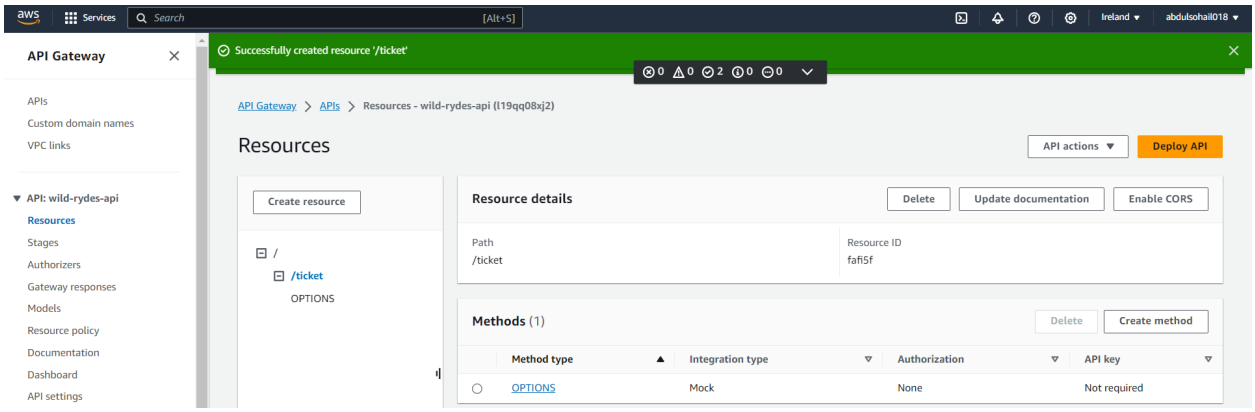


Figure 34: Successfully created the resource named "ticket"

15.1 Reviewing the Created Resource

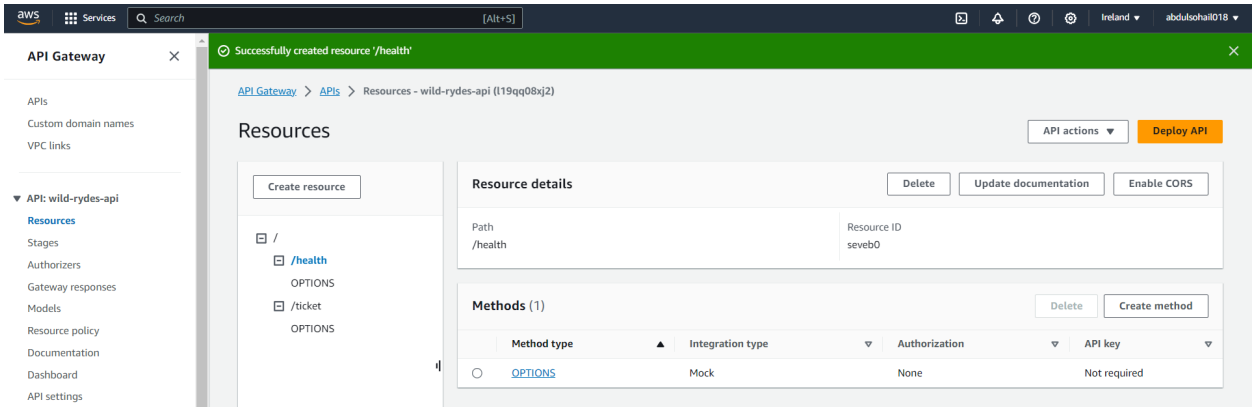


Figure 35: Successfully created the resource named "health"

16. Creating method type “GET” in ticket resource:

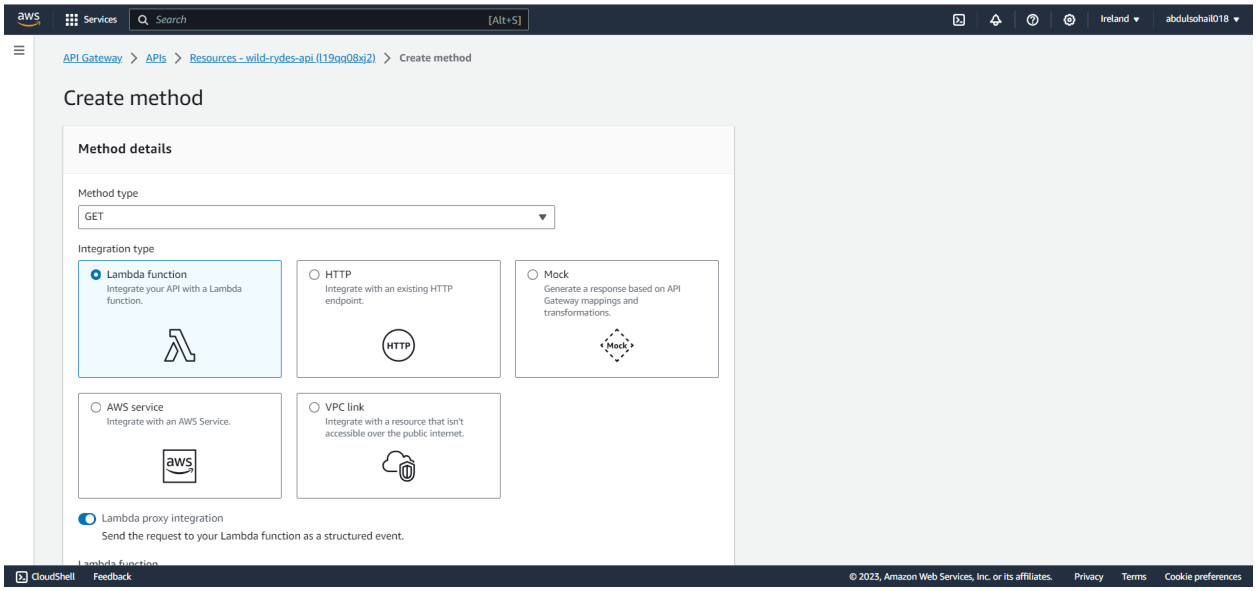


Figure 36: Creating a method in the “ticket” resource by selecting the method type as “GET”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “TicketGetFunction” from the Lambda function

16.1 Creating method type “POST” in ticket resource:

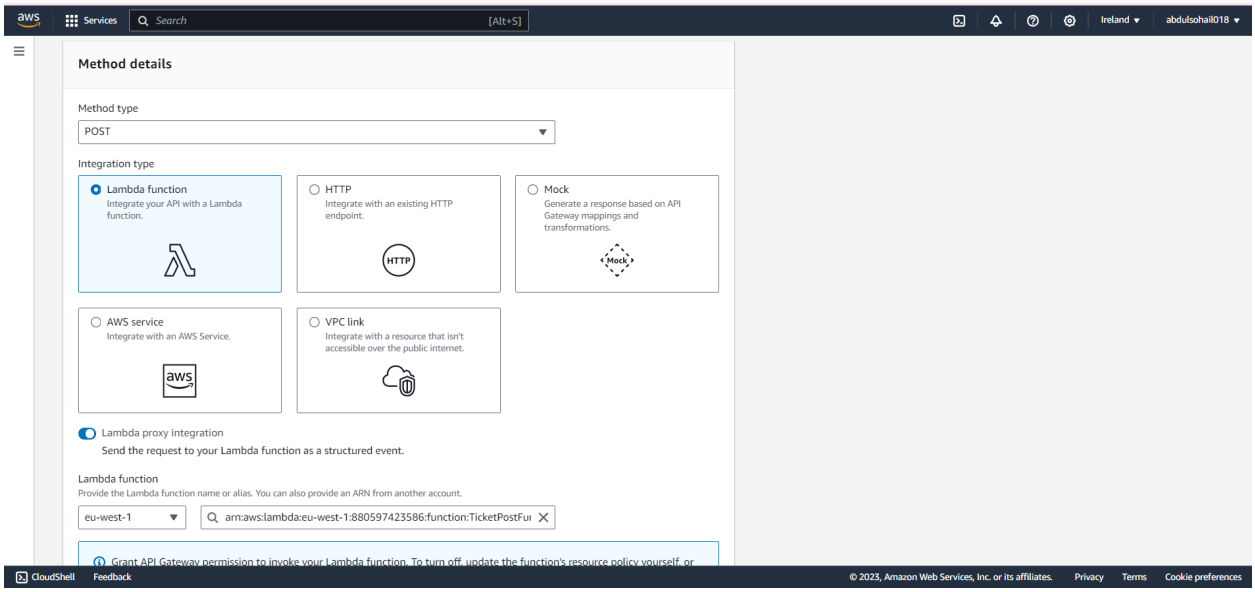


Figure 37: Creating a method in the “ticket” resource by selecting the method type as “POST”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “TicketPostFunction” from the Lambda function

16.2 Creating method type “GET” in Health resource:

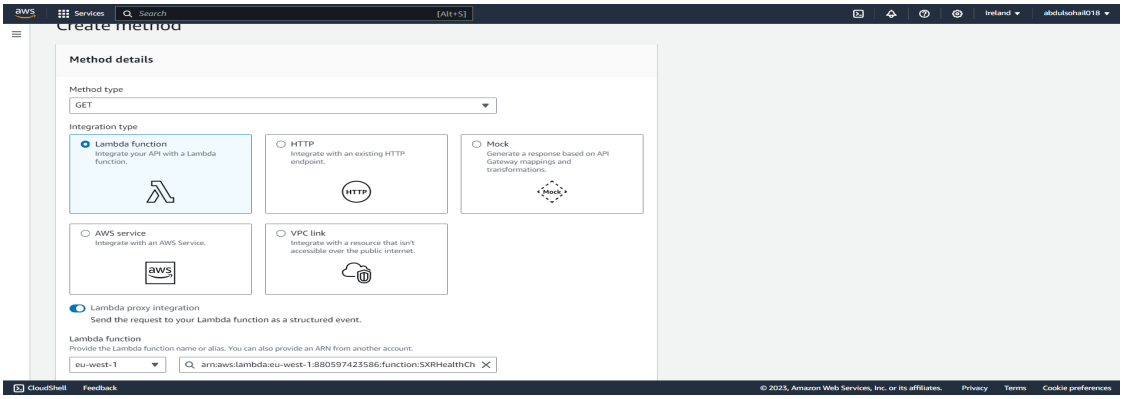


Figure 38: Creating a method in the “health” resource by selecting the method type as “GET”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “SXRHealthCheckFunction” from the Lambda function

17. Created methods in each resource:

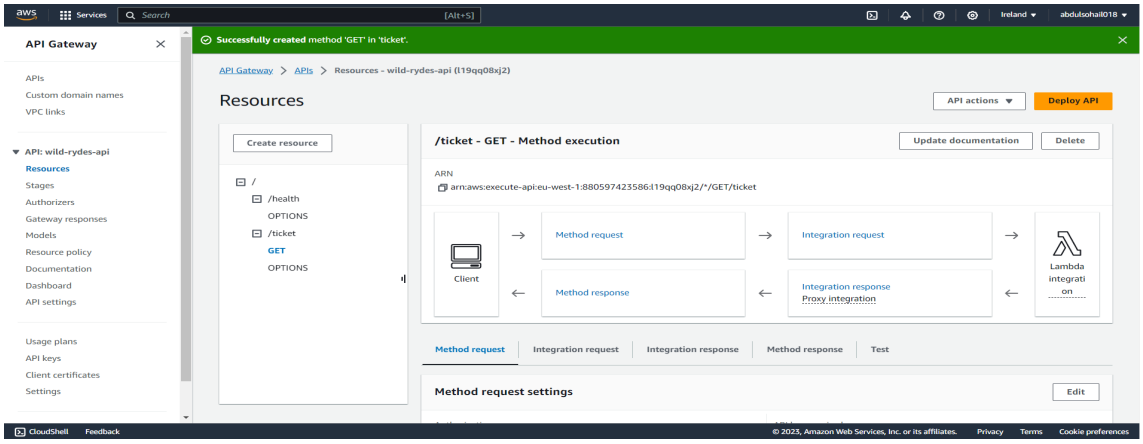


Figure 39: Successfully created GET method in “ticket” resource

17.1 Successful Creation of “POST” Method

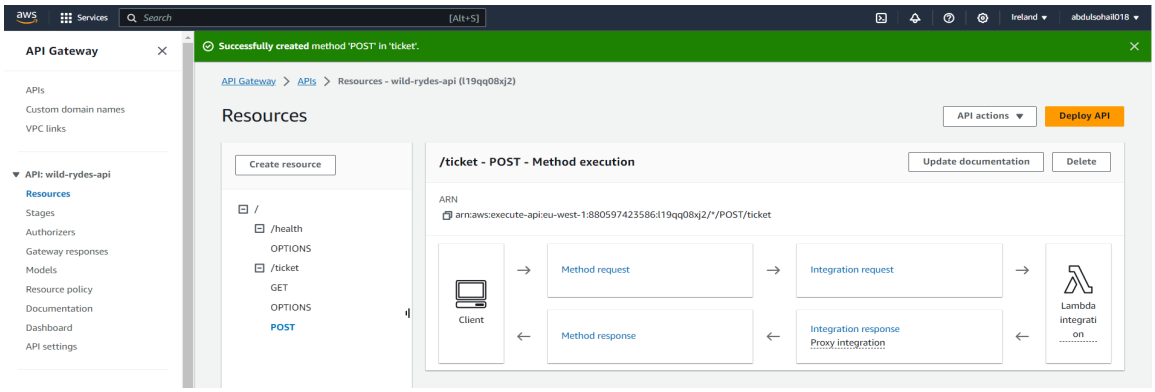


Figure 40: Successfully created POST method in “ticket” resource

17.2 Successful Creation of “GET” Method

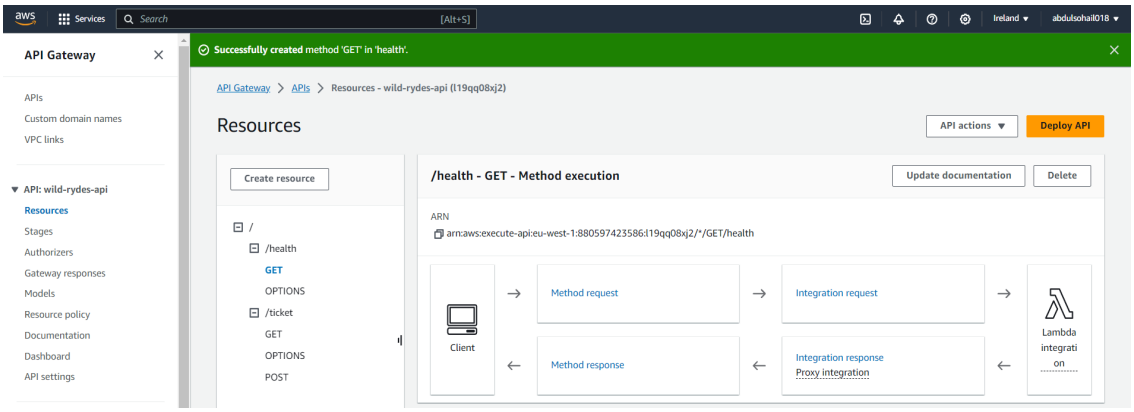


Figure 41: Successfully created GET method in “health” resource

18. Enabling CORS in resources:

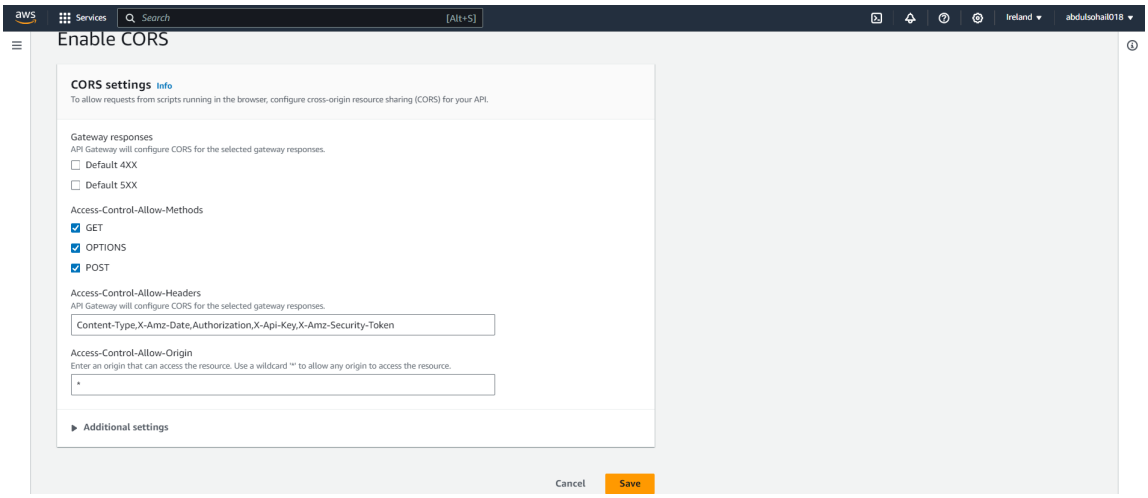


Figure 42: Enabling CORS in the “ticket” resource by selecting all the 3 options “GET”, “OPTIONS” and “POST” in “Access-Control-Allow-Methods”

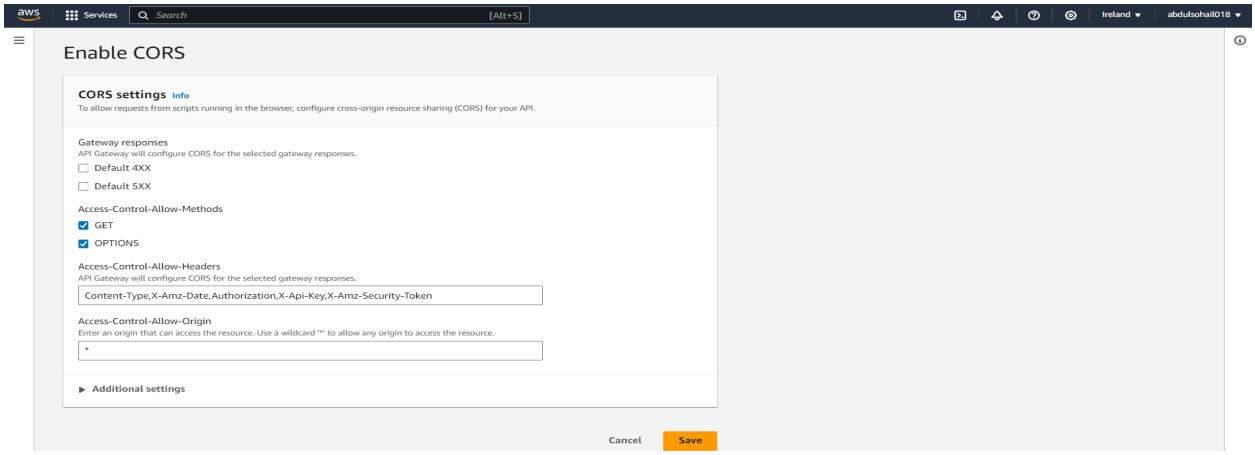


Figure 43: Enabling CORS in the “health” resource by selecting both the options “GET”, and “OPTIONS” in “Access-Control-Allow-Methods”

19. Enable CORS in resources:

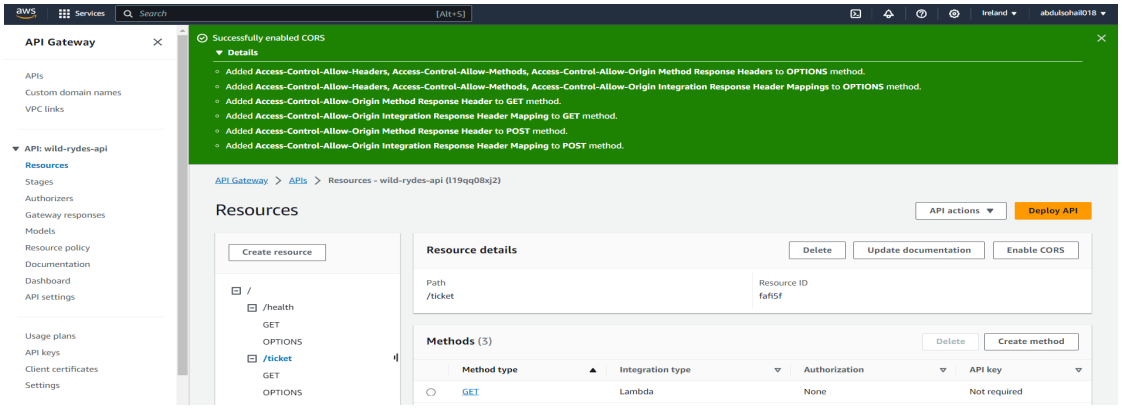


Figure 44: Successfully enabled CORS in "ticket" resource

19.1 Enabling CORS in the Health Resource

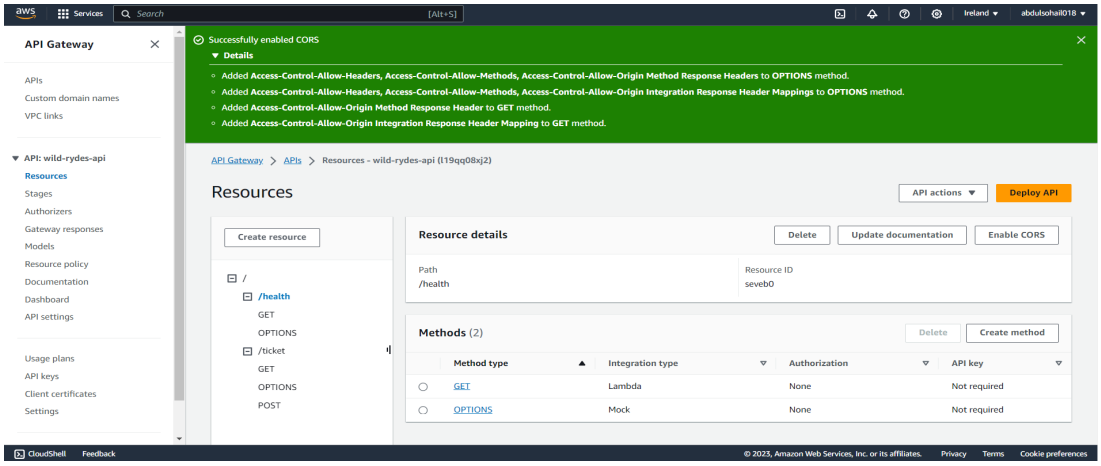


Figure 45: Successfully enabled CORS in "health" resource

20. Deploying API:

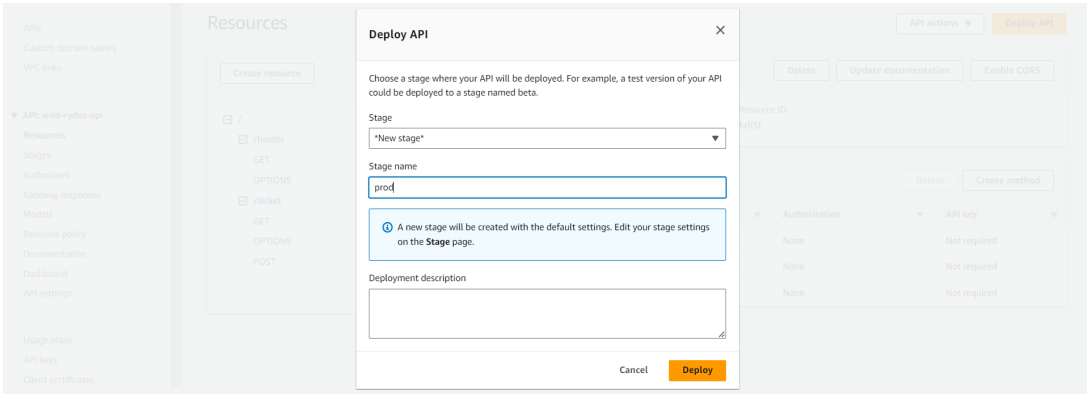


Figure 46: Deploying API by selecting stage as "New Stage" and stage name as "prod"

21. Deployed API:

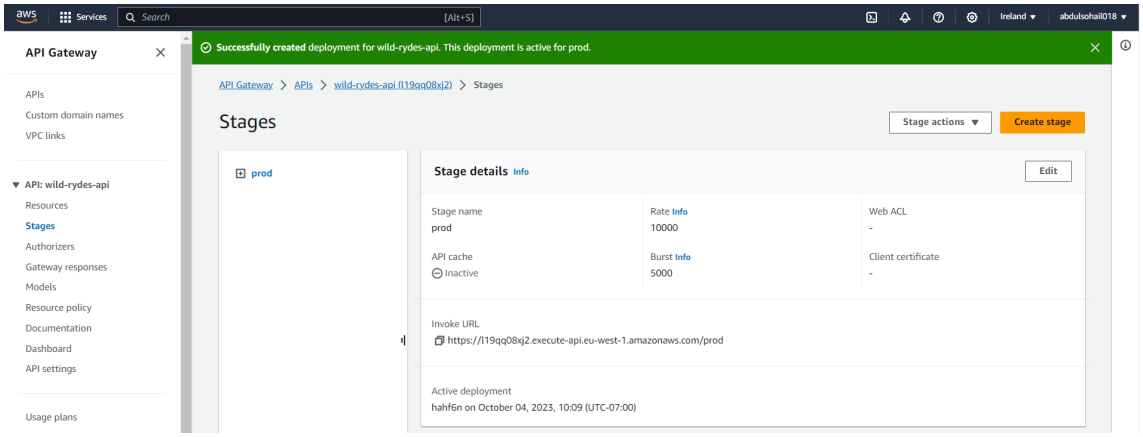


Figure 47: Successfully deployed "wild-rydes-api" API

22. Method execution:

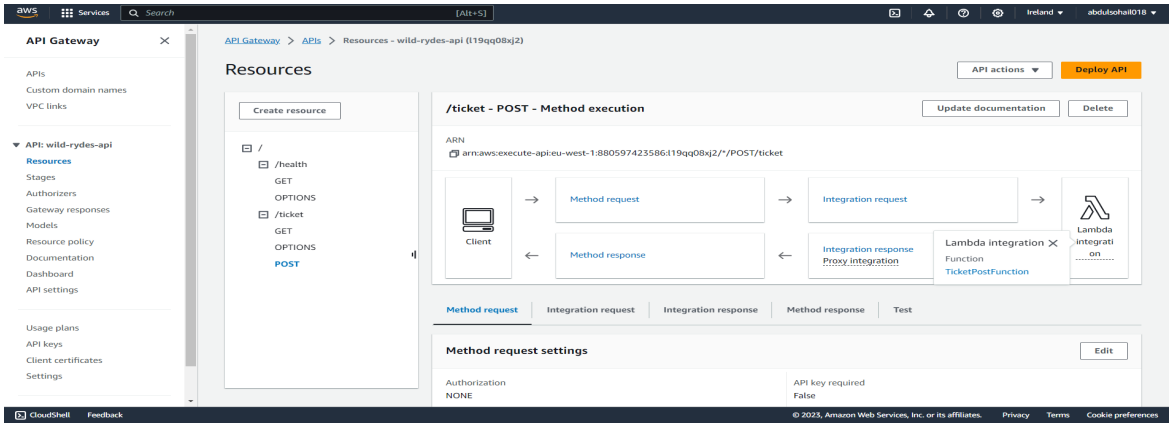


Figure 48: Shows the complete execution of the "POST" method in the "ticket" resource

23. Testing API Gateways Endpoints:

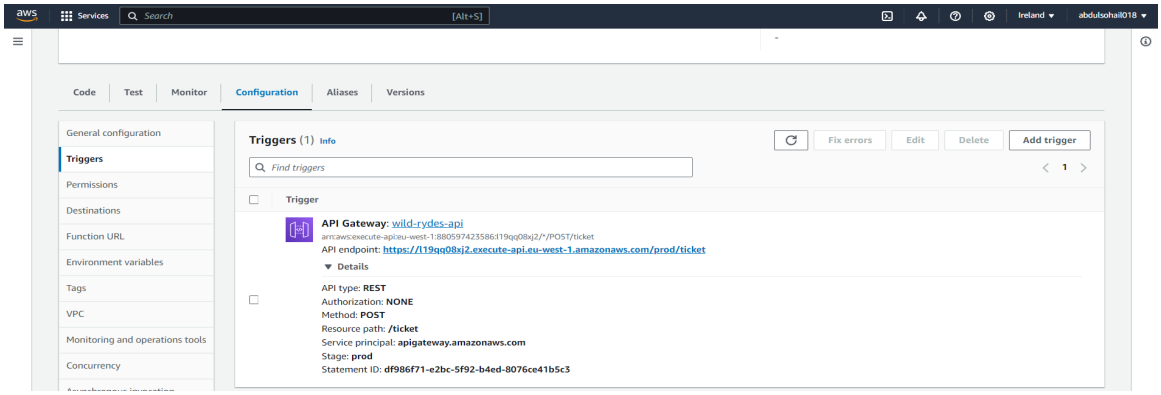


Figure 49: Trigger of "TicketPostFunction" that shows information about API Gateway name, link of API endpoint, etc

23.1 Display Output(Message)

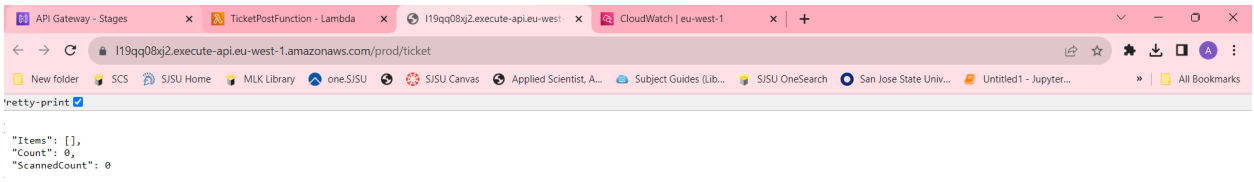


Figure 50: Message showing successful execution of API

23.2 Trigger Creation

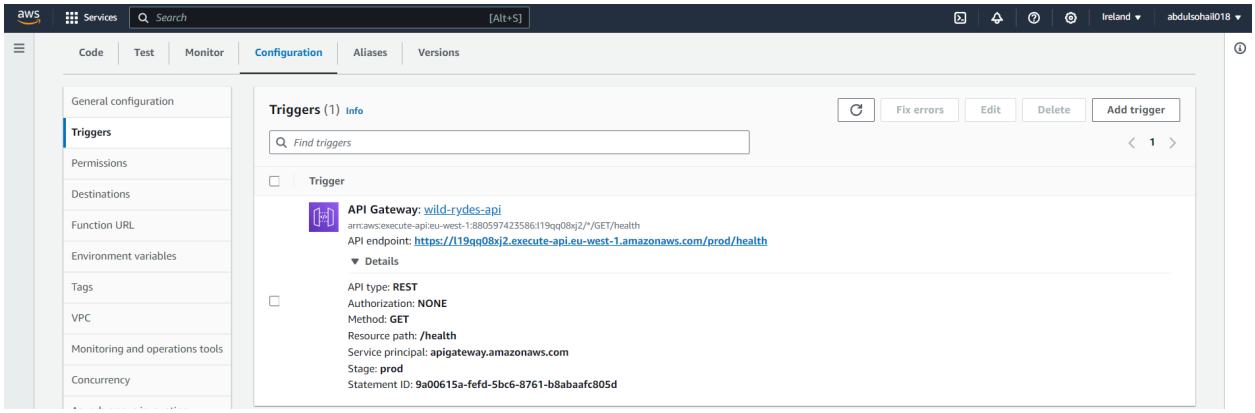


Figure 51: Trigger of "SXRHealthCheckFunction" that shows information about API Gateway name, link of API endpoint, etc

23.3 Display Output(Message)

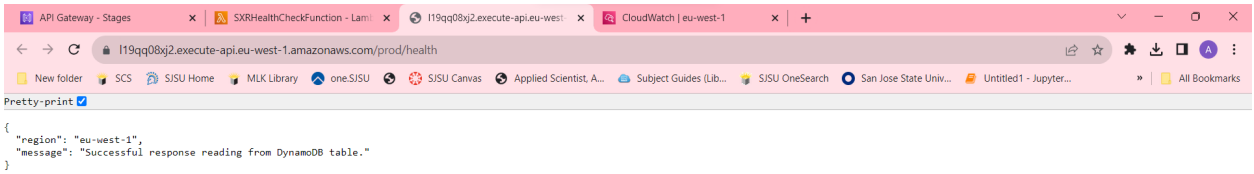


Figure 52: Message showing successful execution of API

Step 3: <u>Building a UI layer:</u>

1. Creating AWS Cognito Identity Pool and S3 bucket:

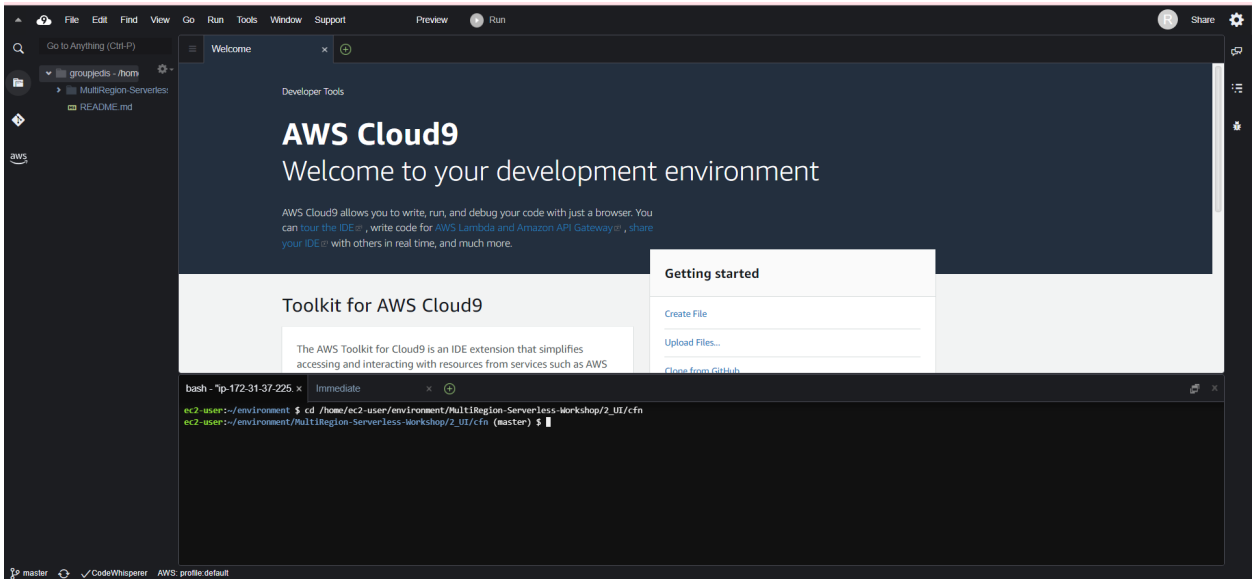


Figure 53: Navigating to Cloud9 IDE and reaching appropriate sub-repository

1.1 Providing the configurations for the Stack

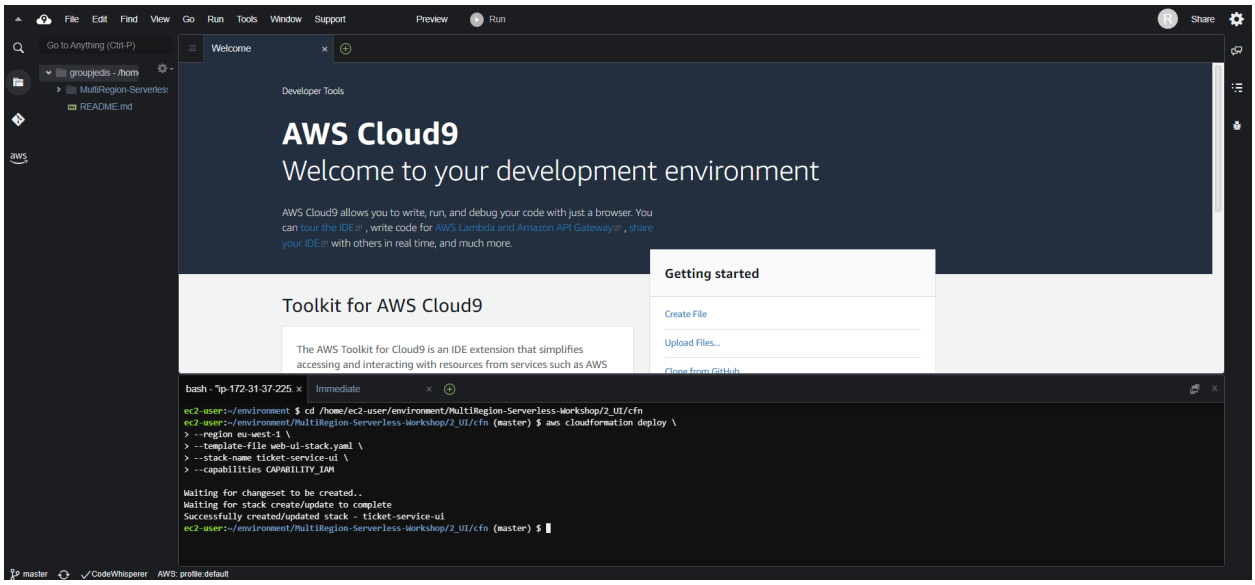


Figure 54: Specifying region, the template file, stack_name "ticket-service-ui" and capabilities of the stack that will be created in CloudFormation

2. Verifying creation of stack:

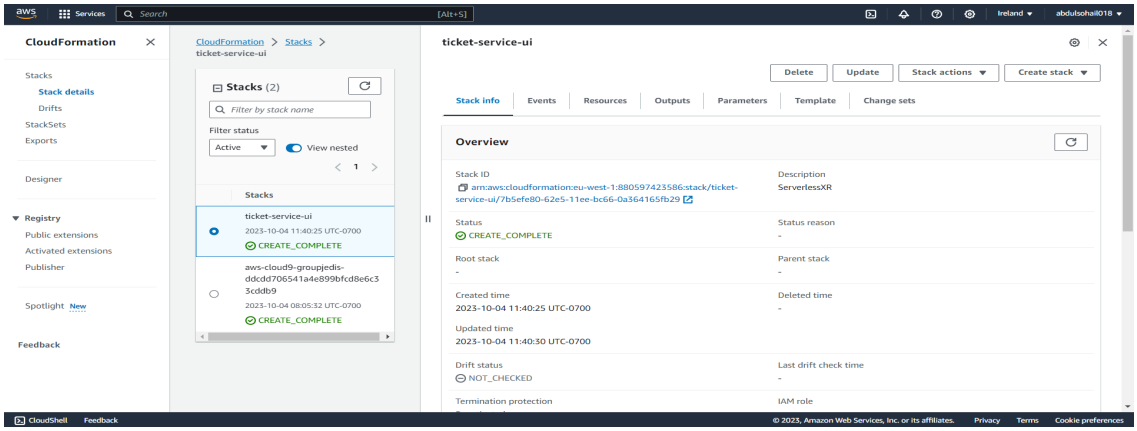


Figure 55: Successfully created stack "ticket-service-ui" in CloudFormation

2.1 Output Generation

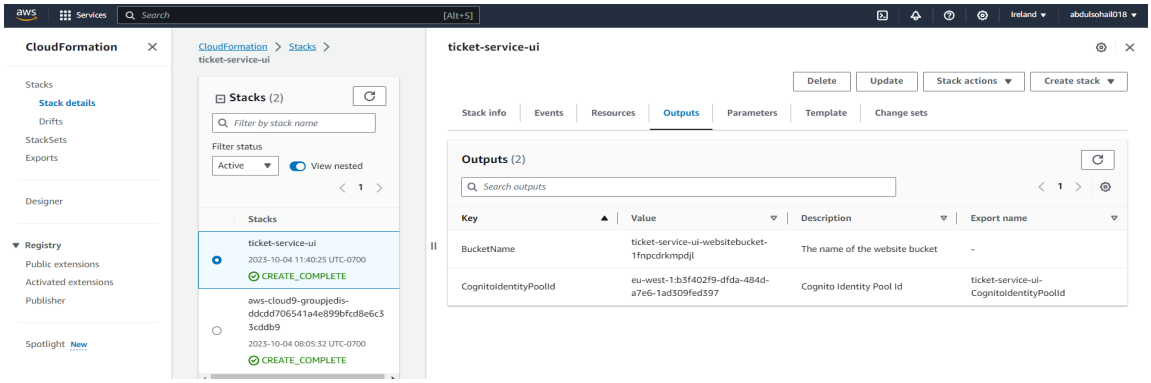


Figure 56: Generated output with BucketName and CognitoIdentityPoolId

3. Created Cloudfront distribution for S3 bucket:

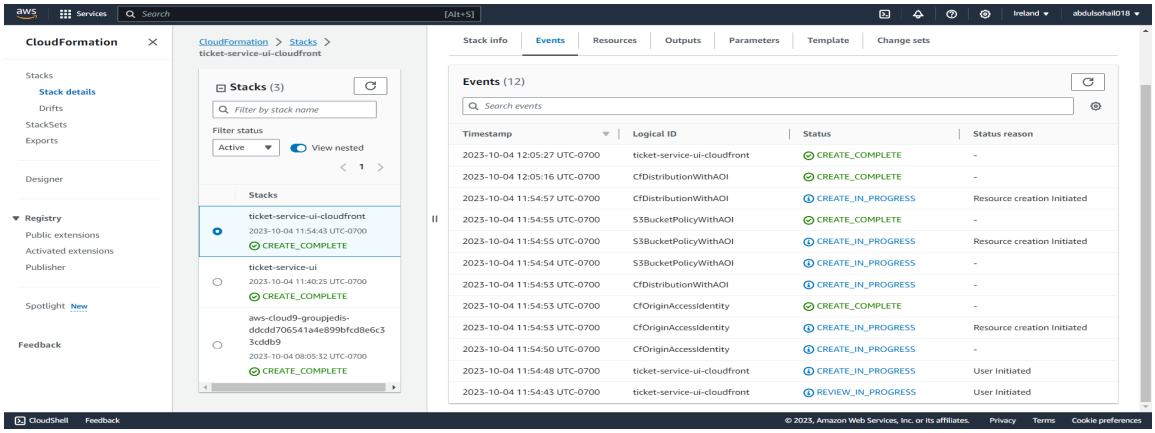


Figure 57: Successfully created cloudfront distribution named "ticket-service-ui-cloudfront"

3.1 Outputs Generated

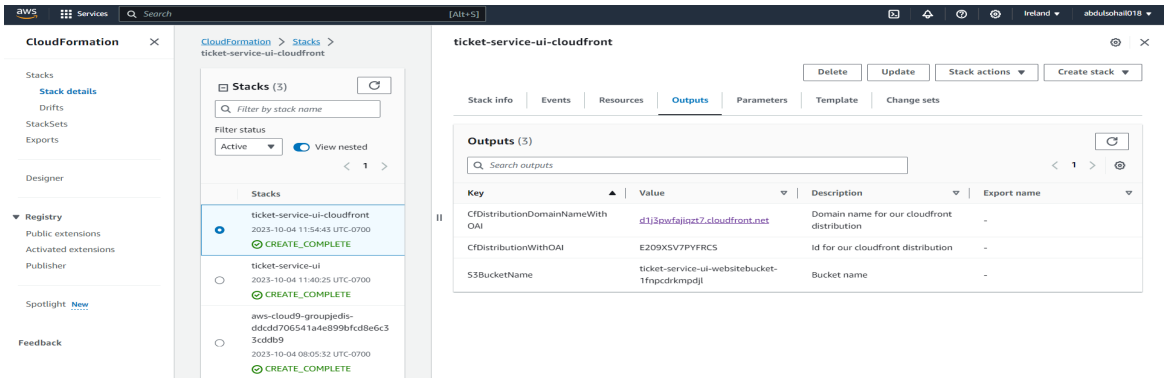


Figure 58: Output section showing domain name and ID of cloudfront distribution as well as S3 Bucket name

4. Configuring Federated Identities with Cognito

4.1 Creating an application in Identity Provider - Facebook

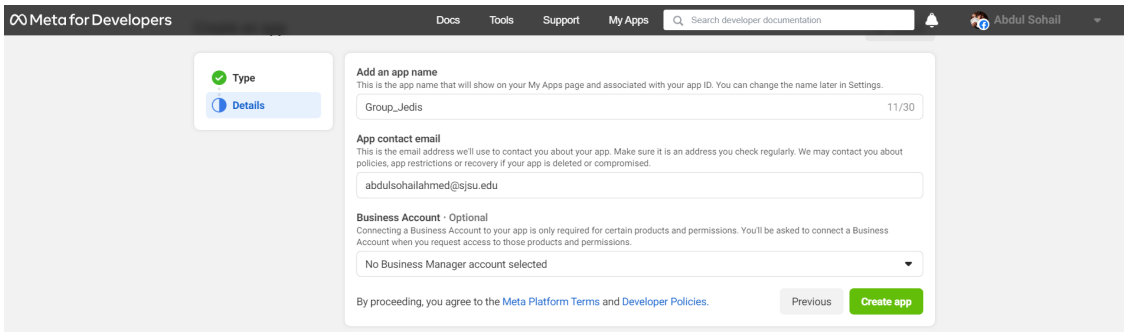


Figure 59: Providing the name of the application as "Group_Jedis" and the email address

4.2 Adding platform:

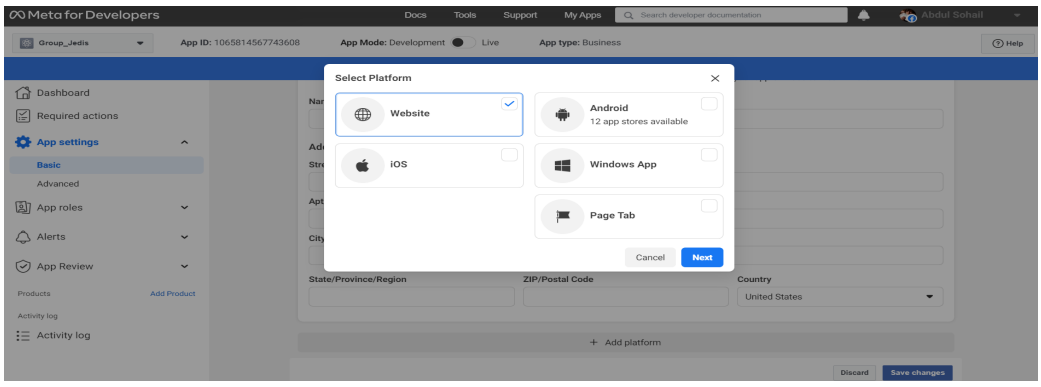


Figure 60: Selecting Website as a platform

4.3 Providing cloudfront URL:

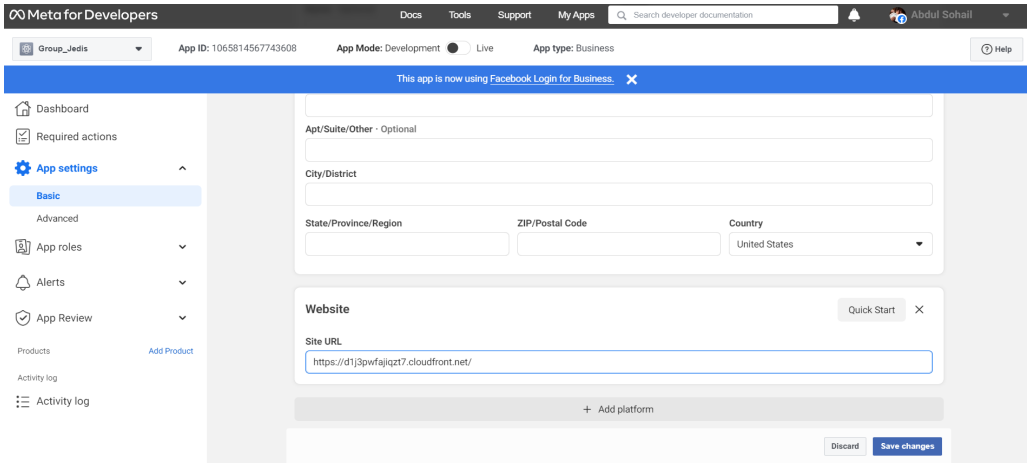


Figure 61: Providing the cloudfront URL obtained earlier in the site URL

4.4 Configuring Cognito Identity Pool to Facebook (Identity Provider):

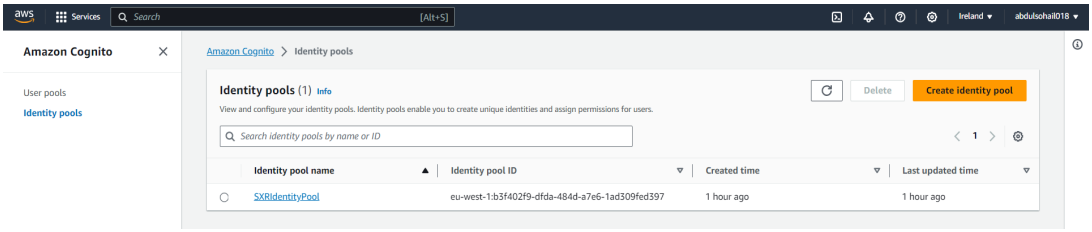


Figure 62: Amazon Cognito Dashboard

4.5 Addition of Identity Provider

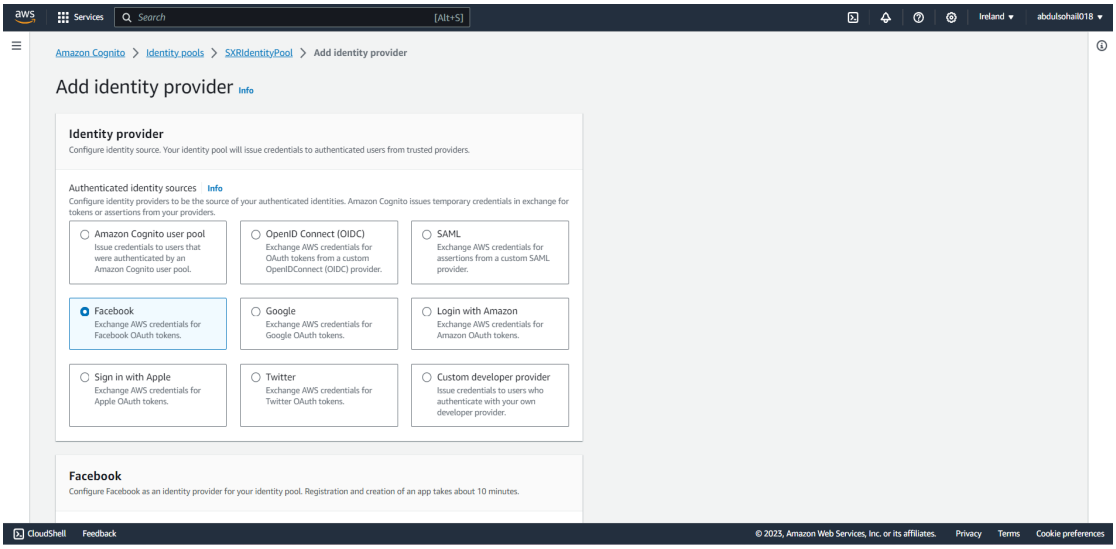


Figure 63: Adding Identity Provider “Facebook”

4.6 Setting up the application id for Facebook

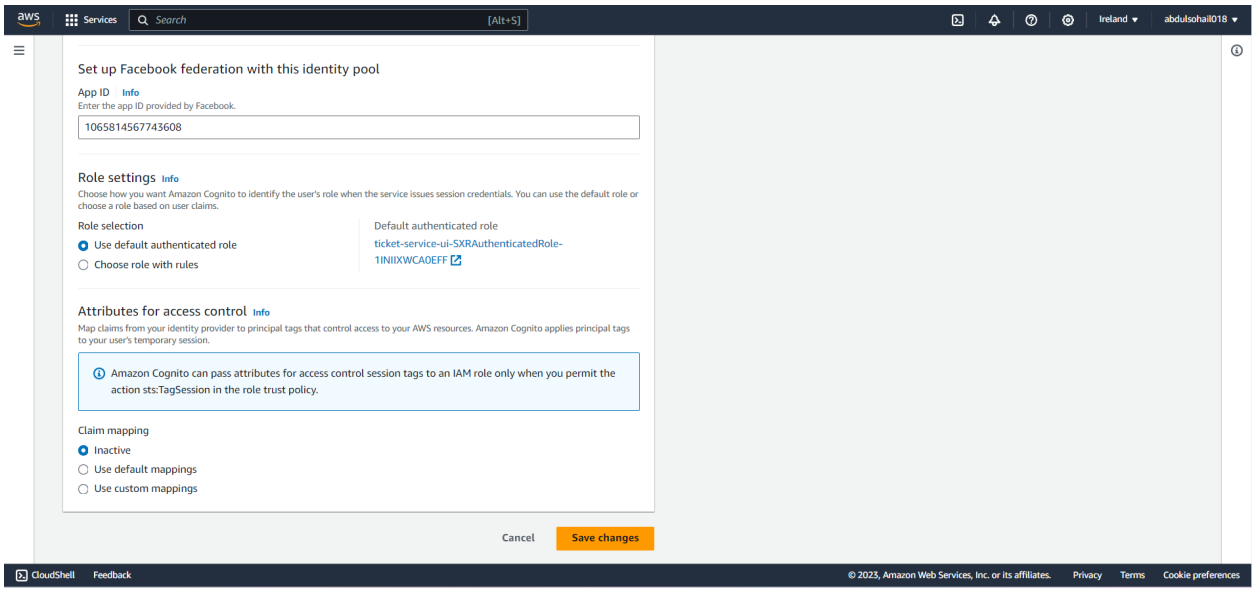


Figure 64: Providing the application ID of the application created earlier in Facebook

5. Configuring and building application code:

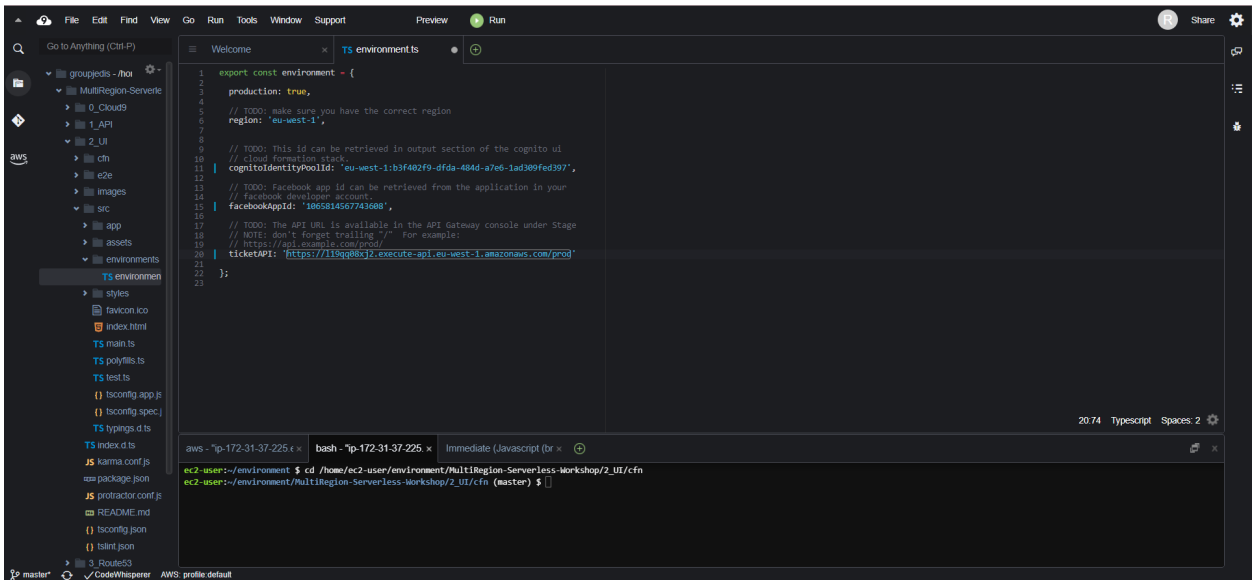


Figure 65: Modifying the "environment.ts" file to provide details like cognitoIdentityPoolId, ticketAPI and facebookAppID

5.1 Code for Setting up the Root folder and Installing Dependencies

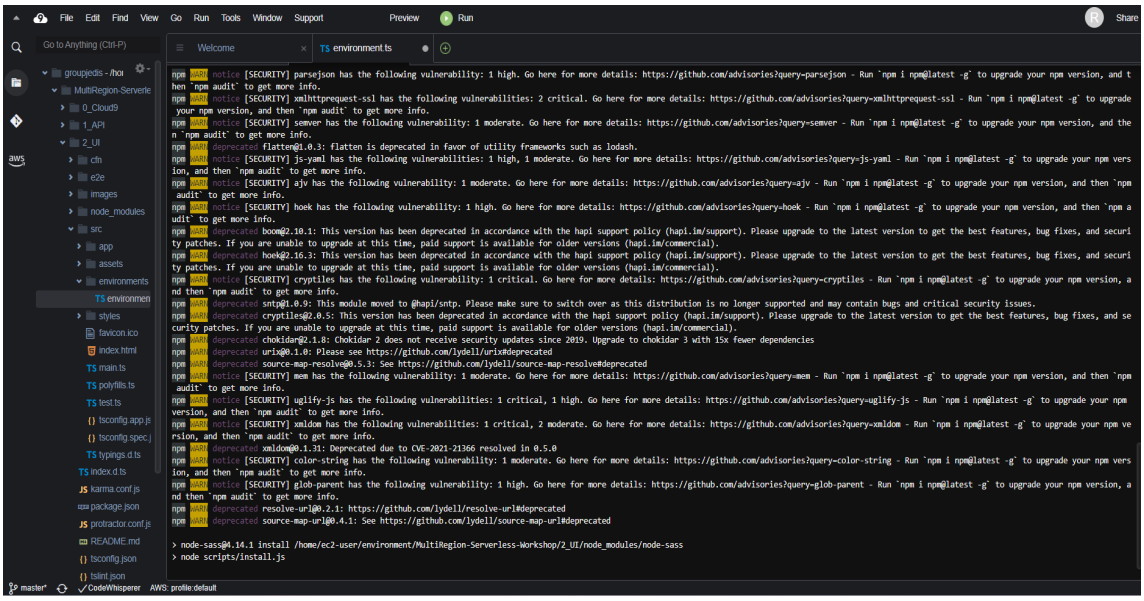


Figure 66: Navigating to root folder of ui project and installing necessary dependencies

5.2 Code to successfully build the application

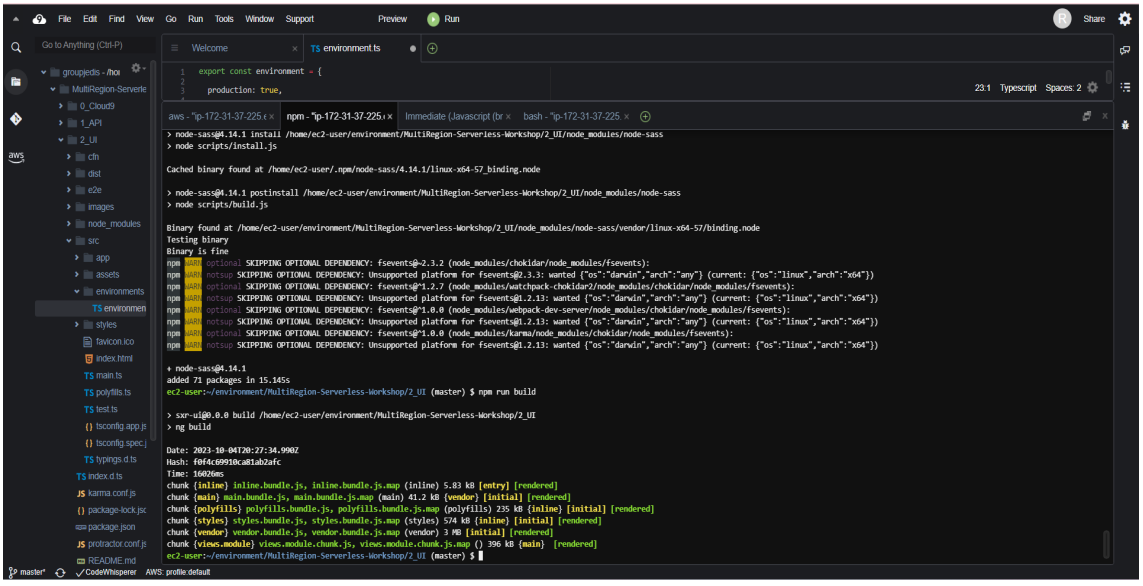


Figure 67: Successfully built the application

6. Uploading the application:

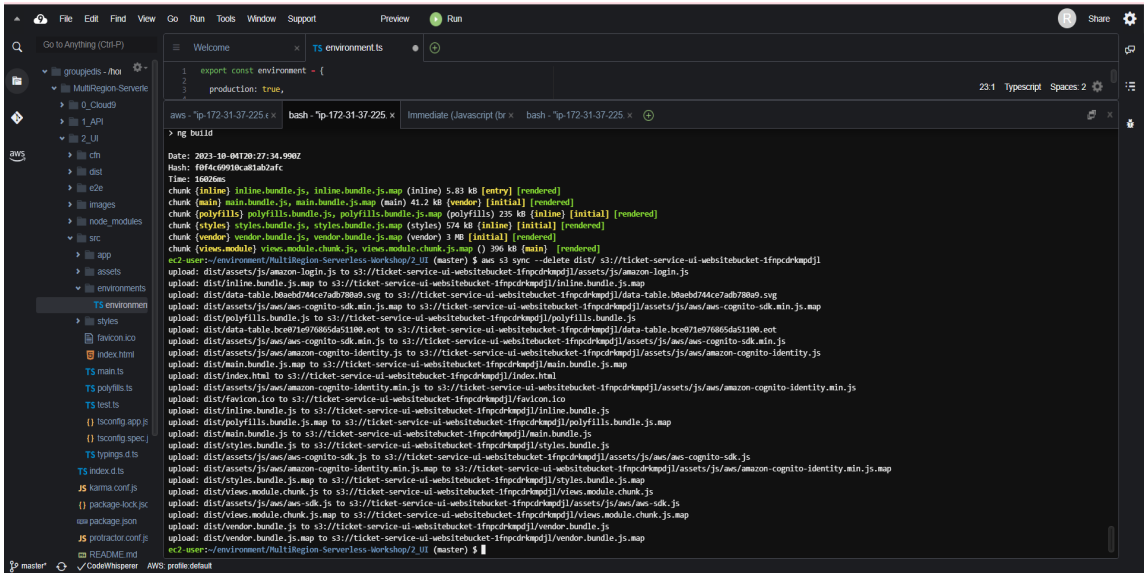


Figure 68: Uploading the UI to S3 bucket

6.1 Updation of Application Settings

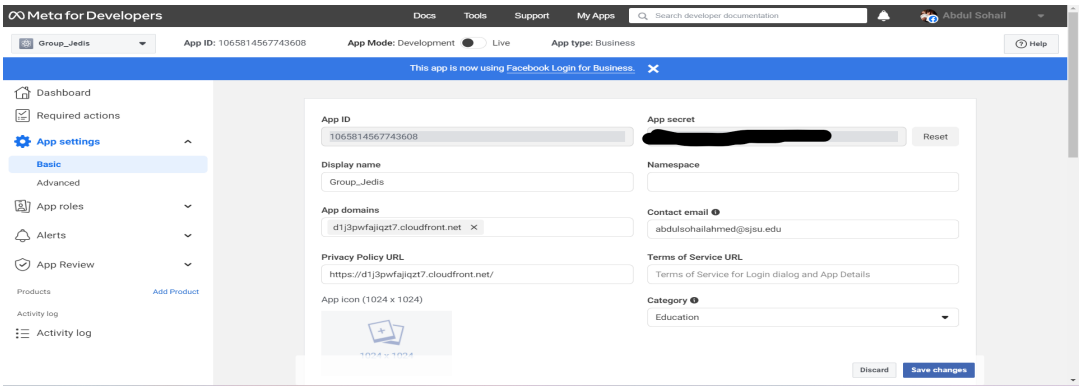


Figure 69: Updating Facebook application settings by adding details like App domains and Privacy Policy URL with cloudfront URL

7. Verifying the application:

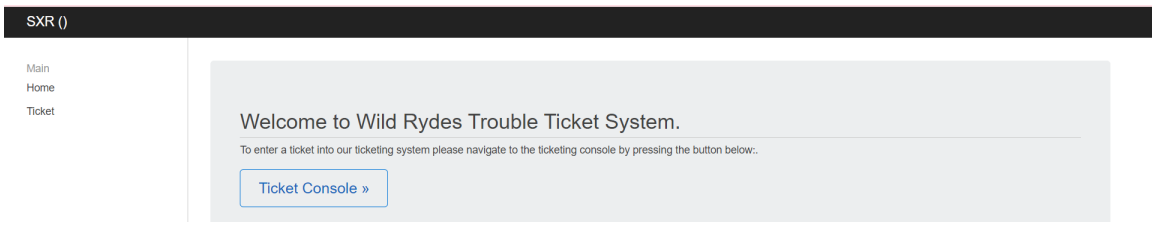


Figure 70: Navigating using cloudfront URL to the Ticket System Console

8. Accessing the Ticket console

Figure 71: Successfully able to access the ticketing system in console

Step 4: <u>Replicate to second region:</u>

1. Configuring Route 53 Domain:

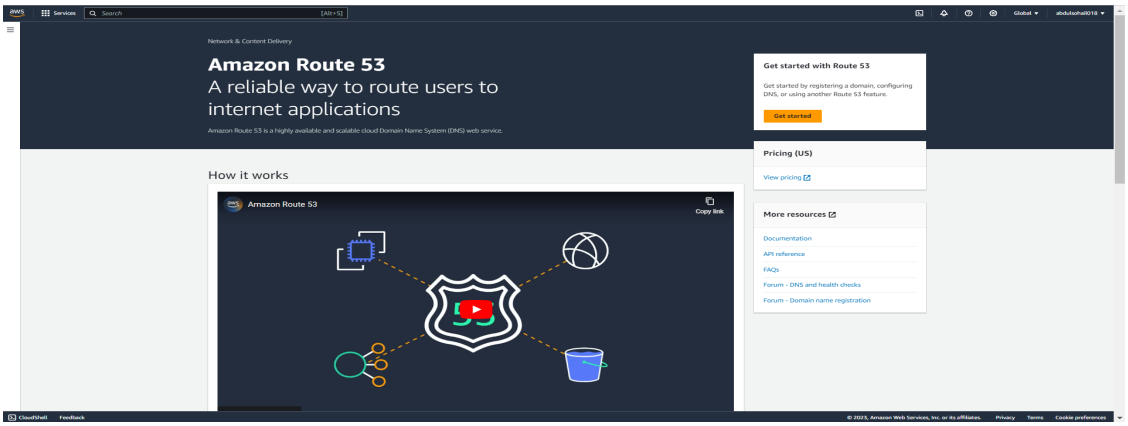


Figure 72: Amazon Route 53 Dashboard

1.1 Purchased Domain Details

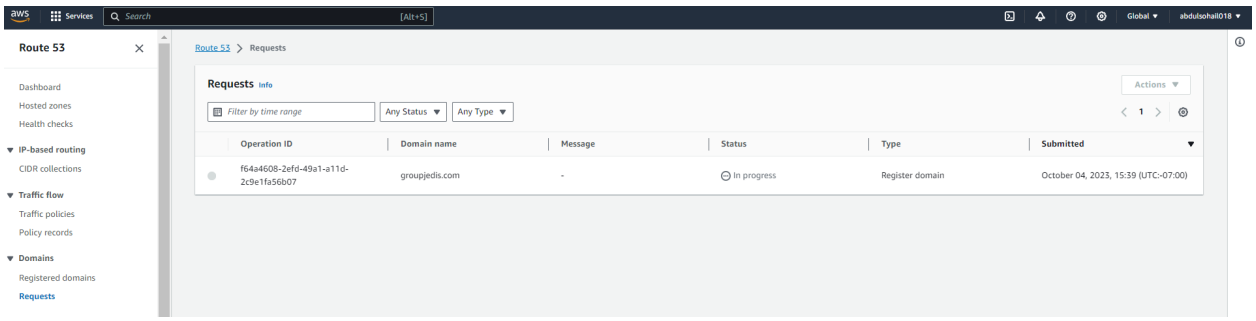


Figure 73: Purchased Domain Name

1.2 Request for the Public Certificate

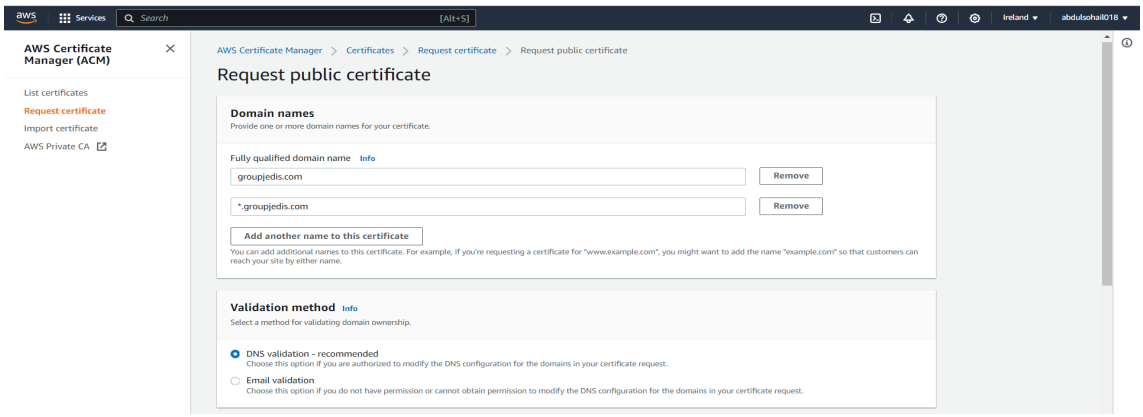


Figure 74: Requesting the Public Certificate for the purchased Domain

1.3 Setting up the Details for the Certificate

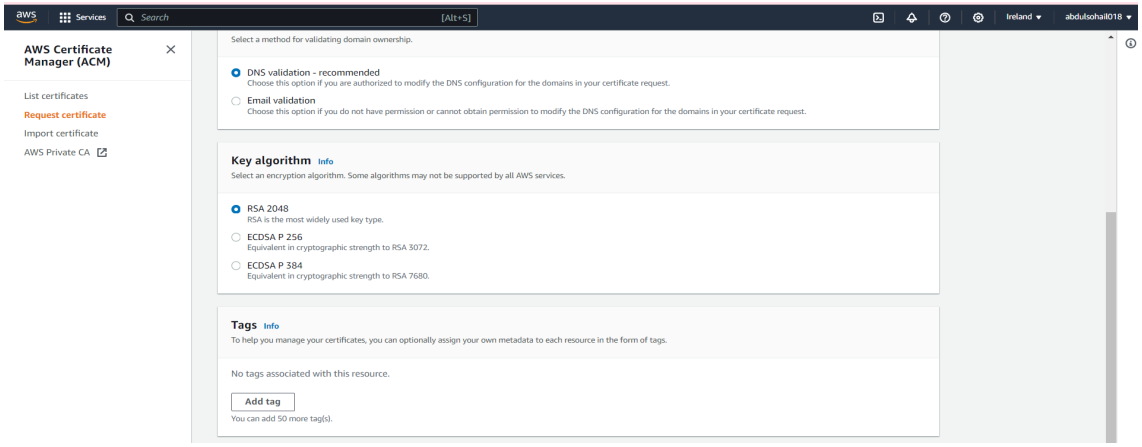


Figure 75: We are providing the details to be included in the certificate

1.4 Checking the Certificate Availability

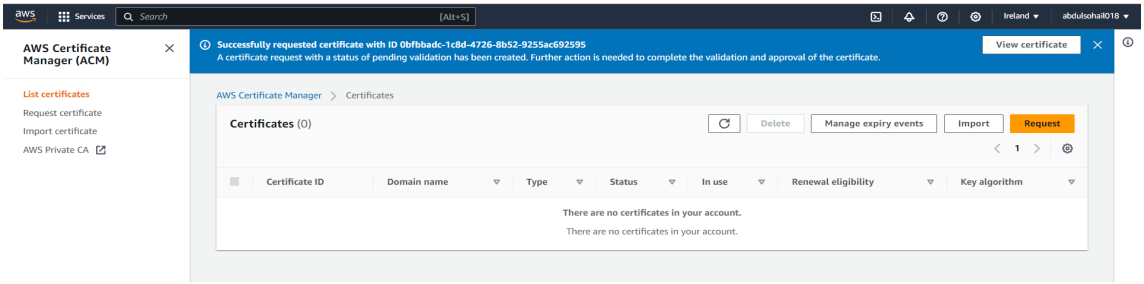


Figure 76: It currently shows that there are no available certificates in the account.

1.5 Certificate Status

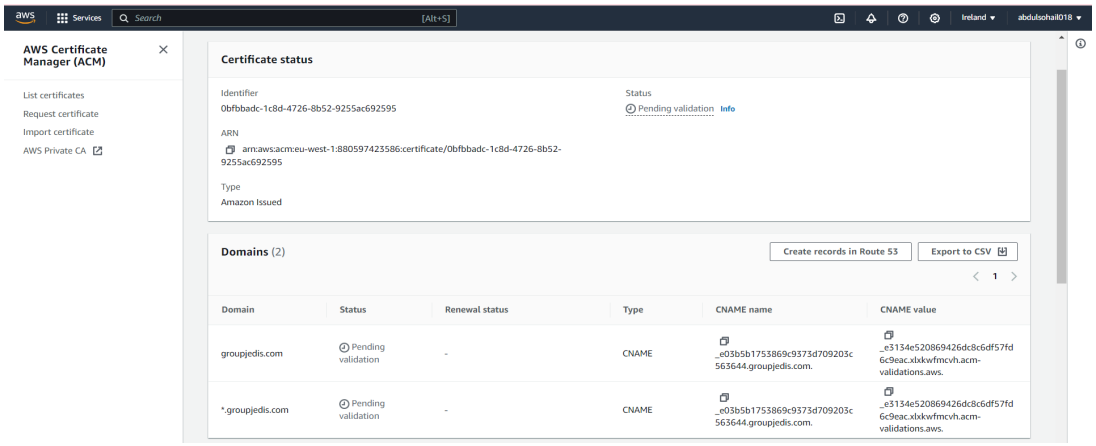


Figure 77: The Certificate Status shows Pending Validation

1.6 Issued Certificate

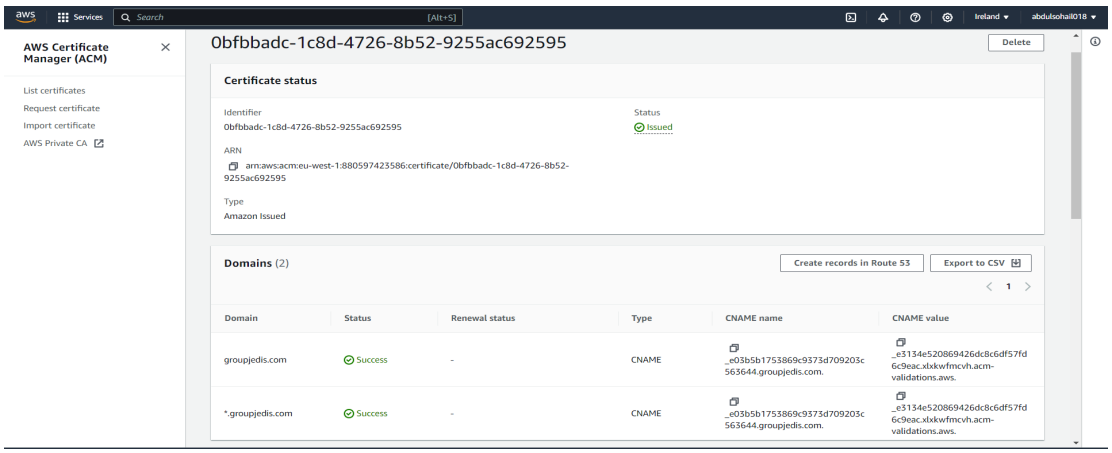


Figure 78: The Certificate Status is updated as "Issued"

1.7 Successful Creation of Certificate Record in Amazon S3

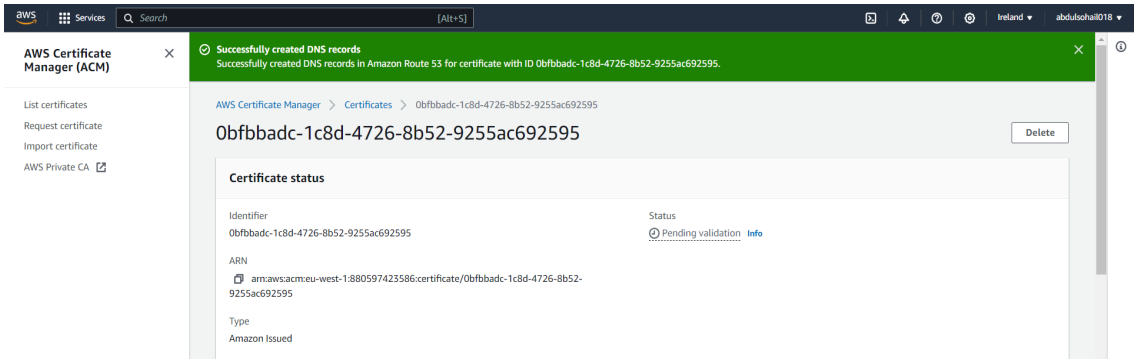


Figure 79: The updation of DNS Records which now includes the Requested Certificate

2. Registered Domain Status

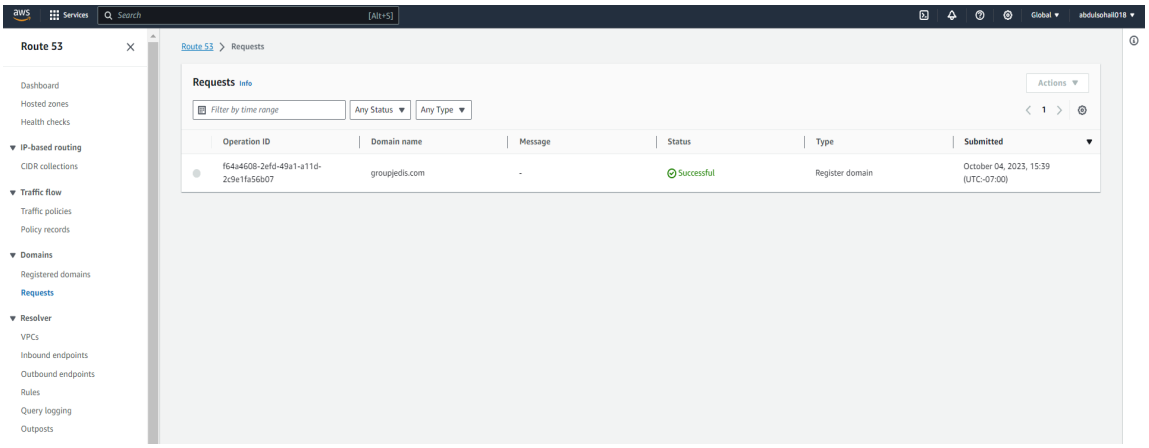


Figure 80: Shows the details of the registered domain as "Successful"

2.1 Certificate Details in the given Region

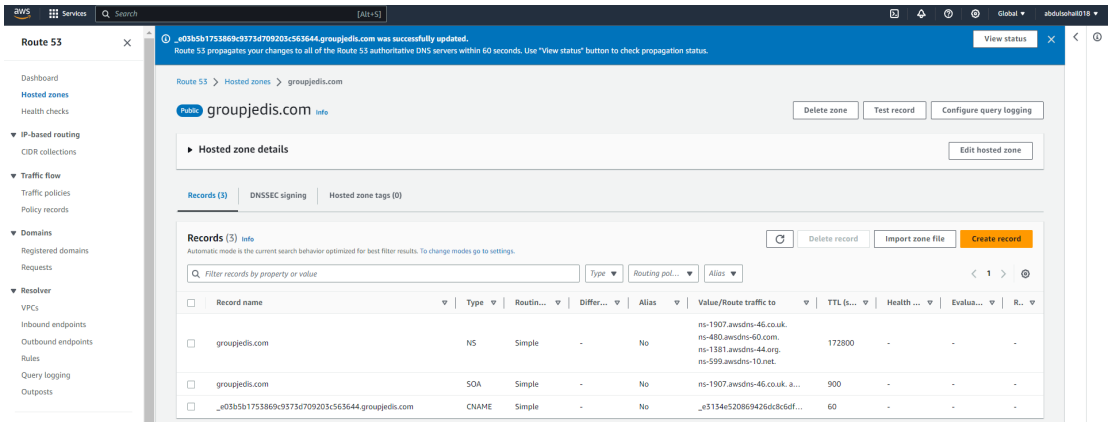


Figure 81: It shows the details of Certificate issued for the particular region in the registered domain

2.2 Creation of Custom Sub-Domain

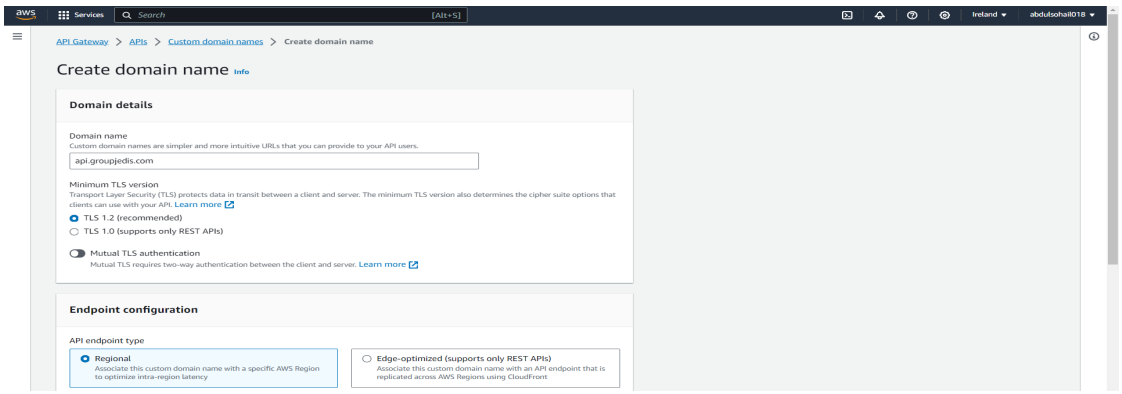


Figure 82: Creating the sub-domain to configure the health checks via each region

2.3 Setting up the End Point Configuration

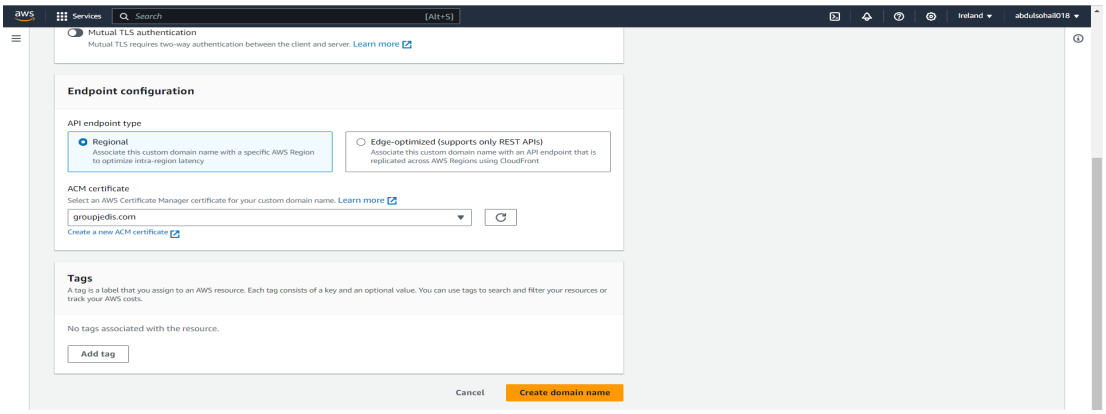


Figure 83: Configuring the End Point

2.4 Reviewing the EndPoint configuration

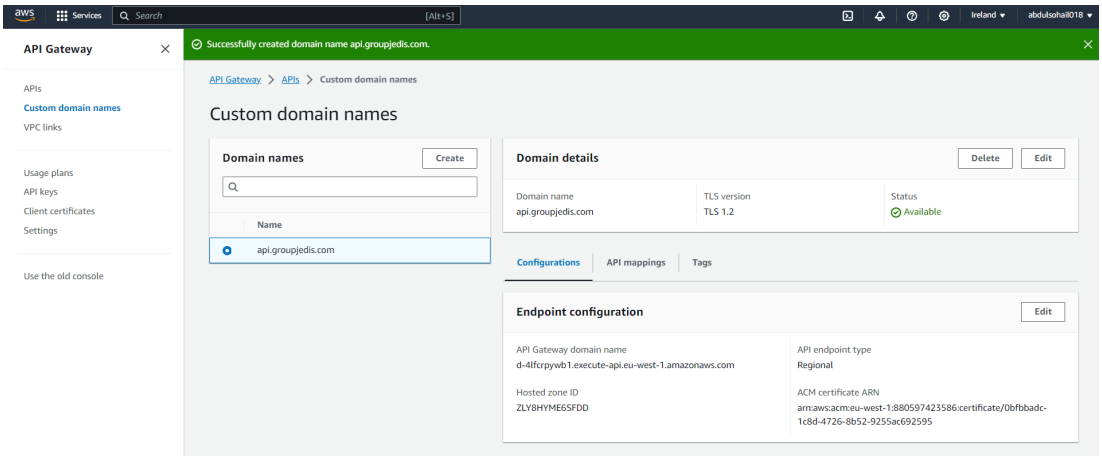


Figure 84: We have reviewed the configuration for the API Endpoint

2.5 Creation of Custom Sub-Domain for Ireland Region

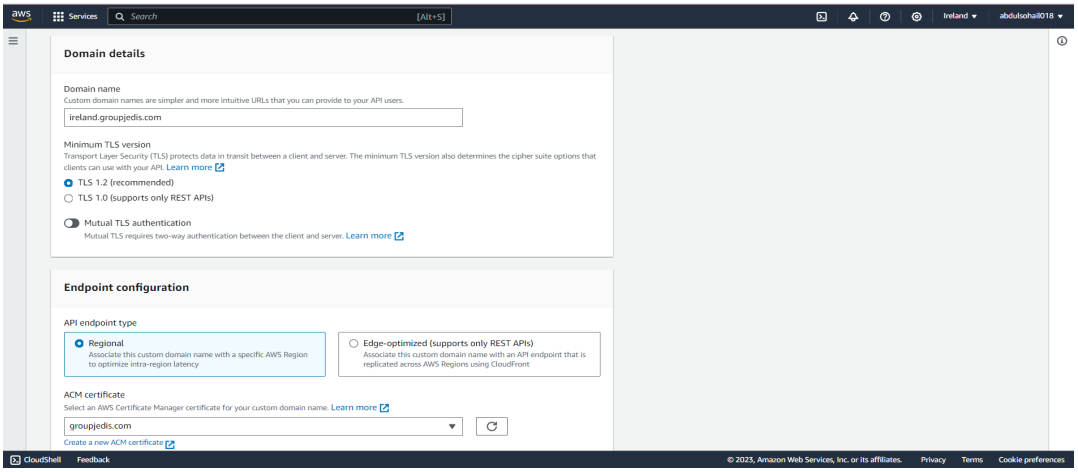


Figure 85: Creating a custom domain along with endpoint configuration

2.6 Created Custom Domains

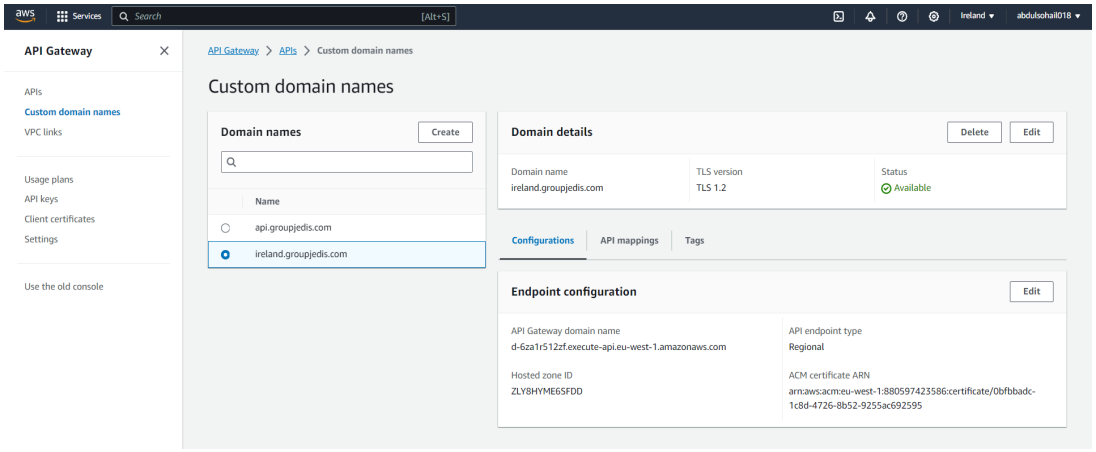


Figure 86: Shows the custom domain created

3. Replicating to Singapore region:

3.1 Naming AWS Cloud 9 environment

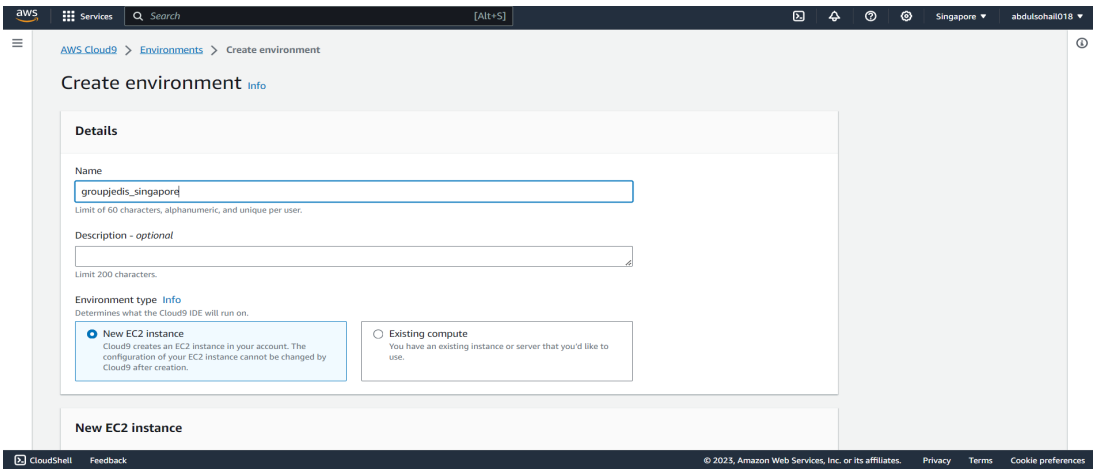


Figure 87: Providing the name of the Cloud9 environment

3.2 Created Cloud9 environment:

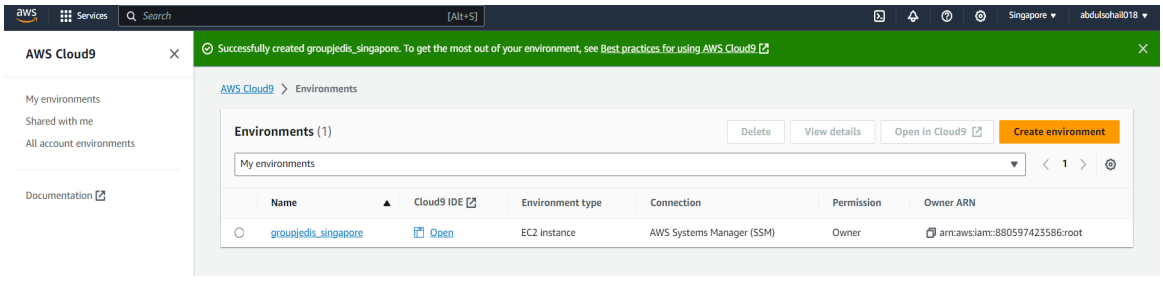


Figure 88: Successfully able to create the Cloud9 environment

3.3 Creating AWS Cognito Identity Pool and S3 bucket:

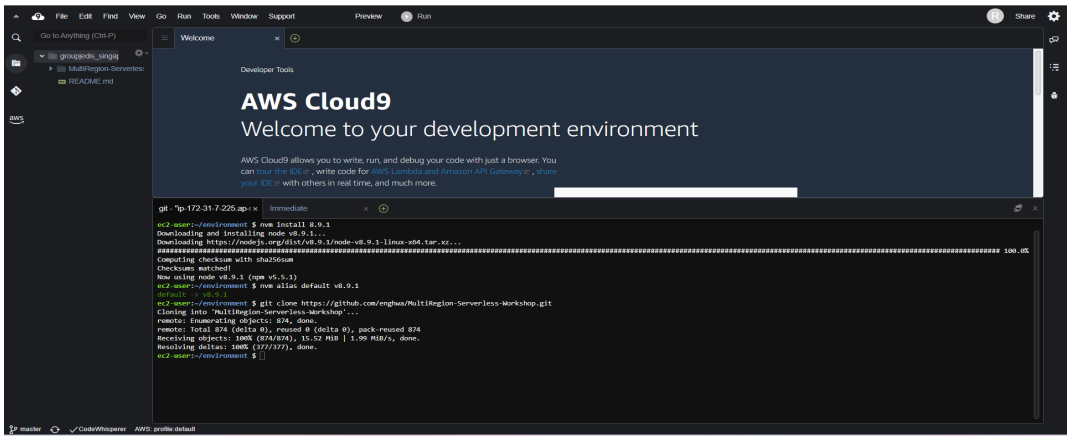


Figure 89: Installing the node and npm required for Angular Front end

3.4 Creating lambda functions:

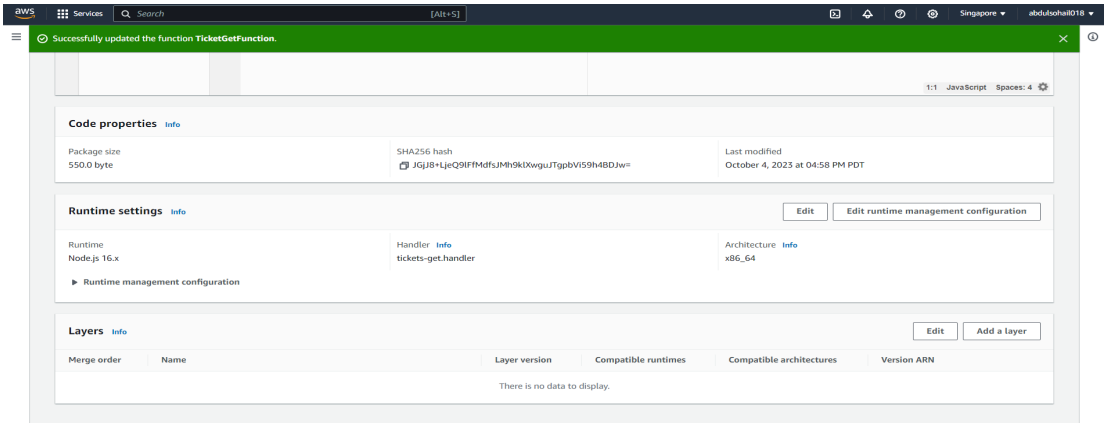


Figure 90: Providing function name "TicketGetFunction" and selecting "Node.js 16.x" to write functions

3.5 Setting the Environment Variables

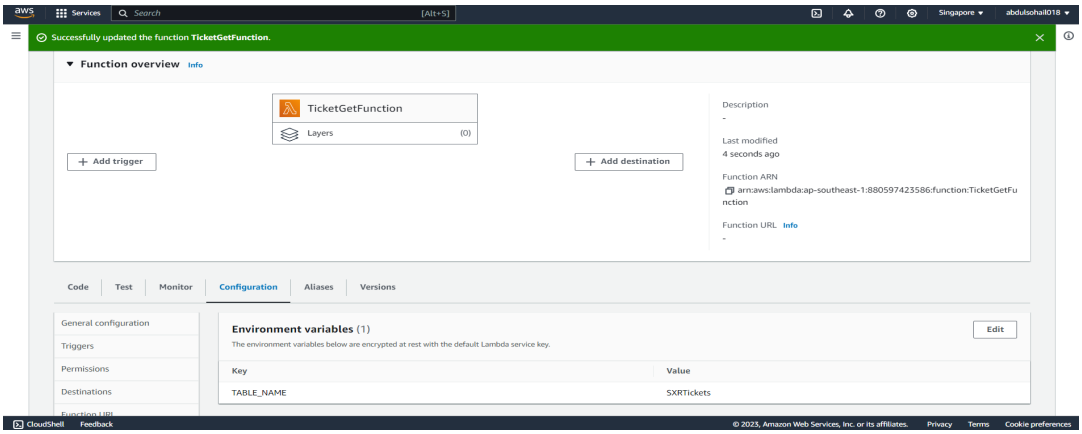


Figure 91: Under environment variables entering the key TABLE_NAME and the value SXRTickets.

3.6 Creation of TicketPostFunction

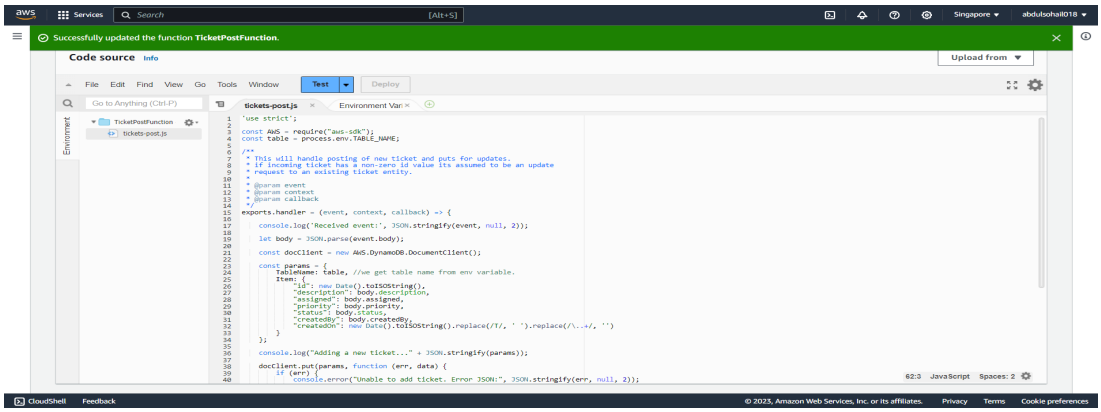


Figure 92: Modifying the "TicketPostFunction" function

3.7 Setting the Environment Variables

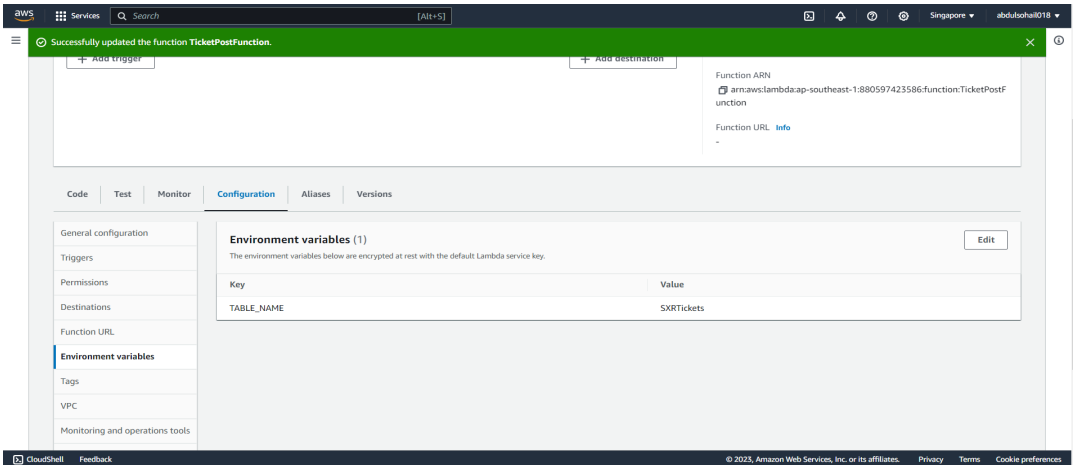


Figure 93: Under environment variables entering the key TABLE_NAME and the value SXRTickets.

3.8 Setting up the Health function

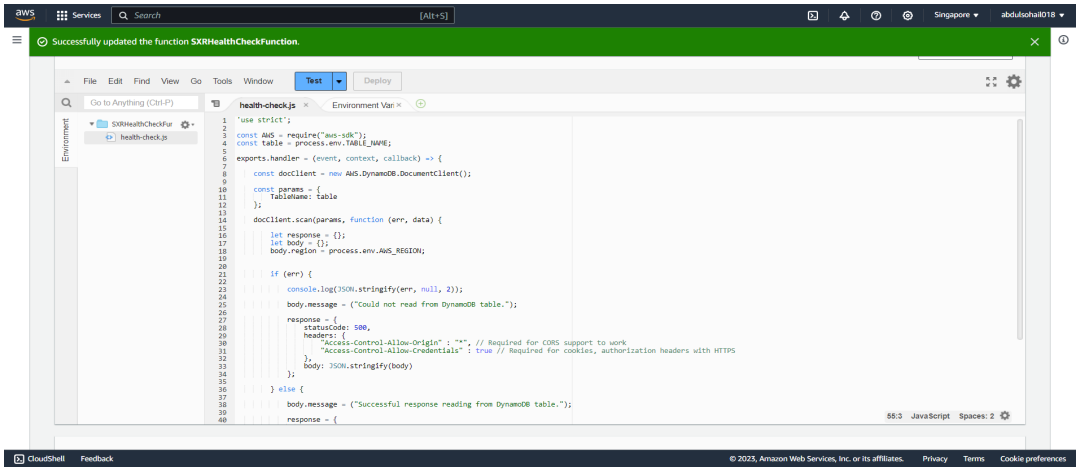


Figure 94: Modifying the "SXRHealthCheckFunction" function

3.9 Updating the Health Check Function

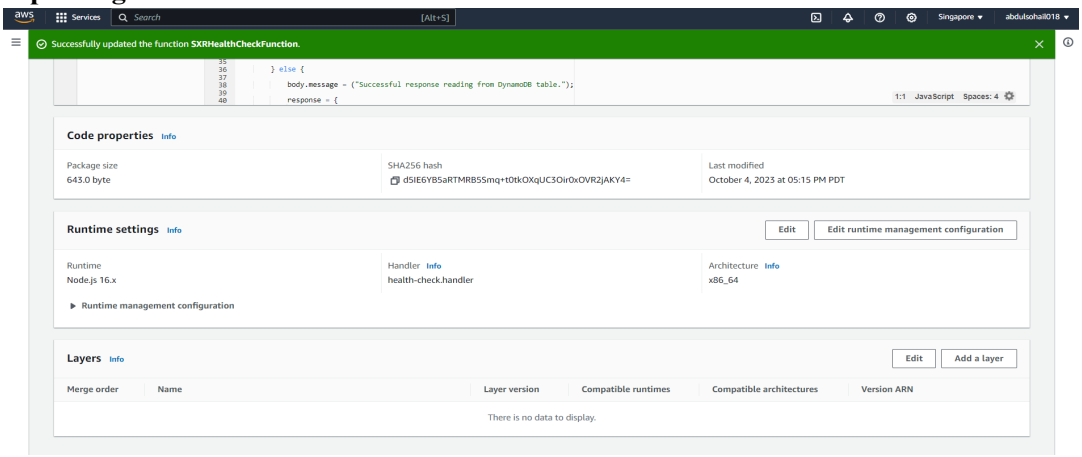


Figure 95: Changed handler to “health-check.handler”

3.10 Providing values for Environment variables

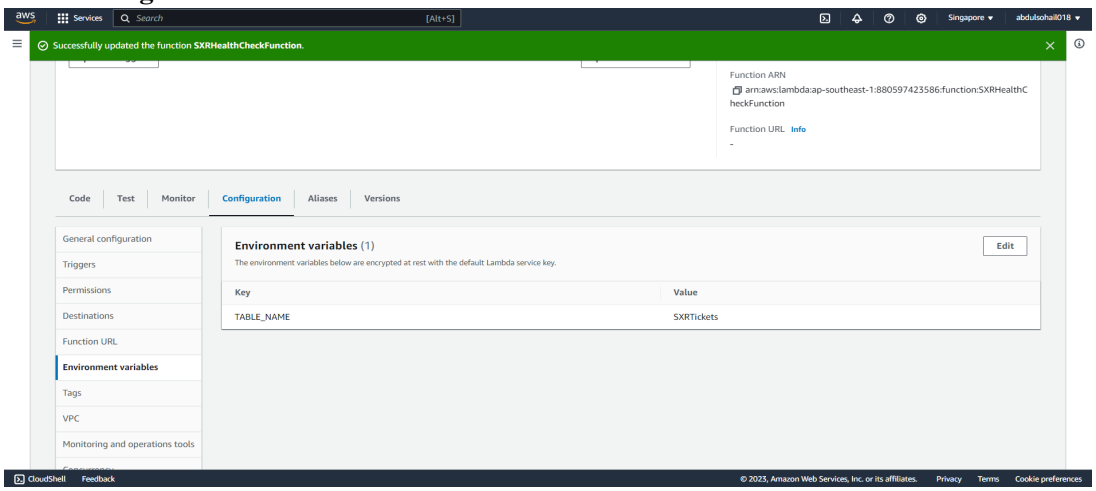


Figure 96: Under environment variables entering the key TABLE_NAME and the value SXRTickets.

3.11 Creating a new API

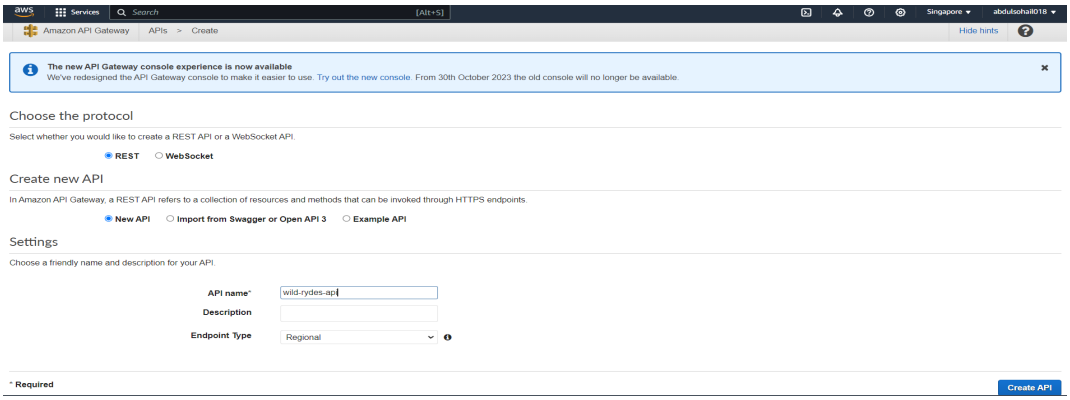


Figure 97: Created a new API using REST Protocol

3.12 Creation of new Resource

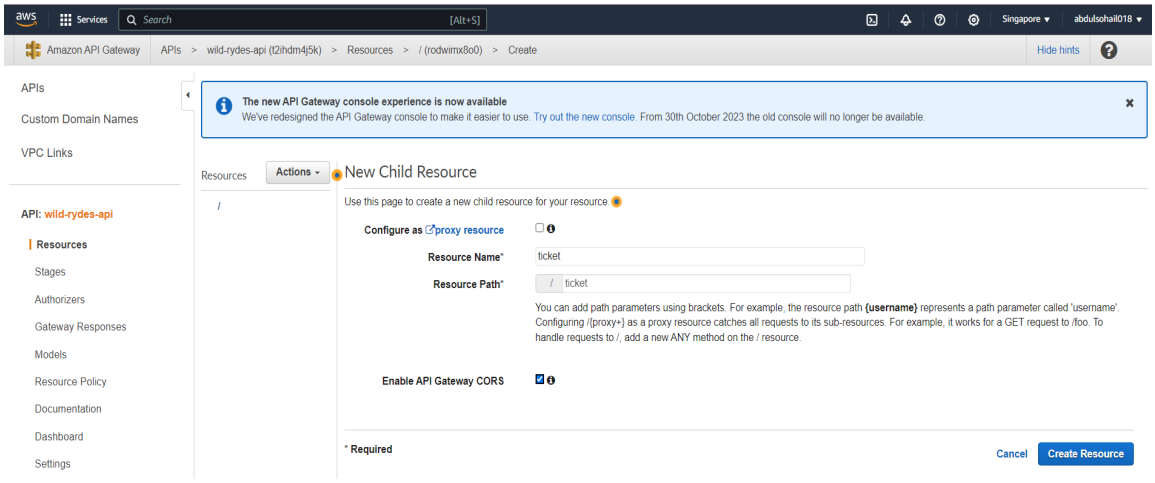


Figure 98: Created a new Resource named “Ticket”

3.13 Creation of new Resource

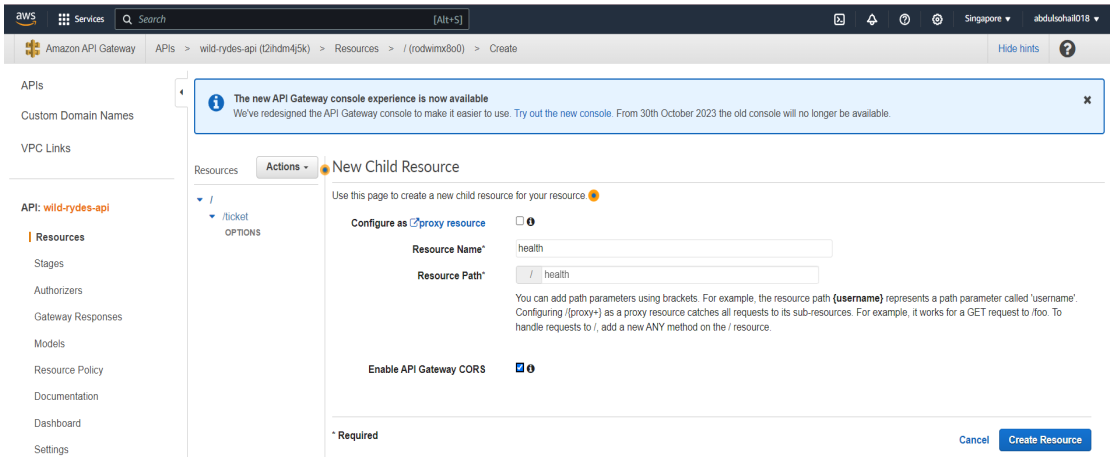


Figure 98: Created a new Resource named “health”

3.14 Creation Of two methods for Ticket Resource

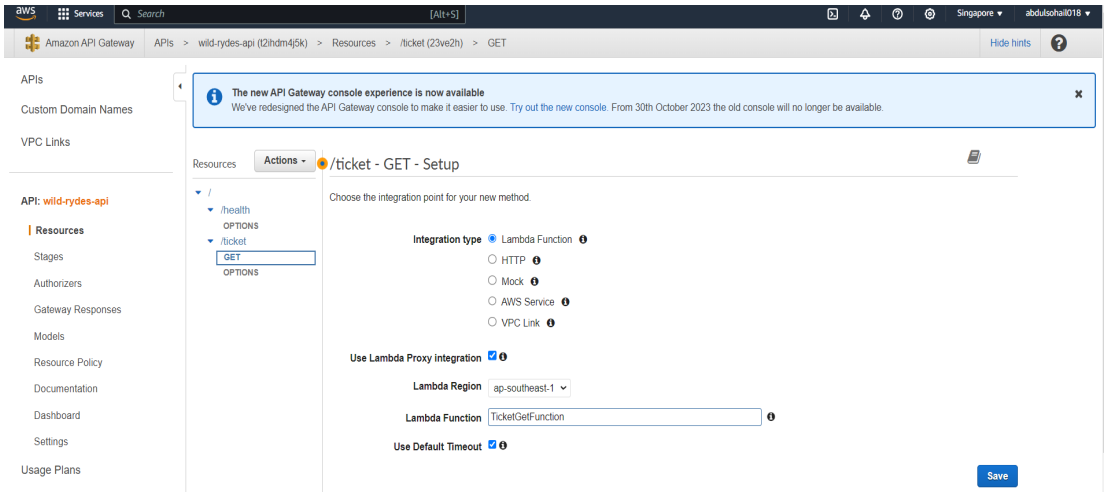


Figure 99: Creating the GET method for the ticket resource

3.15 Creation of Post Method

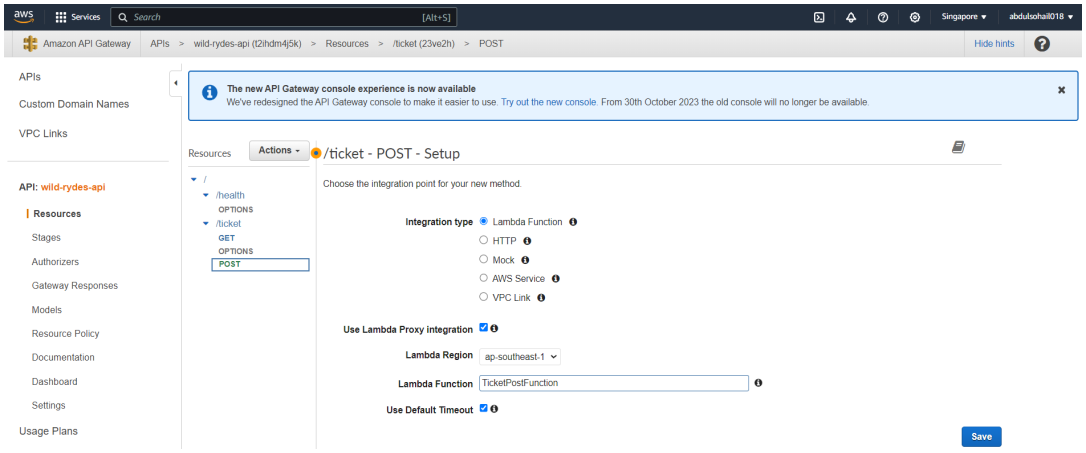


Figure 100: Creating the POST method for the ticket resource

3.16 Creation of GET Method for Health Resource

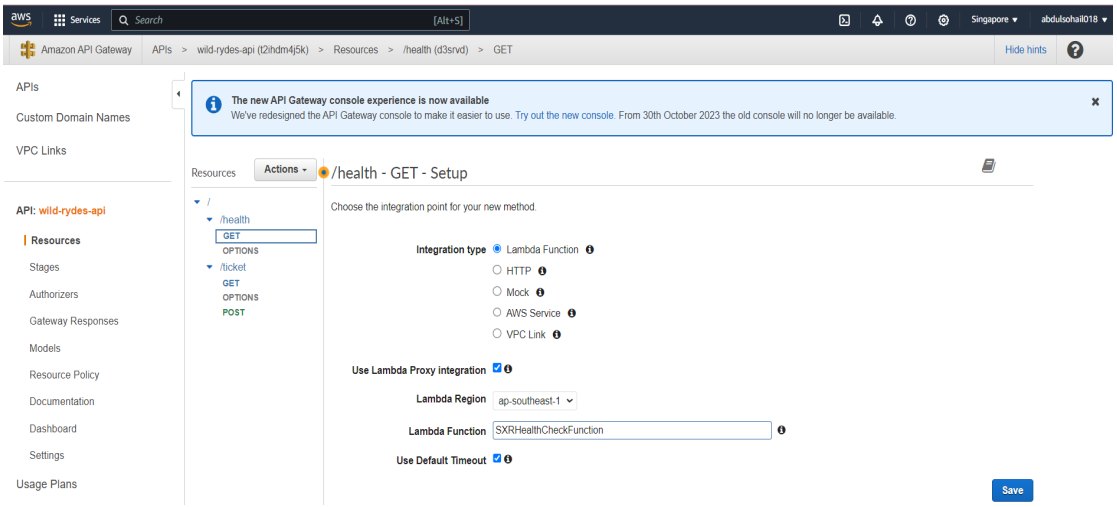


Figure 101: Creating the GET method for the health resource

3.17 Deployment of API

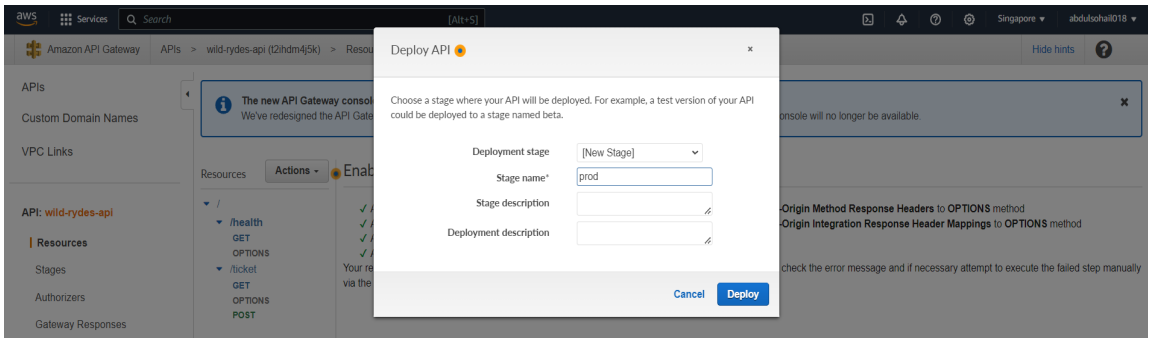


Figure 102: Deploying the API's

3.18 Successful Deployment of API

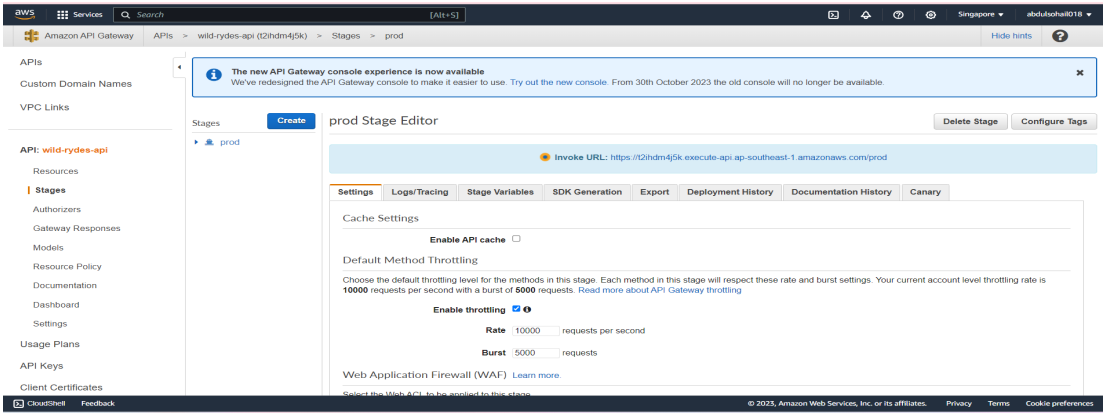


Figure 103: API successfully deployed

3.19 Script for the Ticket Resource

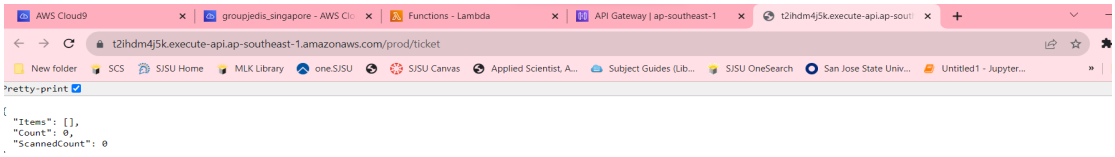


Figure 104: Visualizing the ticket API script

3.20 Script for the Health Resource

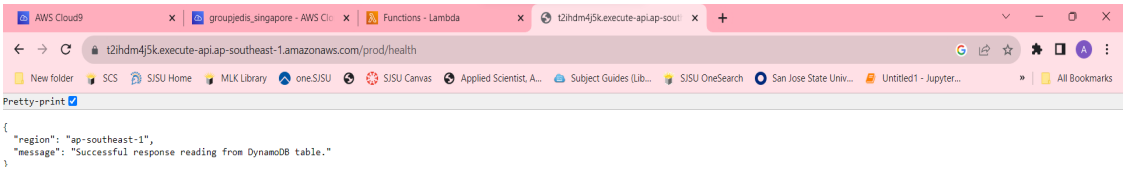


Figure 105: Visualizing the health API script

3.21 Creation of Certificate

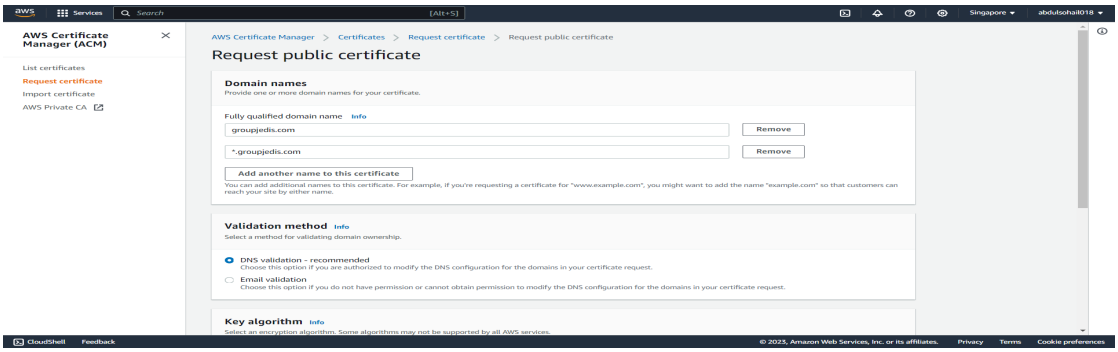


Figure 106: Creating Certificate for Singapore Region

3.22 Created Certificate for the Region

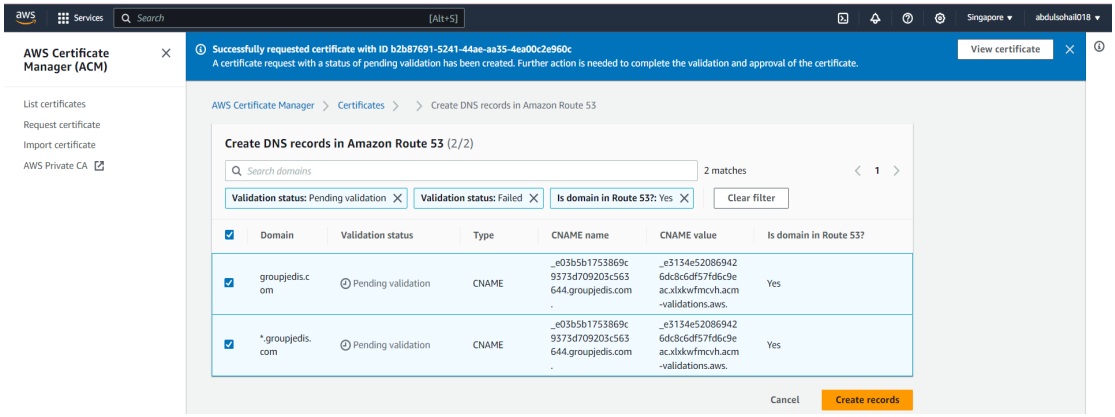


Figure 107: Visualizing certificates created for the Singapore Region

3.23 Certification Status

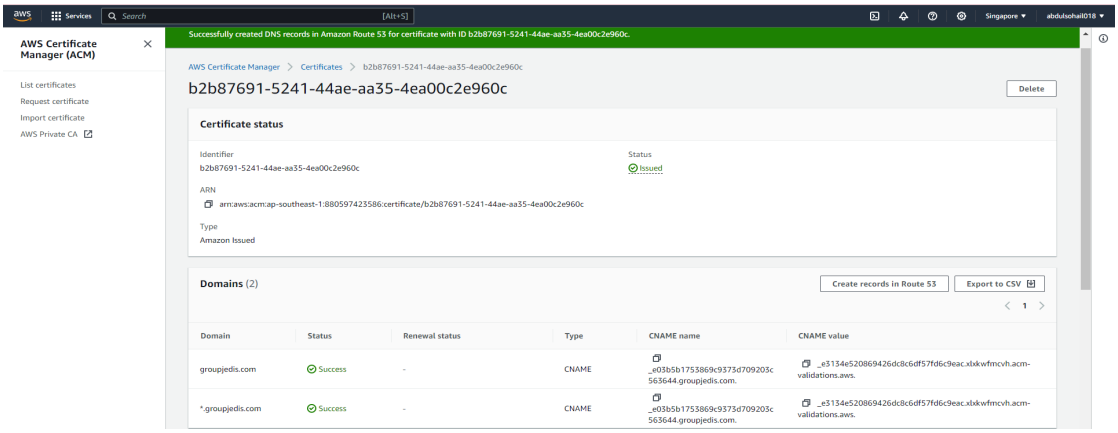


Figure 108: Certificate successfully created

3.24 List of API

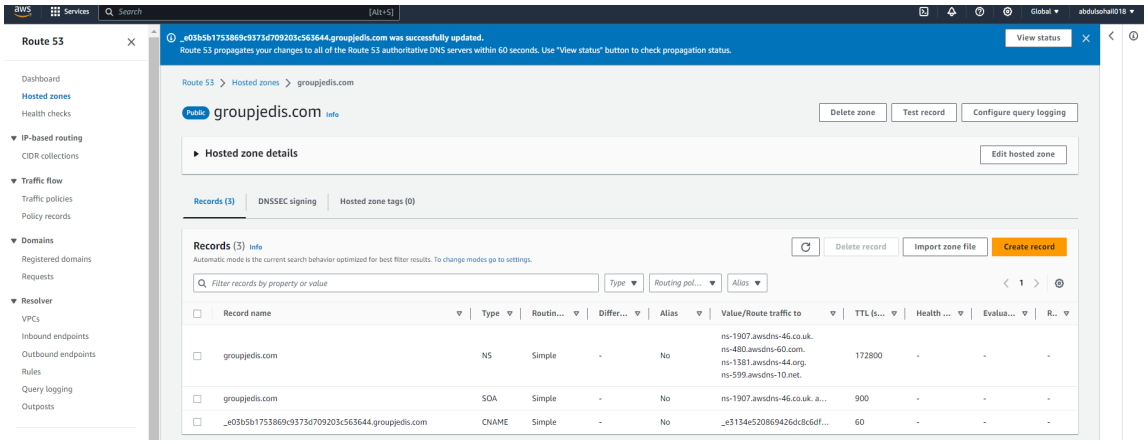


Figure 109: Visualizing the API

3.25 Creation of Custom Domains

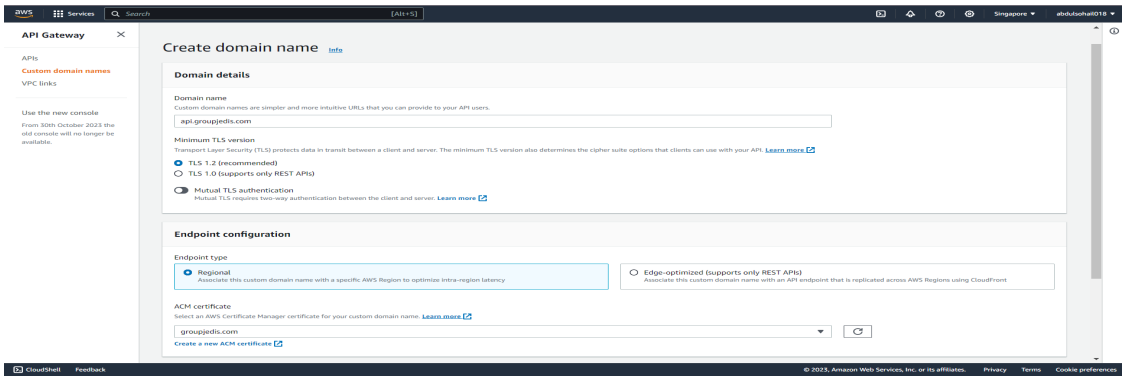


Figure 110: Creating DNS for Singapore API UI

3.26 Configuration Settings for the Domains

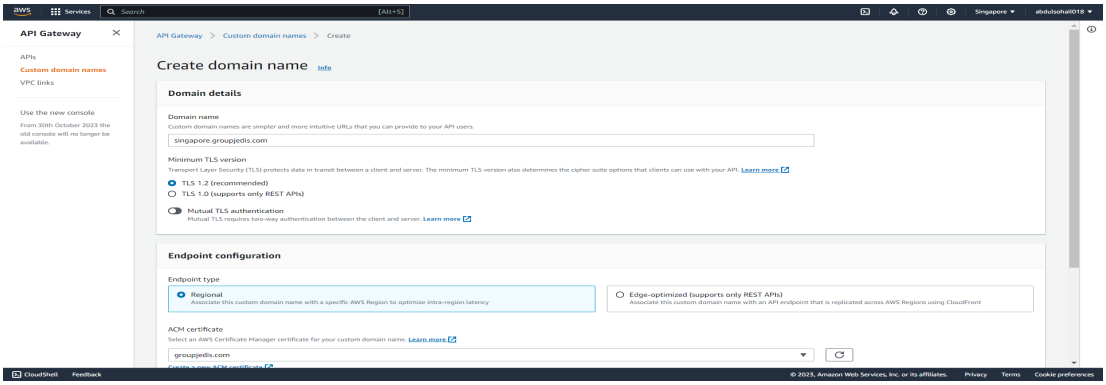


Figure 111: Creating DNS to access Singapore health check API

3.27 Creation of DNS Record

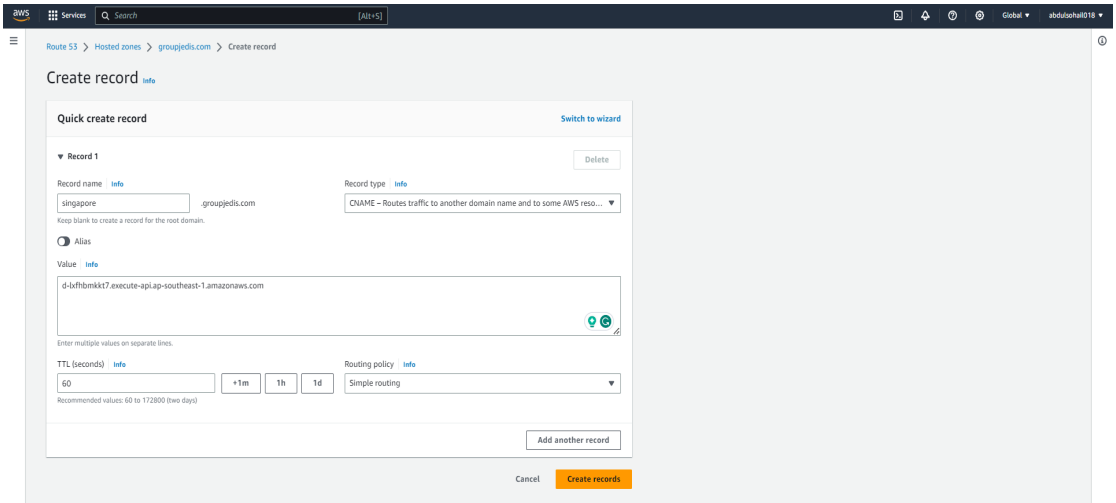


Figure 112: Creating DNS record for Singapore API

3.28 Creation of DNS Record

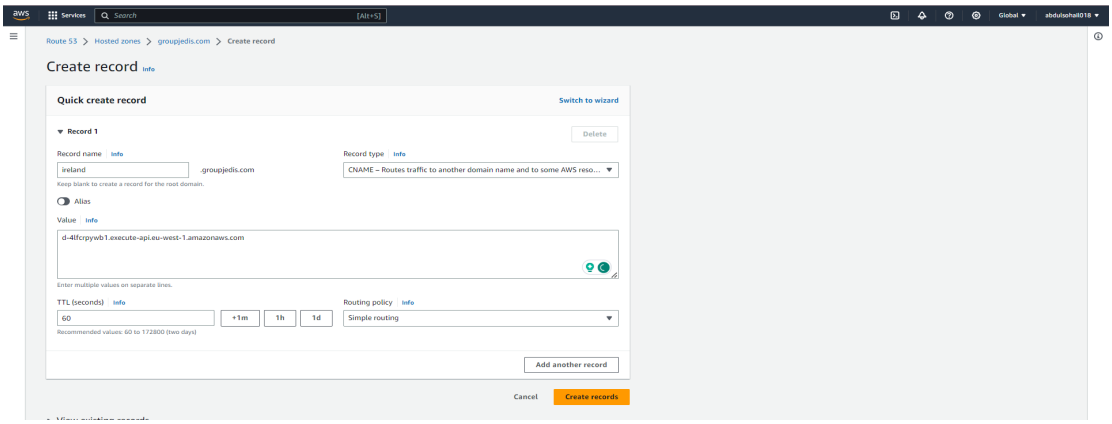


Figure 113: Creating DNS record for Ireland API

3.29 List of all the DNS Records

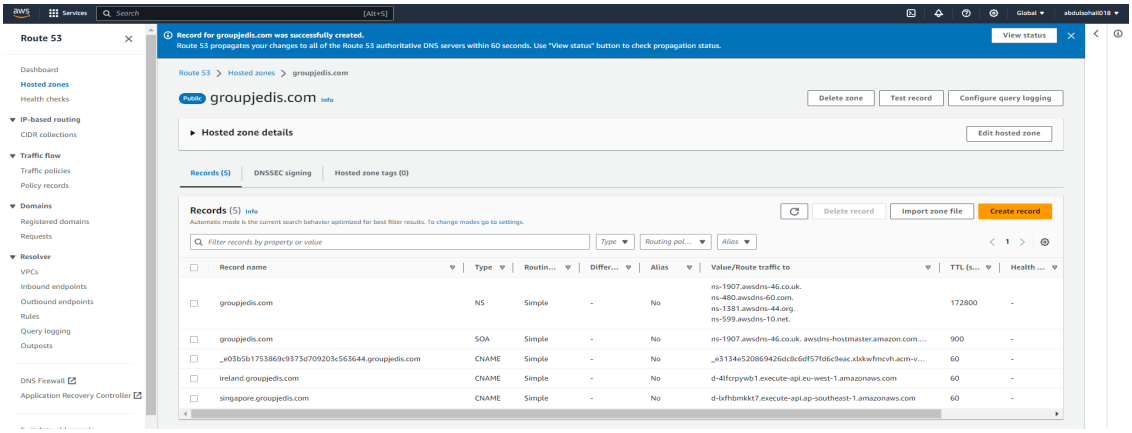


Figure 114: Visualizing Records in Hosted Zones

3.30 Creation of Health Check

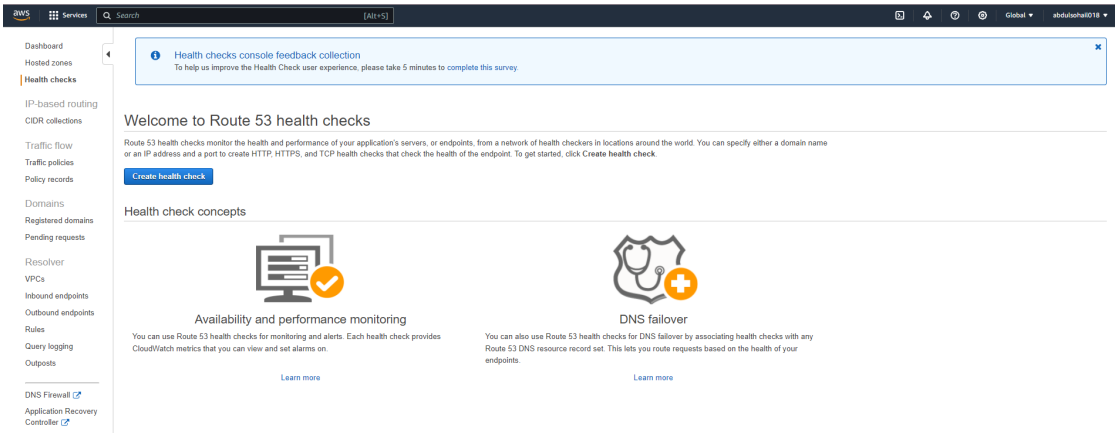


Figure 115: Creating Health checks

3.31 Creation of Health Check for Ireland Region

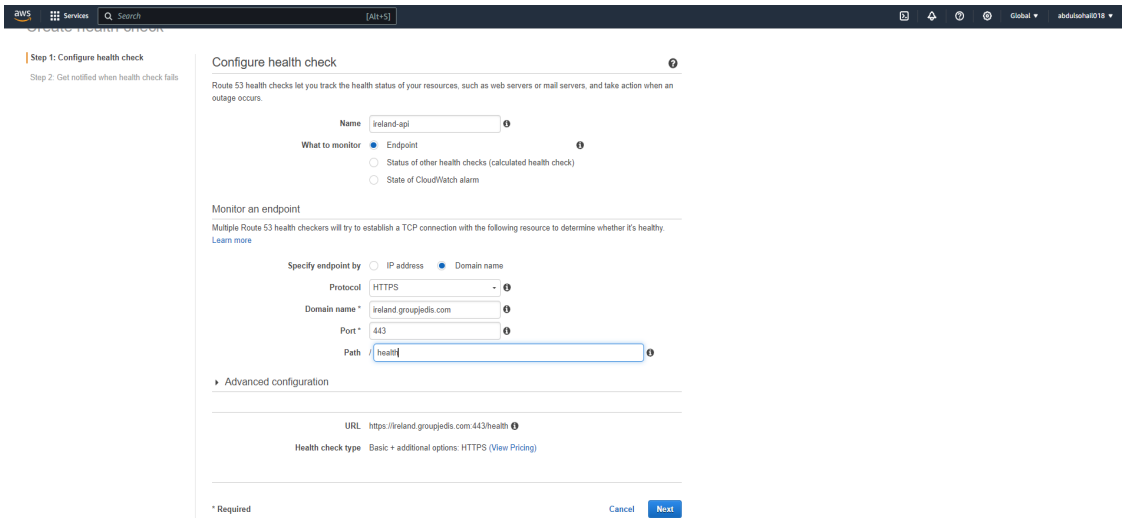


Figure 116: Creating Health Check for Ireland region with all the Configuration Settings

3.32 Advance Configuration Settings

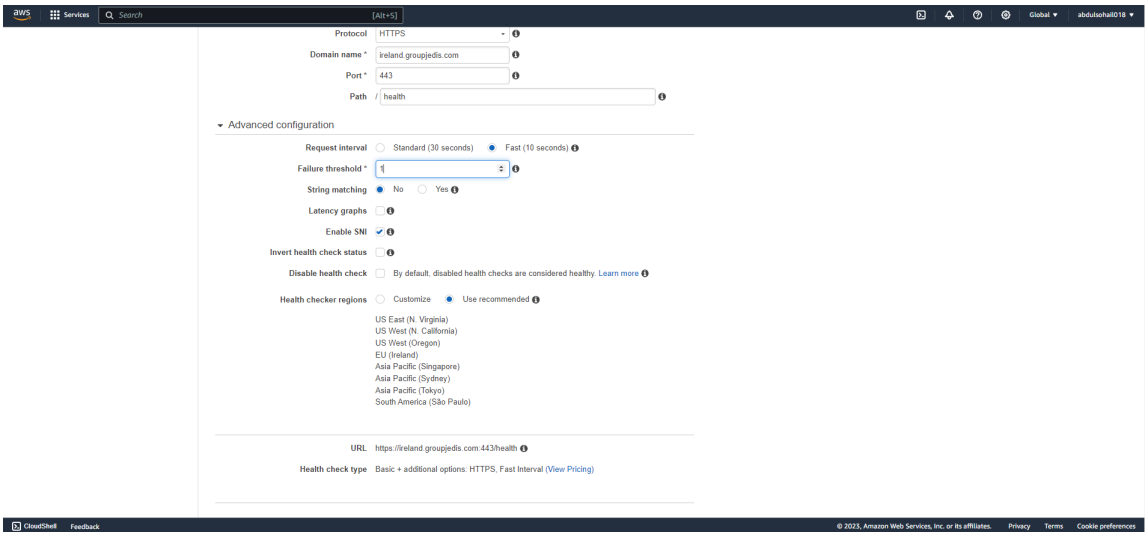


Figure 117: Modifying advanced configuration of Ireland Health Check API

3.33 Setting the Alarm Configuration

Figure 118: Selecting not to create alarm in Ireland Health Check

3.34 Creation of Health check for Singapore Region

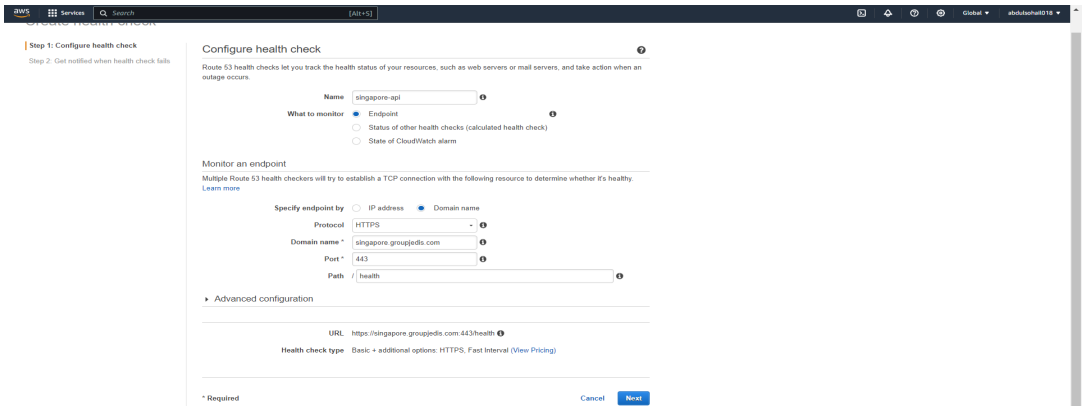


Figure 119: Creating Health Check for Singapore region

3.35 Results for Health Checks

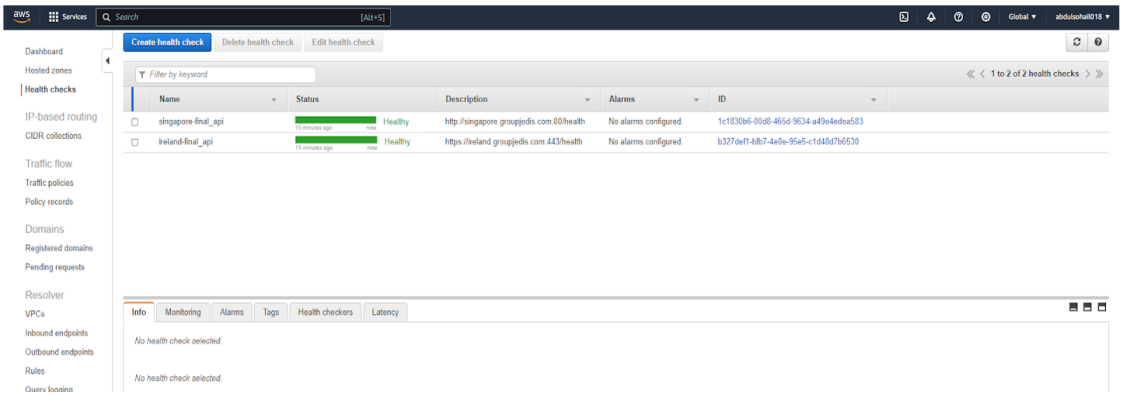


Figure 120: Health Check API is successfully created

3.36 Records for the Ireland Health checks

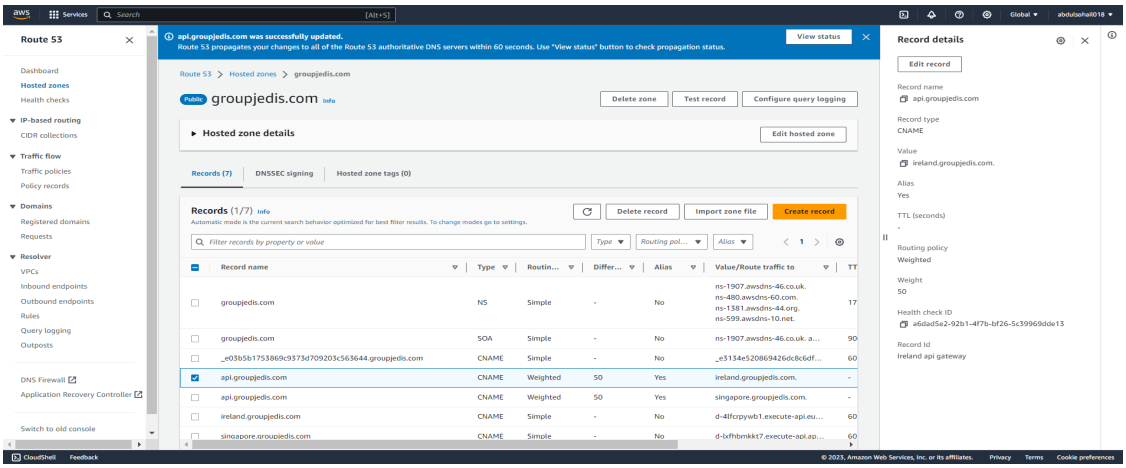


Figure 121: Creating and visualizing domain record for Ireland Health Check API

3.37 Creation of DNS Record For Health Check

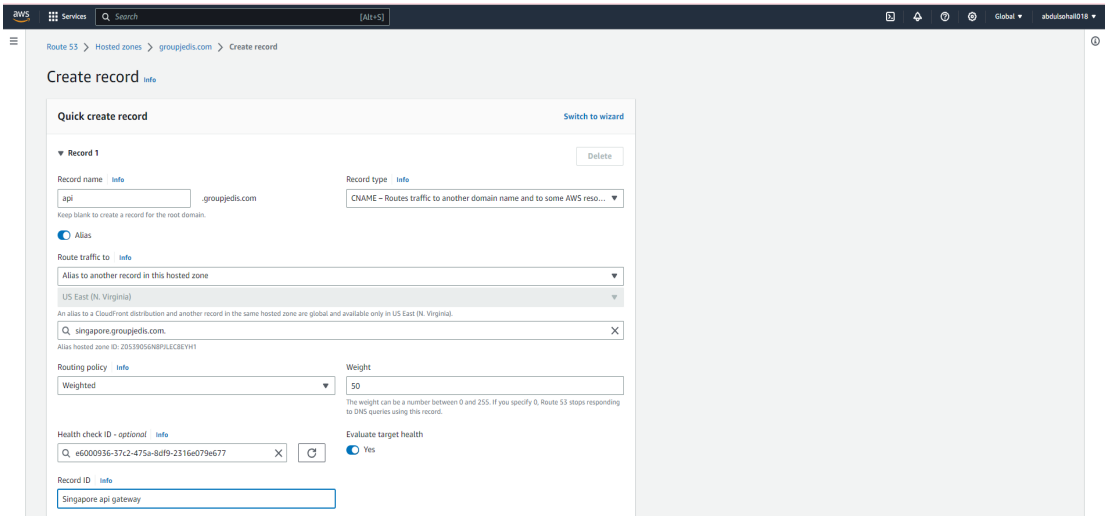


Figure 122: Creating domain record for Singapore Health Check API

3.38 Output Generated

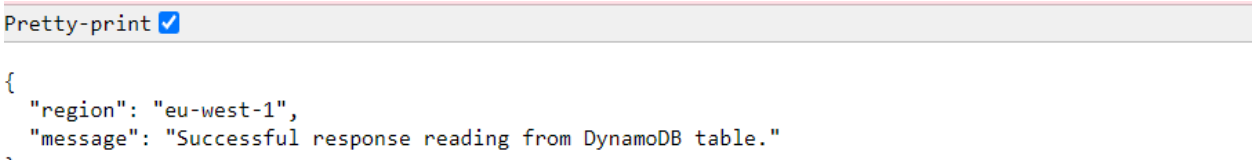


Figure 123: Visualizing record script

3.39 Code for Updation of UI

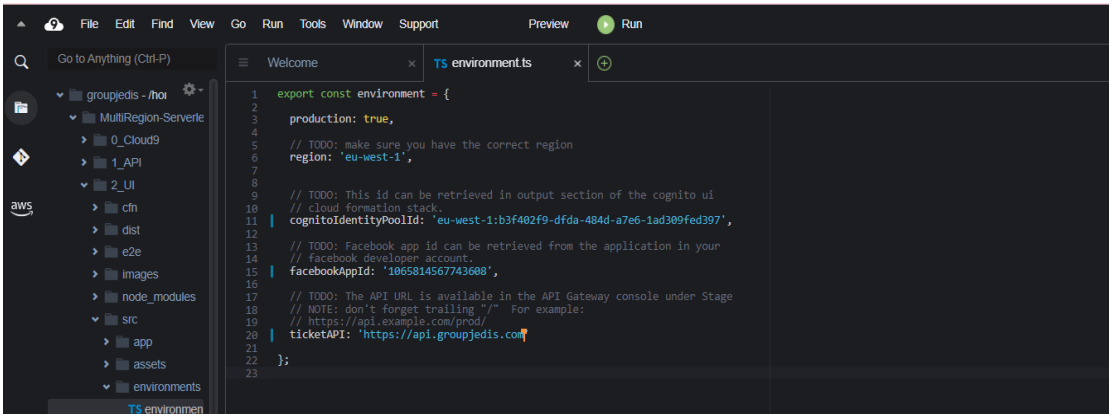


Figure 124: Updating UI by adding API endpoints

3.40 Completion of Built

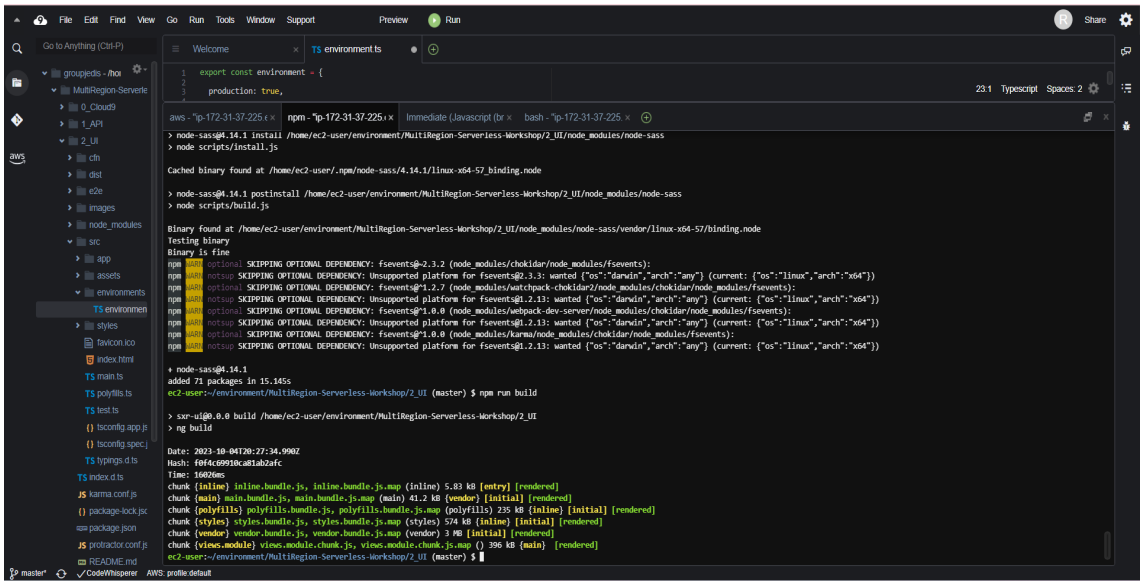


Figure 125: Build successful

Step 5: <u>Test Failover:</u>

1. Breaking the Primary Region

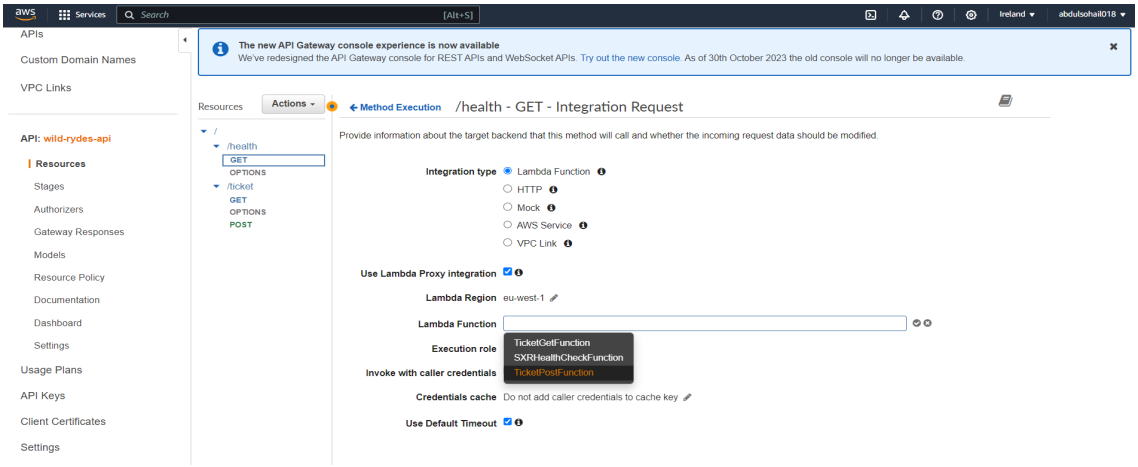


Figure 127: Updated the Lambda Function in Integration Request to check Test Failover

2. Result of the Test Failover

Figure 128: Break successfully done

3. Code to get the Response for the Read Operation

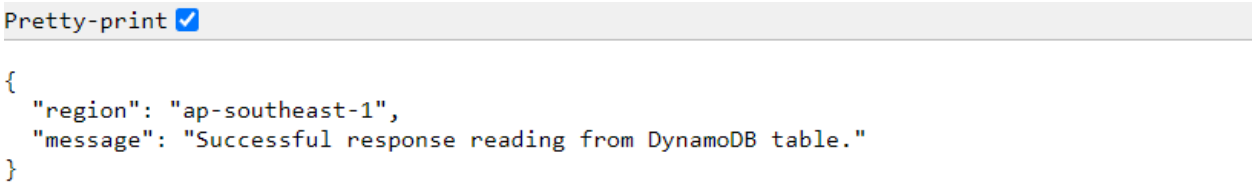


Figure 129: Visualizing script of successful read operation

Link to Source Code repo:

Below mentioned is the path to the GitHub repository for the code:

https://github.com/poojan243/AWS-MultiRegion-Serverless-Workshop