

## DATA 228: Big Data Technologies and Applications

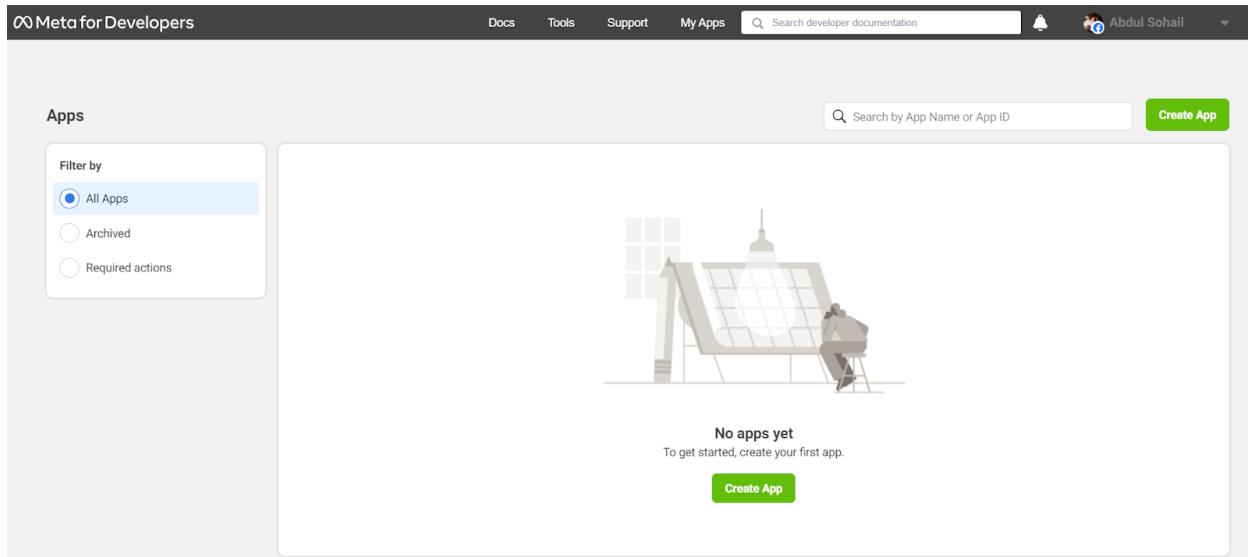
### Homework 4: Multi-Region Application

#### Group Members (Jedi's):

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

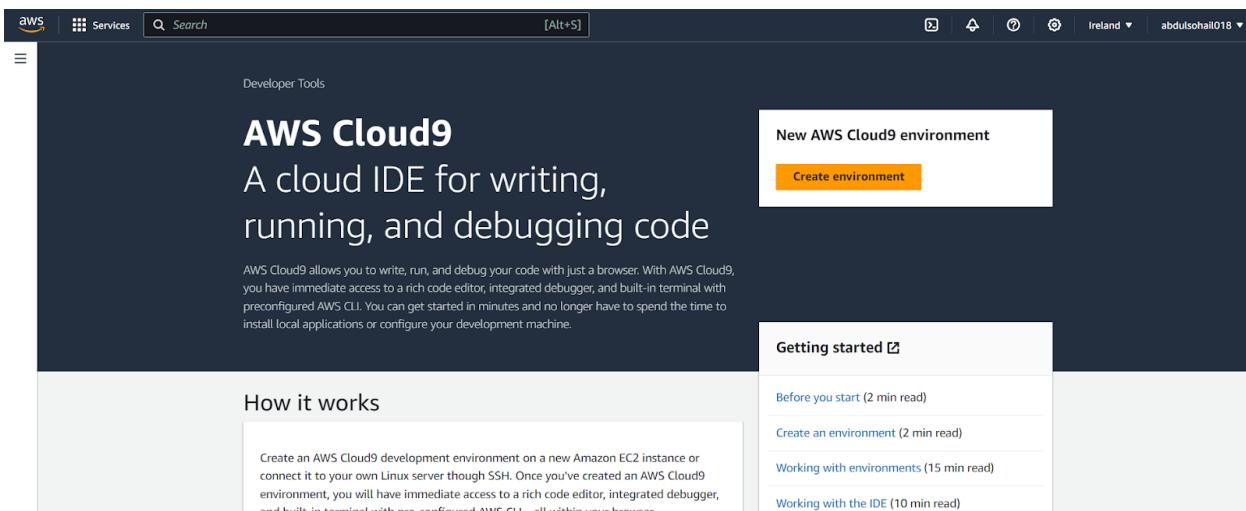
#### Prerequisites:

##### **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:

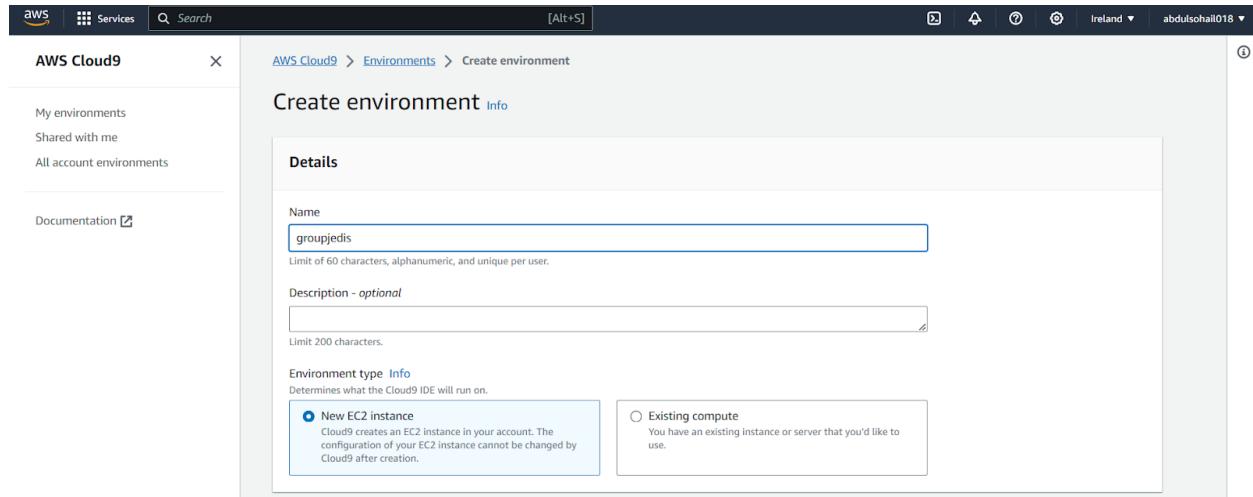


Figure 2: Providing the name of the Cloud9 environment

## 3. Configuring the settings:

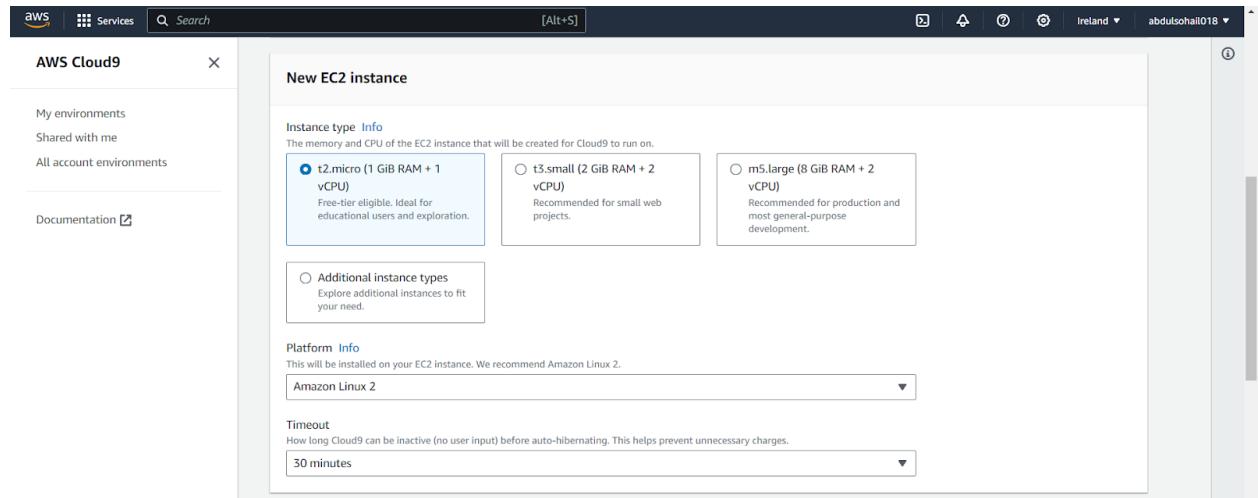


Figure 3: Selecting EC2 instance, platform installed, and timeout

## 4. Created Cloud9 environment:

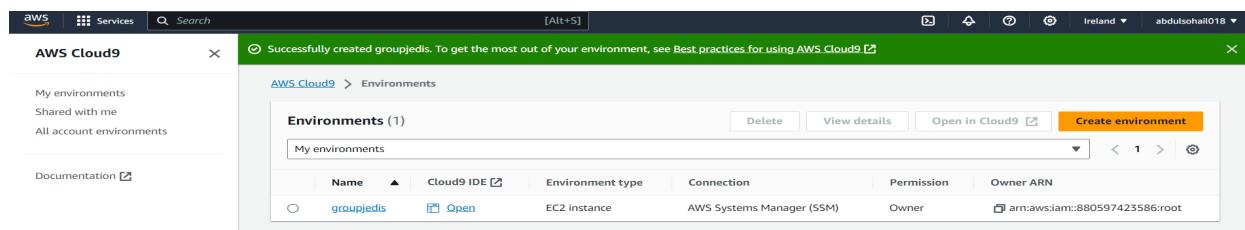
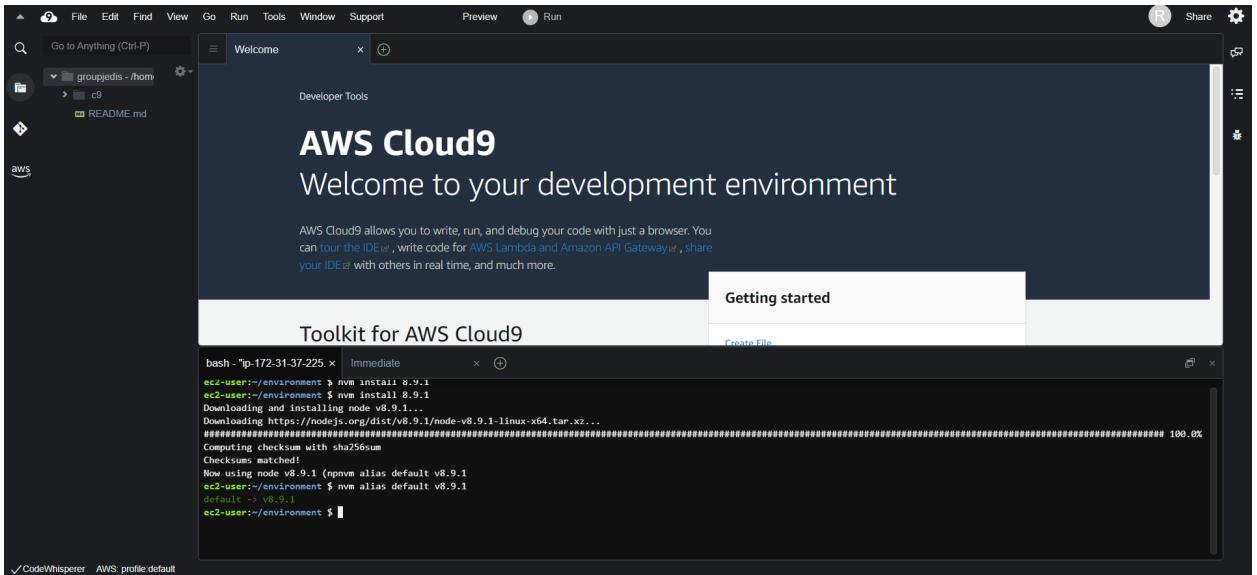


Figure 4: Successfully able to create the Cloud9 environment

## 5. Installed packages:

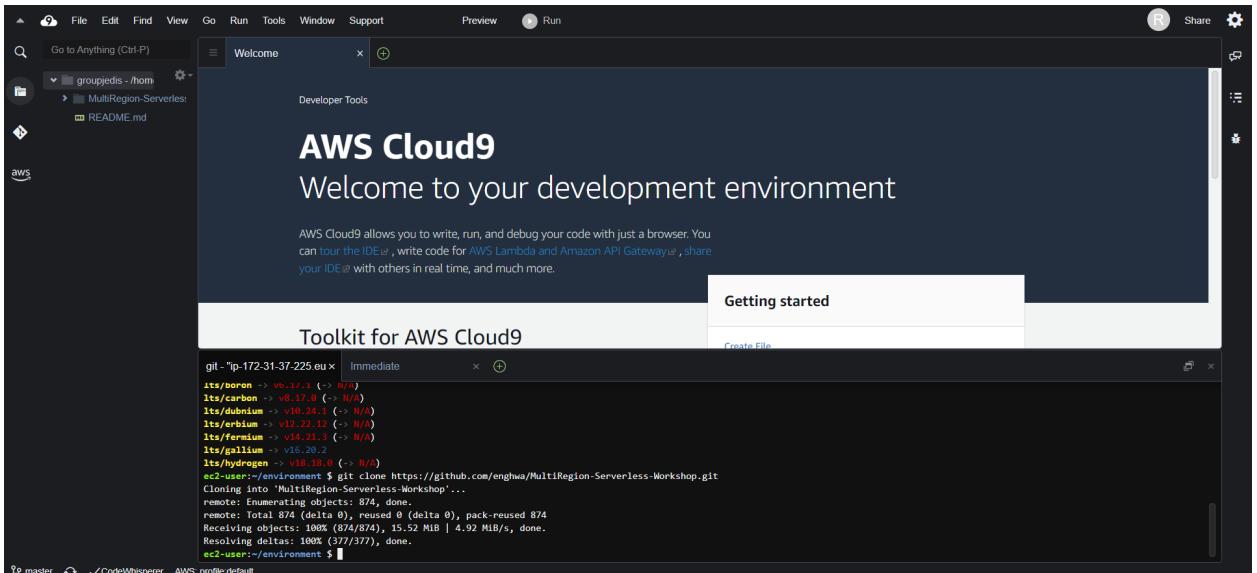


The screenshot shows the AWS Cloud9 developer tools interface. On the left, there's a file explorer with a folder named 'groupjedis - /home/groupjedis' containing files 'c9' and 'README.md'. The main area displays the 'Welcome' screen for AWS Cloud9, which includes a 'Getting started' section and a 'Toolkit for AWS Cloud9' terminal window. The terminal window shows the command 'nvm install 8.9.1' being run, followed by the output of the Node.js download and installation process. The output indicates that Node.js v8.9.1 has been successfully installed.

```
bash - "ip-172-31-37-225.x"  Immediate
ec2-user:~/environment $ nvm install 8.9.1
ec2-user:~/environment $ nvm install 8.9.1
Downloading and installing node v8.9.1...
Downloaded https://nodejs.org/dist/v8.9.1/node-v8.9.1-linux-x64.tar.xz...
#####
Computing checksum with sha256sum
Checksums matched!
Now using node v8.9.1 (npmv alias default v8.9.1
ec2-user:~/environment $ nvm alias default v8.9.1
default -> v8.9.1
ec2-user:~/environment $
```

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

## 6. Cloning git repository:



The screenshot shows the AWS Cloud9 developer tools interface. On the left, there's a file explorer with a folder named 'groupjedis - /home/groupjedis' containing files 'MultiRegion-Serverless' and 'README.md'. The main area displays the 'Welcome' screen for AWS Cloud9. The terminal window at the bottom shows the command 'git clone https://github.com/engwha/MultiRegion-Serverless-Workshop.git' being run. The output shows the cloning process, including the enumeration of objects from the remote repository and the receiving of objects into the local repository.

```
git - "ip-172-31-37-225.eu"  Immediate
its/poron --> v0.11.1 (-> N/A)
its/carbon --> v0.17.0 (-> N/A)
its/dubnium --> v0.9.24.1 (-> N/A)
its/erbium --> v12.22.12 (-> N/A)
its/fermium --> v14.21.3 (-> N/A)
its/gallium --> v16.20.2
its/hydrogen --> v18.18.0 (-> N/A)
ec2-user:~/environment $ git clone https://github.com/engwha/MultiRegion-Serverless-Workshop.git
Cloning into 'MultiRegion-Serverless-Workshop'...
remote: Enumerating objects: 874, done.
remote: Total 874 (delta 0), reused 0 (delta 0), pack-reused 874
Receiving objects: 100% (874/874), 15.52 MB | 4.92 MB/s, done.
Resolving deltas: 100% (377/377), done.
ec2-user:~/environment $
```

Figure 6: Cloning the MultiRegion Serverless Workshop git repository

## Step 2: Build an API layer:

### 1. Creating IAM Policy:

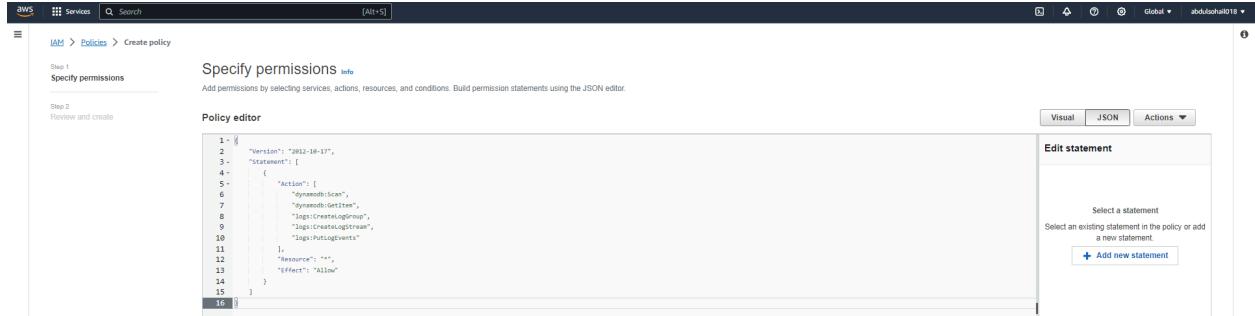


Figure 7: Creating a custom policy in the IAM service

### 2. Created custom policy named TicketGetPolicy:

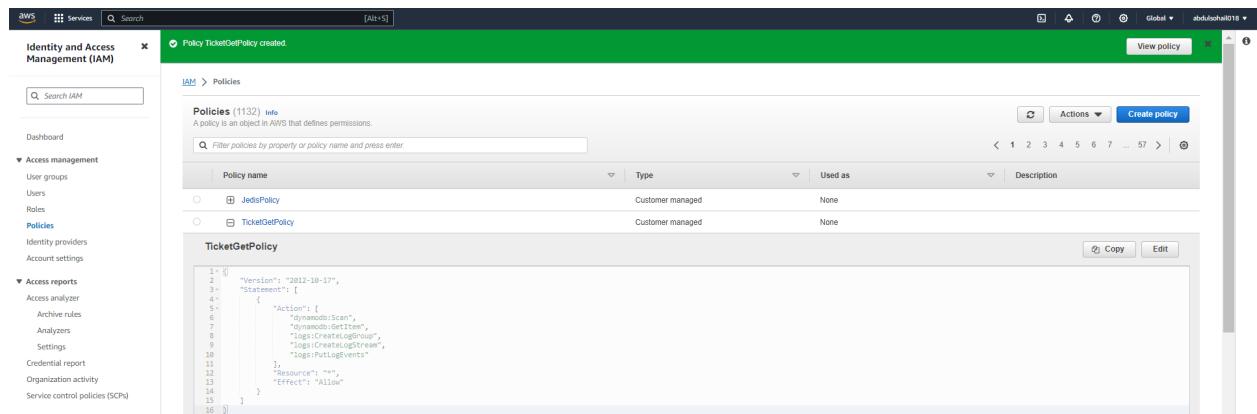


Figure 8: Successfully created custom policy named “TicketGetPolicy”

#### 2.1 Created custom policy named TicketPostPolicy:

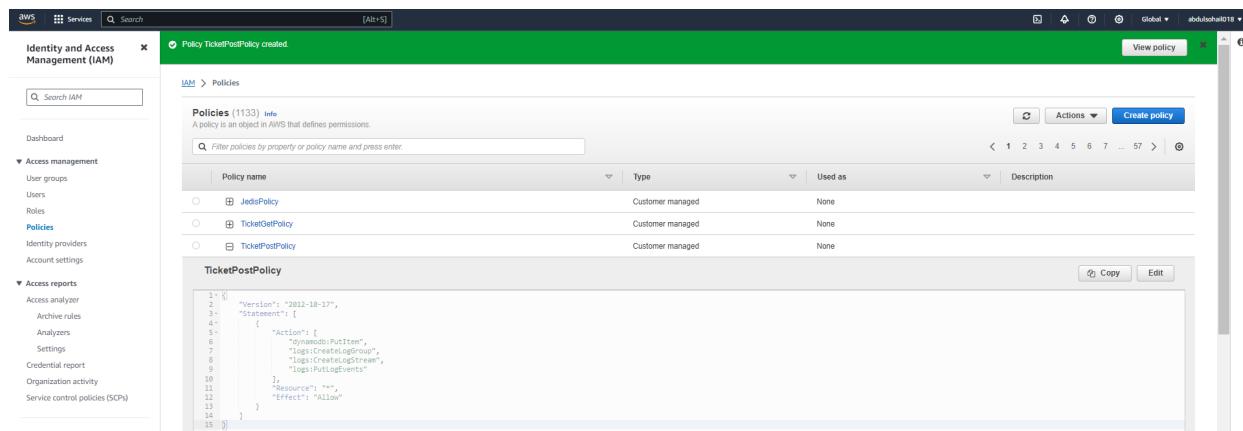
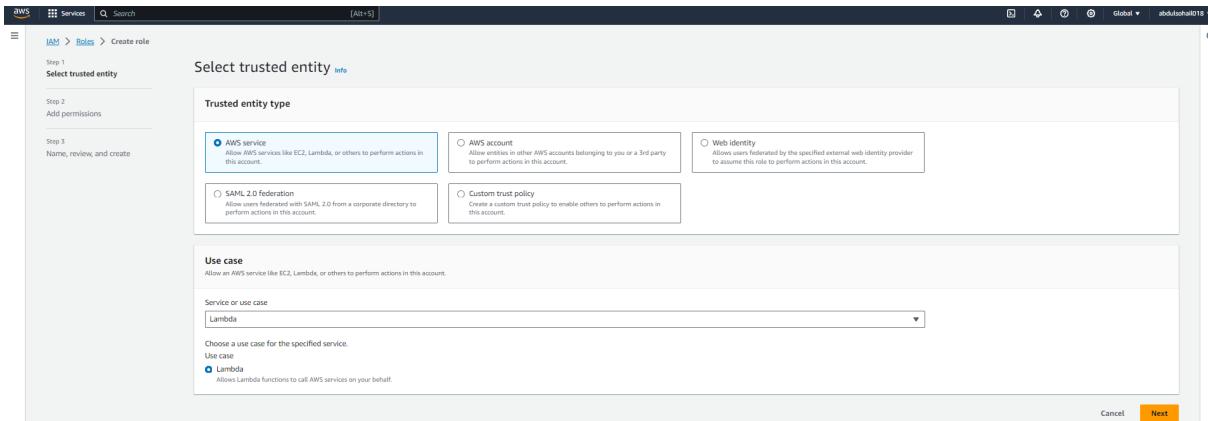


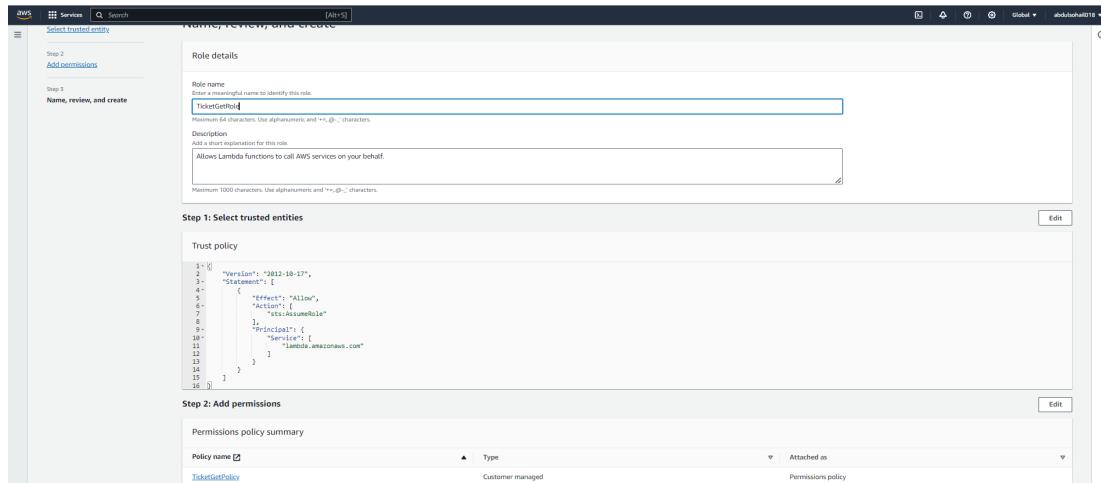
Figure 9: Successfully created custom policy named “TicketPostPolicy”

### 3. Creating roles for the custom policies:



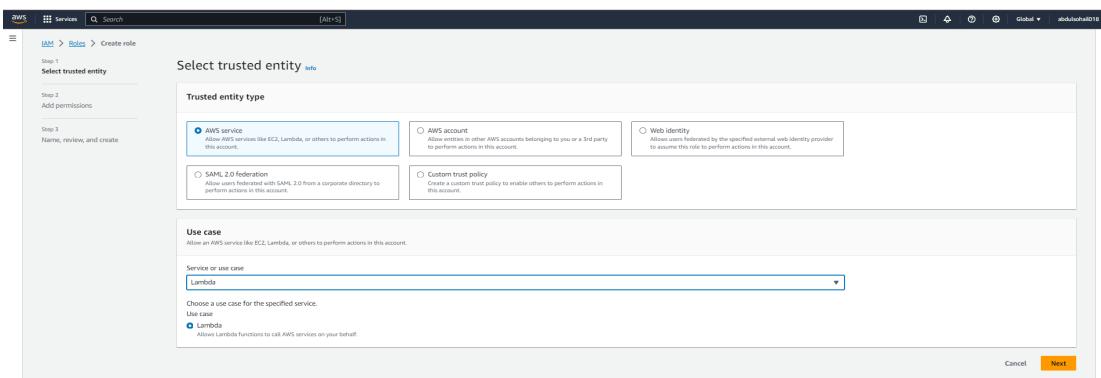
**Figure 10:** Create an IAM role for the policy “TicketGetPolicy” with the use case “Lambda”

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



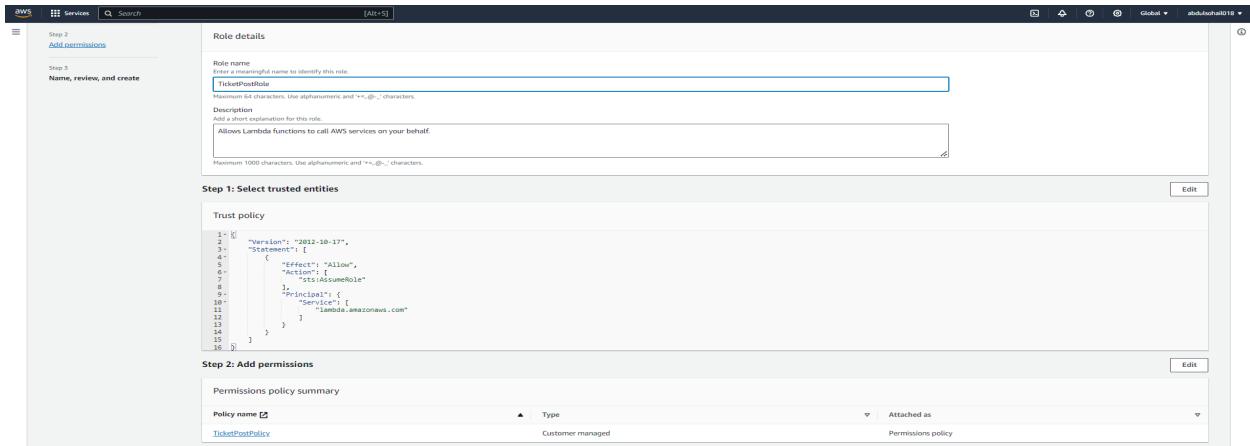
**Figure 11:** Creating a role named “TicketGetRole”

#### 3.2 Selection of Trusted Entity and Use Case



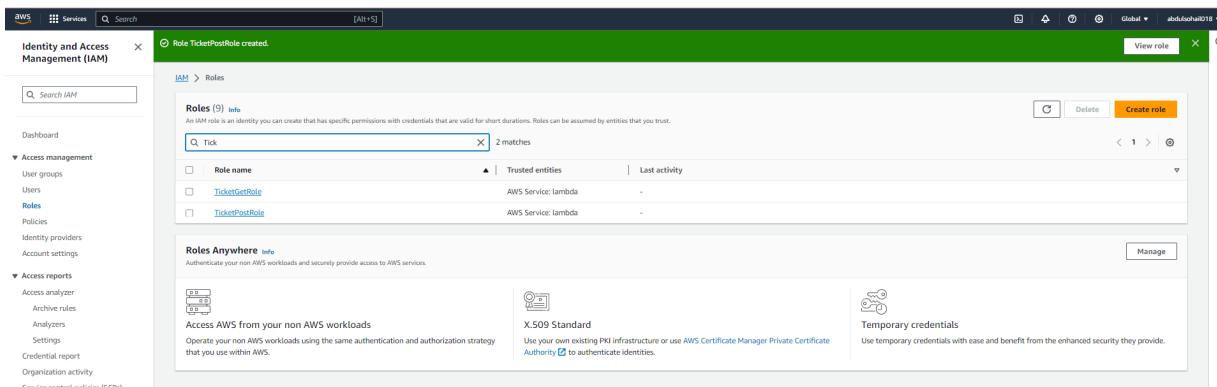
**Figure 12:** Selecting the trusted entity as AWS Service and use case as Lambda

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



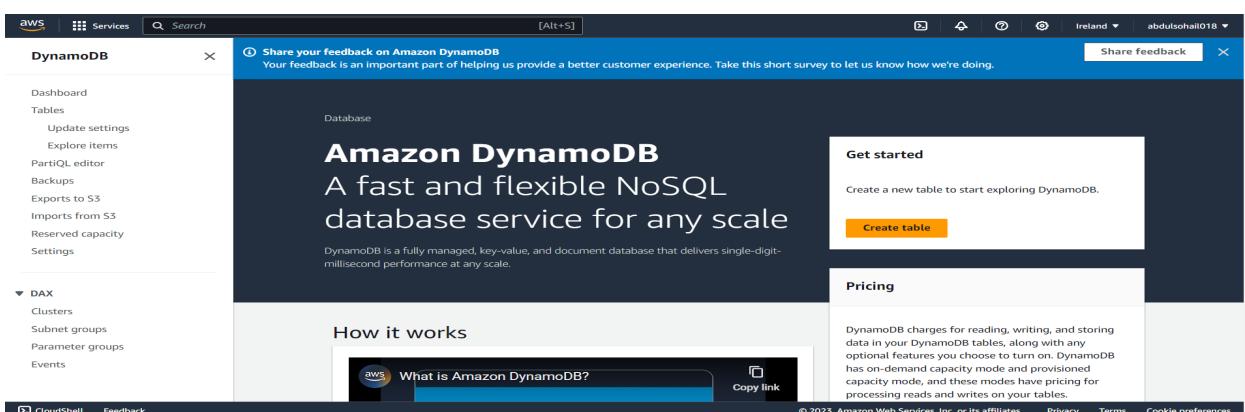
**Figure 13:** Creating a role named “TicketPostRole”

#### **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

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. In the 'Table name' field, 'SXRTickets' is entered. Under 'Partition key', 'id' is selected as the primary partition key, and under 'Sort key - optional', an empty field is shown.

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

## 5.2 Configuring the Table Settings

The screenshot shows the 'Default settings' configuration page for the newly created table. The 'Default settings' tab is selected. It displays the default table settings, which are all set to 'Yes' for 'Editable after creation'. These include 'Table class' (DynamoDB Standard), 'Capacity mode' (Provisioned), 'Provisioned read capacity' (5 RCU), 'Provisioned write capacity' (5 WCU), 'Auto scaling' (On), 'Local secondary indexes' (-), 'Global secondary indexes' (-), 'Encryption key management' (Owned by Amazon DynamoDB), and 'Deletion protection' (Off).

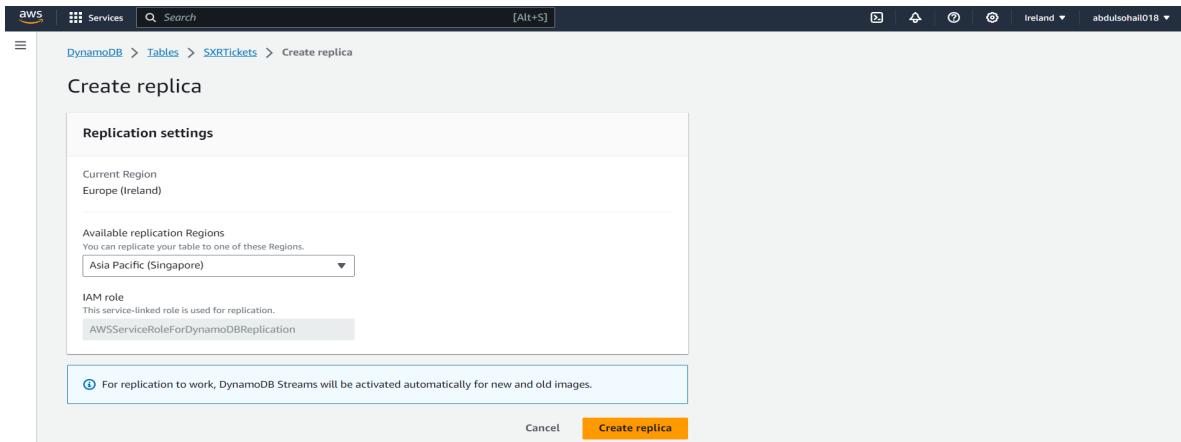
**Figure 17:** Keeping the table settings as “Default”

## 6. Created DynamoDB table:

The screenshot shows the 'Tables' list in the AWS DynamoDB console. A green banner at the top indicates that the 'SXRTickets' table was created successfully. The table list shows one item: 'SXRTickets' (Name, Active Status, Partition key 'id (\$)', Sort key '-', Indexes 0, Deletion protection Off, Read capacity mode Provisioned with auto scaling (5), Write capacity mode Provisioned with auto scaling (5)).

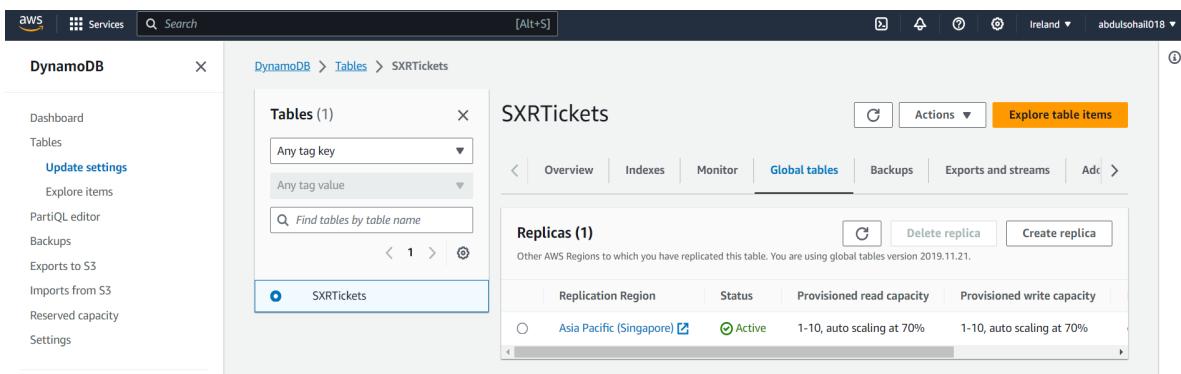
**Figure 18:** Successfully created “SXRTickets” DynamoDB table

## 7. Creating a replica of the DynamoDB table:



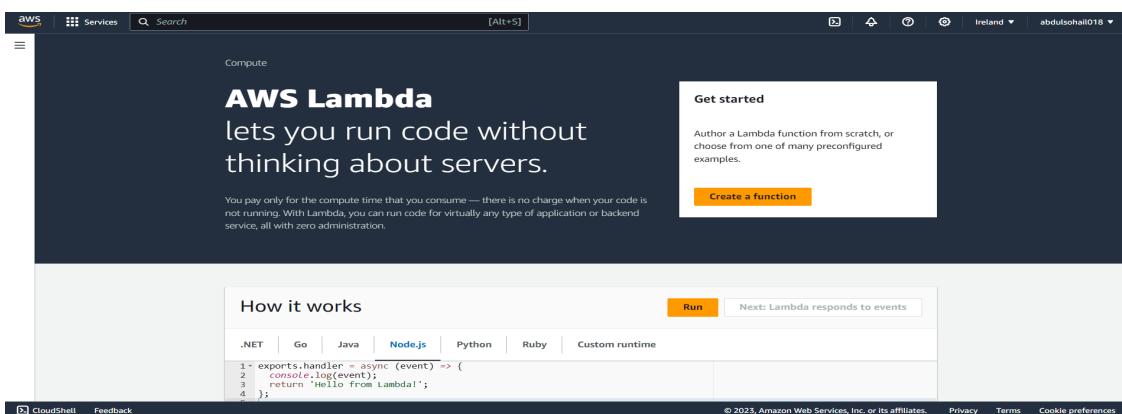
**Figure 19:** Creating a replica of the original table created in “Europe(Ireland)” in “Asia Pacific (Singapore)”

## 8. Created a replica of the DynamoDB table:



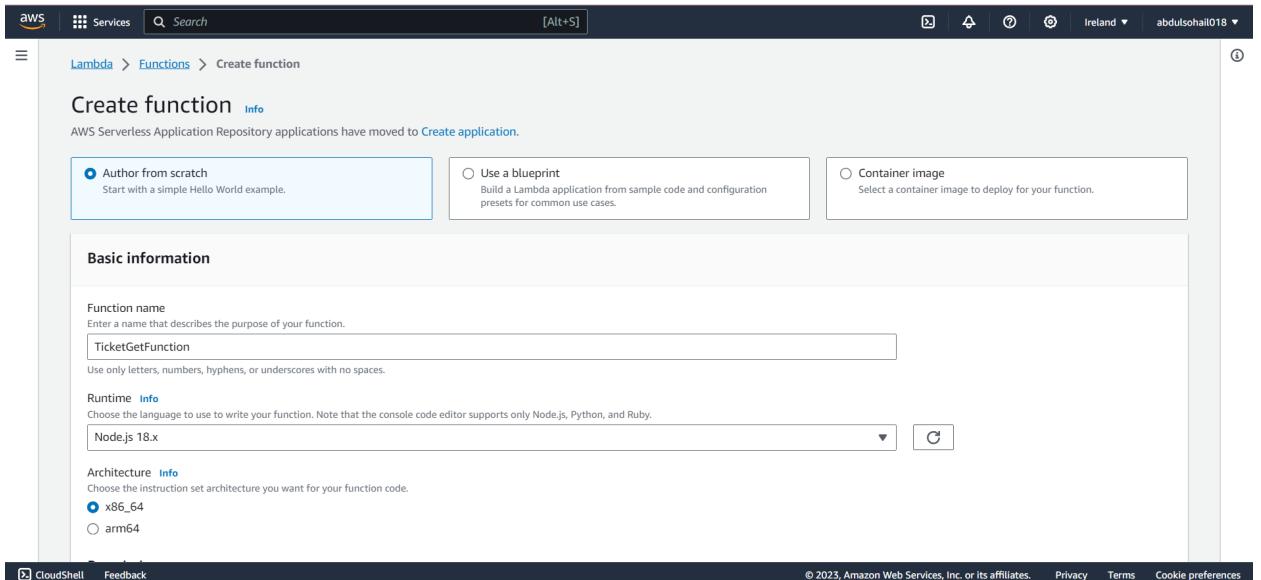
**Figure 20:** Successfully created the replica of the DynamoDB table in “Asia Pacific (Singapore)”

## 9. Creating lambda functions:



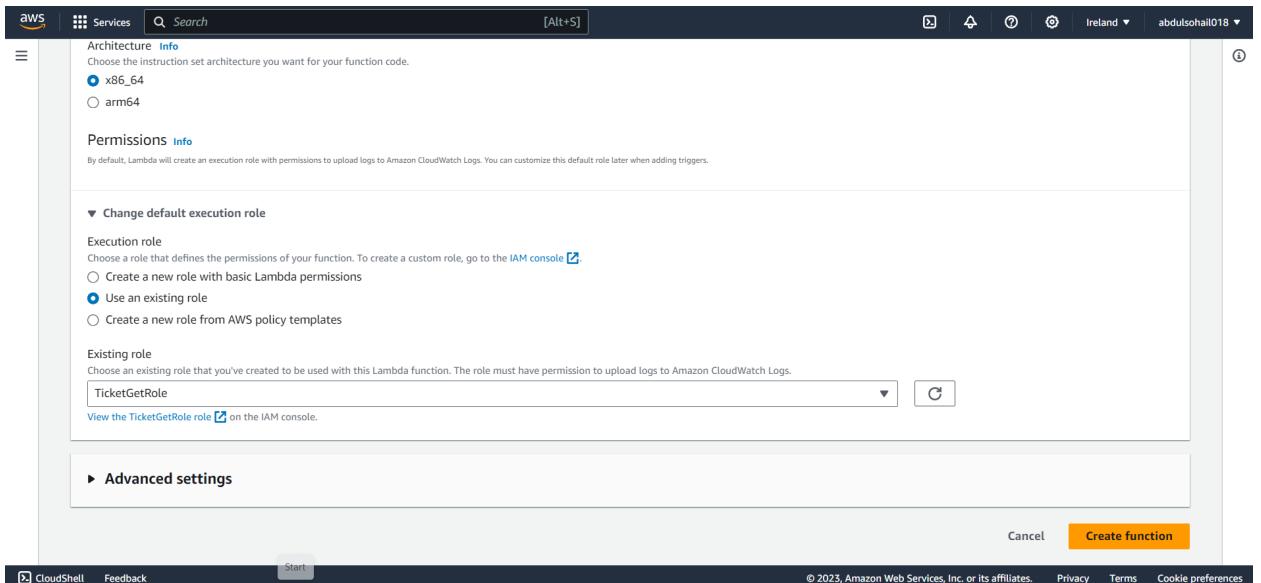
**Figure 21:** AWS lambda function

## 9.1 Creation of custom Lambda Function



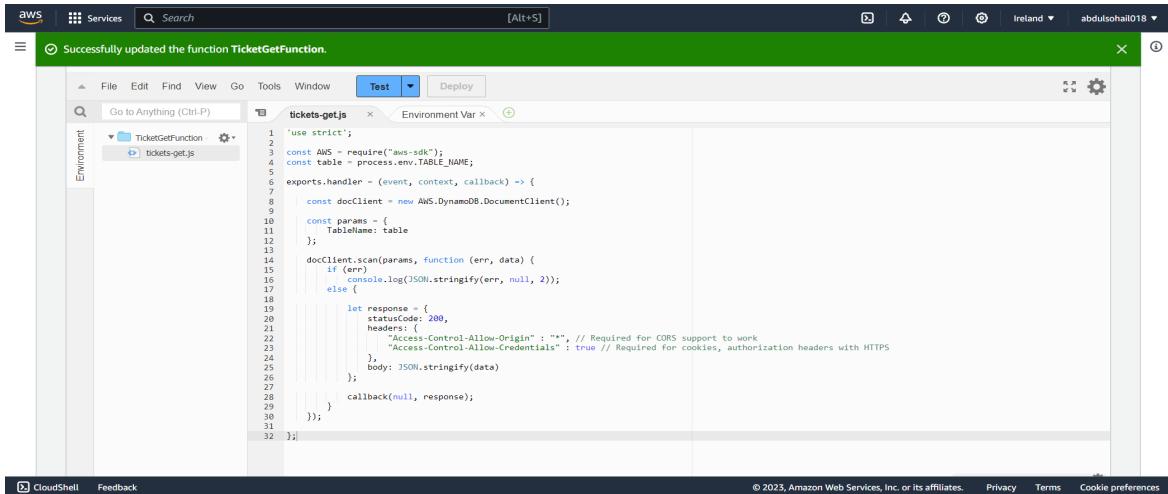
**Figure 22:** Providing function name “TicketGetFunction” and selecting “Node.js 18.x” to write functions

## 9.2 Configuring the Function Settings and Completion of Creation



**Figure 23:** Using the existing role “TicketGetRole” created before

## 10. Modify lambda function:



The screenshot shows the AWS Lambda function editor for the 'TicketGetFunction'. The code editor displays the 'tickets-get.js' file with the following content:

```
'use strict';
const AWS = require("aws-sdk");
const table = process.env.TABLE_NAME;
exports.handler = (event, context, callback) => {
  const docClient = new AWS.DynamoDB.DocumentClient();
  const params = {
    TableName: table
  };
  docClient.scan(params, function (err, data) {
    if (err)
      console.log(JSON.stringify(err, null, 2));
    else {
      let response = {
        statusCode: 200,
        headers: {
          "Access-Control-Allow-Origin": "*",
          "Access-Control-Allow-Credentials": true // Required for cookies, authorization headers with HTTPS
        },
        body: JSON.stringify(data)
      };
      callback(null, response);
    }
  });
};
```

Figure 24: Modifying the “TicketGetFunction” function

### 10.1 Configuring the Runtime Setting

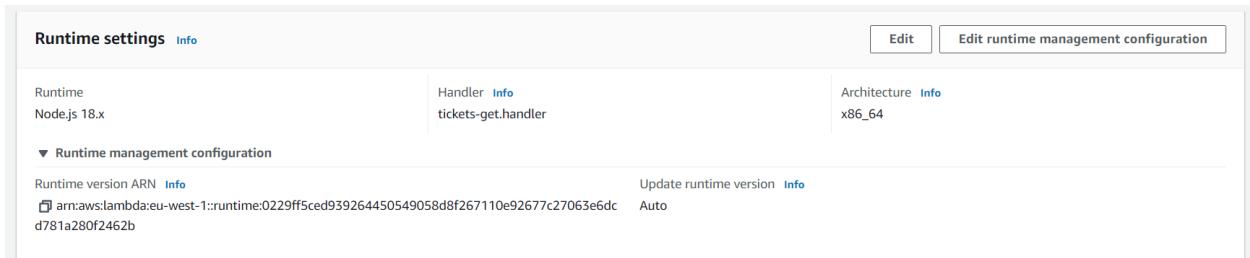


Figure 25: Changed handler to “ticket-get.handler”

### 10.2 Editing the Environment Variables

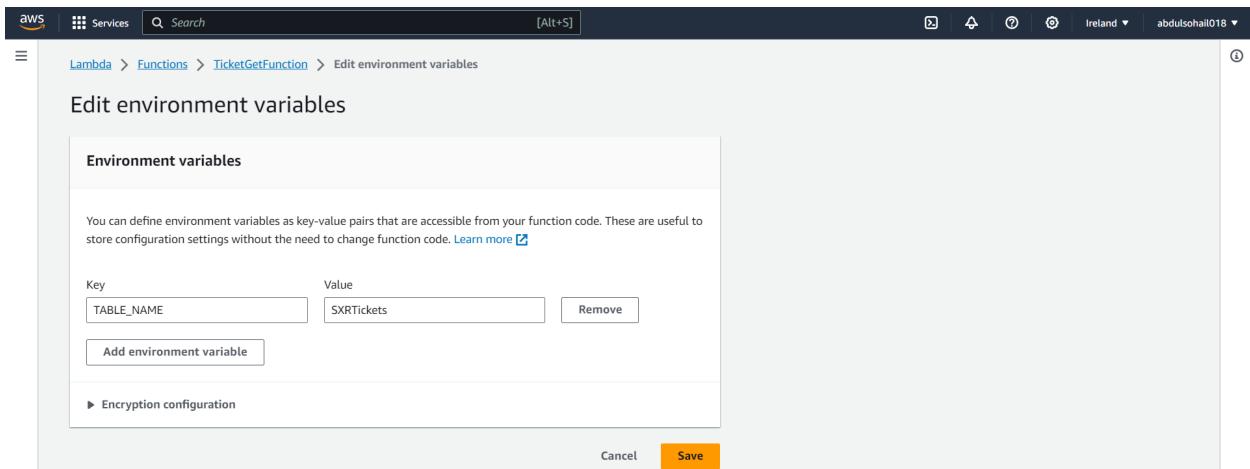


Figure 26: Adding environment variables with key as “TABLE\_NAME” and value as “SXRTickets”

### 10.3 Updation of the Environment Variables

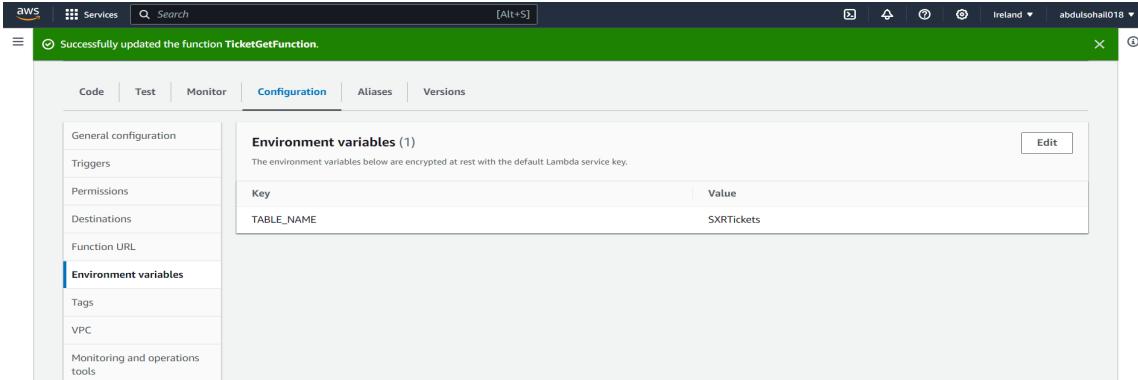


Figure 27: Successfully added environment variables

### 11. Created lambda functions:

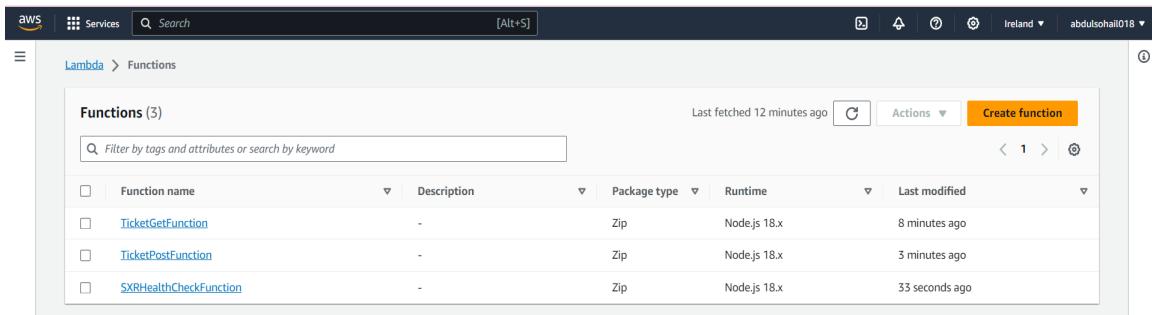


Figure 28: Successfully created all three lambda functions

### 12. Creating API Gateway Endpoint:

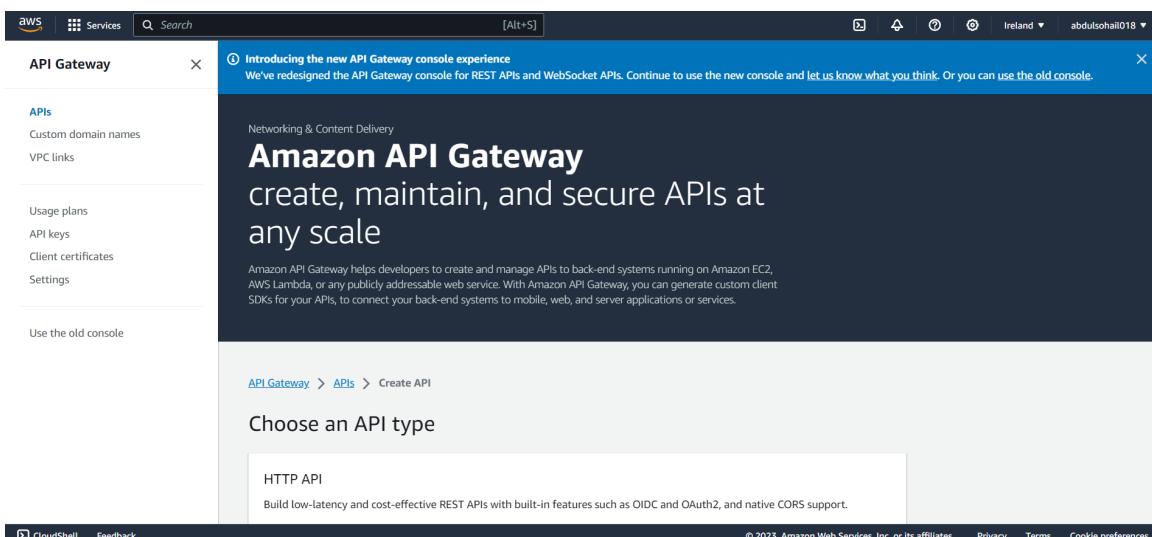


Figure 29: API Gateway Endpoint dashboard

## 12.1 Creation of the new API

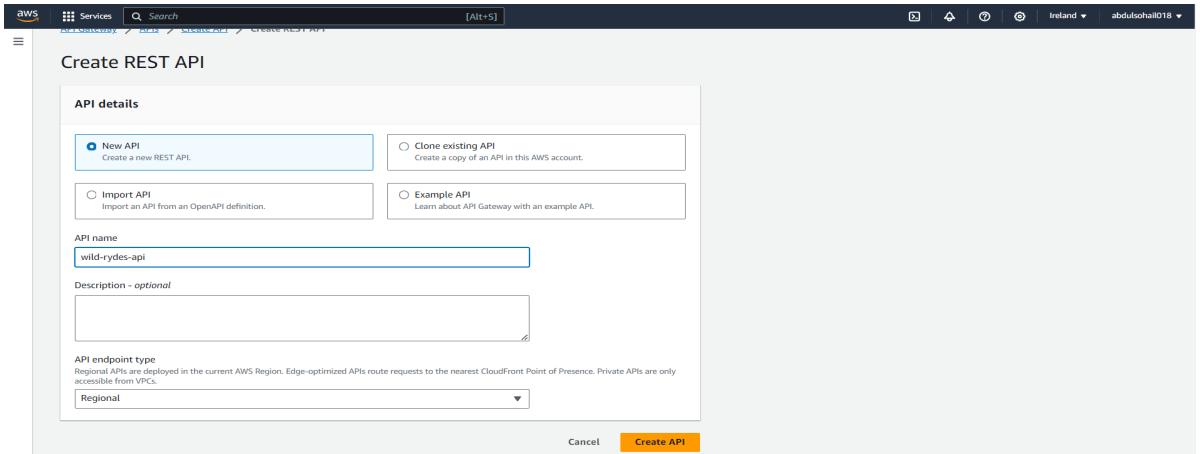


Figure 30: Providing the name of the API as “wild-rydes-api” & selecting the endpoint type as “Regional”

## 13. Created API Gateway Endpoint:

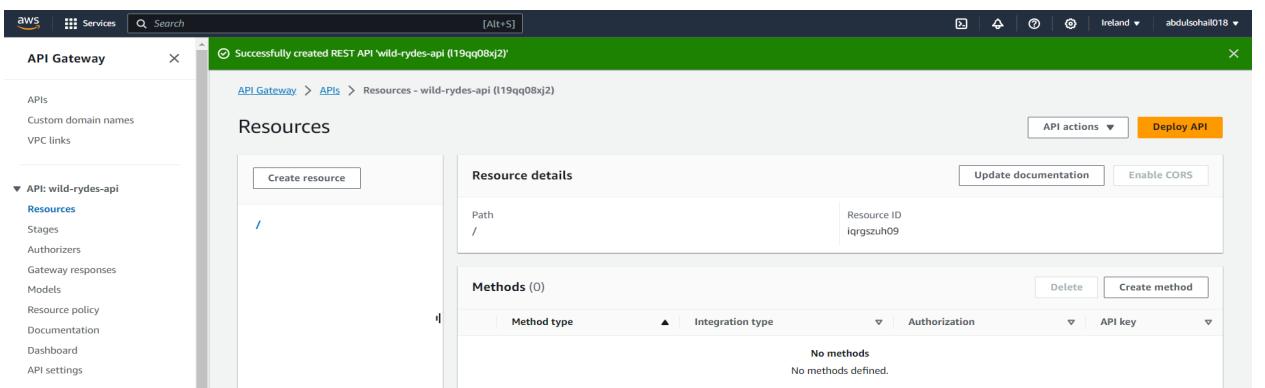


Figure 31: Successfully able to create the API Gateway Endpoint

## 14. Creating resource:

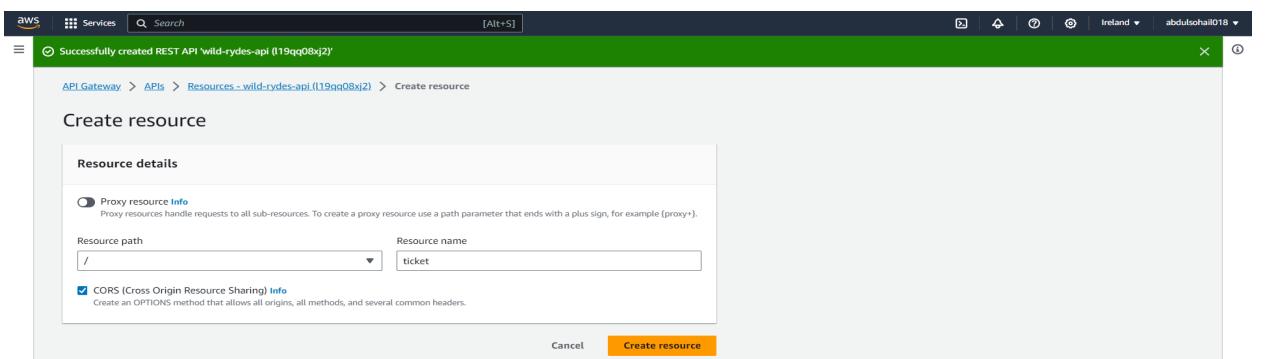


Figure 32: Providing resource name as “ticket” and enabling API Gateways CORS

## 14.1 Providing the Resource Details

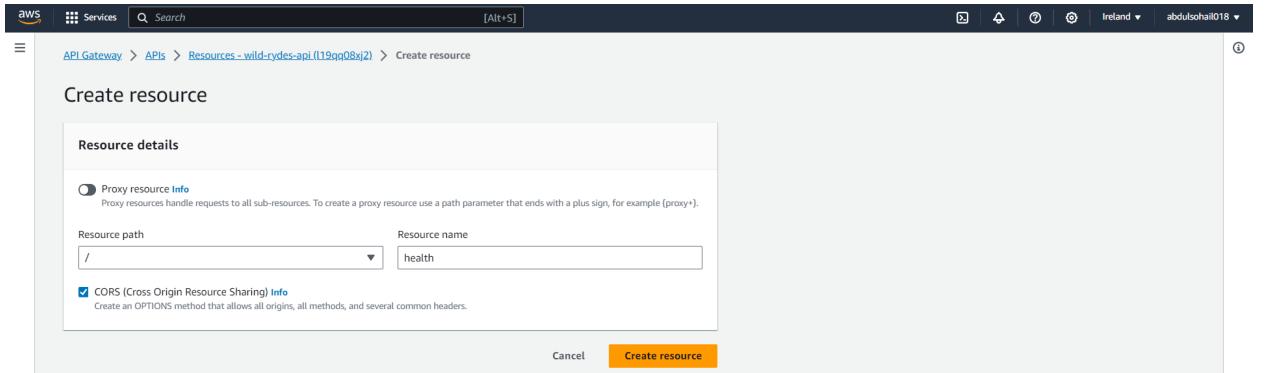


Figure 33: Providing resource name as “health” and enabling API Gateways CORS

## 15. Created resources:

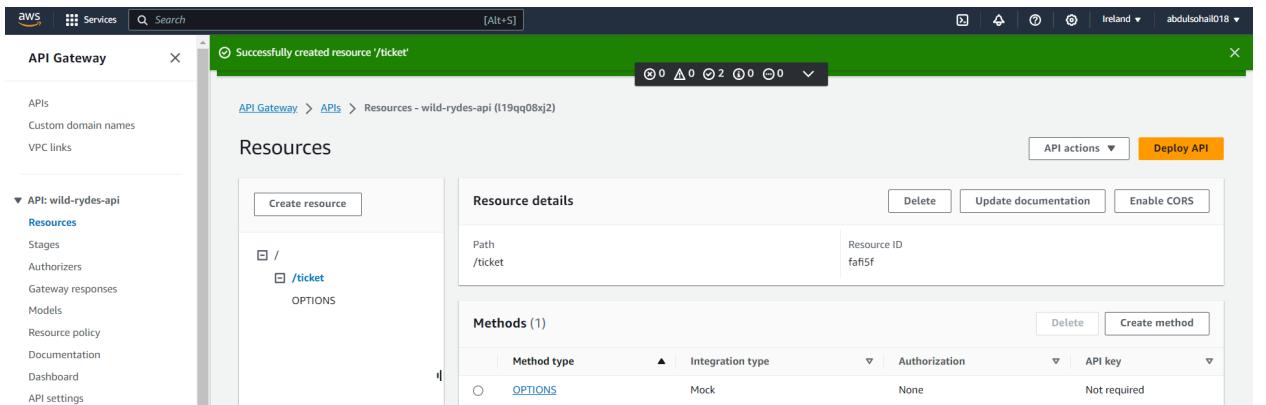


Figure 34: Successfully created the resource named “ticket”

## 15.1 Reviewing the Created Resource

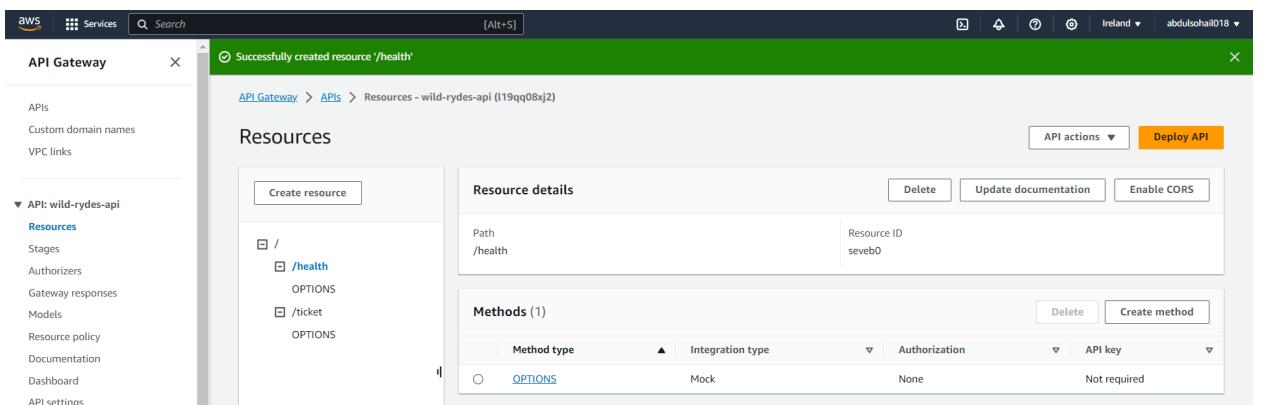
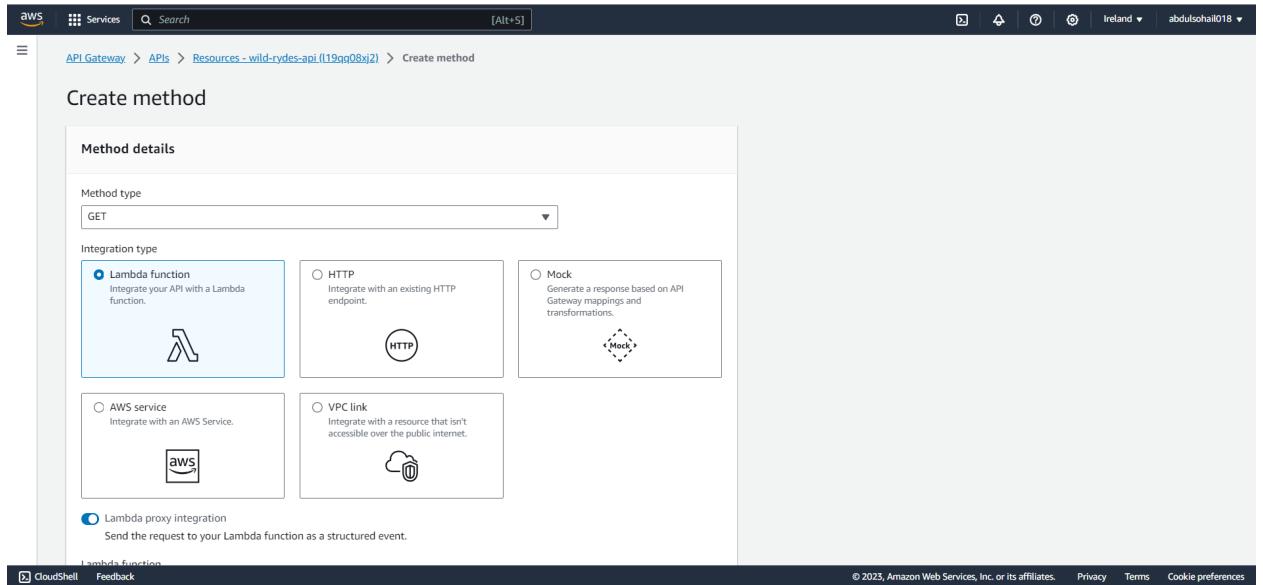


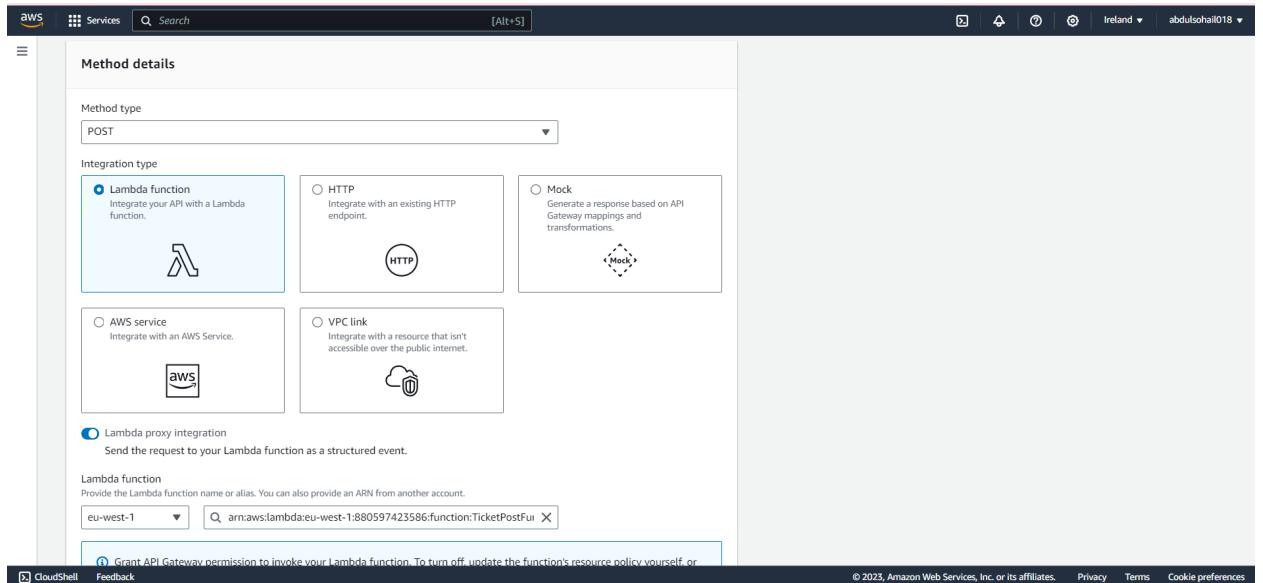
Figure 35: Successfully created the resource named “health”

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



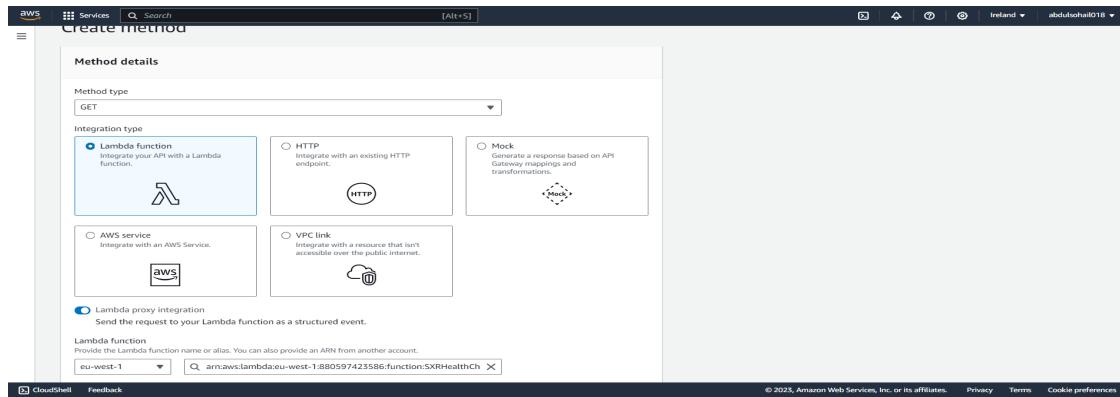
**Figure 36:** Creating a method in the “ticket” resource by selecting the method type as “GET”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “TicketGetFunction” from the Lambda function

### 16.1 Creating method type “POST” in ticket resource:



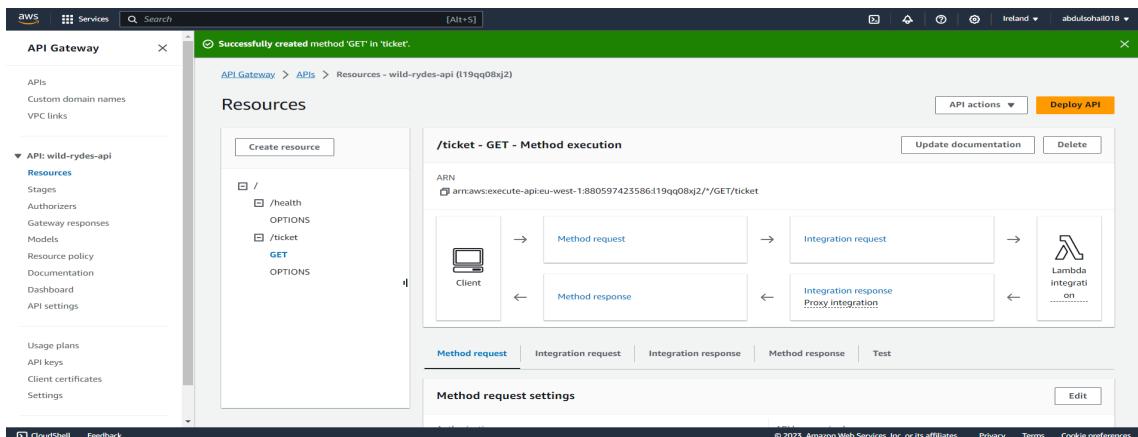
**Figure 37:** Creating a method in the “ticket” resource by selecting the method type as “POST”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “TicketPostFunction” from the Lambda function

## 16.2 Creating method type “GET” in Health resource:



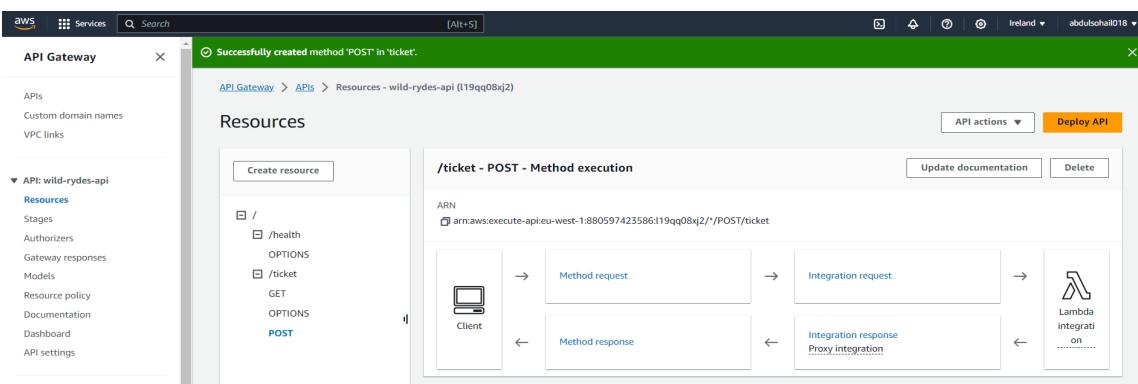
**Figure 38:** Creating a method in the “health” resource by selecting the method type as “GET”, integration type as “Lambda function”, enabling Lambda proxy integration, and selecting “SXRHealthCheckFunction” from the Lambda function

## 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

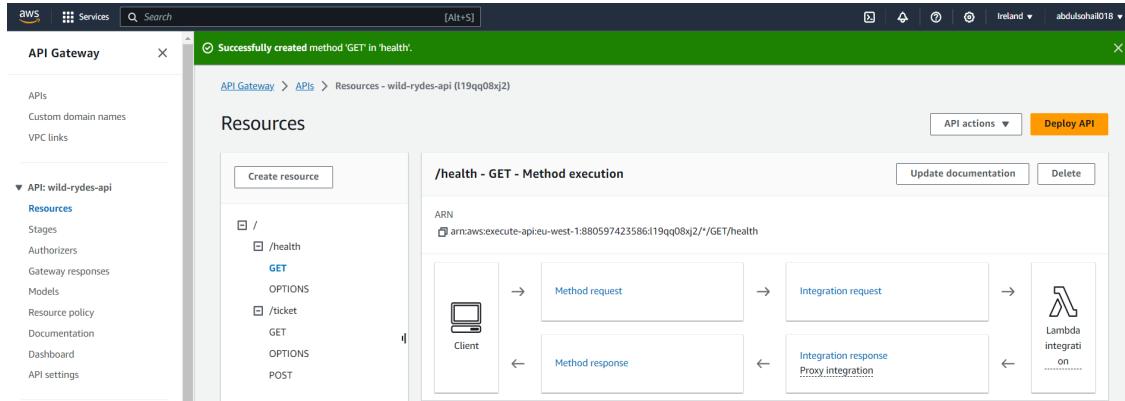


Figure 41: Successfully created GET method in “health” resource

## 18. Enabling CORS in resources:

The screenshot shows the 'Enable CORS' configuration page for the 'ticket' resource. Under 'Access-Control-Allow-Methods', the options 'GET', 'OPTIONS', and 'POST' are checked. Under 'Access-Control-Allow-Origin', a wildcard '\*' is entered. At the bottom right, there are 'Cancel' and 'Save' buttons.

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

The screenshot shows the 'Enable CORS' configuration page for the 'health' resource. Under 'Access-Control-Allow-Methods', the options 'GET' and 'OPTIONS' are checked. Under 'Access-Control-Allow-Origin', a wildcard '\*' is entered. At the bottom right, there are 'Cancel' and 'Save' buttons.

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:

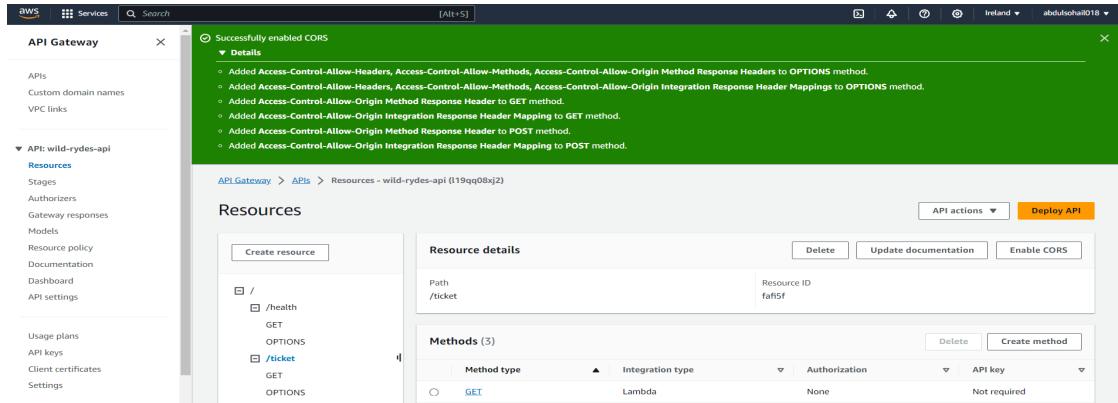


Figure 44: Successfully enabled CORS in “ticket” resource

### 19.1 Enabling CORS in the Health Resource

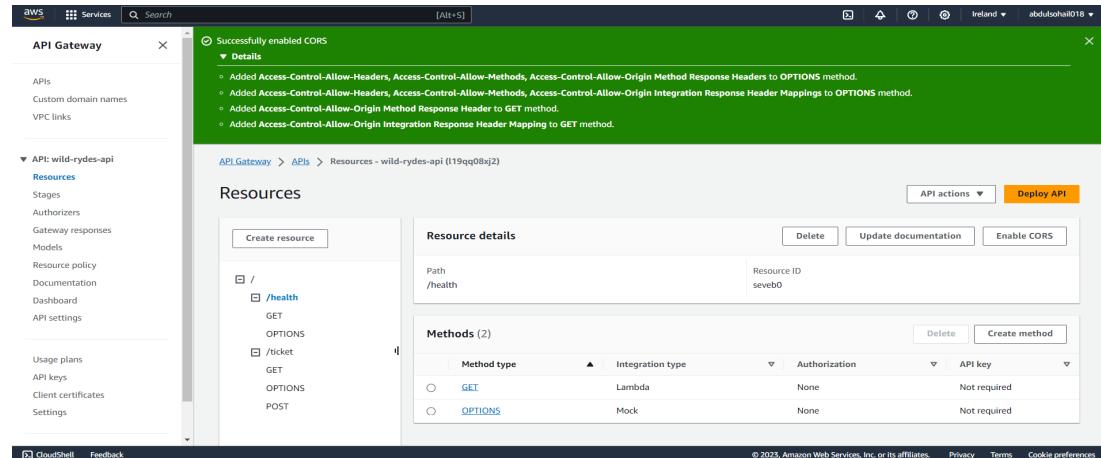


Figure 45: Successfully enabled CORS in “health” resource

## 20. Deploying API:

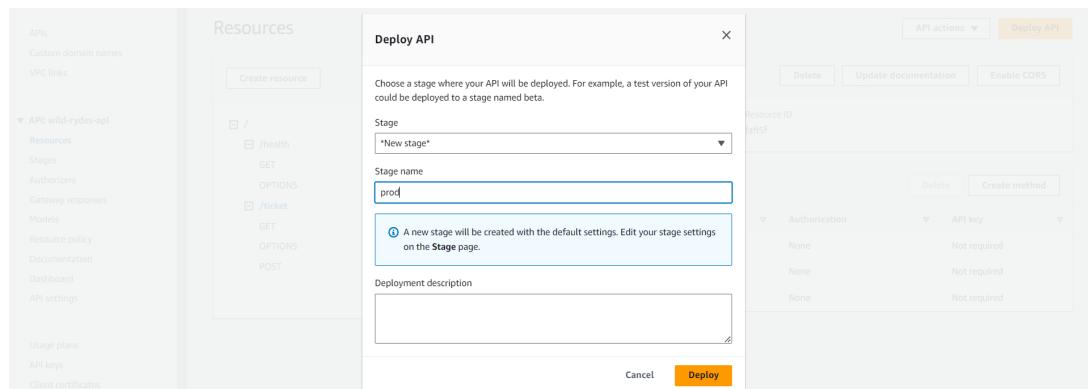


Figure 46: Deploying API by selecting stage as “New Stage” and stage name as “prod”

## 21. Deployed API:

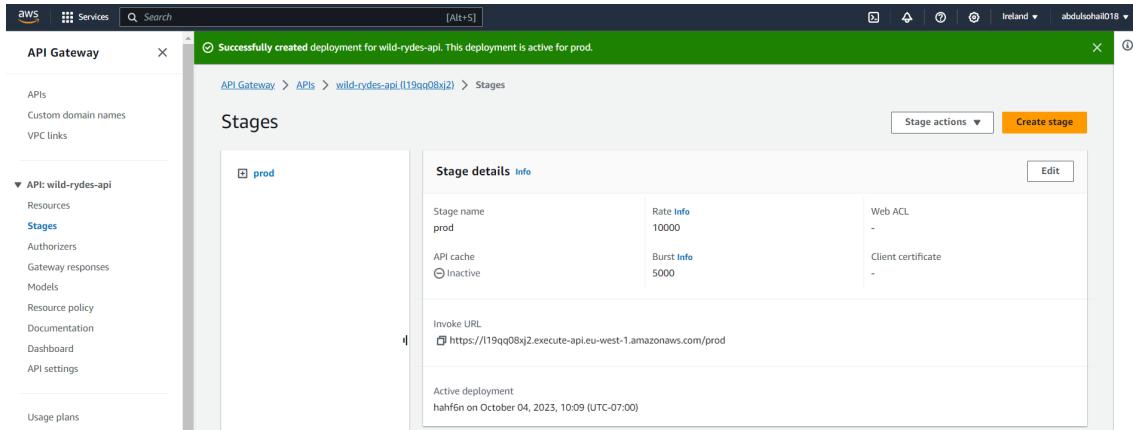


Figure 47: Successfully deployed “wild-rydes-api” API

## 22. Method execution:

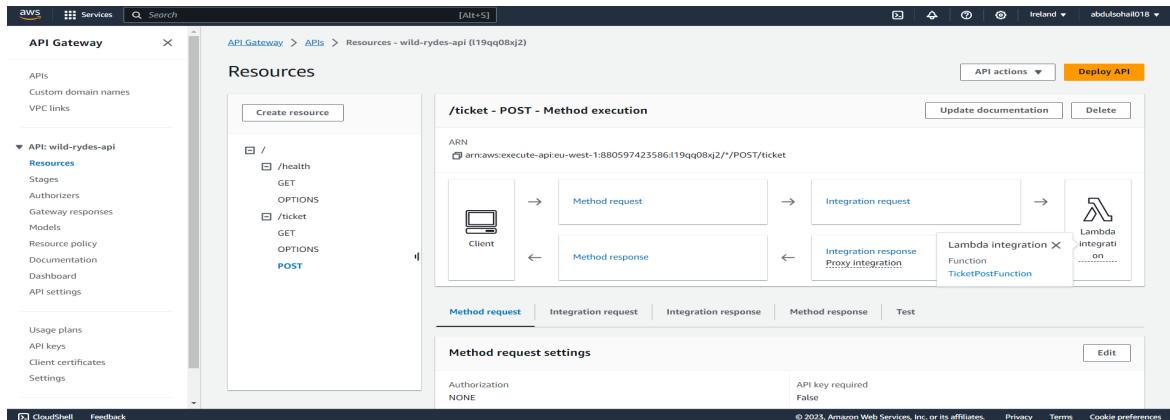


Figure 48: Shows the complete execution of the “POST” method in the “ticket” resource

## 23. Testing API Gateways Endpoints:

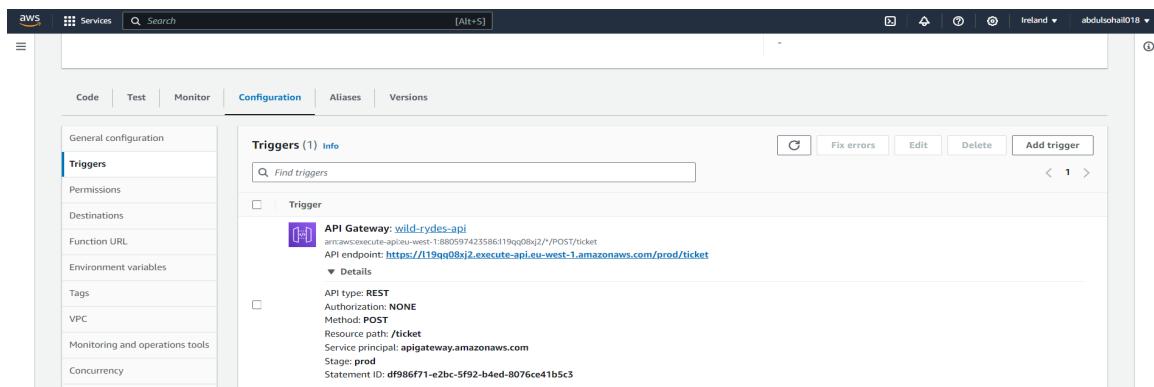
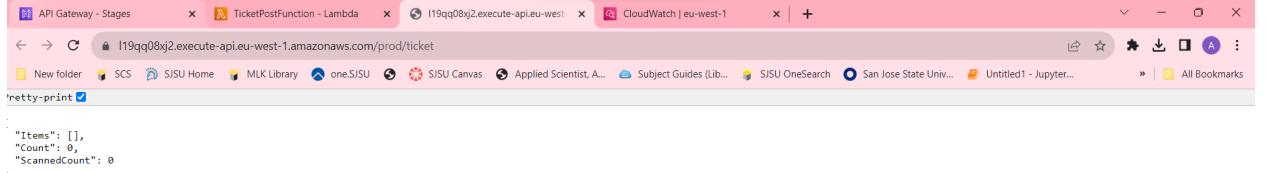


Figure 49: Trigger of “TicketPostFunction” that shows information about API Gateway name, link of API endpoint, etc

### 23.1 Display Output(Message)



```
"Items": [],
"Count": 0,
"ScannedCount": 0
```

Figure 50: Message showing successful execution of API

### 23.2 Trigger Creation

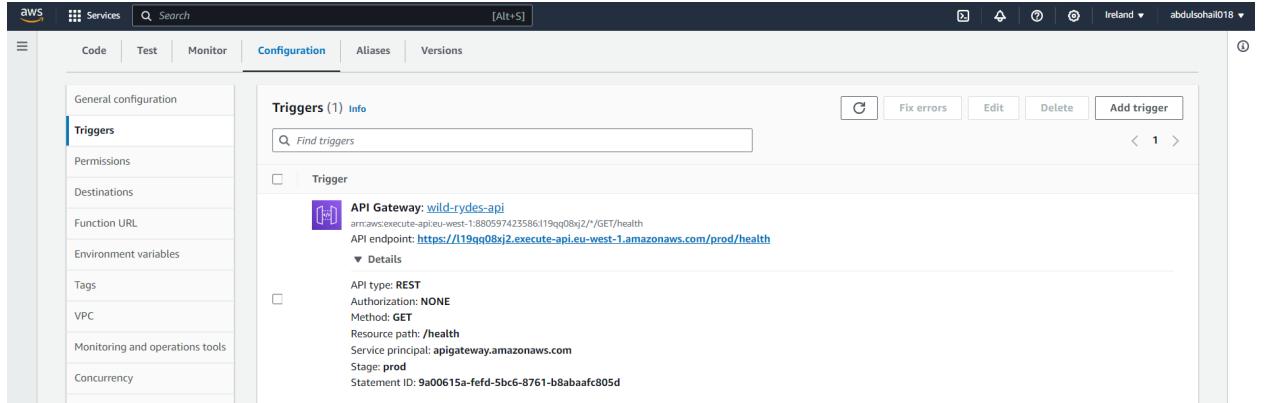
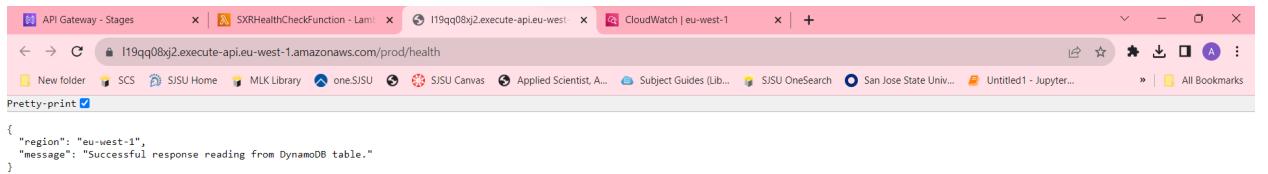


Figure 51: Trigger of “SXRHealthCheckFunction” that shows information about API Gateway name, link of API endpoint, etc

### 23.3 Display Output(Message)



```
{
  "region": "eu-west-1",
  "message": "Successful response reading from DynamoDB table."
}
```

Figure 52: Message showing successful execution of API

## Step 3: Building a UI layer:

### 1. Creating AWS Cognito Identity Pool and S3 bucket:

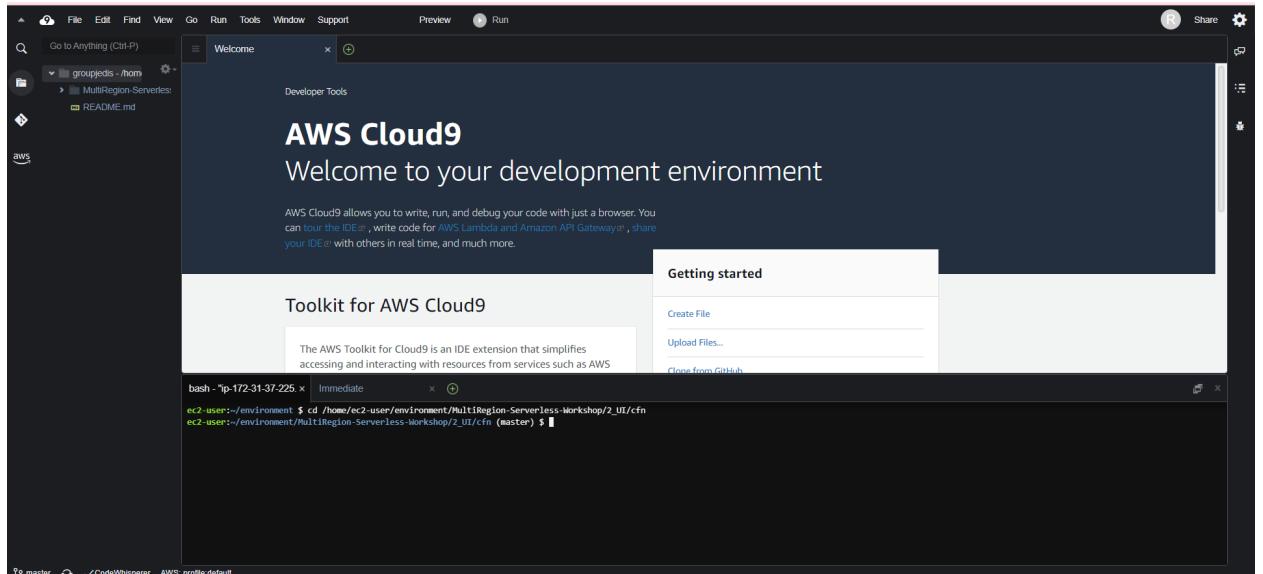


Figure 53: Navigating to Cloud9 IDE and reaching appropriate sub-repository

#### 1.1 Providing the configurations for the Stack

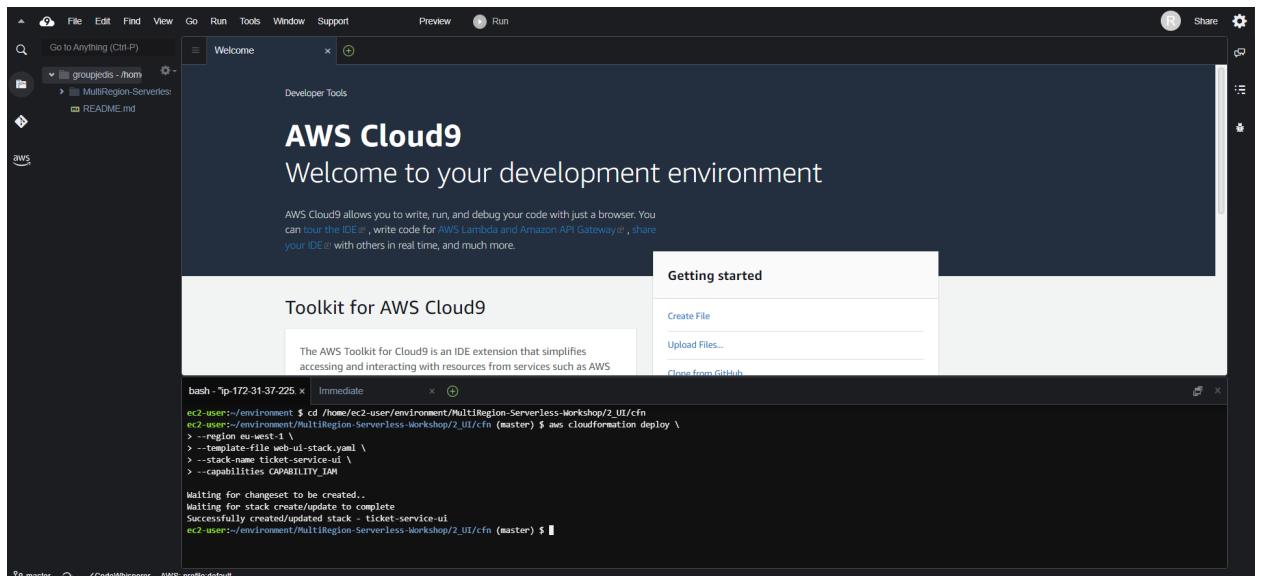


Figure 54: Specifying region, the template file, stack\_name “ticket-service-ui” and capabilities of the stack that will be created in CloudFormation

## 2. Verifying creation of stack:

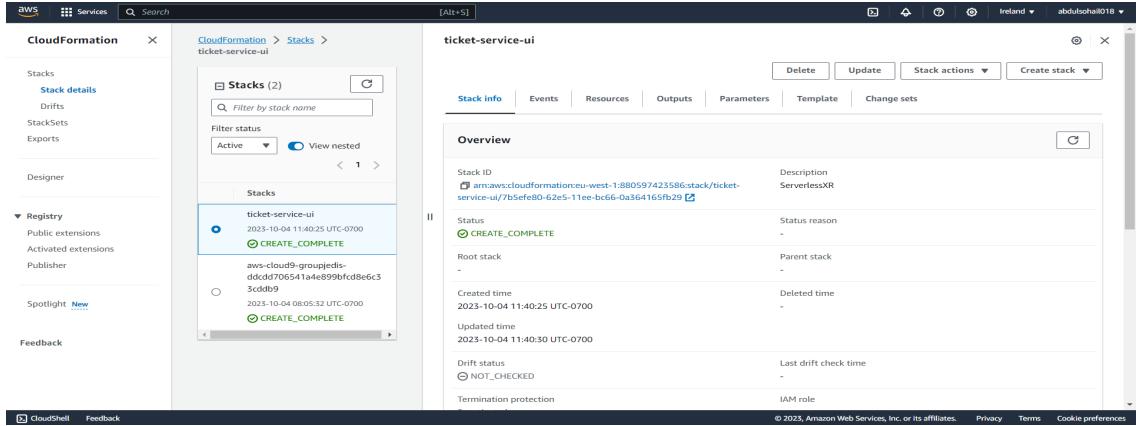


Figure 55: Successfully created stack “ticket-service-ui” in CloudFormation

### 2.1 Output Generation

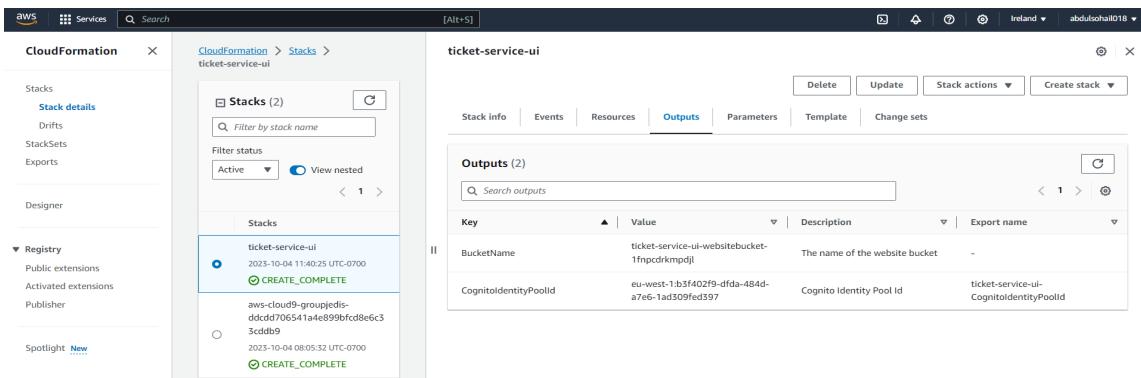


Figure 56: Generated output with BucketName and CognitoIdentityPoolId

## 3. Created Cloudfront distribution for S3 bucket:

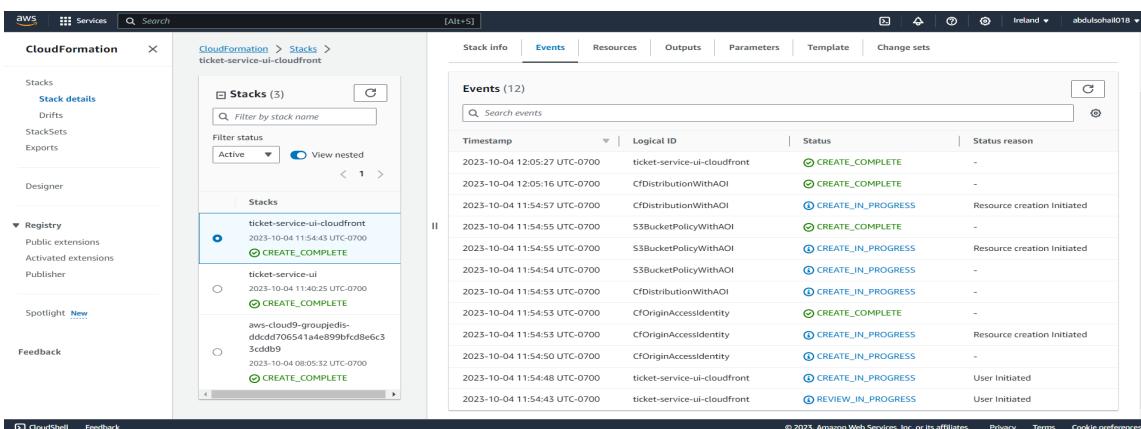


Figure 57: Successfully created cloudfront distribution named “ticket-service-ui-cloudfront”

### 3.1 Outputs Generated

The screenshot shows the AWS CloudFormation console. On the left, the navigation pane includes 'CloudFormation', 'Stacks', 'Drifts', 'StackSets', 'Exports', 'Designer', 'Registry', 'Public extensions', 'Activated extensions', 'Publisher', 'Spotlight', and 'Feedback'. The main area displays a stack named 'ticket-service-ui-cloudfront' with three stacks listed: 'ticket-service-ui-cloudfront' (status: CREATE\_COMPLETE), 'ticket-service-ui' (status: CREATE\_COMPLETE), and 'aws-cloud9-grouped-3cd9d9' (status: CREATE\_COMPLETE). The 'Outputs' tab is selected, showing three outputs:

Key	Value	Description	Export name
CFDistributionDomainNameWithOAI	d113pwfajj0z7.cloudfront.net	Domain name for our cloudfront distribution	-
CfDistributionWithOAI	E209XSV7PYFRCS	Id for our cloudfront distribution	-
S3BucketName	ticket-service-ui-websitebucket-1fnpcdrkmpdjl	Bucket name	-

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

The screenshot shows the 'Meta for Developers' application creation interface. The 'Type' tab is selected. The 'Details' tab is shown below it. The 'Add an app name' field contains 'Group\_Jedis'. The 'App contact email' field contains 'abdulsohalahmed@sjtu.edu'. The 'Business Account' section is optional and currently empty. At the bottom, there is a note about agreeing to terms and policies, and a 'Create app' button.

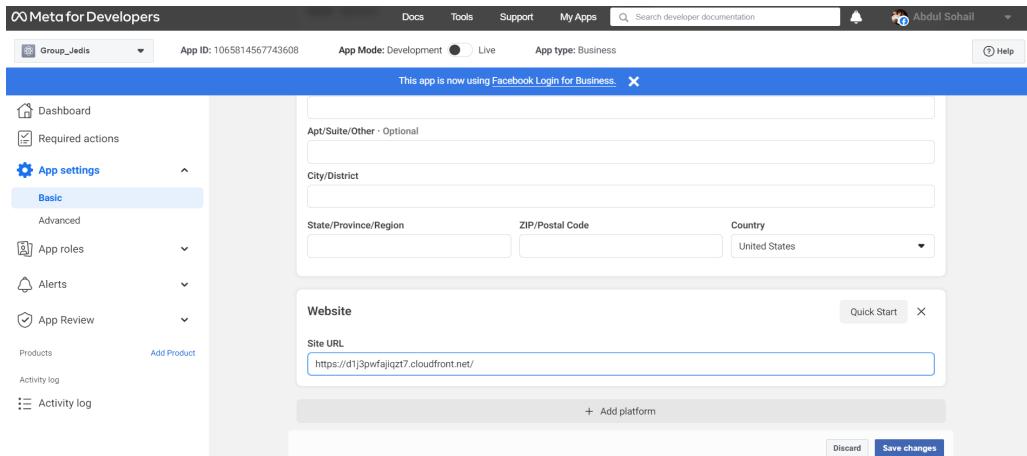
Figure 59: Providing the name of the application as “Group\_Jedis” and the email address

### 4.2 Adding platform:

The screenshot shows the 'Select Platform' dialog box. It lists several options: Website (selected), Android, iOS, Windows App, and Page Tab. Below the platforms, there are fields for 'State/Province/Region', 'ZIP/Postal Code', and 'Country' (United States). A 'Next' button is visible at the bottom right of the dialog.

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):

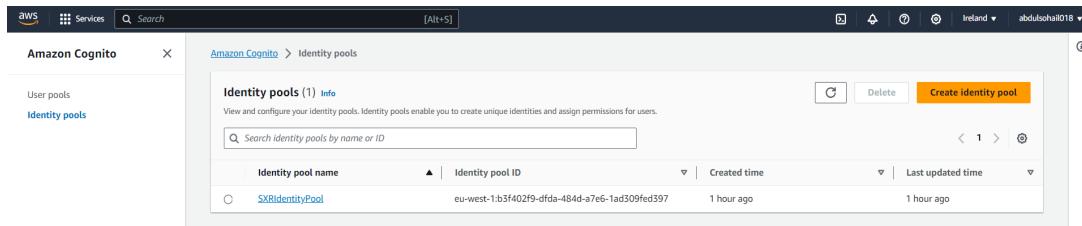


Figure 62: Amazon Cognito Dashboard

### 4.5 Addition of Identity Provider

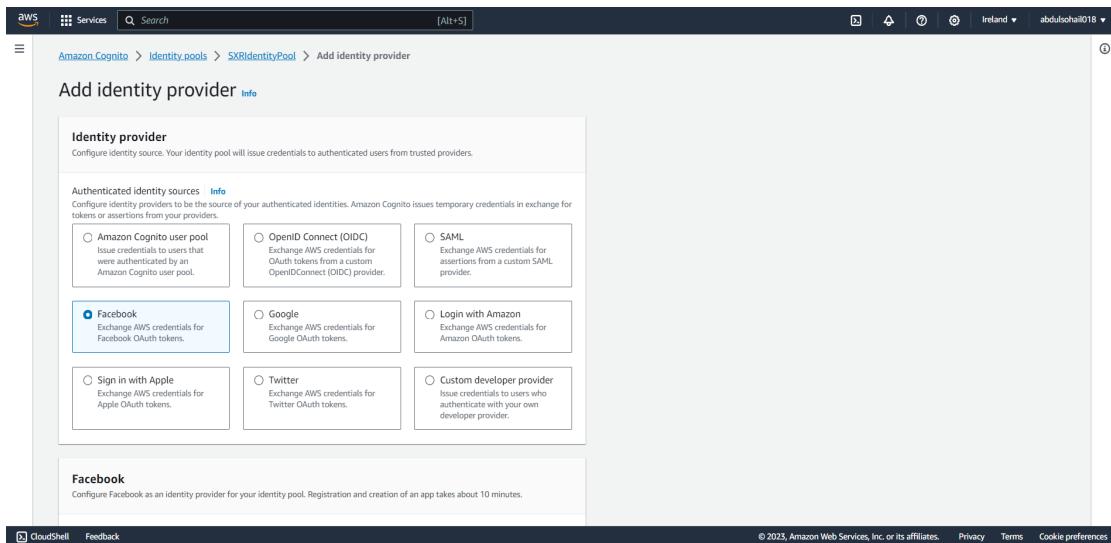
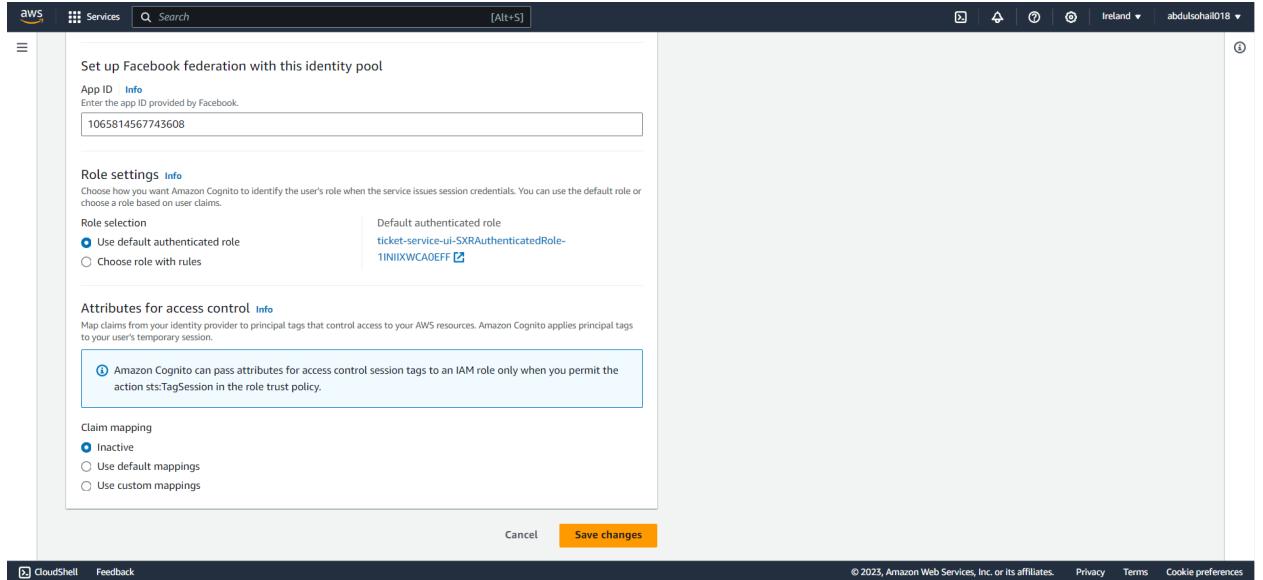


Figure 63: Adding Identity Provider "Facebook"

## 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:

```

export const environment = {
  production: true,
  // TODO: make sure you have the correct region
  region: 'eu-west-1',
  // TODO: This id can be retrieved in output section of the cognito ui
  cognitoidentityPoolId: 'eu-west-1:b3f402f9-d7da-484d-a7e5-1ad309fed397',
  // TODO: Facebook app id can be retrieved from the application in your
  // Facebook developer account.
  facebookAppId: '1065814567743608',
  // TODO: The API URL is available in the API Gateway console under Stage
  // variables. You can also retrieve it via the AWS CLI, for example:
  // https://ap1.example.com/prod/
  ticketAPI: 'https://119qg8sxj2.execute-api.eu-west-1.amazonaws.com/prod'
};

```

**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

```

Go to Anything (Ctrl-P)
File Edit Find View Go Run Tools Window Support Preview Run

Welcome x TS environments

npm WARN notice [SECURITY] parsejson has the following vulnerability: 1 high. Go here for more details: https://github.com/advisories?query=parsejson - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] xmlhttprequest-ssl has the following vulnerabilities: 2 critical. Go here for more details: https://github.com/advisories?query=xmlhttprequest-ssl - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] sever has the following vulnerability: 1 moderate. Go here for more details: https://github.com/advisories?query=sever - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] flatland@0.3: flatlet is deprecated in favor of utility frameworks such as lodash.
npm WARN notice [SECURITY] fs-yaml has the following vulnerabilities: 1 high, 1 moderate. Go here for more details: https://github.com/advisories?query=js-yaml - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] ajv has the following vulnerability: 1 moderate. Go here for more details: https://github.com/advisories?query=ajv - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] hook has the following vulnerability: 1 high. Go here for more details: https://github.com/advisories?query=hook - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] boxcpm2@0.1.0: This version has been deprecated in accordance with the npm support policy (npm.im/support). Please upgrade to the latest version to get the best features, bug fixes, and security patches. If you are unable to upgrade at this time, paid support is available for older versions (npm.im/commercial).
npm WARN notice [SECURITY] deprecated hook@0.5.3: This version has been deprecated in accordance with the npm support policy (npm.im/support). Please upgrade to the latest version to get the best features, bug fixes, and security patches. If you are unable to upgrade at this time, paid support is available for older versions (npm.im/commercial).
npm WARN notice [SECURITY] cryptiles@3.0.3: See https://github.com/lydell/source-map#deprecated
npm WARN notice [SECURITY] mem has the following vulnerability: 1 critical. Go here for more details: https://github.com/advisories?query=mem - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] uglify-js has the following vulnerabilities: 1 critical, 1 high. Go here for more details: https://github.com/advisories?query=uglify-js - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] xaldon@0.1.3: Deprecation due to CVE-2021-21366 resolved in 0.5.0
npm WARN notice [SECURITY] color-string has the following vulnerability: 1 moderate. Go here for more details: https://github.com/advisories?query=color-string - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] glob-parent@5.1.0: This version has the following vulnerability: 1 high. Go here for more details: https://github.com/advisories?query=glob-parent - Run 'npm i npm@latest -g' to upgrade your npm version, and then 'npm audit' to get more info.
npm WARN notice [SECURITY] deprecated resolve-url@0.2.1: https://github.com/lydell/source-map#deprecated
npm WARN notice [SECURITY] source-map-url@0.4.1: See https://github.com/lydell/source-map#deprecated
> node-sass@4.14.1 install /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2_UI/node_modules/node-sass
> node scripts/install.js

```

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

## 5.2 Code to successfully build the application

```

File Edit Find View Go Run Tools Window Support Preview Run

Welcome x TS environments

1 export const environment = {
2   production: true,
3 }

aws:~$ ip-172-31-37-225:x npm -ip-172-31-37-225:x Immediate Javascript (br x bash:~$ ip-172-31-37-225:x
> node-sass@4.14.1 install /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2_UI/node_modules/node-sass
> node scripts/install.js

Cached binary found at /home/ec2-user/.npm/node-sass/4.14.1/linux-x64-57_binding.node
> node-sass@4.14.1 postInstall /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2_UI/node_modules/node-sass
> node scripts/build.js

Binary found at /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2_UI/node_modules/node-sass/vendor/linux-x64-57_binding.node
Testing binary
Binary is fine
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/chokidar/node_modules/fsevents):
npm WARN noImplicitRequire SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN noImplicitType SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.0 (node_modules/webpack-dev-server/node_modules/chokidar/node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.0 (node_modules/karma/node_modules/chokidar/node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ node-sass@4.14.1
added 71 packages in 15.145s
ec2-user:~/environment/MultiRegion-Serverless-Workshop/2_UI (master) $ npm run build
> sxr-ut@0.0.0 build /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2_UI
> ng build
Date: 2023-10-04T20:27:34.990Z
Hash: f0f4c69910ca81ab2afac
Time: 1602ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 21.2 kB [rendered] [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 235 kB [inline] [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 574 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 3 MB [initial] [rendered]
chunk {views.module} views.module.chunk.js, views.module.chunk.js.map () 396 kB [main] [rendered]
ec2-user:~/environment/MultiRegion-Serverless-Workshop/2_UI (master) $ 

```

Figure 67: Successfully built the application

## 6. Uploading the application:

```

Go to Anything (Ctrl-P)          Run
File Edit Find View Go Run Tools Window Support Preview
Welcome x TS environmentts
export const environment = [
  ...
  production: true,
]

aws -T ip-172-31-37-225.x bash -T ip-172-31-37-225.x Immediate Javascript (br x bash -T ip-172-31-37-225.x
> ng build

Date: 2022-10-04T20:27:24.990Z
Hash: f044c69910ca01ab2a7c
Time: 16026ms

chunk {inline} inline.bundle.js.map (inline) 5.4 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 41.2 kB [vendor] [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 235 kB [inline] [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 1.1 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 3 MB [initial] [rendered]
chunk {views.module} views.module.chunk.js, views.module.chunk.js.map () 306 kB [main] [rendered]
ec2-user:~/environment/MultiRegion-Serverless-Workshop/2_U (master)$ $ sync --delete dist/s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl
upload:dist/assets/js/amazon-login.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/amazon-login.js
upload:dist/assets/js/aws-cognito-identity.min.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/aws-cognito-identity.min.js
upload:dist/assets/js/index.html to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/index.html
upload:dist/assets/js/main.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/main.bundle.js
upload:dist/assets/js/polyfills.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/polyfills.bundle.js
upload:dist/assets/js/styles.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/styles.bundle.js
upload:dist/assets/js/vendor.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/vendor.bundle.js
upload:dist/assets/js/views.module.chunk.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/views.module.chunk.js
upload:dist/favico.ico to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/favicon.ico
upload:dist/polyfills.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/polyfills.bundle.js
upload:dist/data-table.bce07e970865d51100.eot to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/amazon-login.eot
upload:dist/assets/js/aws-cognito-identity.min.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/aws-cognito-identity.min.js
upload:dist/assets/js/index.html to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/index.html
upload:dist/main.bundle.js, main.bundle.js.map to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/main.bundle.js.map
upload:dist/index.html to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/index.html
upload:dist/assets/js/aws/amazon-cognito-identity.min.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/aws/amazon-cognito-identity.min.js
upload:dist/favico.ico to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/favicon.ico
upload:dist/polyfills.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/polyfills.bundle.js
upload:dist/main.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/main.bundle.js
upload:dist/main.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/styles.bundle.js
upload:dist/assets/js/aws/cognito-sdk.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/aws/cognito-sdk.js
upload:dist/assets/js/aws/amazon-cognito-identity.min.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/assets/js/aws/amazon-cognito-identity.min.js.map
upload:dist/assets/js/index.html to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/index.html
upload:dist/assets/js/vendor.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/vendor.bundle.js
upload:dist/assets/js/views.module.chunk.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/views.module.chunk.js.map
upload:dist/vendor.bundle.js to s3://ticket-service-ui-websitebucket-ifnpscdrkmpdjl/vendor.bundle.js.map
ec2-user:~/environment/MultiRegion-Serverless-Workshop/2_U (master)$

```

Figure 68: Uploading the UI to S3 bucket

## 6.1 Updation of Application Settings

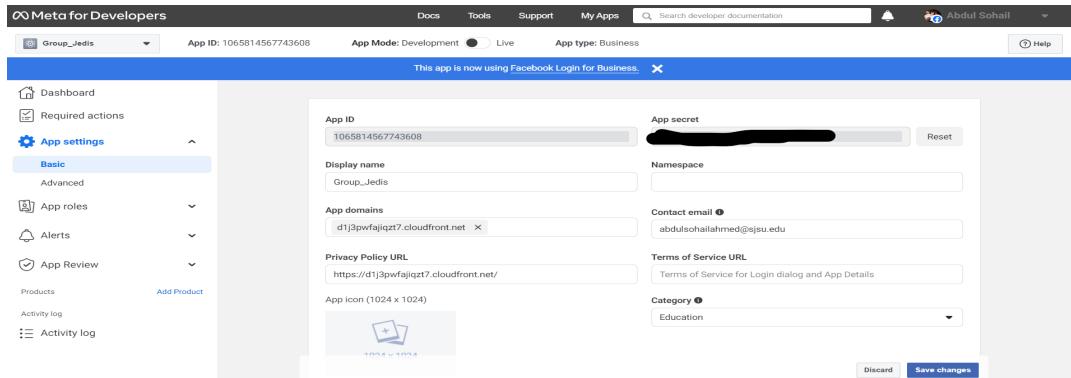


Figure 69: Updating Facebook application settings by adding details like App domains and Privacy Policy URL with cloudfront URL

## 7. Verifying the application:

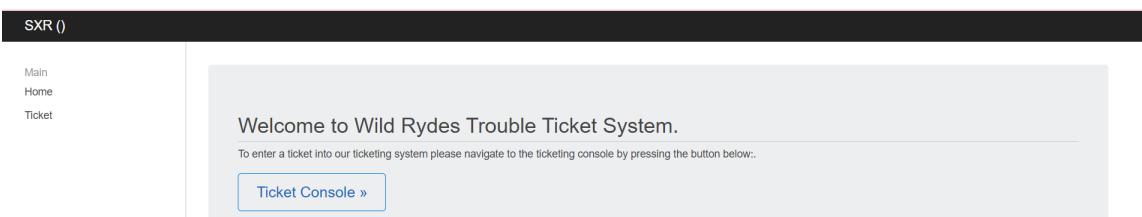


Figure 70: Navigating using cloudfront URL to the Ticket System Console

## 8. Accessing the Ticket console

The screenshot shows a web-based ticketing system. On the left, there's a sidebar with 'Main', 'Home', and 'Ticket'. The main area has a 'Create Ticket' form with fields for 'Description' (with a note: '\* Description (min 10 characters)'), 'Assigned' (dropdown), 'Priority' (dropdown), and 'Status' (dropdown). A note at the bottom says '\* = required field' and there's a 'Submit' button. Below the form is a table titled 'All Tickets' with columns: assigned, priority, status, createdOn, and createdBy. The table shows 'No data to display' and '0 total'.

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

### Step 4: Replicate to second region:

#### 1. Configuring Route 53 Domain:

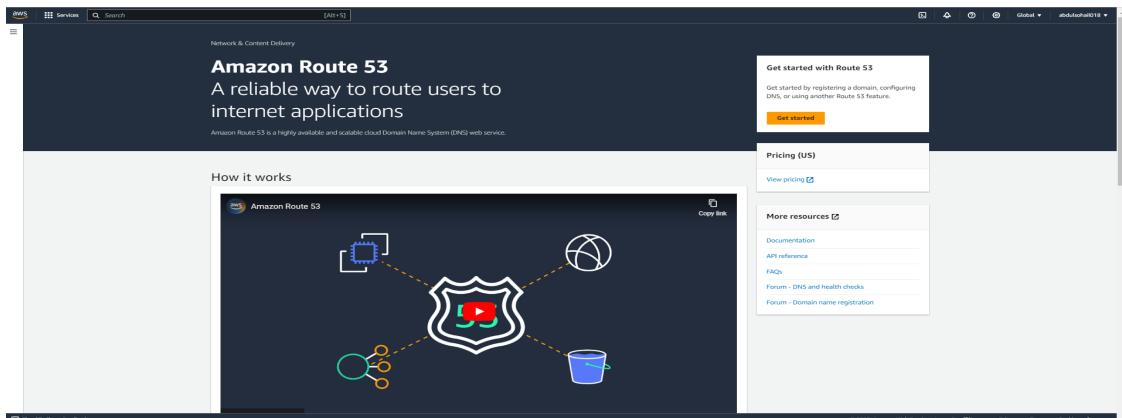


Figure 72: Amazon Route 53 Dashboard

#### 1.1 Purchased Domain Details

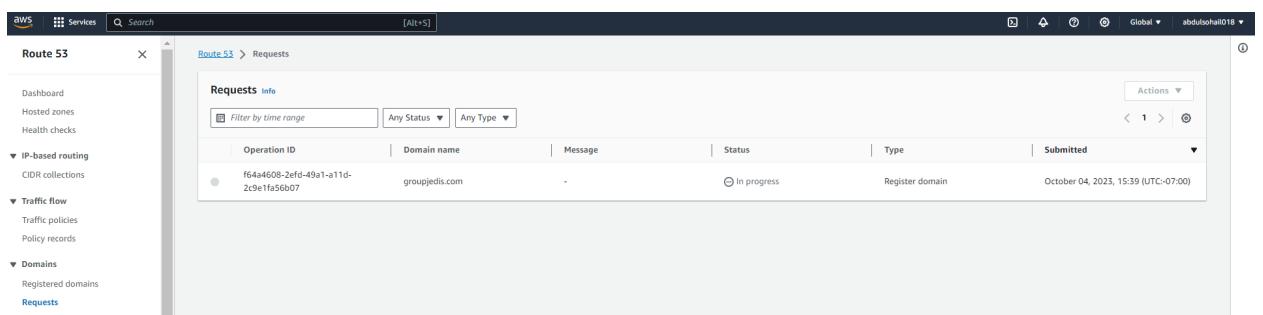


Figure 73: Purchased Domain Name

## 1.2 Request for the Public Certificate

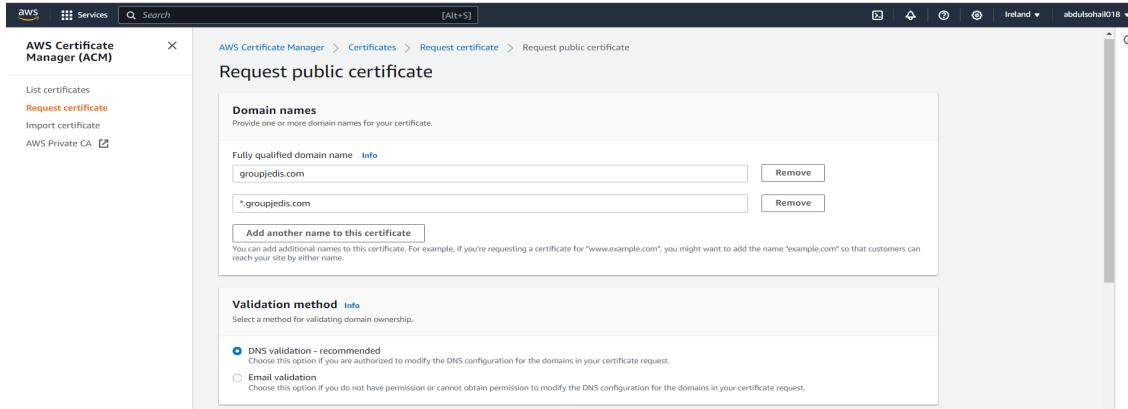


Figure 74: Requesting the Public Certificate for the purchased Domain

## 1.3 Setting up the Details for the Certificate

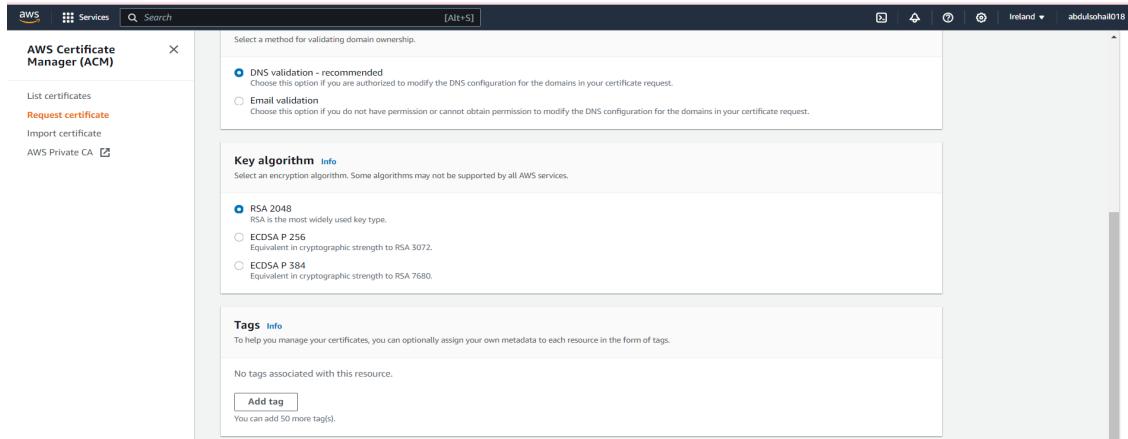


Figure 75: We are providing the details to be included in the certificate

## 1.4 Checking the Certificate Availability

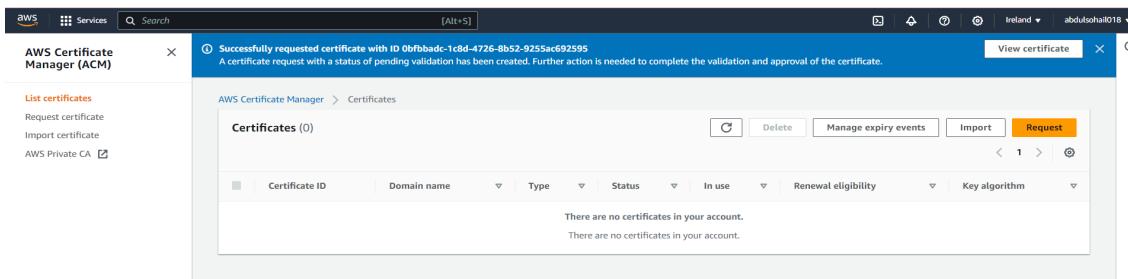


Figure 76: It currently shows that there are no available certificates in the account.

## 1.5 Certificate Status

The screenshot shows the AWS Certificate Manager (ACM) interface. On the left, a sidebar lists options: List certificates, Request certificate, Import certificate, and AWS Private CA. The main panel displays a certificate's status. The identifier is 0fbfbadc-1c8d-4726-8b52-9255ac692595. The ARN is arn:aws:acm:eu-west-1:1880597423586:certificate/0fbfbadc-1c8d-4726-8b52-9255ac692595. The type is Amazon Issued. The status is Pending validation, indicated by a blue circular icon with a question mark. Below this, a table titled 'Domains (2)' lists two entries: groupjedis.com and \*.groupjedis.com, both with pending validation status. A 'Create records in Route 53' button is available.

Figure 77: The Certificate Status shows Pending Validation

## 1.6 Issued Certificate

This screenshot shows the same certificate after its status has been updated to 'Issued'. The identifier, ARN, and type remain the same. The status is now green with a checkmark icon and the word 'Issued'. The domains listed in the table below also show a green checkmark icon and the word 'Success' next to them, indicating successful DNS record creation.

Figure 78: The Certificate Status is updated as “Issued”

## 1.7 Successful Creation of Certificate Record in Amazon S3

The screenshot shows a success message overlay: 'Successfully created DNS records' and 'Successfully created DNS records in Amazon Route 53 for certificate with ID 0fbfbadc-1c8d-4726-8b52-9255ac692595.' Below this, the certificate status is shown again with the identifier 0fbfbadc-1c8d-4726-8b52-9255ac692595, ARN arn:aws:acm:eu-west-1:1880597423586:certificate/0fbfbadc-1c8d-4726-8b52-9255ac692595, and type Amazon Issued. The status is Pending validation, indicated by a blue circular icon with a question mark.

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

## 2. Registered Domain Status

The screenshot shows the AWS Route 53 Requests page. The left sidebar includes options like Dashboard, Hosted zones, Health checks, IP-based routing, Traffic flow, Domains, Requests, and Resolver. The main content area displays a table titled 'Requests info' with columns: Operation ID, Domain name, Message, Status, Type, and Submitted. A single row is shown: 'f64a4608-2efd-49a1-a11d-2c9e1fa5a607' for 'groupjedis.com' with a status of 'Successful' and a type of 'Register domain'. The date is 'October 04, 2023, 15:39 (UTC-07:00)'.

**Figure 80:** Shows the details of the registered domain as “Successful”

### 2.1 Certificate Details in the given Region

The screenshot shows the AWS Route 53 Hosted Zone Details page for 'groupjedis.com'. The left sidebar lists various AWS services. The main content area shows a table of 'Records (3) info' with columns: Record name, Type, Routing policy, Alias, Value/Route traffic to, TTL, Health, Evaluation, and R.. The records listed are: 'groupjedis.com' (NS, Simple, No, ns-1907.awsdns-46.co.uk, ns-480.awsdns-60.com, ns-1581.awsdns-44.org, ns-599.awsdns-10.net), 'groupjedis.com' (SOA, Simple, No, ns-1907.awsdns-46.co.uk...), and '\_e03b5b1753869c9373d709203c563644.groupjedis.com' (CNAME, Simple, No, \_e3134e520869425de8c6df...).

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

### 2.2 Creation of Custom Sub-Domain

The screenshot shows the AWS API Gateway Create domain name page. The left sidebar includes options like Services, API Gateway, APIs, and Custom domain names. The main content area has two sections: 'Domain details' and 'Endpoint configuration'. In 'Domain details', the 'Domain name' field is set to 'api.groupjedis.com'. Under 'Minimum TLS version', 'TLS 1.2 (recommended)' is selected. In 'Endpoint configuration', 'Regional' is selected as the API endpoint type. There is also an option for 'Edge-optimized (supports only REST APIs)' which is currently unselected.

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

## 2.3 Setting up the End Point Configuration

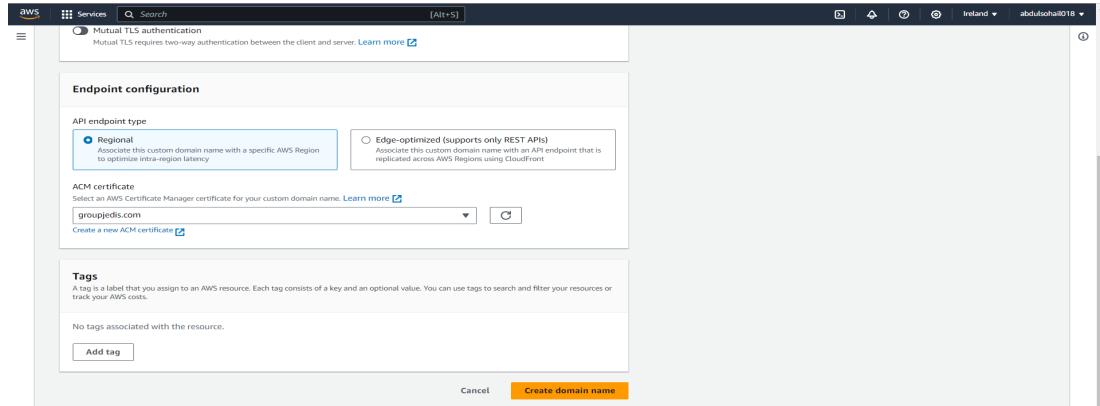


Figure 83: Configuring the End Point

## 2.4 Reviewing the EndPoint configuration

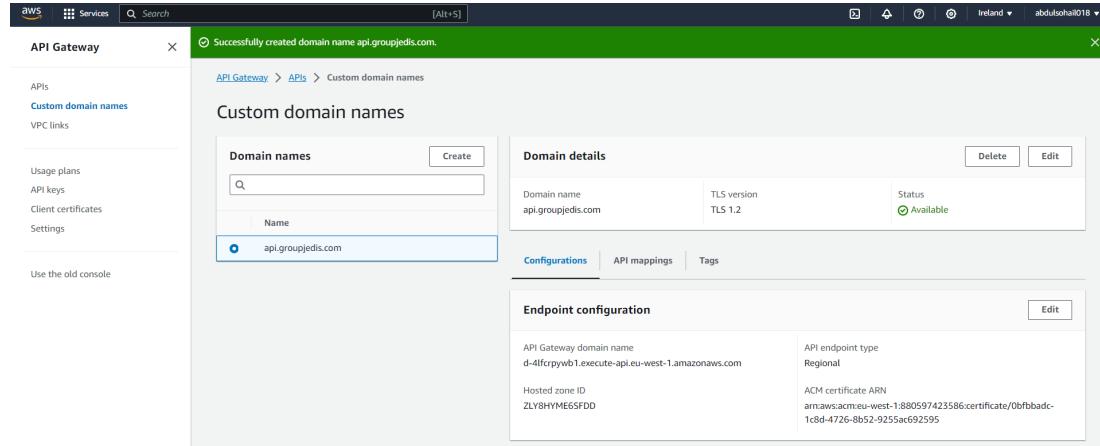


Figure 84: We have reviewed the configuration for the API Endpoint

## 2.5 Creation of Custom Sub-Domain for Ireland Region

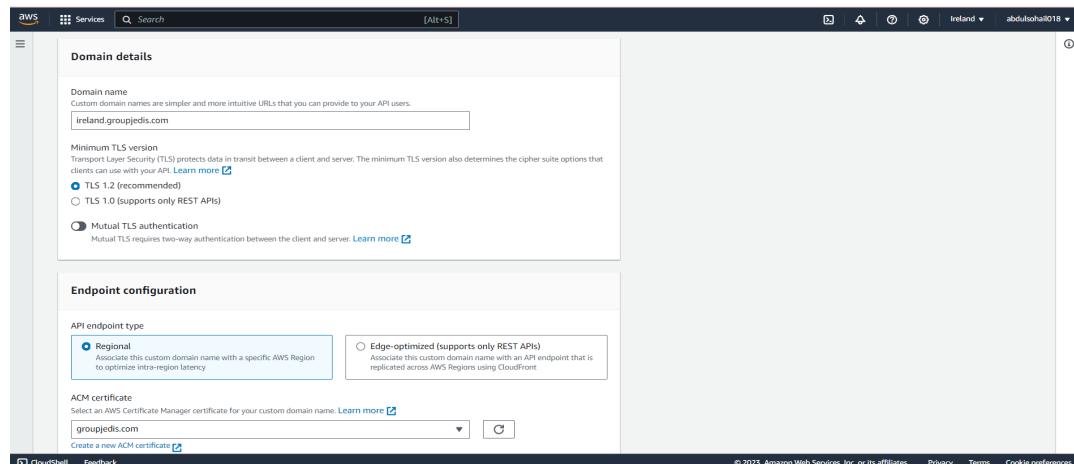


Figure 85: Creating a custom domain along with endpoint configuration

## 2.6 Created Custom Domains

The screenshot shows the 'Custom domain names' section of the AWS API Gateway console. On the left, a sidebar lists 'APIs', 'Custom domain names' (which is selected), and 'VPC links'. The main area has a search bar and a 'Create' button. A table displays the custom domain 'ireland.groupjedis.com' with details: Domain name (ireland.groupjedis.com), TLS version (TLS 1.2), and Status (Available). Below the table are tabs for 'Configurations', 'API mappings', and 'Tags'. At the bottom, there's an 'Endpoint configuration' section with fields for API Gateway domain name, API endpoint type (Regional), and ACM certificate ARN.

Figure 86: Shows the custom domain created

## 3. Replicating to Singapore region:

### 3.1 Naming AWS Cloud 9 environment

The screenshot shows the 'Create environment' page in the AWS Cloud 9 console. The 'Details' section includes a 'Name' field containing 'groupjedis\_singapore'. There are two options for 'Environment type': 'New EC2 instance' (selected) and 'Existing compute'. The 'New EC2 instance' section is expanded, showing a note about Cloud9 creating an EC2 instance. At the bottom, there are buttons for 'CloudShell' and 'Feedback'.

Figure 87: Providing the name of the Cloud9 environment

### 3.2 Created Cloud9 environment:

The screenshot shows the 'Environments' page in the AWS Cloud 9 console. A green banner at the top says 'Successfully created groupjedis\_singapore. To get the most out of your environment, see Best practices for using AWS Cloud9.' The main table lists one environment: 'groupjedis\_singapore' (Status: Open, Type: EC2 instance, Connection: AWS Systems Manager (SSM), Permission: Owner, Owner ARN: arn:aws:iam:880597423586:root). There are buttons for 'Delete', 'View details', 'Open in Cloud9', and 'Create environment'.

Figure 88: Successfully able to create the Cloud9 environment

### 3.3 Creating AWS Cognito Identity Pool and S3 bucket:

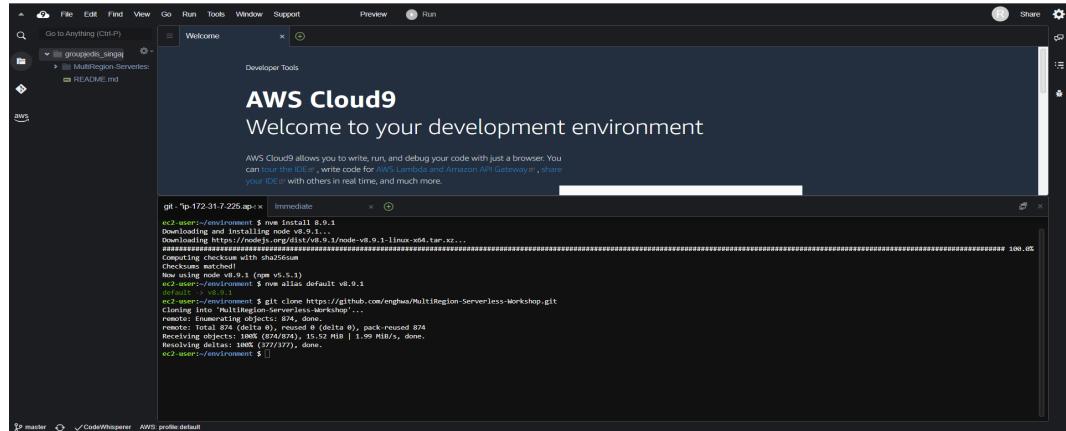


Figure 89: Installing the node and npm required for Angular Front end

### 3.4 Creating lambda functions:

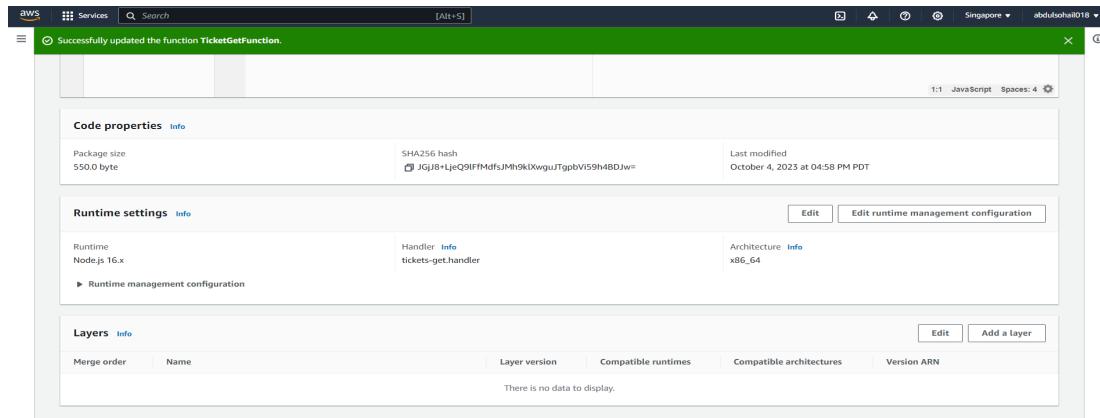


Figure 90: Providing function name “TicketGetFunction” and selecting “Node.js 16.x” to write functions

### 3.5 Setting the Environment Variables

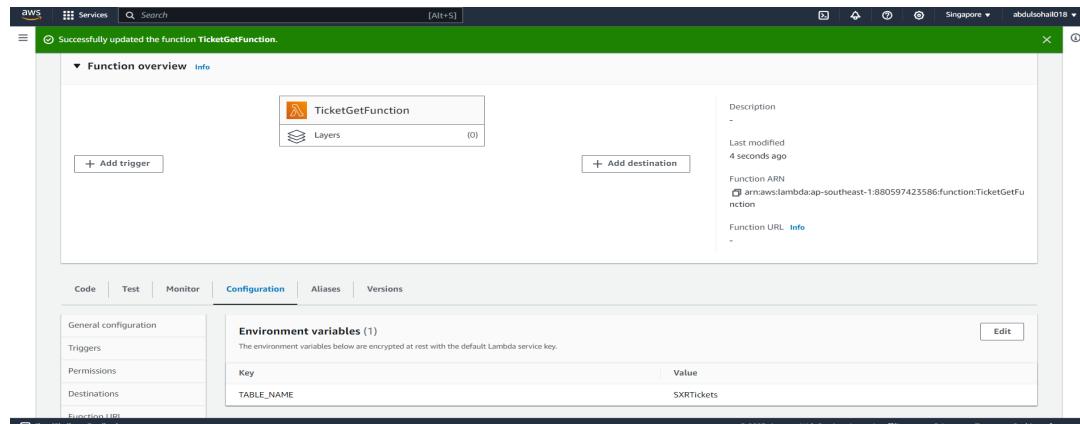


Figure 91: Under environment variables entering the key TABLE\_NAME and the value SXRTickets.

### 3.6 Creation of TicketPostFunction

The screenshot shows the AWS Lambda function editor for the 'TicketPostFunction'. The code is written in JavaScript and handles posting new tickets to a DynamoDB table. It uses environment variables to determine the table name.

```

1  'use strict';
2  const AWS = require('aws-sdk');
3  const table = process.env.TABLE_NAME;
4
5  /* This will handle posting of new ticket and puts for updates.
6   * If the ticket already exists, it will update its assumed to be an update
7   * request to an existing ticket write.
8   */
9  // @param event
10 // @param context
11 // @param callback
12
13 exports.handler = (event, context, callback) => {
14
15     console.log('Received event:', JSON.stringify(event, null, 2));
16
17     let body = JSON.parse(event.body);
18
19     const docClient = new AWS.DynamoDB.DocumentClient();
20
21     const params = {
22         TableName: table, // get table name from env variable.
23         Item: {
24             "id": new Date().toISOString(),
25             "body": body,
26             "assigned": false,
27             "status": "open",
28             "createdOn": new Date().toISOString().replace(/\//, '-').replace(/\.:/, '')
29         }
30     };
31
32     console.log("Adding a new ticket..." + JSON.stringify(params));
33
34     docClient.put(params, function (err, data) {
35         if (err) {
36             callback(`Unable to add ticket. Error ${err}: ${JSON.stringify(err, null, 2)}`);
37         } else {
38             callback(null, {
39                 statusCode: 200,
40                 headers: {
41                     "Access-Control-Allow-Origin": "*",
42                     "Access-Control-Allow-Credentials": true // Required for cookies, authorization headers with HTTPS
43                 },
44                 body: JSON.stringify(data)
45             });
46         }
47     });
48 }

```

Figure 92: Modifying the “TicketPostFunction” function

### 3.7 Setting the Environment Variables

The screenshot shows the AWS Lambda function configuration page for 'TicketPostFunction'. The 'Environment variables' section is selected, showing a key 'TABLE\_NAME' with the value 'SXRTickets'.

Key	Value
TABLE_NAME	SXRTickets

Figure 93: Under environment variables entering the key TABLE\_NAME and the value SXRTickets.

### 3.8 Setting up the Health function

The screenshot shows the AWS Lambda function editor for the 'SXRHealthCheckFunction'. The code is written in JavaScript and performs a scan operation on a DynamoDB table to check if it can be read.

```

1  'use strict';
2  const AWS = require('aws-sdk');
3  const table = process.env.TABLE_NAME;
4
5  exports.handler = (event, context, callback) => {
6
7      const docClient = new AWS.DynamoDB.DocumentClient();
8
9      const params = {
10          TableName: table
11      };
12
13      docClient.scan(params, function (err, data) {
14
15          if (err) {
16              console.log(JSON.stringify(err, null, 2));
17              body.message = ("Could not read from DynamoDB table.");
18              body.region = process.env.AWS_REGION;
19
20              response = {
21                  statusCode: 500,
22                  headers: {
23                      "Access-Control-Allow-Origin": "*",
24                      "Access-Control-Allow-Credentials": true // Required for cookies, authorization headers with HTTPS
25                  },
26                  body: JSON.stringify(data)
27              };
28          } else {
29              body.message = ("Successful response reading from DynamoDB table.");
30
31              response = {
32                  statusCode: 200,
33                  headers: {
34                      "Access-Control-Allow-Origin": "*",
35                      "Access-Control-Allow-Credentials": true // Required for cookies, authorization headers with HTTPS
36                  },
37                  body: JSON.stringify(data)
38              };
39          }
40      });
41
42  }

```

Figure 94: Modifying the “SXRHealthCheckFunction” function

### 3.9 Updating the Health Check Function

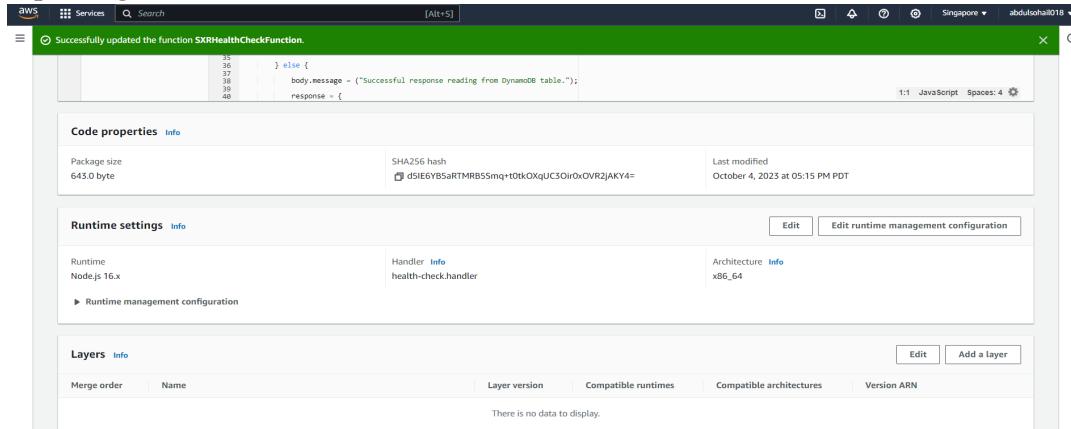


Figure 95: Changed handler to “health-check.handler”

### 3.10 Providing values for Environment variables

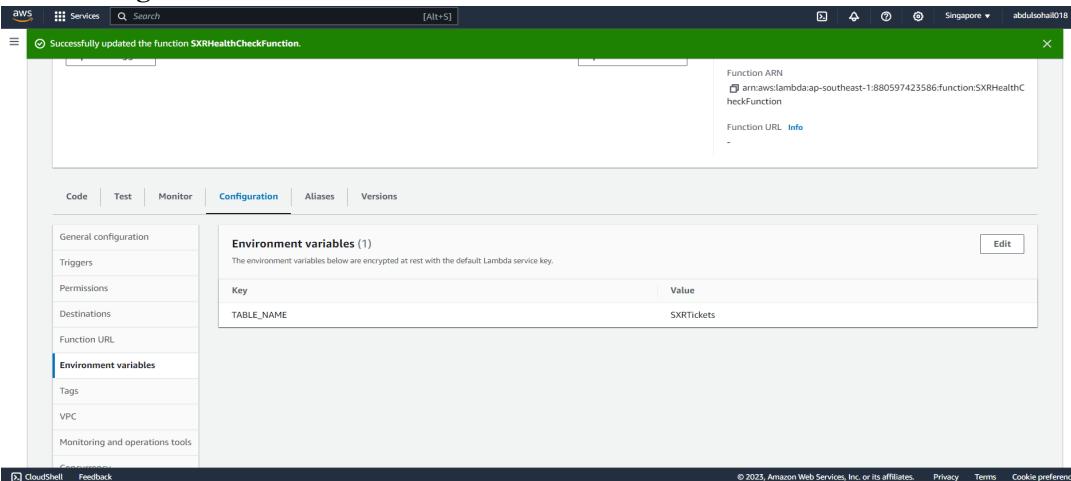


Figure 96: Under environment variables entering the key TABLE\_NAME and the value SXRTickets.

### 3.11 Creating a new API

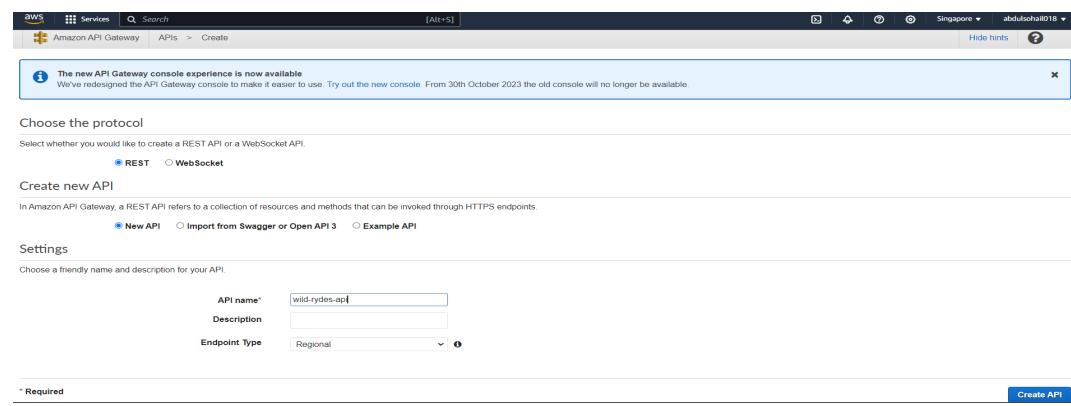


Figure 97: Created a new API using REST Protocol

### 3.12 Creation of new Resource

The screenshot shows the AWS API Gateway console interface. On the left, a sidebar lists the API 'wild-rydes-api' and its resources, including 'Stages', 'Authorizers', 'Gateway Responses', 'Models', 'Resource Policy', 'Documentation', 'Dashboard', and 'Settings'. The main area is titled 'New Child Resource' and shows the path configuration. The 'Resource Name\*' field is set to 'ticket' and the 'Resource Path\*' field is set to '/ticket'. A note explains that path parameters can be added using brackets, such as '{username}' for a path parameter called 'username'. Below the fields are 'Enable API Gateway CORS' and 'Create Resource' buttons.

Figure 98: Created a new Resource named “Ticket”

### 3.13 Creation of new Resource

This screenshot is similar to Figure 98, showing the creation of a new child resource. The path is now '/ticket' and the resource name is 'health'. The 'Resource Path\*' field is set to '/health'. The 'Enable API Gateway CORS' checkbox is checked. The 'Create Resource' button is visible at the bottom right.

Figure 98: Created a new Resource named “health”

### 3.14 Creation Of two methods for Ticket Resource

This screenshot shows the configuration for a new method under the '/ticket' resource. The path is '/ticket' and the method is 'GET'. The 'Integration type' is set to 'Lambda Function'. The 'Lambda Region' is 'ap-southeast-1' and the 'Lambda Function' is 'TicketGetFunction'. The 'Save' button is at the bottom right.

Figure 99: Creating the GET method for the ticket resource

### 3.15 Creation of Post Method

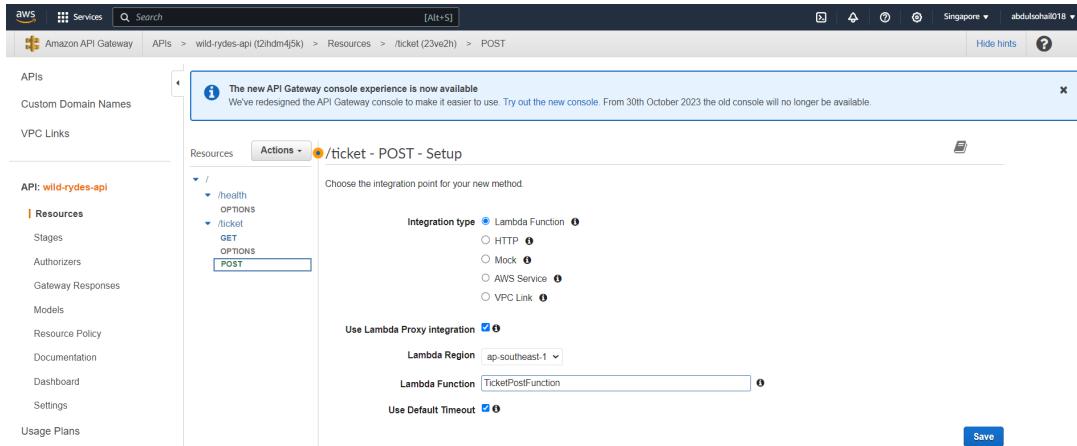


Figure 100: Creating the POST method for the ticket resource

### 3.16 Creation of GET Method for Health Resource

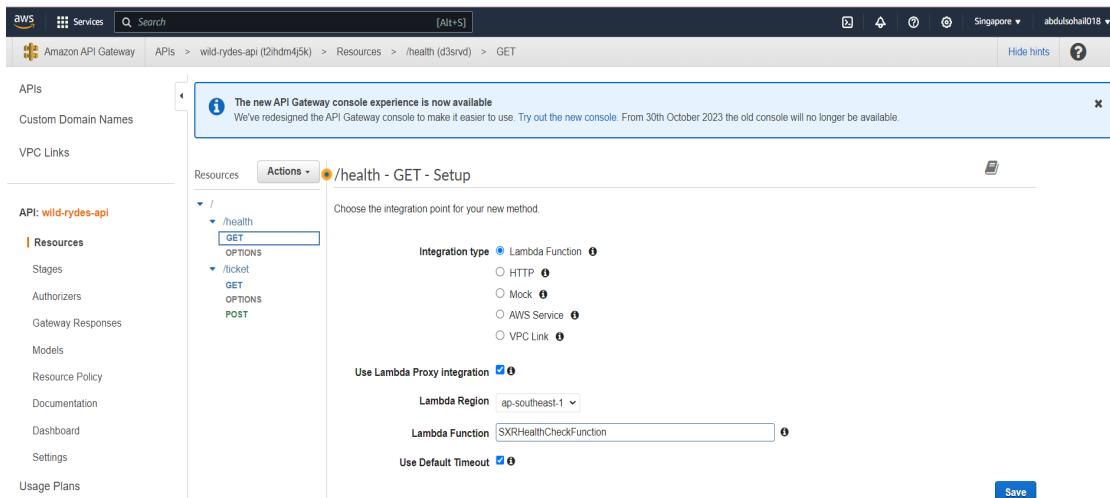


Figure 101: Creating the GET method for the health resource

### 3.17 Deployment of API

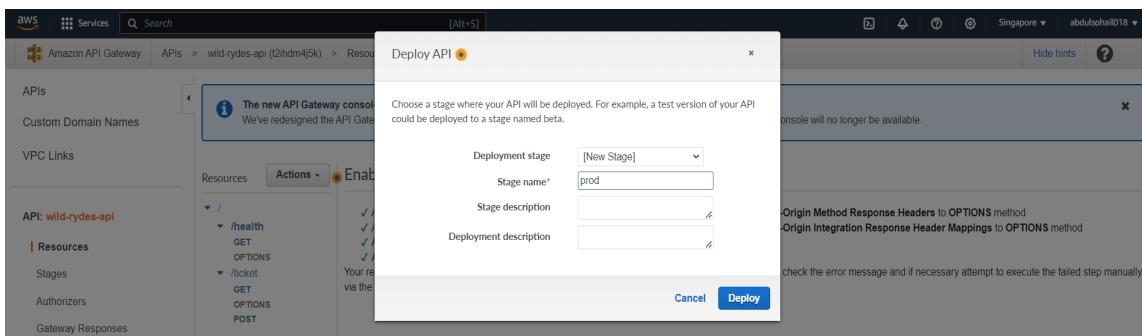


Figure 102: Deploying the API's

### 3.18 Successful Deployment of API

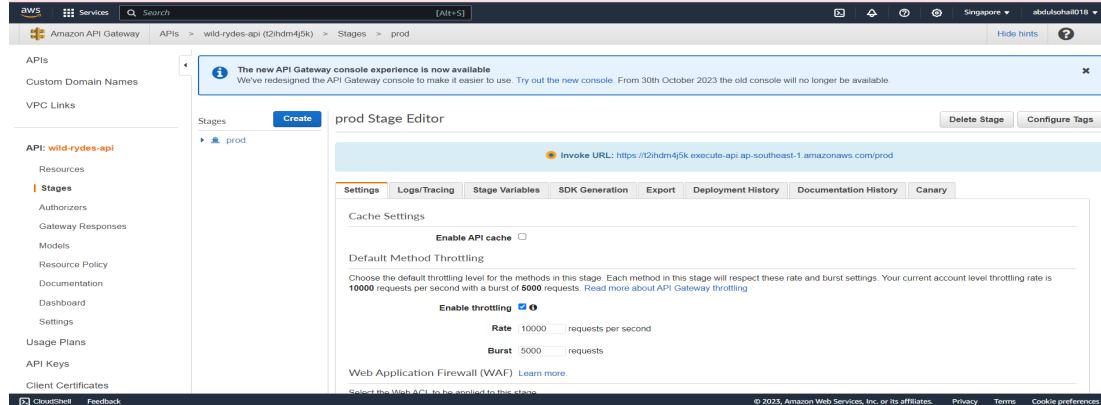


Figure 103: API successfully deployed

### 3.19 Script for the Ticket Resource

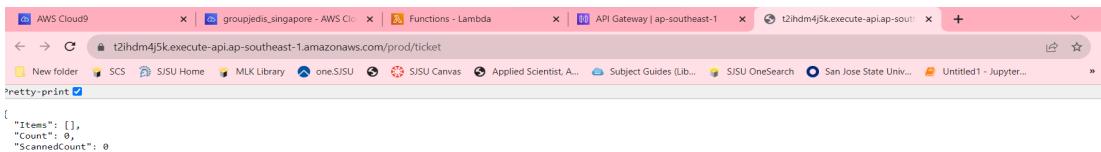


Figure 104: Visualizing the ticket API script

### 3.20 Script for the Health Resource



Figure 105: Visualizing the health API script

### 3.21 Creation of Certificate

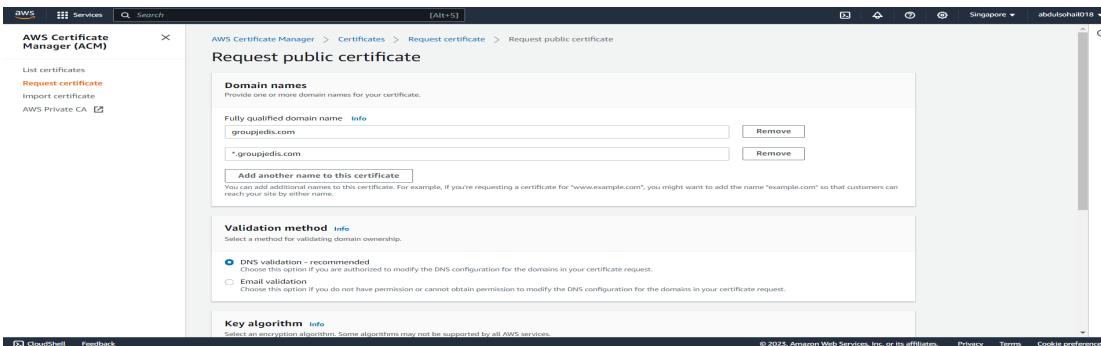


Figure 106: Creating Certificate for Singapore Region

### 3.22 Created Certificate for the Region

The screenshot shows the AWS Certificate Manager interface. A success message at the top states: "Successfully requested certificate with ID b2b87691-5241-44ae-aa35-4ea00c2e960c. A certificate request with a status of pending validation has been created. Further action is needed to complete the validation and approval of the certificate." Below this, the "Certificates" section shows the newly created certificate. A sub-section titled "Create DNS records in Amazon Route 53 (2/2)" displays two pending validation records. The first record is for the domain "groupjedids.com" with a CNAME value of "e3134e520869426ddc6df57fd6c9eac.xlkwfmvcv.acm-validations.aws.". The second record is for the domain "\*groupjedids.com" with a CNAME value of "e3134e520869426ddc6df57fd6c9eac.xlkwfmvcv.acm-validations.aws.". Both records have a status of "Pending validation".

Figure 107: Visualizing certificates created for the Singapore Region

### 3.23 Certification Status

The screenshot shows the AWS Certificate Manager interface after the certificate has been issued. A green success message at the top states: "Successfully created DNS records in Amazon Route 53 for certificate with ID b2b87691-5241-44ae-aa35-4ea00c2e960c." Below this, the "Certificates" section shows the issued certificate with the ID "b2b87691-5241-44ae-aa35-4ea00c2e960c". The "Certificate status" section indicates the status is "Issued". The "Domains" section lists two domains: "groupjedids.com" and "\*groupjedids.com", both with a status of "Success". The screenshot also shows the ARN of the certificate: "arn:aws:acm:ap-southeast-1:1880597423586:certificate/b2b87691-5241-44ae-aa35-4ea00c2e960c".

Figure 108: Certificate successfully created

### 3.24 List of API

The screenshot shows the AWS Route 53 interface. A success message at the top states: "e03b5b1753869c9373d709203c563644.groupjedids.com was successfully updated. Route 53 propagates your changes to all of the Route 53 authoritative DNS servers within 60 seconds. Use 'View status' button to check propagation status." Below this, the "Hosted zones" section shows the hosted zone for "groupjedids.com". The "Records" tab is selected, displaying three records: "groupjedids.com" (NS, Simple, Value: ns-1907.awsdns-46.co.uk., TTL: 172800), "groupjedids.com" (SOA, Simple, Value: ns-480.awsdns-60.com., TTL: 900), and "groupjedids.com" (CNAME, Simple, Value: e3134e520869426ddc6df57fd6c9eac.xlkwfmvcv.acm-validations.aws., TTL: 60). The "DNSSEC signing" and "Hosted zone tags (0)" tabs are also visible.

Figure 109: Visualizing the API

### 3.25 Creation of Custom Domains

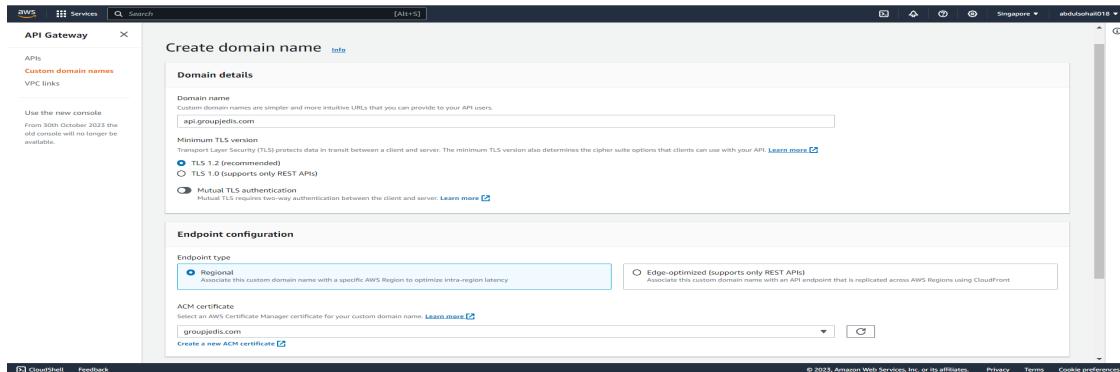


Figure 110: Creating DNS for Singapore API UI

### 3.26 Configuration Settings for the Domains

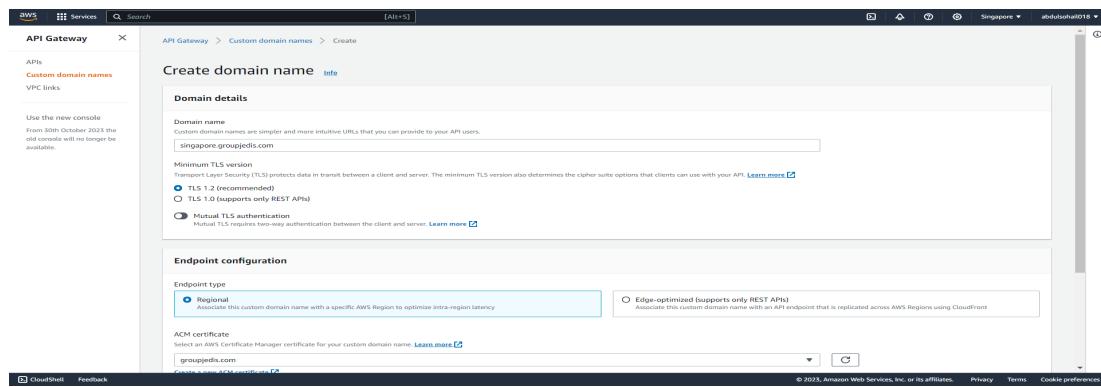


Figure 111: Creating DNS to access Singapore health check API

### 3.27 Creation of DNS Record

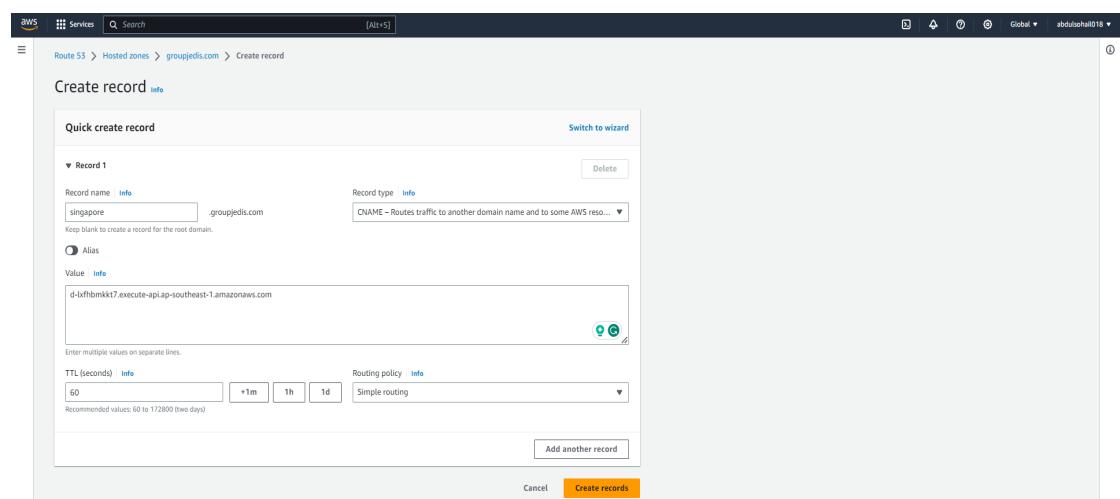


Figure 112: Creating DNS record for Singapore API

### 3.28 Creation of DNS Record

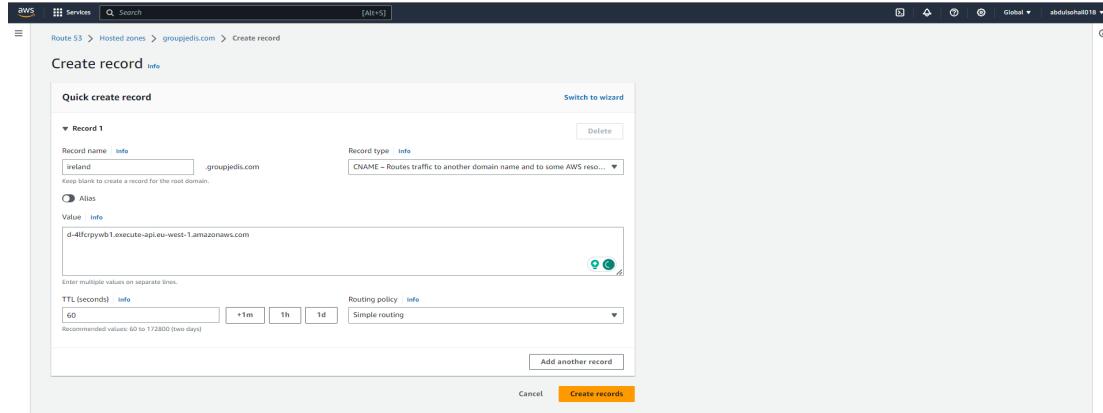


Figure 113: Creating DNS record for Ireland API

### 3.29 List of all the DNS Records

The screenshot shows the 'Hosted zone details' page for the 'groupjedis.com' zone. It lists 5 records: 1 NS record for 'groupjedis.com' with type Simple, TTL 172800, and Alias No; 1 SOA record for 'groupjedis.com' with type Simple, TTL 900, and Alias No; 2 CNAME records for 'ireland.groupjedis.com' and 'singapore.groupjedis.com' with type Simple, TTL 60, and Alias Yes; and 1 CNAME record for 'groupjedis.com' with type Simple, TTL 60, and Alias Yes. The left sidebar shows navigation options like Dashboard, Hosted zones, and Health checks.

Figure 114: Visualizing Records in Hosted Zones

### 3.30 Creation of Health Check

The screenshot shows the 'Health check concepts' page. It features two main sections: 'Availability and performance monitoring' (with a monitor icon) and 'DNS failover' (with a shield and plus icon). Both sections have 'Learn more' links. The left sidebar includes 'Create health check' and other navigation items like Dashboard, Hosted zones, and IP-based routing.

Figure 115: Creating Health checks

### 3.31 Creation of Health Check for Ireland Region

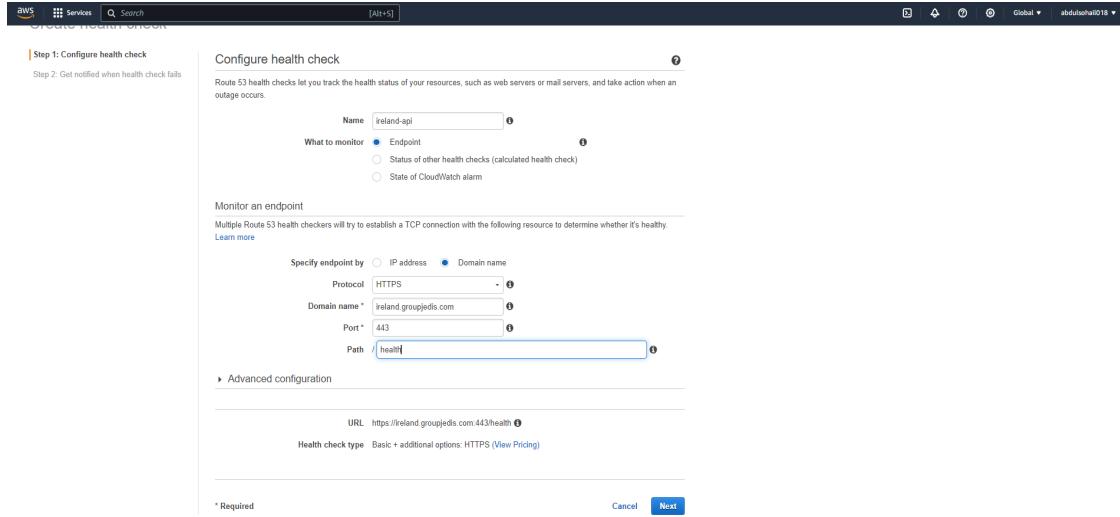


Figure 116: Creating Health Check for Ireland region with all the Configuration Settings

### 3.32 Advance Configuration Settings

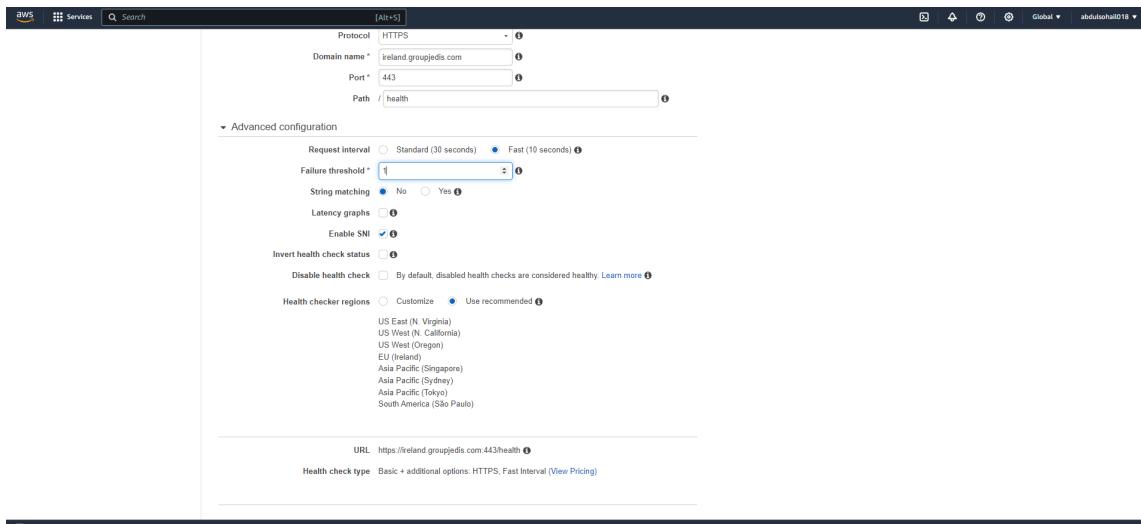


Figure 117: Modifying advanced configuration of Ireland Health Check API

### 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

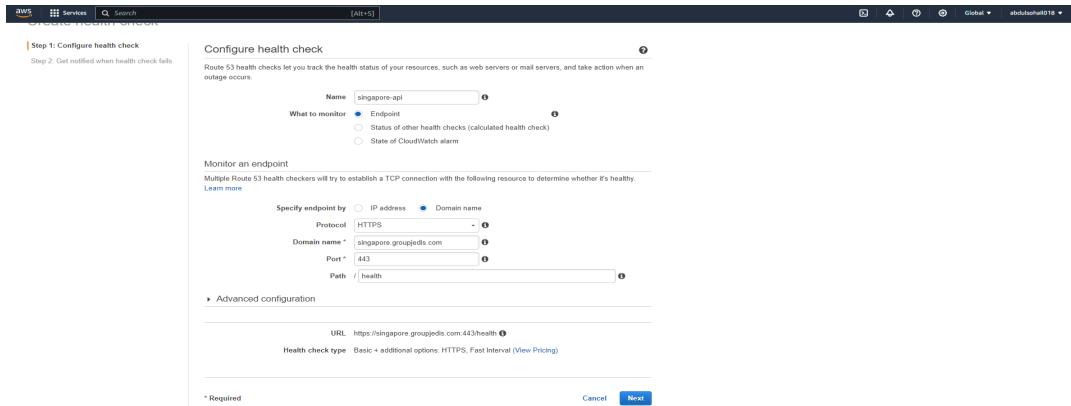


Figure 119: Creating Health Check for Singapore region

### 3.35 Results for Health Checks

The screenshot shows the 'Health checks' page in the AWS Route 53 console. It lists two health checks:

Name	Status	Description	Alarms	ID
singapore-final_api	Healthy	http://singapore.groupjedis.com:80/health	No alarms configured	1c1030b6-00d0-465d-9c34-a49e4dea503
ireland-final_api	Healthy	https://ireland.groupjedis.com:443/health	No alarms configured	b327de1-tfb7-4e0e-95e5-c1d48d7b6530

Figure 120: Health Check API is successfully created

### 3.36 Records for the Ireland Health checks

The screenshot shows the 'Hosted zone details' page for the 'groupjedis.com' hosted zone in the AWS Route 53 console. It displays the following records:

Record name	Type	Routing policy	Alias	Value/Route traffic to	TTL
groupjedis.com	NS	Simple	No	ns-190.awsdns-46.co.uk... ns-480.awsdns-60.com... ns-1381.awsdns-44.org... ns-599.awsdns-10.net...	17
groupjedis.com	SOA	Simple	No	ns-1907.awsdns-46.co.uk... ns-1907.awsdns-46.co.uk... ns-1907.awsdns-46.co.uk... ns-1907.awsdns-46.co.uk...	90
e03b5a1753869e5375d709203c563644.groupjedis.com	CNAME	Simple	No	e134e520869426dc6cf...	60
api.groupjedis.com	CNAME	Weighted	Yes	ireland.groupjedis.com...	
api.groupjedis.com	CNAME	Weighted	Yes	singapore.groupjedis.com...	
ireland.groupjedis.com	CNAME	Simple	No	d-4fcfpywb1.execute-api.eu...	60
singapore.groupjedis.com	CNAME	Simple	No	d-lsfhbmkk17.execute-api.eu...	60

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

### 3.37 Creation of DNS Record For Health Check

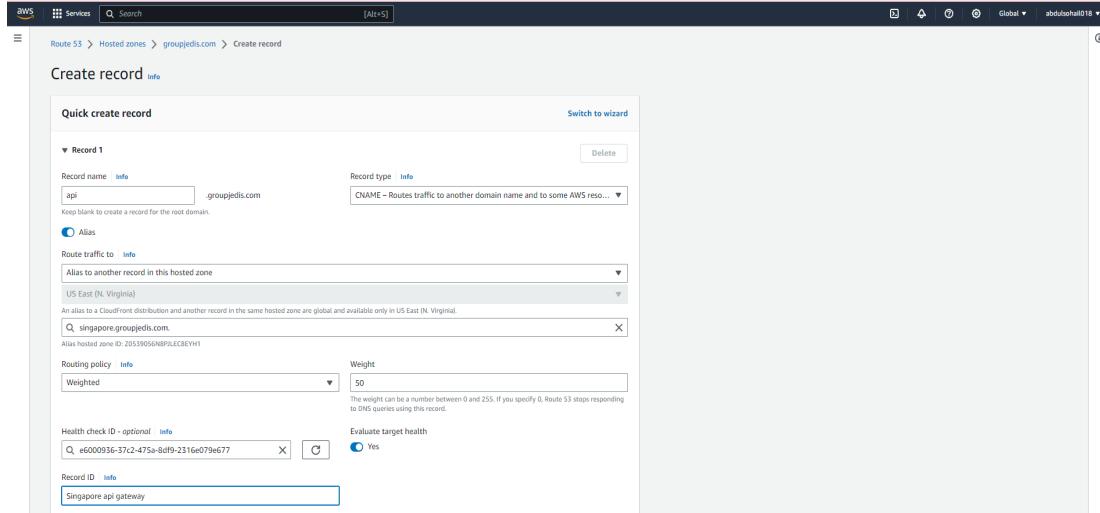


Figure 122: Creating domain record for Singapore Health Check API

### 3.38 Output Generated

```
Pretty-print   
{  
  "region": "eu-west-1",  
  "message": "Successful response reading from DynamoDB table."  
}
```

Figure 123: Visualizing record script

### 3.39 Code for Updation of UI

```
export const environment = {  
  production: true,  
  region: 'eu-west-1',  
  // TODO: make sure you have the correct region  
  cognitoIdentityPoolId: 'eu-west-1:b3f402f9-dff4-494d-a7e6-1ad309fed397',  
  // TODO: This id can be retrieved in output section of the cognito ui  
  // cloud formation stack.  
  // TODO: Facebook app id can be retrieved from the application in your  
  // Facebook developer account.  
  facebookAppId: '106581456745668',  
  // TODO: The API URL is available in the API Gateway console under Stage  
  // NOTE: don't forget trailing "/" For example:  
  // https://api.example.com/prod/  
  ticketAPI: 'https://api.groupjedis.com'  
};
```

Figure 124: Updating UI by adding API endpoints

### 3.40 Completion of Built

```

Go to Anything (Ctrl+P)
File Edit Find View Go Run Tools Window Support Preview Run
Welcome x TS environments
export const environment = {
  production: true,
}

aws -lp-172-31-37-225 x npm -lp-172-31-37-225 x Immediate Javascript (br x bash -lp-172-31-37-225 x +
> node-sass@14.1 postinstall /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2.UI/node_modules/node-sass
> node scripts/install.js
Cached binary found at /home/ec2-user/.npm/node-sass/4.14.1/linux-x64-57_binding.node
> node-sass@14.1 postinstall /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2.UI/node_modules/node-sass
> node scripts/build.js

Binary found at /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2.UI/node_modules/node-sass/vendor/linux-x64-57_binding.node
Testing binary
Binary is fine
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node modules/chokidar/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node modules/watchpack-chokidar2/node_modules/chokidar/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"linux","arch":"x64"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node modules/chokidar/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node modules/karma/node_modules/chokidar/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ node-sass@14.1
added 71 packages in 15.345s
ec2-user~/environment/MultiRegion-Serverless-Workshop/2.UI (master) $ npm run build
> srx-ui@0.0.0 build /home/ec2-user/environment/MultiRegion-Serverless-Workshop/2.UI
> ng build

Date: 2023-10-04T20:27:34.990Z
Hash: f9f4e69910k81ab2afc
Time: 1602ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 41.2 kB [vendor] [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 228 kB [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 574 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 3 MB [initial] [rendered]
chunk {views} views.module.chunk.js, views.module.chunk.js.map () 396 kB [main] [rendered]
ec2-user~/environment/MultiRegion-Serverless-Workshop/2.UI (master) $ 

```

Figure 125: Build successful

### Step 5: Test Failover:

#### 1. Breaking the Primary Region

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

#### 2. Result of the Test Failover

Name	Status	Description	Alarms	ID
ireland-api-total	Unhealthy	https://ireland.groupjedis.com:443/health	No alarms configured.	ace9b492-a6c3-446b-ab9-7bf44f3cf0a1

Figure 128: Break successfully done

### 3. Code to get the Response for the Read Operation

```
Pretty-print   
  
{  
  "region": "ap-southeast-1",  
  "message": "Successful response reading from DynamoDB table."  
}
```

**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>