8/14/2023

ESDE CA2 Report

Bee Design Website Integrated With AWS

Name: Tan Yong Kit, Eden

StudentID: P2243605

Class: DIT/FT/2A/06

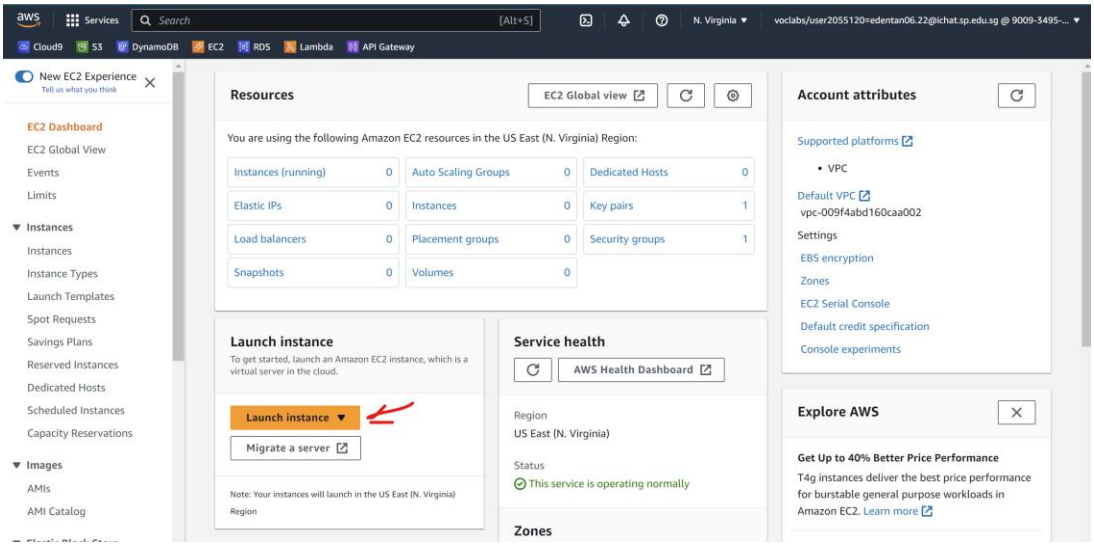
Table of Contents

Part 1 – Set up EC2 Instance on AWS

1.1 Creating an EC2 Instance

Step 1:

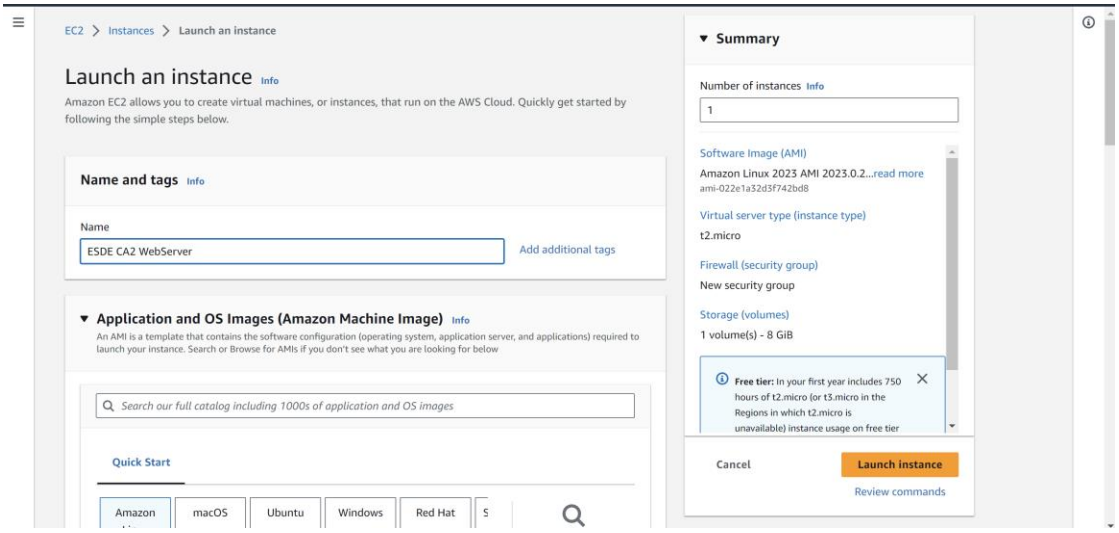
Go into the EC2 Instance Dashboard and click on “Launch Instance” to start a new EC2 Instance.



1.2 Configure EC2 Instance Settings

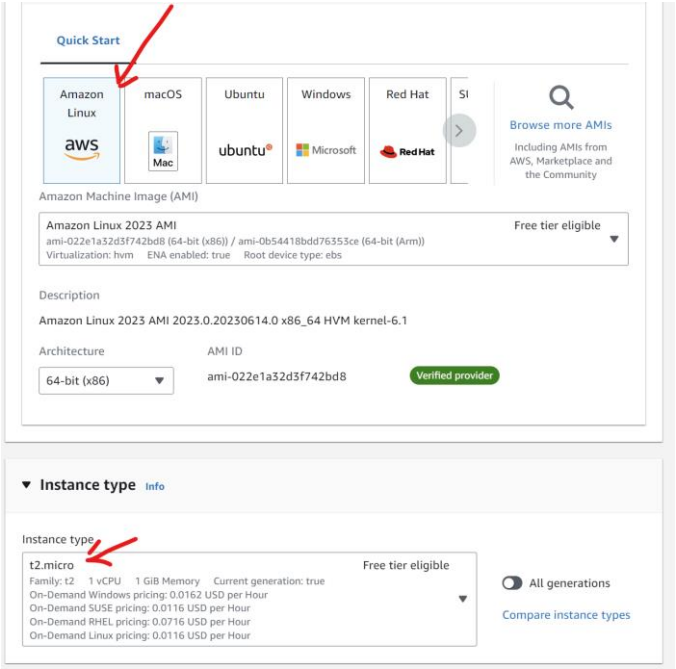
Step 1:

Add a name for your EC2 Instance.



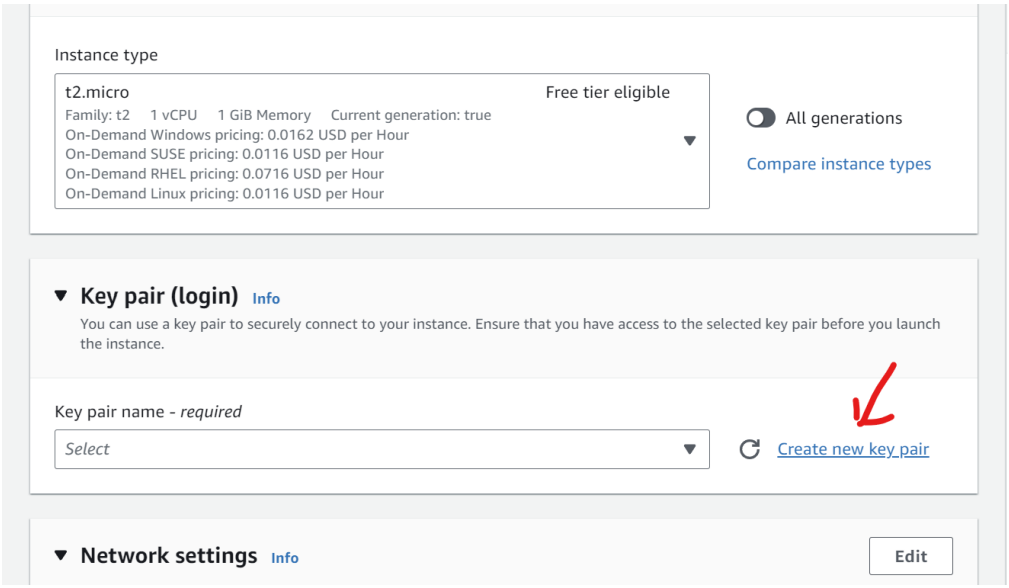
Step 2:

Scroll down and make sure that you have the following default configurations selected, as they are one of the cheapest cost that AWS offers.



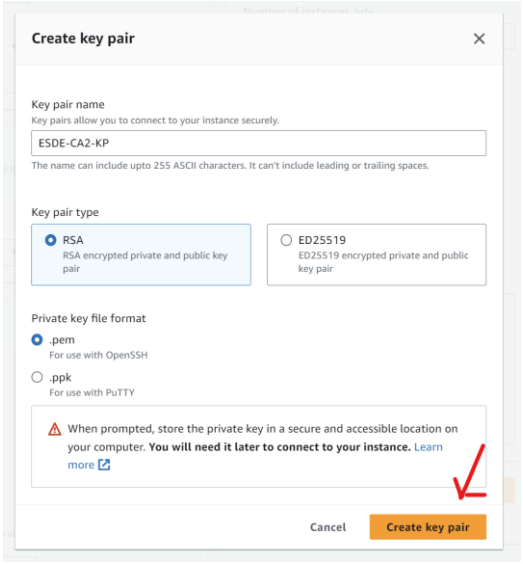
Step 3:

Create a new Key-Pair for you EC2 Instance, click on “Create new key pair”.



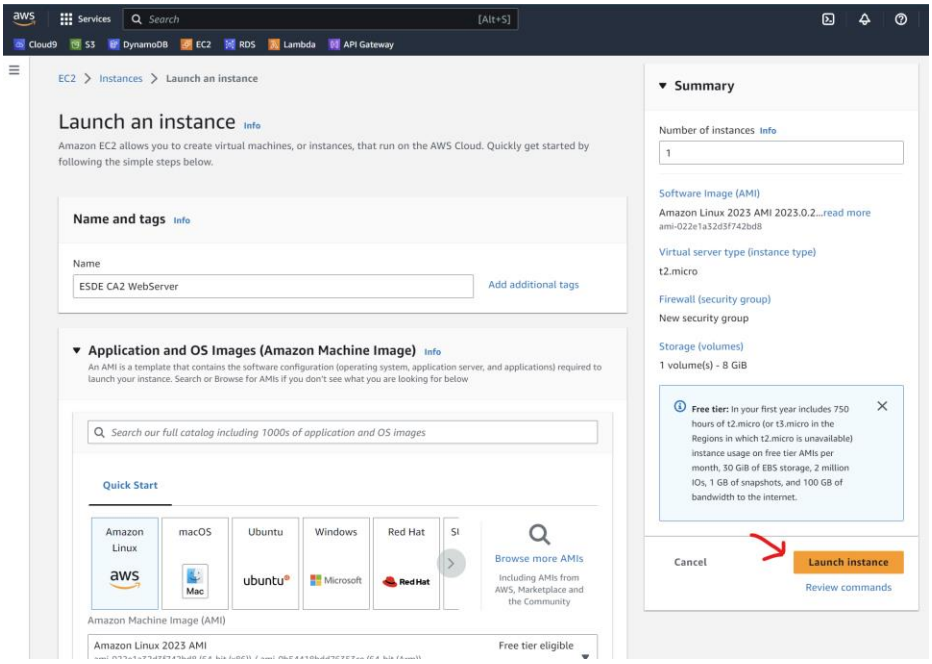
Step 4:

Click on “Create Key Pair” after adding your Key-Pair Name below and save the .pem file at a directory of your choice.



Step 5:

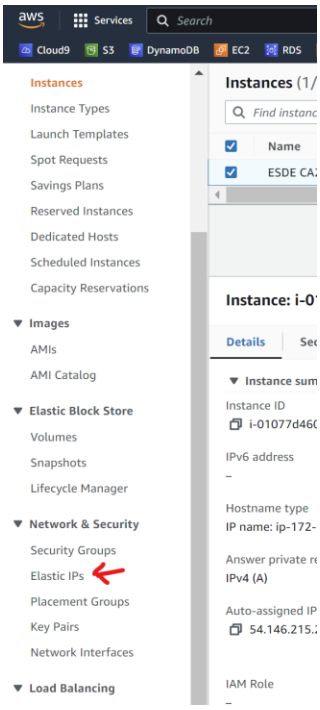
Finally click on the “Launch Instance” on the right side to finish creating your EC2 Instance.



1.3 Creating an Elastic IP Address for your Web Server

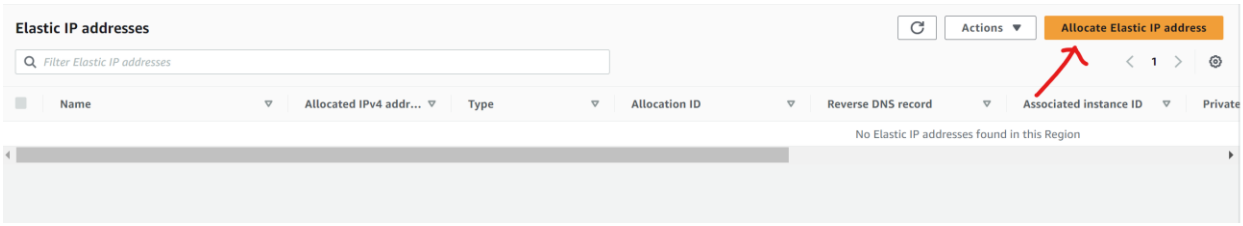
Step 1:

Click on the Elastic IPs on the left side.



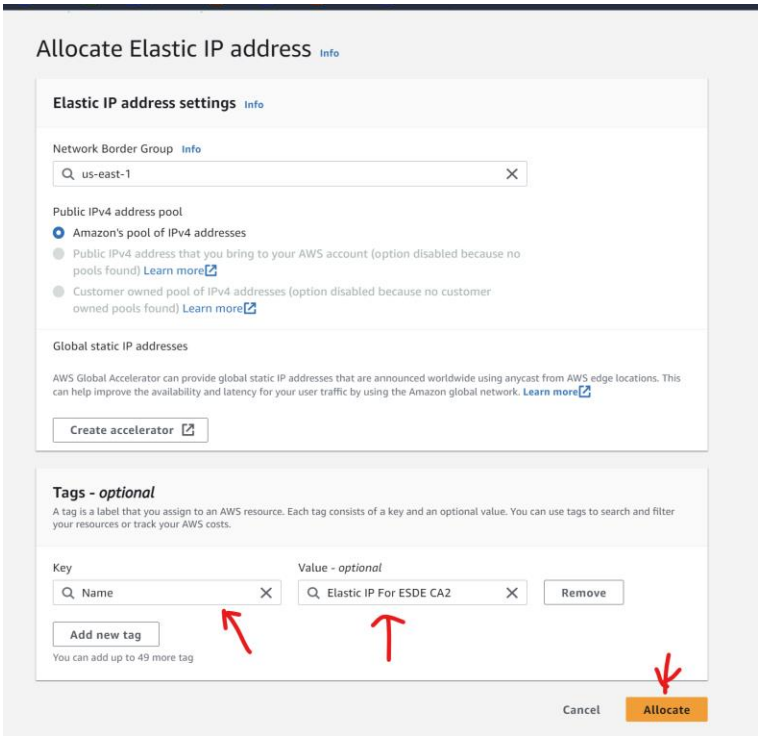
Step 2:

Click on “Allocate Elastic IP address” to create a Elastic IP address.



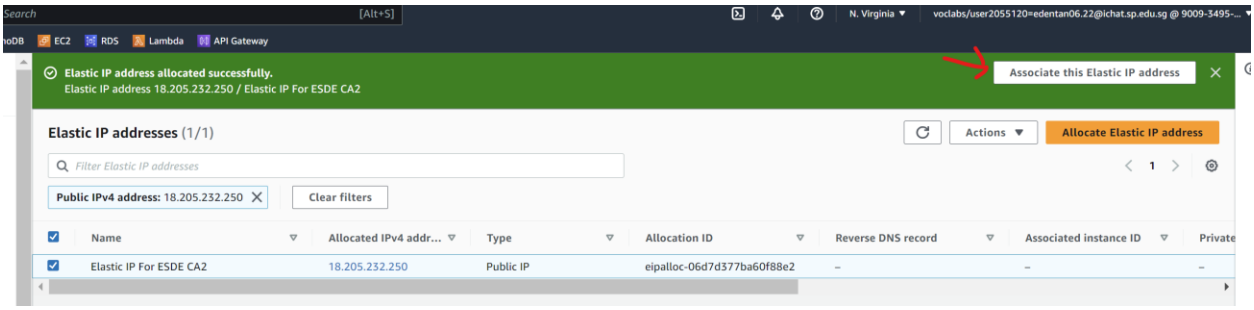
Step 3:

Create a new tag name as shown below and click on the “Allocate” button to create an Elastic IP address.



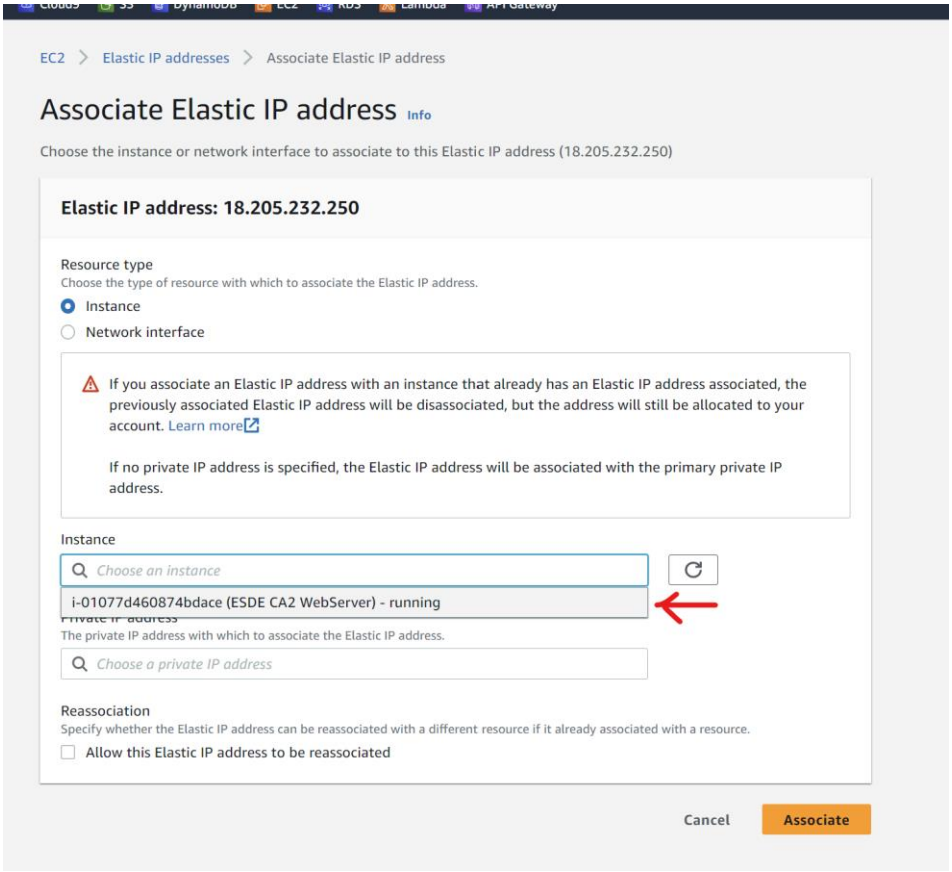
Step 4:

Next, click on the “Associate this Elastic IP Address” on the top right to associate this Elastic IP Address to your EC2 Instance.



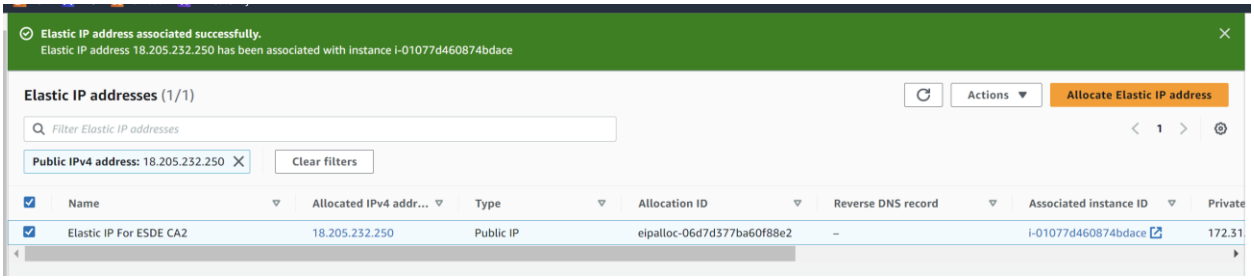
Step 5:

Next, Choose the current running instance and associate the Elastic IP Address we created to it and then click on the “Associate” button to associate it.



Step 6:

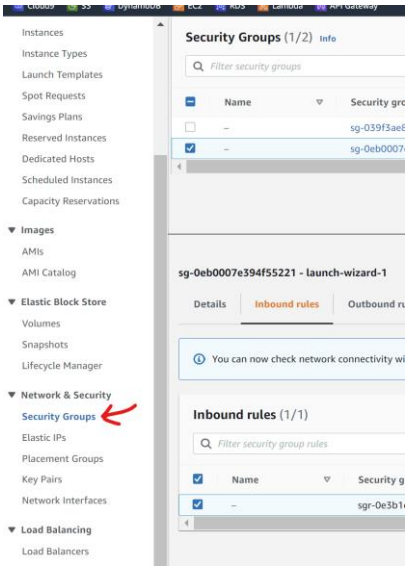
You should see the following Successful Message indicating that the Elastic IP Address has been associated with our EC2 Instance.



1.4 Setting the Inbound Rules for our Ports 3001 and 5000

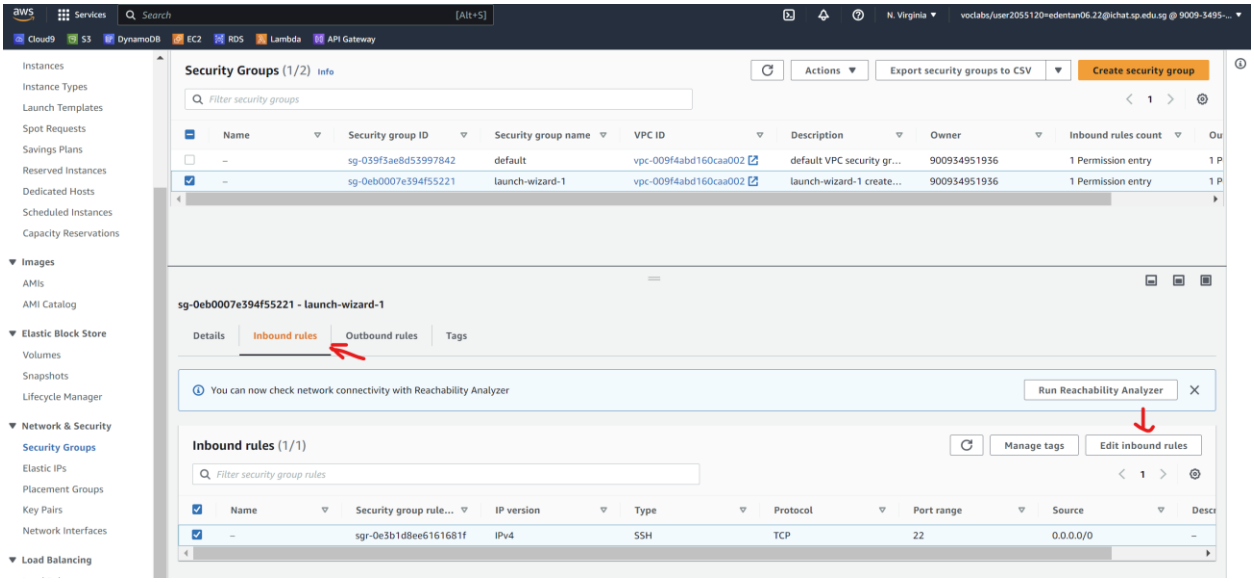
Step 1:

Go over to Security Groups under Network and Security



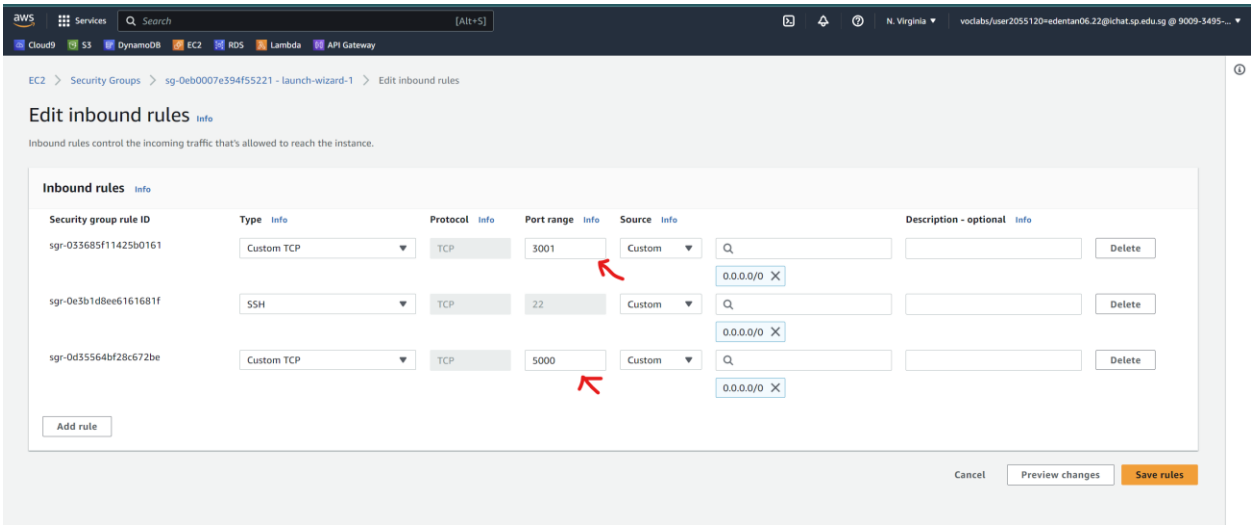
Step 2:

Select Inbound Rules and click on the “Edit inbound rules” button.



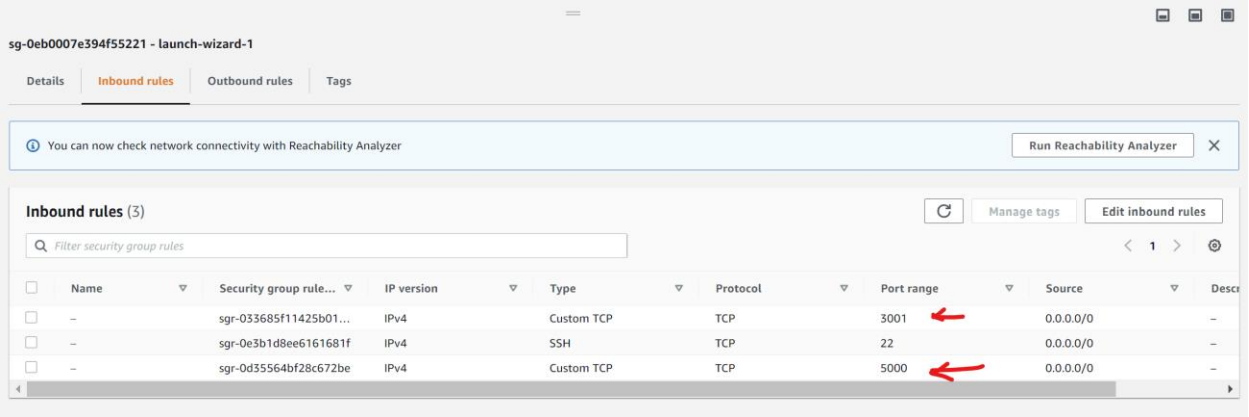
Step 3:

Add the following 2 new rules, Ports 3001 and 5000 as shown below. And click on the “Save rules” button.



Step 4:

You should be able to see the added rules under the Inbound Rules as shown below.



1.5 Download the following applications

Step 1:

Install Putty and WinSCP using the given links below.

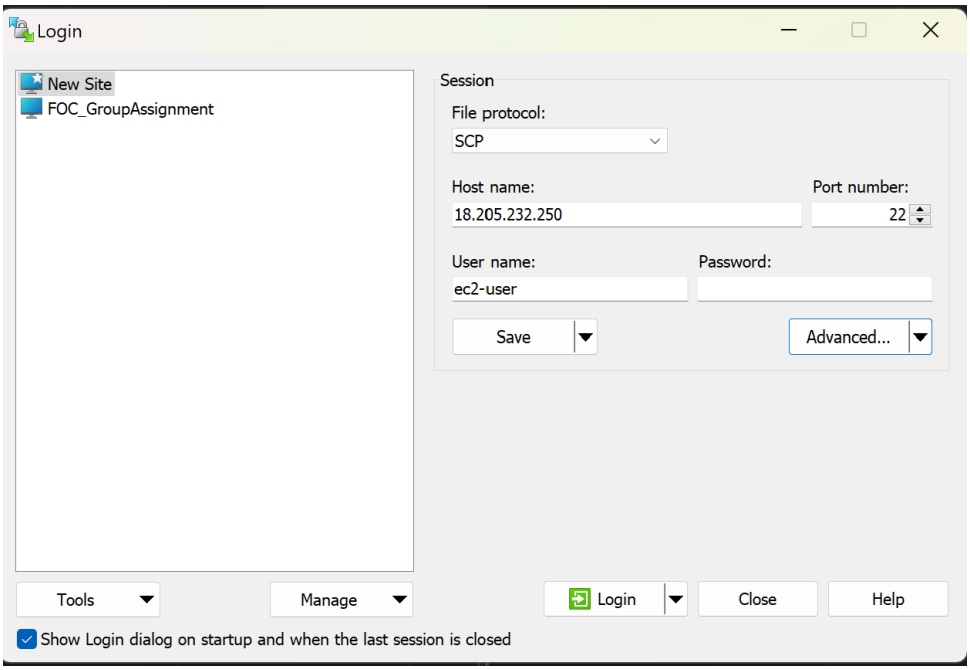
Putty: https://the.earth.li/~sgtatham/putty/latest/w64/putty-64bit-0.78-installer.msi

WinSCP: https://winscp.net/eng/download.php

1.6 Using Putty and WinSCP for our EC2 Web Server

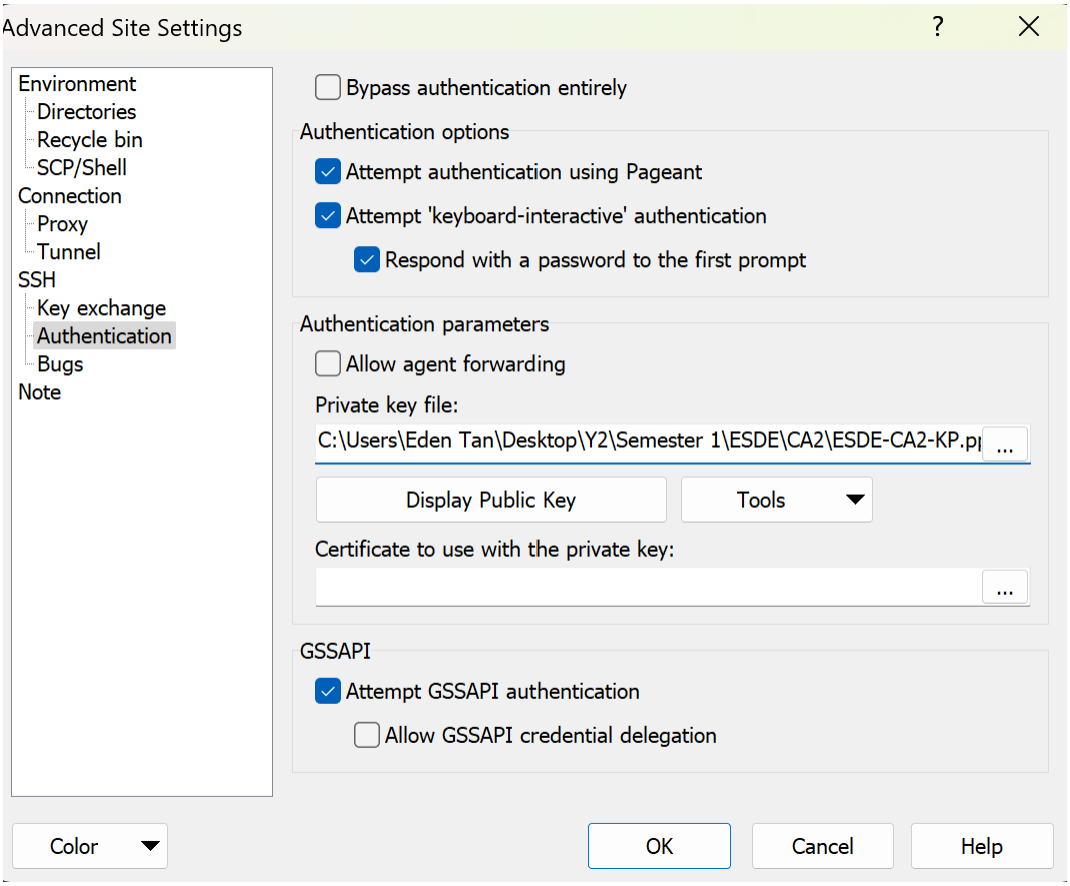
Step 1:

Open up WinSCP and you should see the following below, add in the following details as shown.

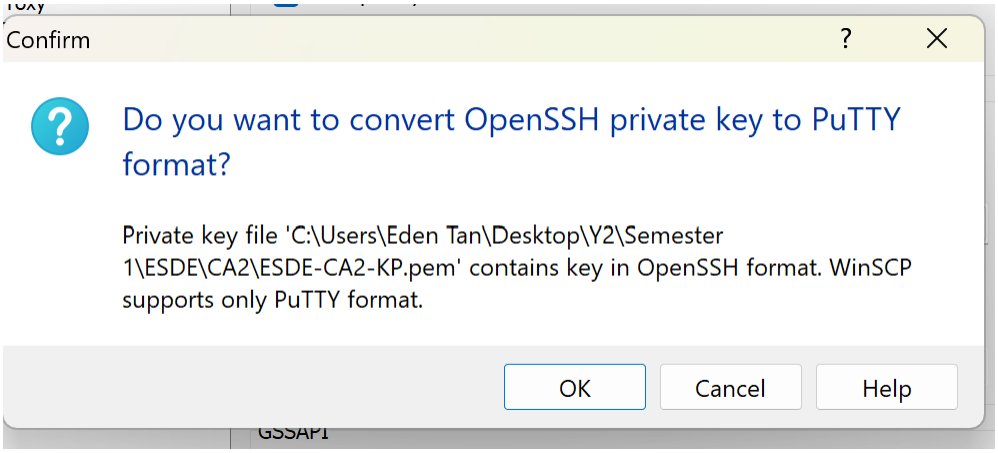


Step 2:

Click on the “Advanced…” button to add in your .pem Key-Pair that you created previously under SSH > Authentication.

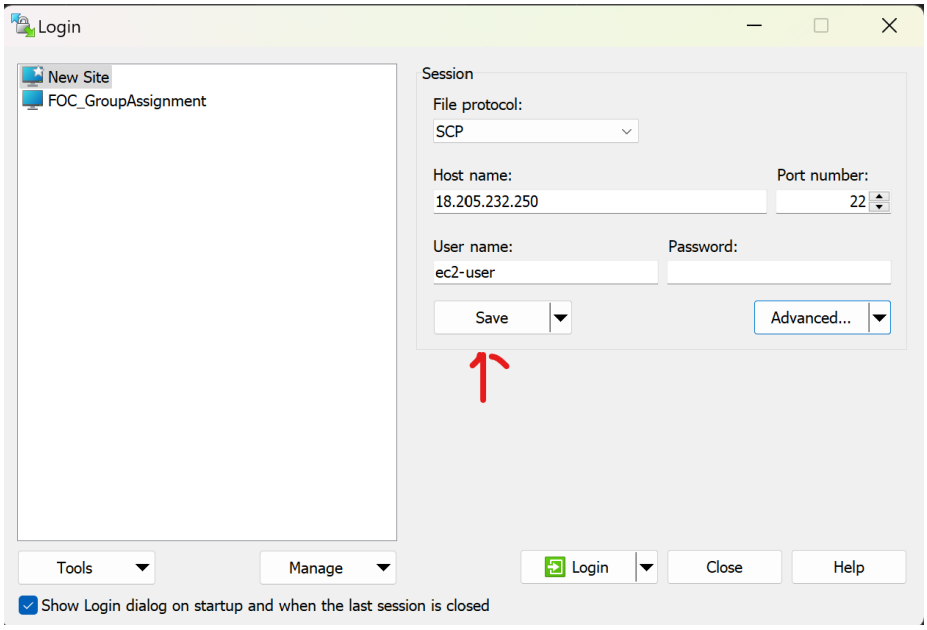


*Note: You will be prompted to convert your .pem key to Putty Format, click on “ok” to proceed.



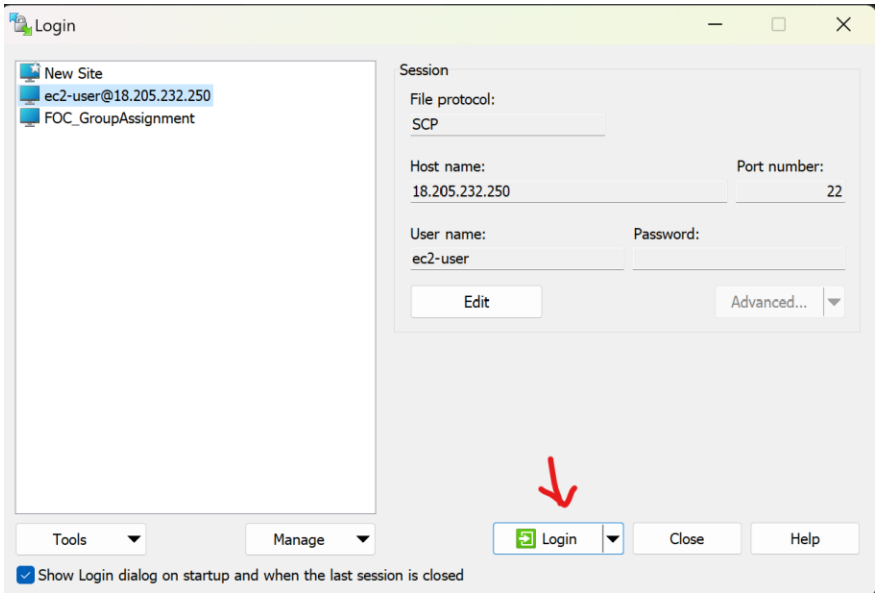
Step 3:

Once done, click “save” and “ok” in the pop up.

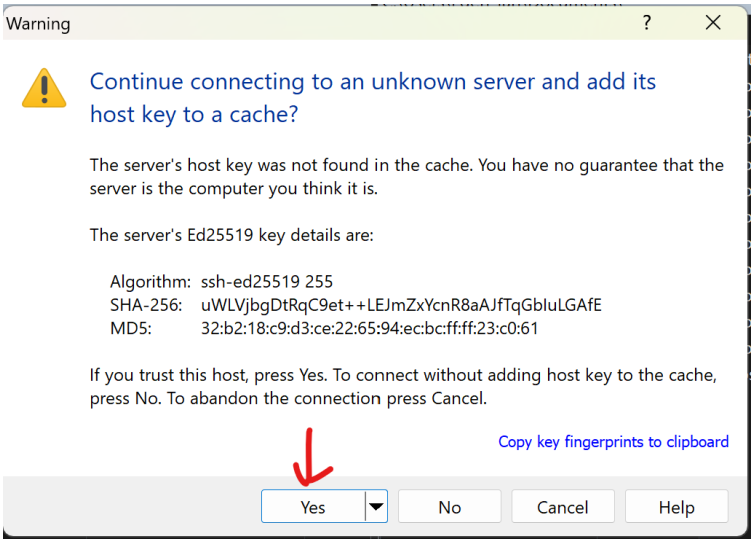


Step 4:

Click on “Login” to see if you can access the EC2 Server.

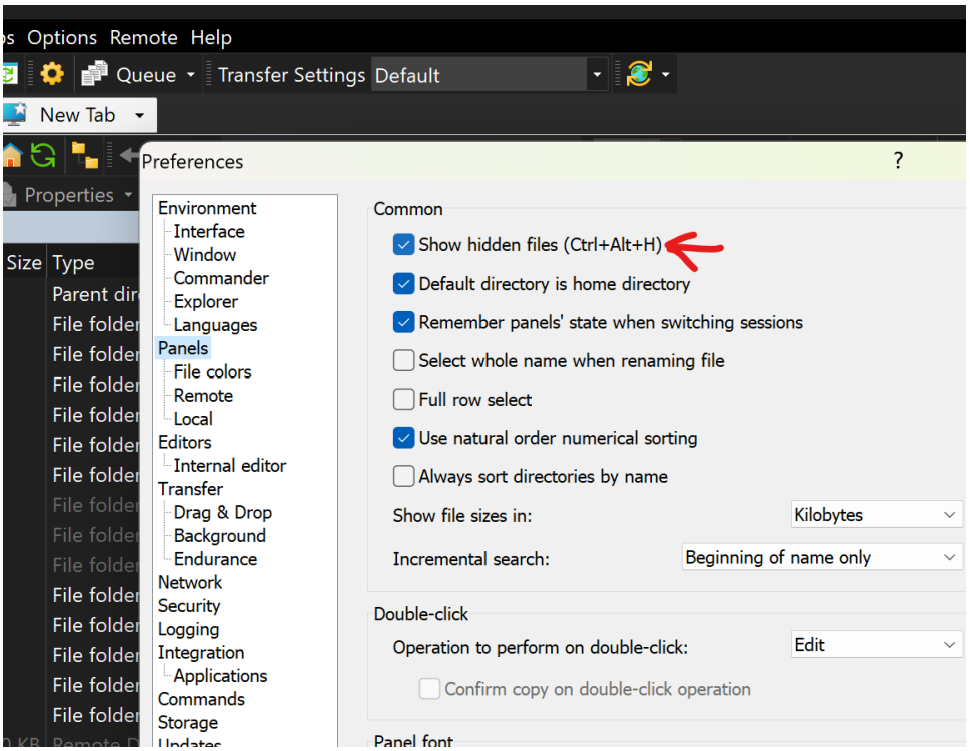


*Note: You will see the following warning, click on “Yes” to continue



Step 5:

Finally, at the top tabs go to Options > Preferences > Panels and check “Show hidden files”

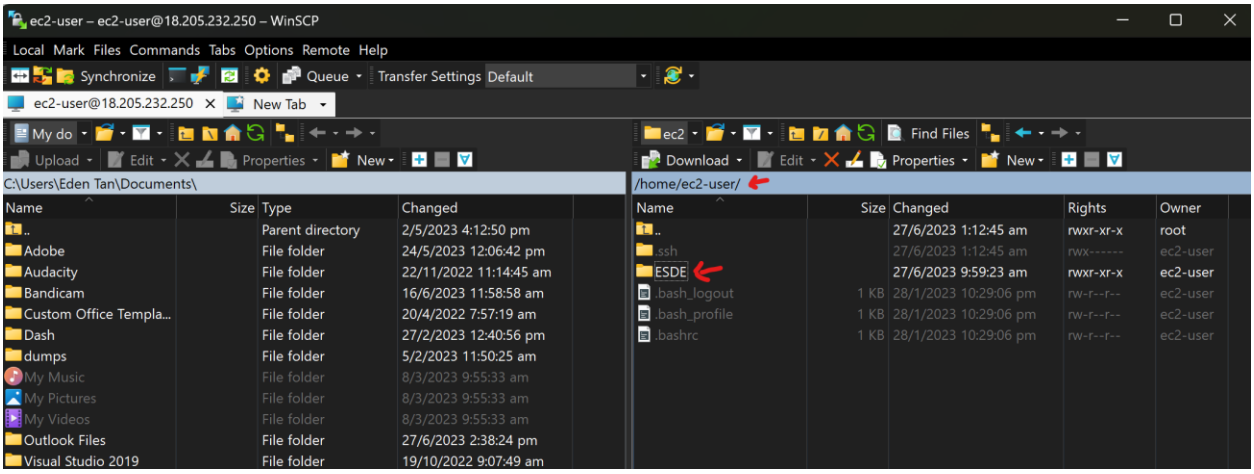


Part 2 – Install NodeJS and copy Bee Design app to EC2

2.1 Creating the folder directories for Bee Design

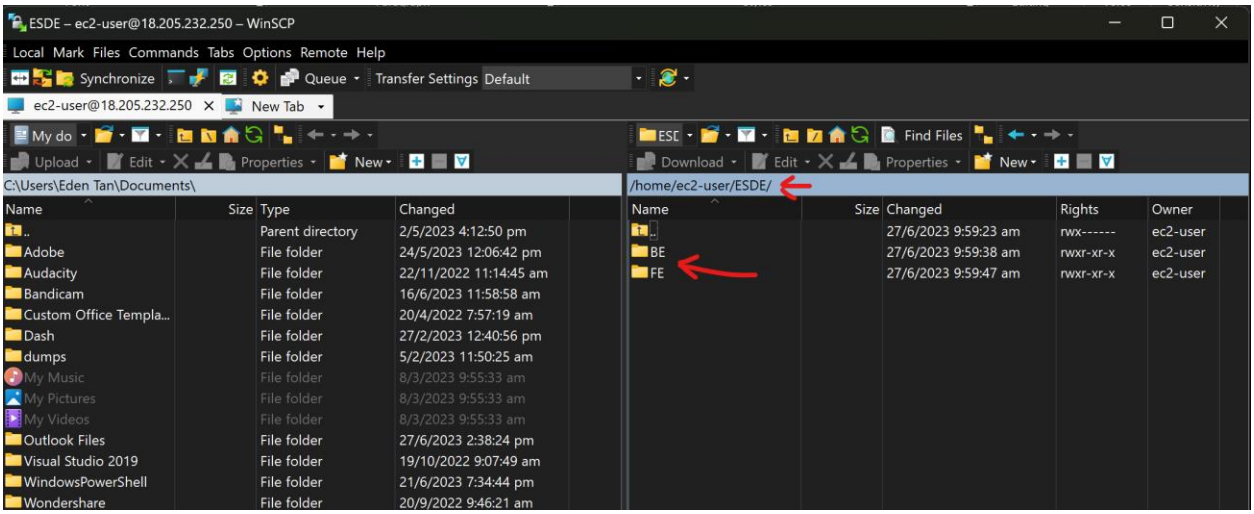
Step 1:

Go to /home/ec2-user/ and create a directory called ESDE.



Step 2:

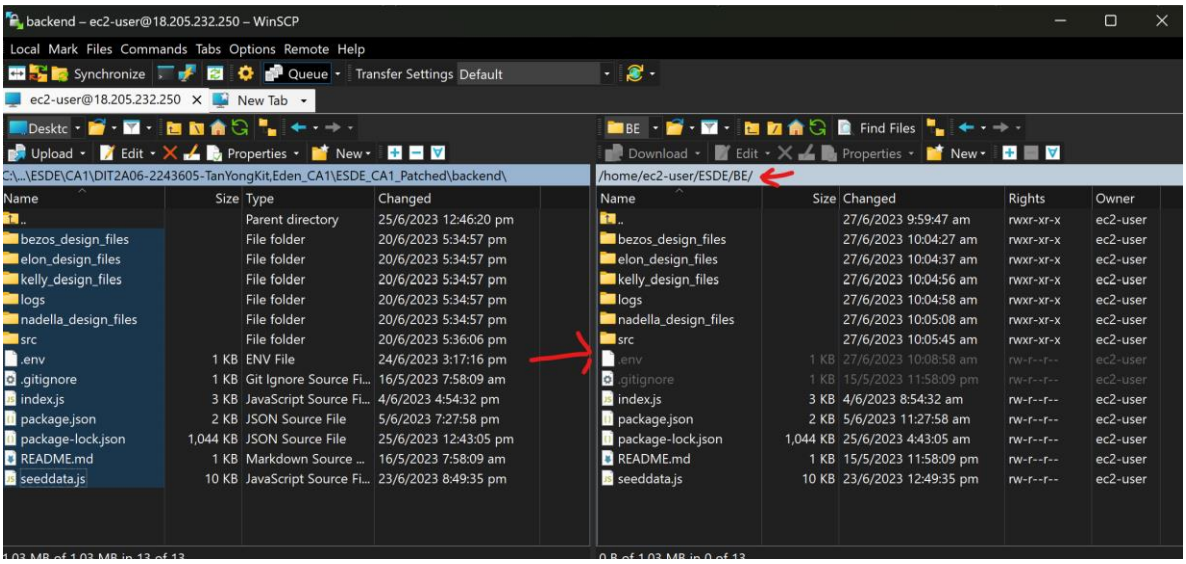
Next, go into the ESDE Directory and create 2 more directories BE and FE respectively.



2.2 Bringing respective Backend and Frontend Bee Design Folders to ESDE Folder

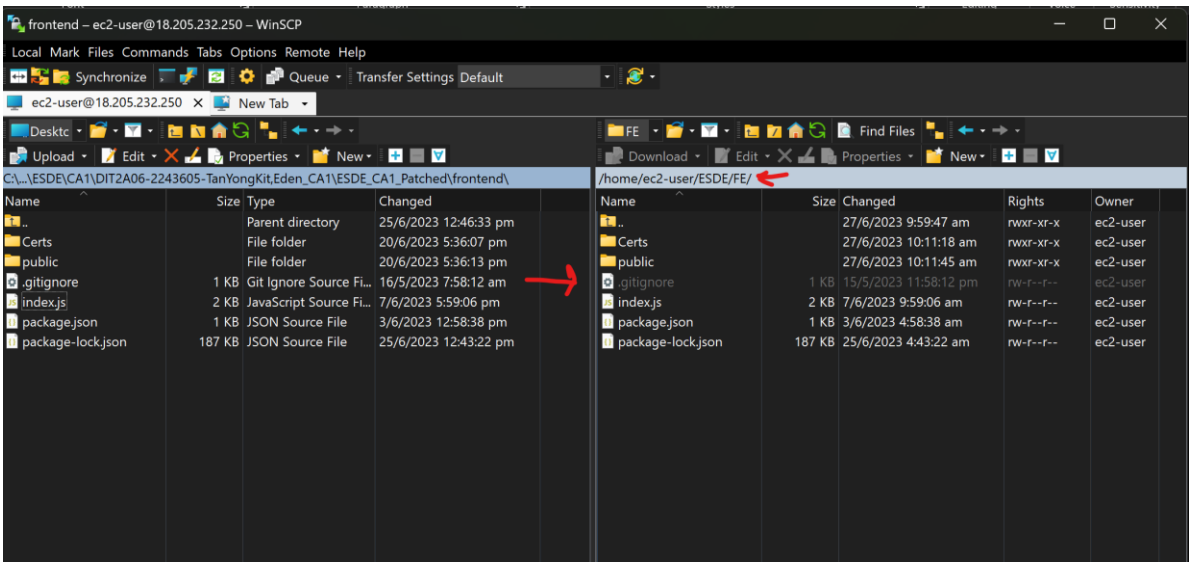
Step 1:

Go to the directory of your CA1 Bee Design Folder and drag the files in the ESDE_CA1_Patched\backend\ into the ESDE/BE Folder.



Step 2:

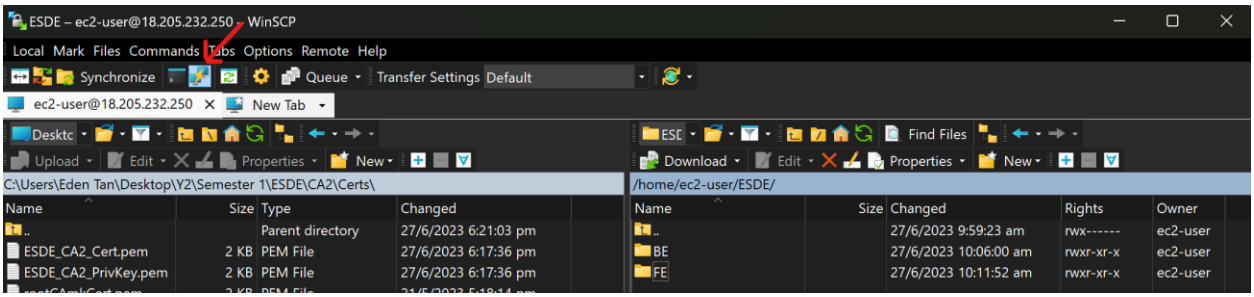
Go to the directory of your CA1 Bee Design Folder and drag the files in the ESDE_CA1_Patched\frontend\ into the ESDE/FE Folder.



2.3 Set up and install node modules and NodeJS with Putty

Step 1:

Click on the Open session with Putty Icon at the top to open a session terminal

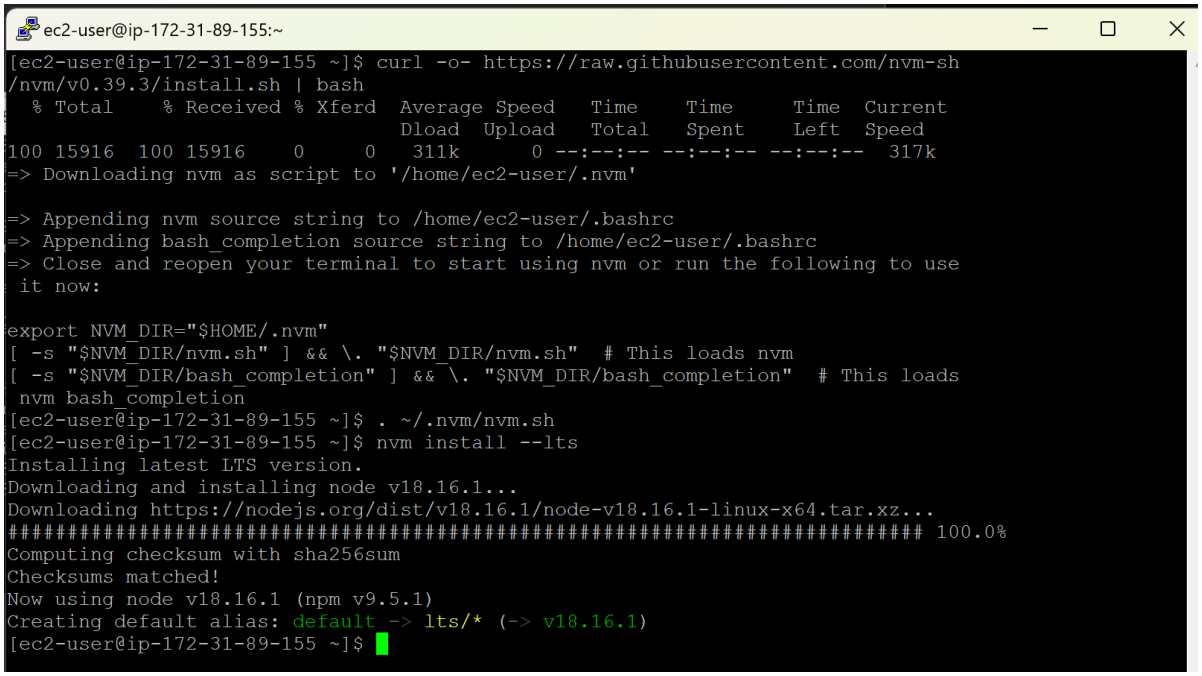


Step 2:

Next, type in the following in order into the terminal to install NodeJS.

1. curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
2. . ~/.nvm/nvm.sh
3. nvm install –lts

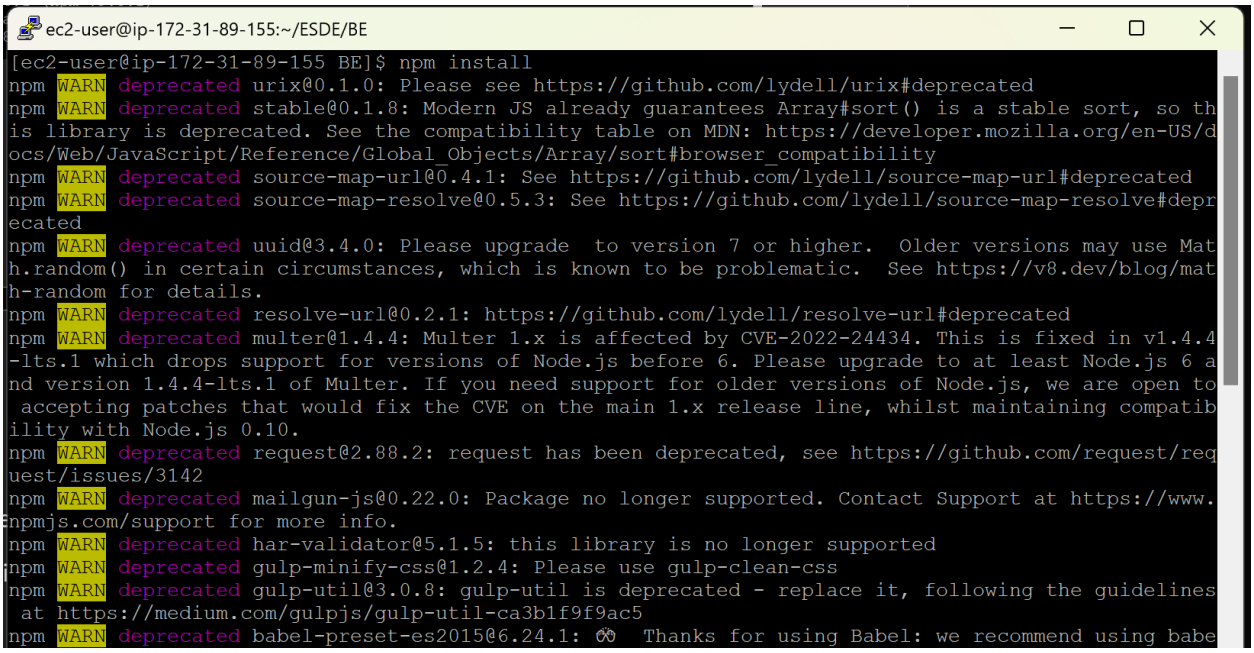
Reference: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html



Step 3:

Cd into the backend directory ESDE/BE and run the following command.

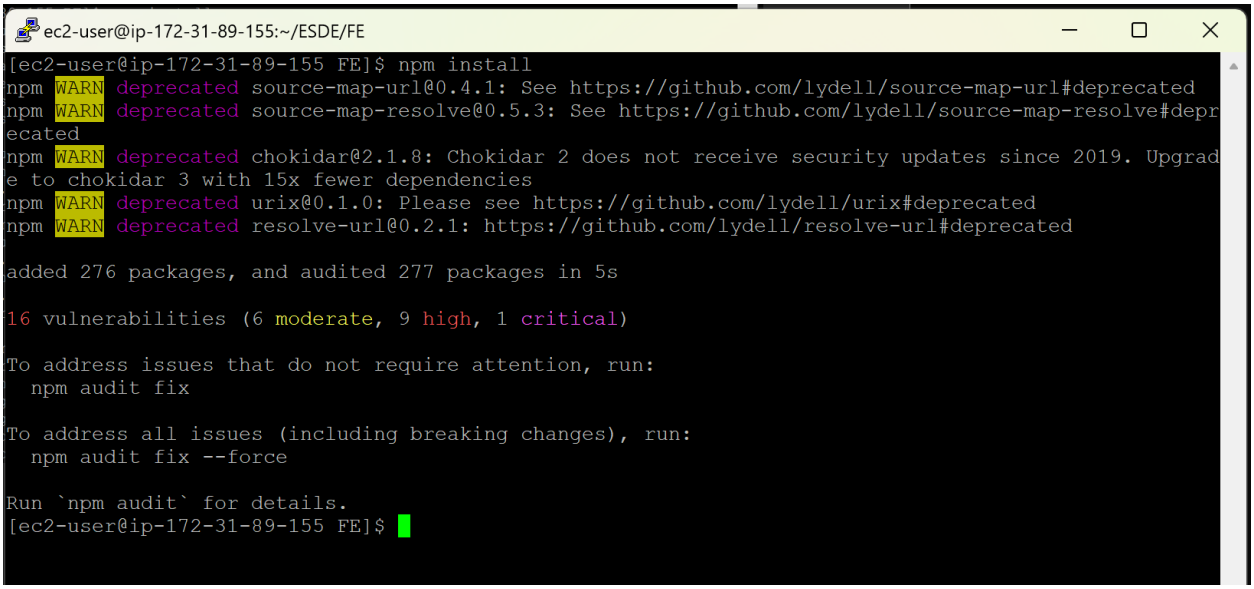
Command: npm install



Step 4:

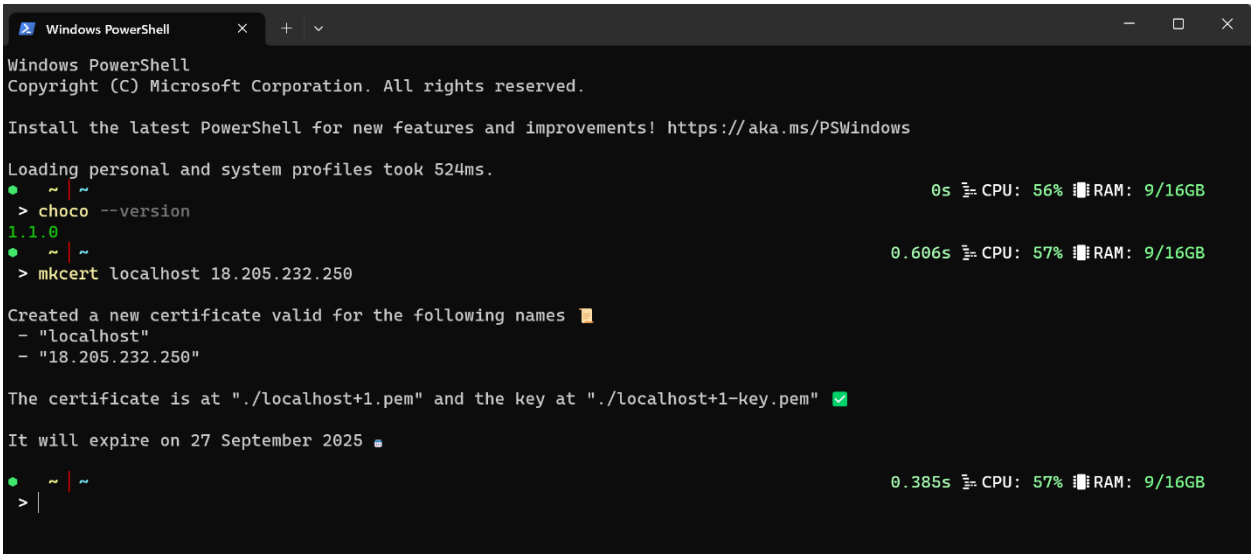
Cd into the frontend directory ESDE/FE and run the following command.

Command: npm install



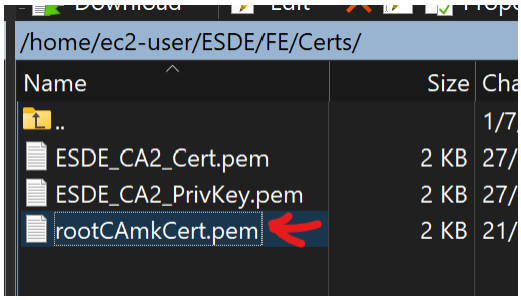
Step 5:

Create a new SSL Cert using Mkcert in your command prompt and bring it into the respective Backend and Frontend folders in the WinSCP with the elastic ip address created in EC2.



*Note: **Remember to trust mkcert in your trusted SSL certificates** in your browser.

Import the rootCAmkcert.pem into the trusted SSL Certificates.



Step 6:

In the frontend folder under index.js, add in the credentials to read the key and cert file created and change the .listen function by **removing the hostname** to the following below as we are no longer using localhost and save it.

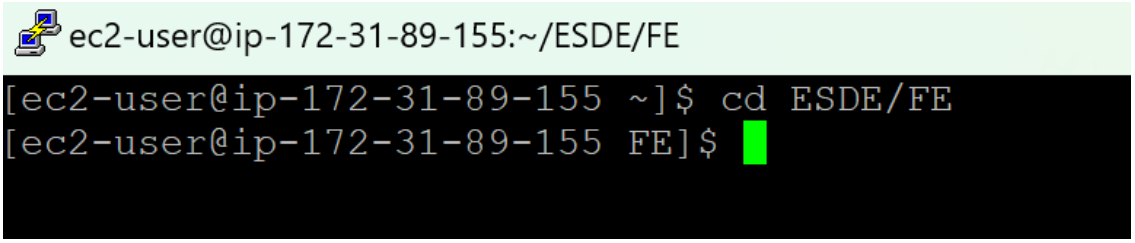
```
//Read private key and certificate for HTTPS ~ Sensitive Data Exposure Prevention
const credentials = {
  key: fs.readFileSync("./Certs/ESDE_CA2_PrivKey.pem"),
  cert: fs.readFileSync("./Certs/ESDE_CA2_Cert.pem"),
  requestCert: false,
  rejectUnauthorized: false
};
//End of Sensitive Data Exposure Prevention
```

```
// Create an HTTPS server with the 'credentials' for a secure connection
const httpsServer = https.createServer(credentials, app);
httpsServer.listen(port, function () {
  console.log(`Server hosted at https://18.205.232.250:${port}`);
});
```

*Note: Do the same for the backend folder index.js and change the localhost to your Elastic IP address.

Step 7 :

CD in the FE folder in the terminal to test if your site is currently working.



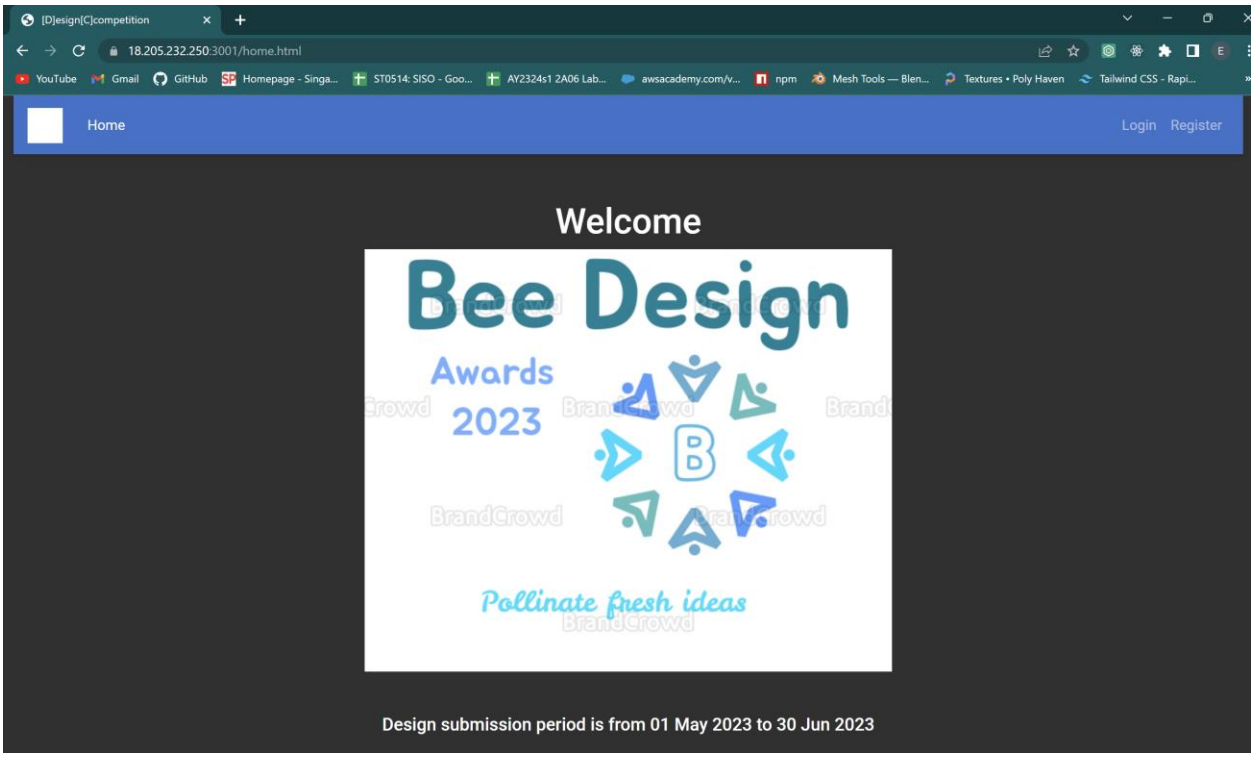
Step 8:

Run node index.js and go to the link to see if your front end site is working.

```
ec2-user@ip-172-31-89-155:~/ESDE/FE
[ec2-user@ip-172-31-89-155 ~]$ cd ESDE/FE
[ec2-user@ip-172-31-89-155 FE]$ node index.js
Server hosted at https://18.205.232.250:3001
```

Step 9:

If all is set up correctly, you should be able to see your Front end site in your browser. When you type https://18.205.232.250:3001/home.html.

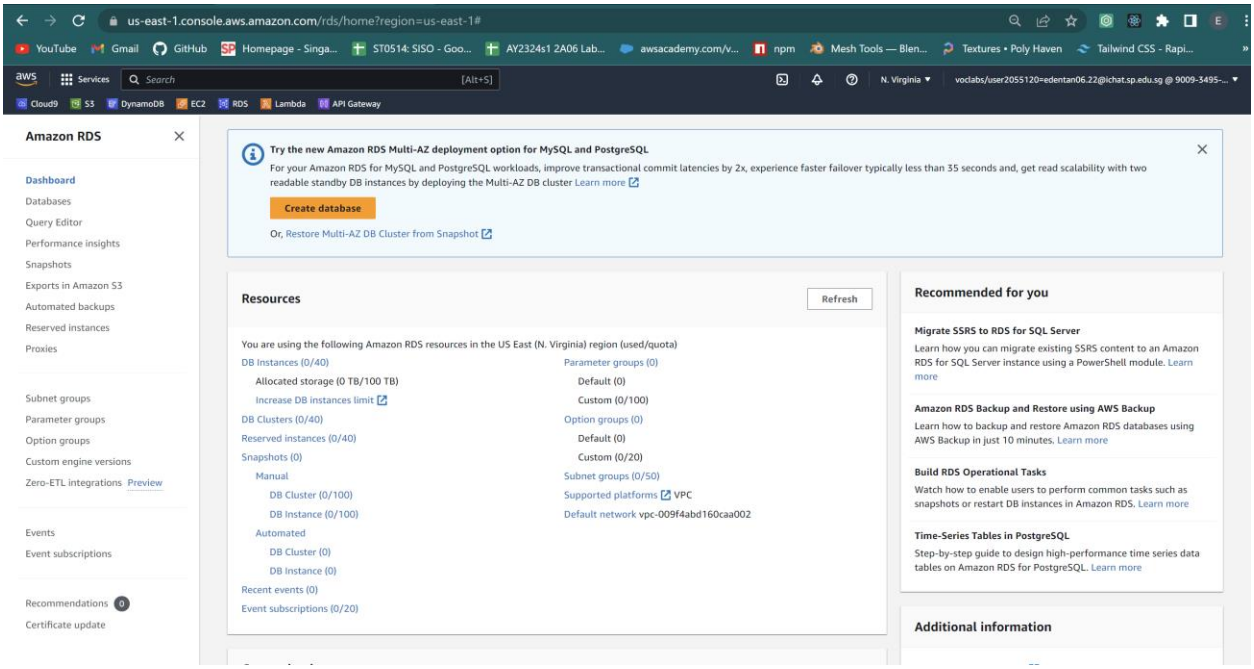


Part 3 – Set up RDS

3.1 Creating an AWS RDS

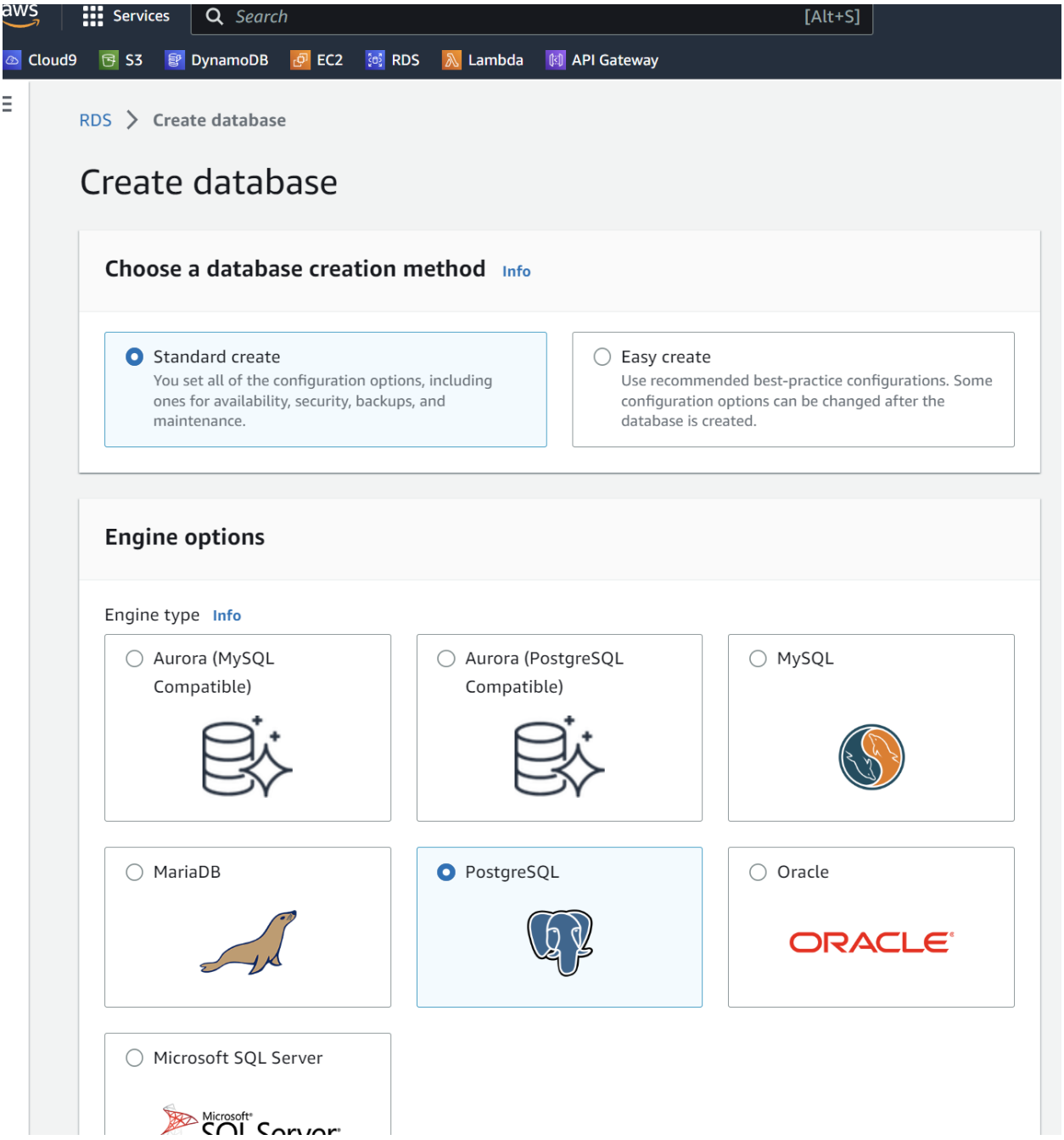
Step 1:

Select the RDS Service by AWS and go to the dashboard as shown below.



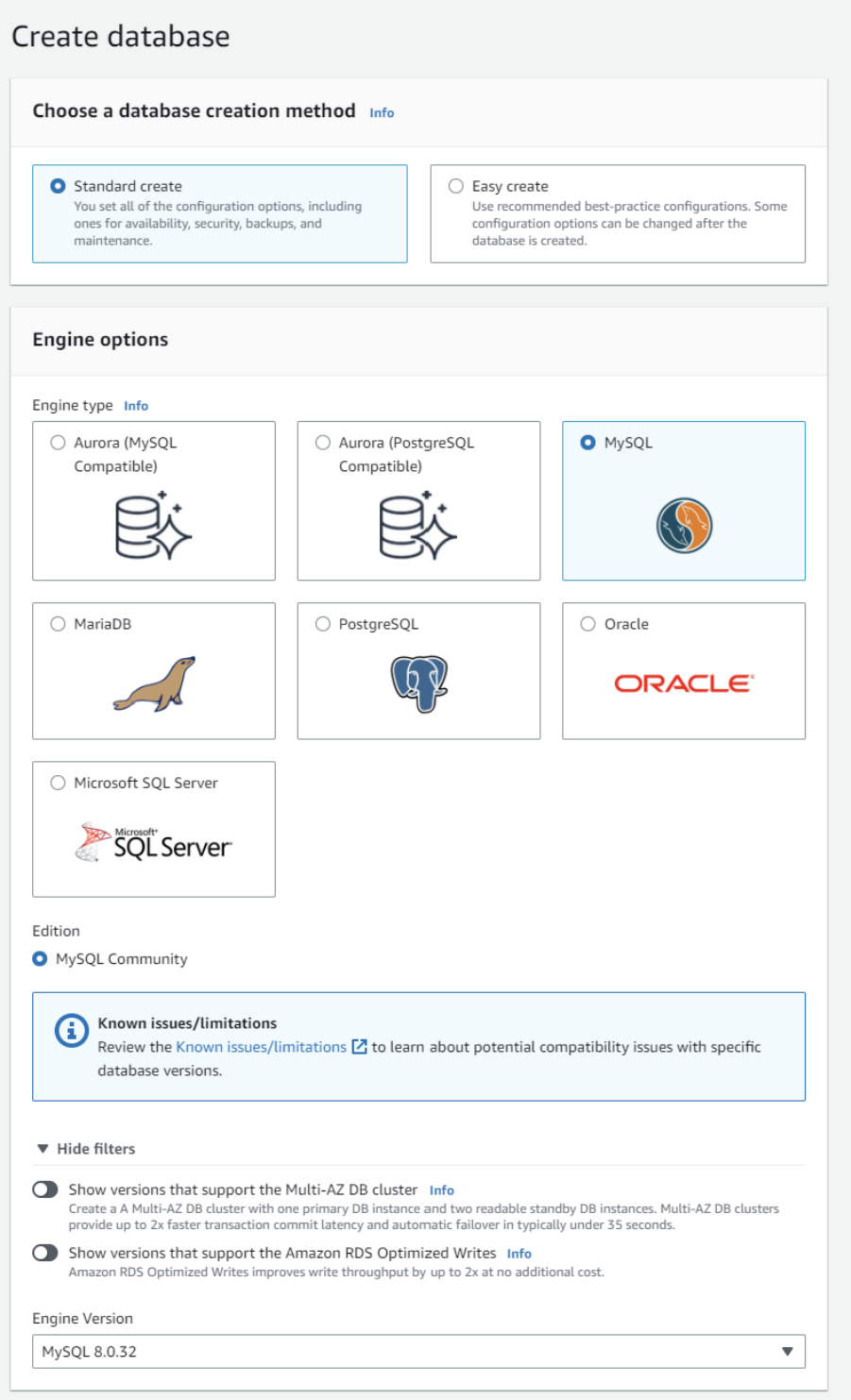
Step 2:

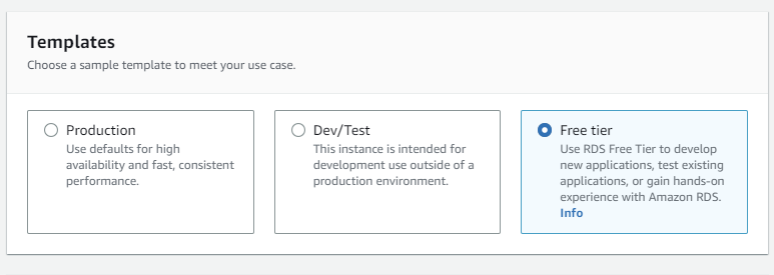
Click on the orange button “Create database” to create a new database you should see the following as shown below.



Step 3:

Make sure that you **choose Standard create**, under Engine options **choose MySQL**, under Templates **choose Free tier**.



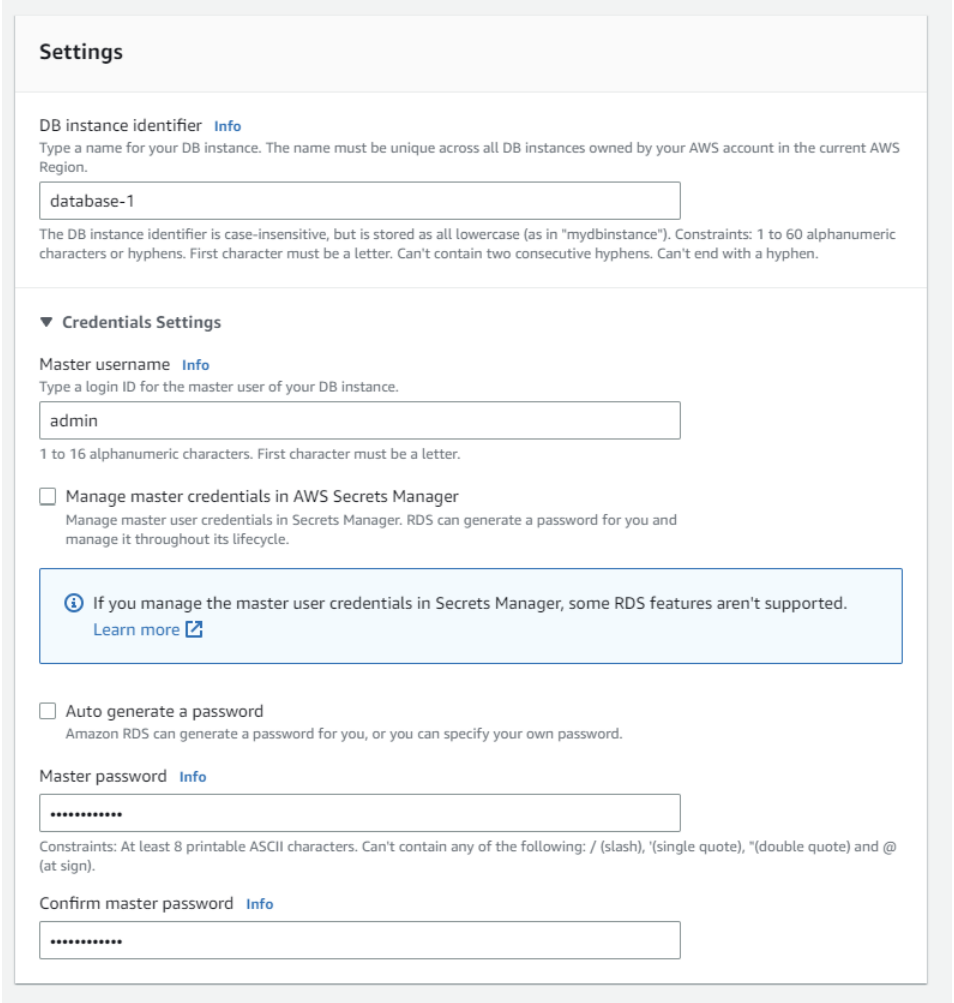


Step 4:

For the settings follow the image shown below.

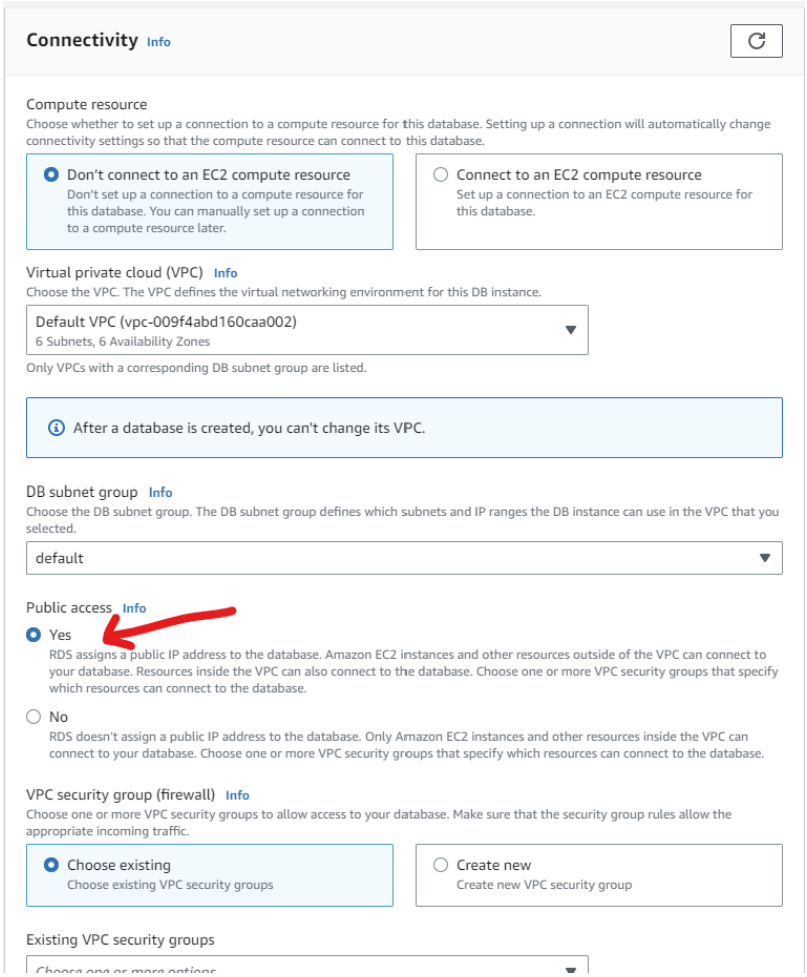
Password: Edentan2003!

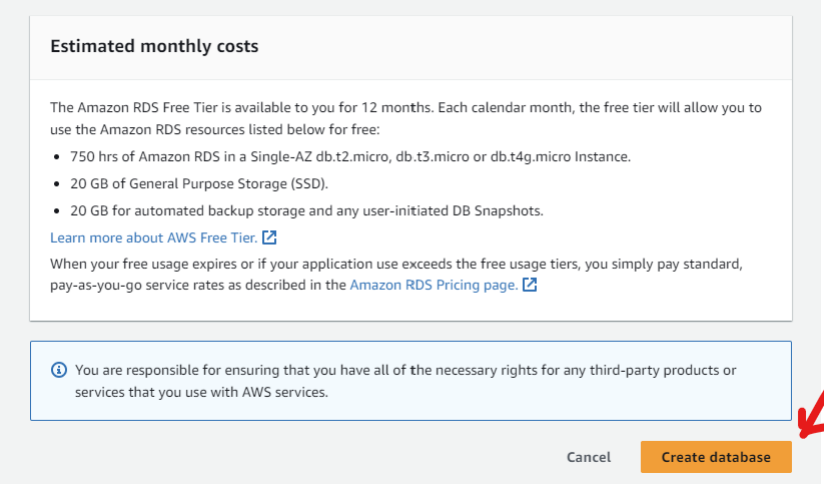
*Note: For the password we will use it when we go to MySQL Workbench later.



Step 5:

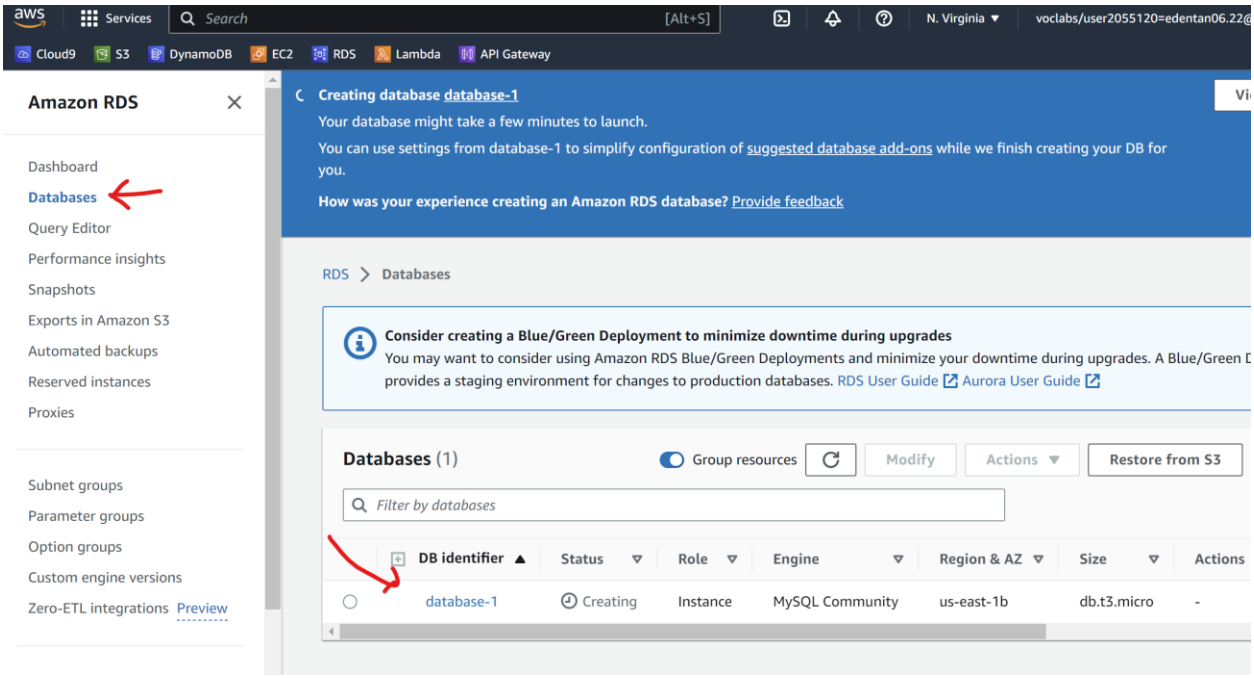
Scroll down to Connectivity, select “Yes” for public access as shown below and click on “Create database” at the bottom.





Step 6:

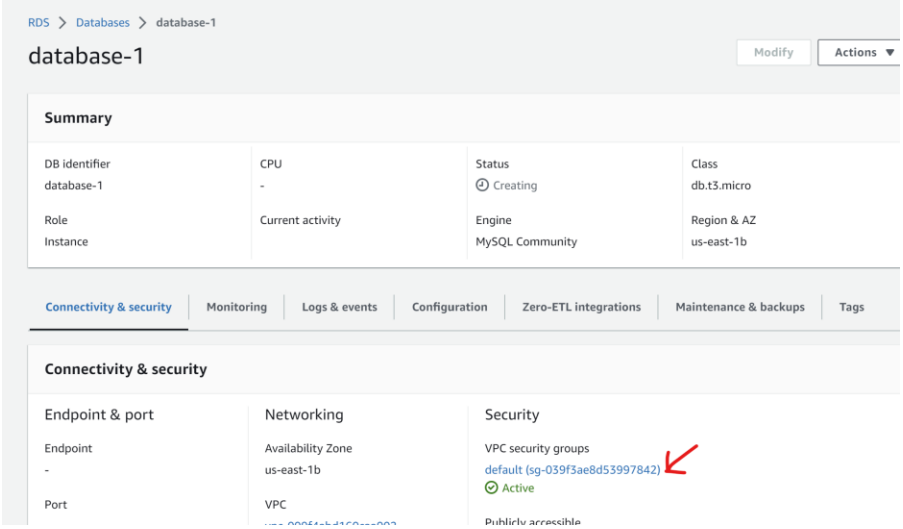
Under Databases in Amazon RDS, you should be able to see the newly created database.



3.2 Configure Inbound rules for database

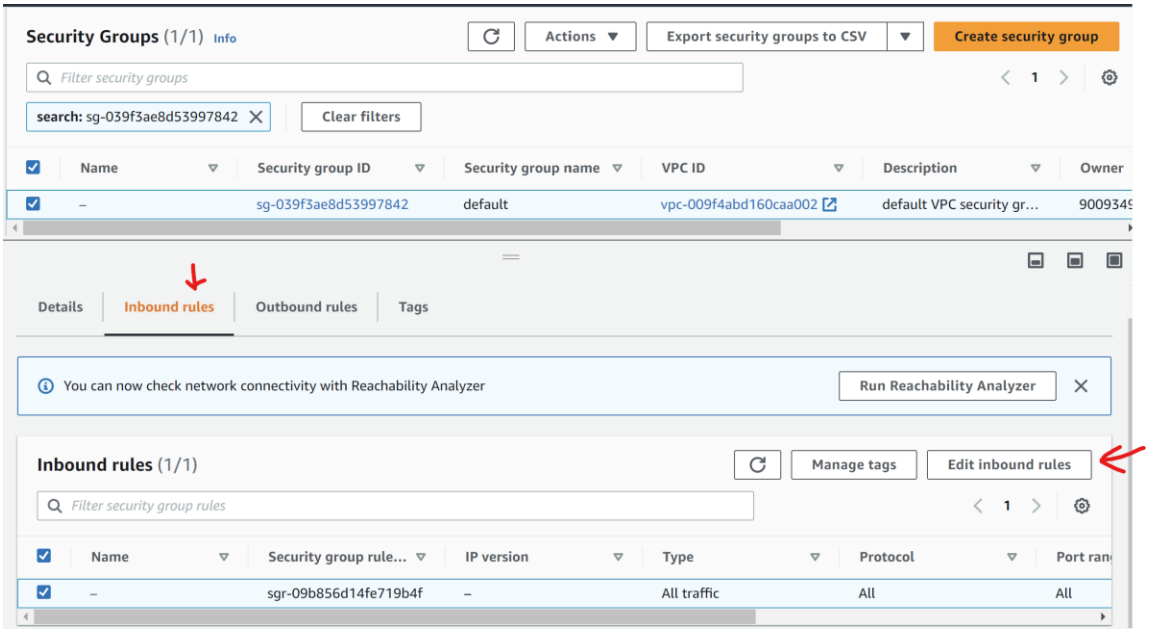
Step 1:

Click on the database-1, and then go to VPC security groups and click the link as shown below.



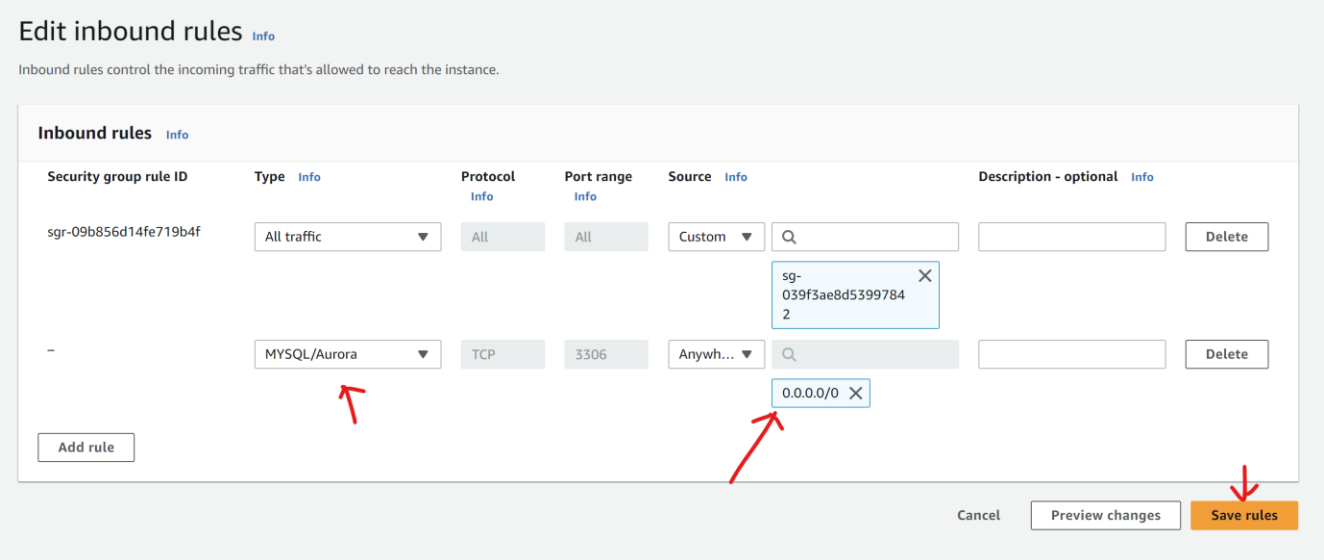
Step 2:

Select Inbound rules and click on the “Edit inbound rules” button.



Step 3:

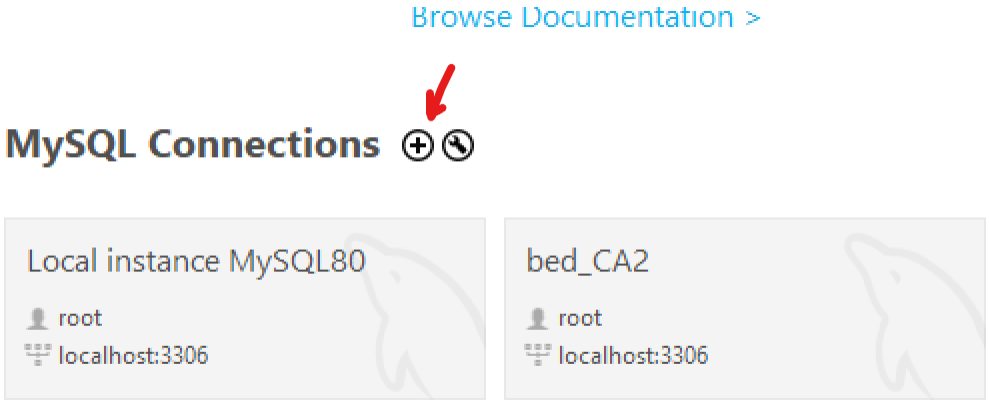
Add a new rule, for MYSQL/Aurora as shown below and click on “Save rules” when done.



3.3 Setting up MySQL Workbench for seeding of data

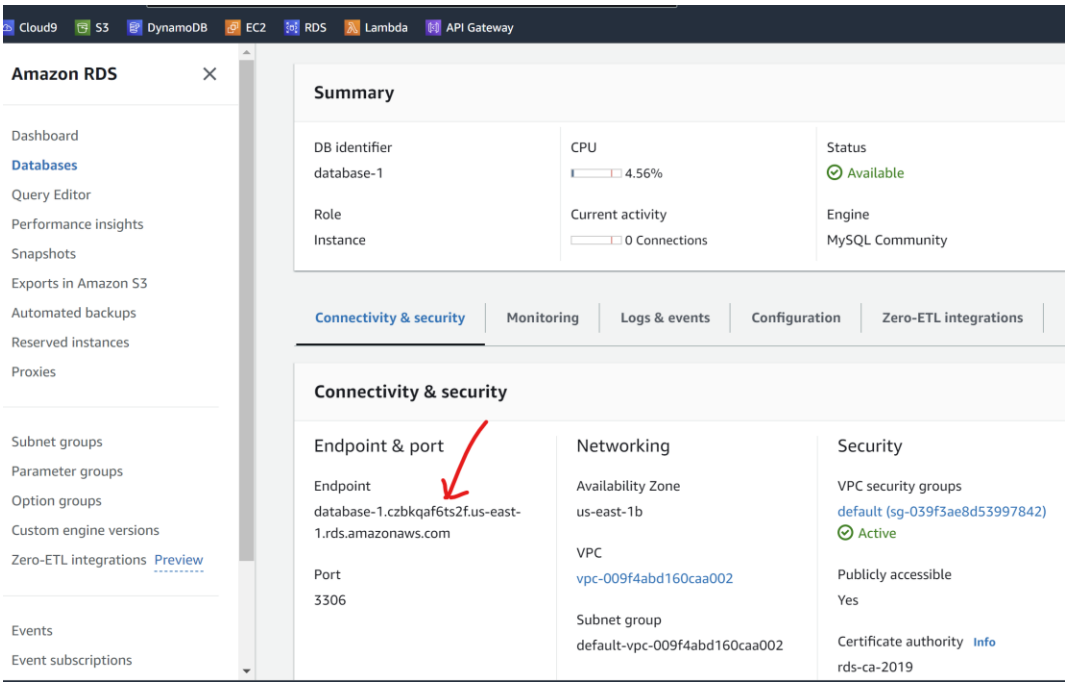
Step 1:

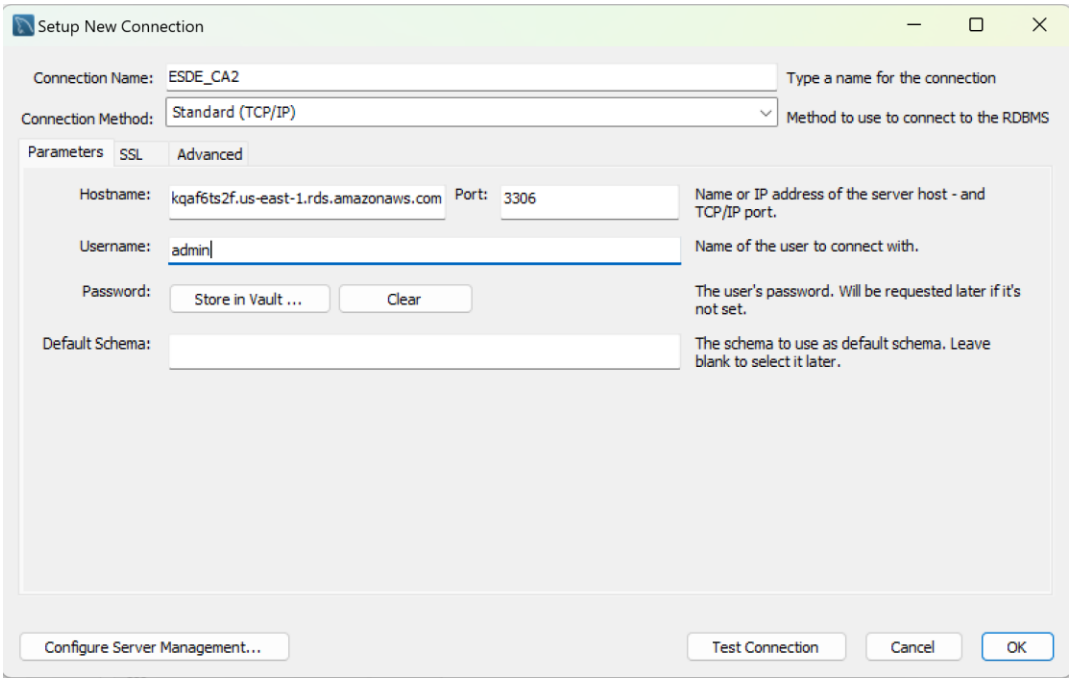
Open MySQL Workbench and create a new connection.



Step 2:

Add a connection name. Next, for your Hostname copy the database endpoint you created just now and paste it in Hostname as shown below.



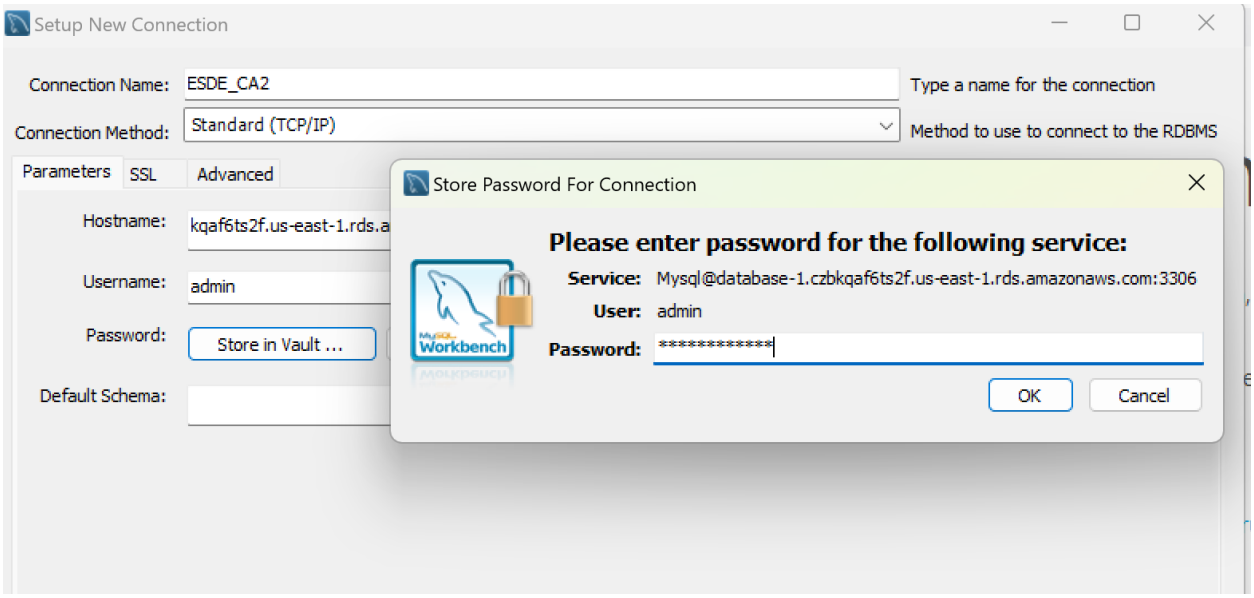


Step 3:

For the Username and Password, we will use the following which we used just now when creating the RDS.

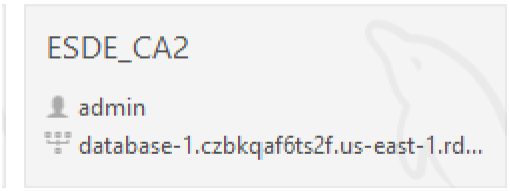
Username: admin

Password: Edentan2003!



Step 4:

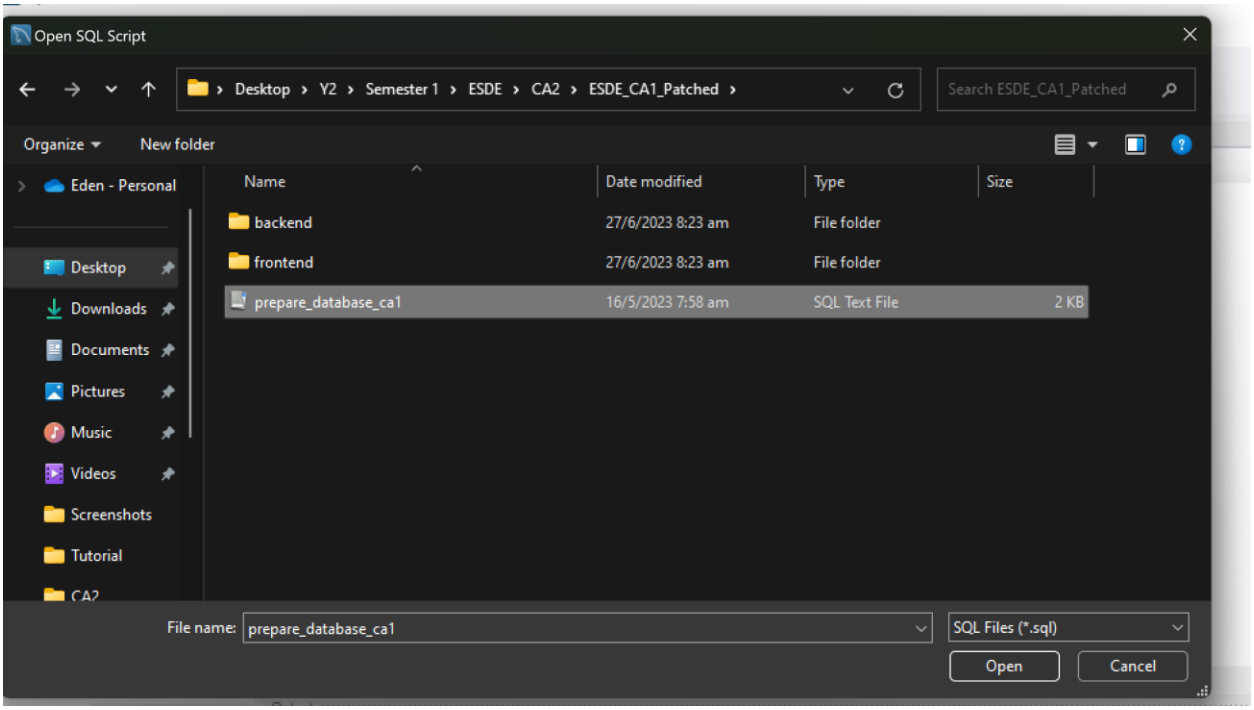
Finally click on Ok and you should see your ESDE_CA2 connection being created as shown below.



3.4 Running the SQL Script in the connection to create Database etc.

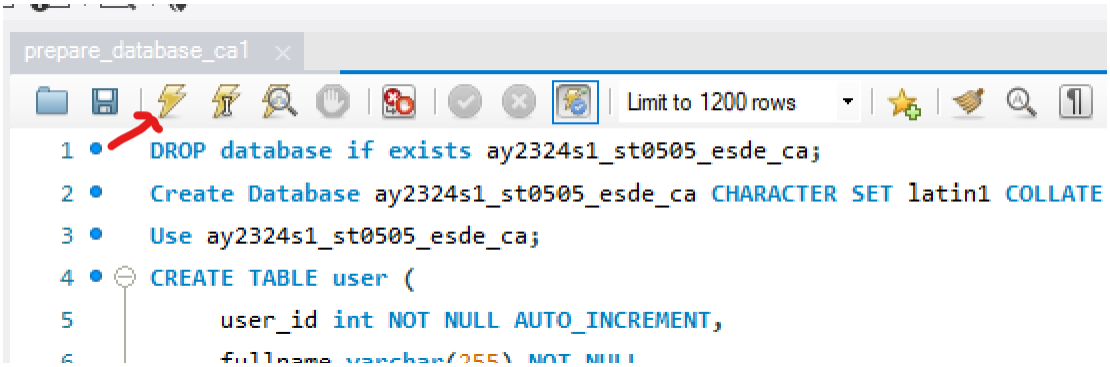
Step 1:

Go into the ESDE_CA2 connection and open the SQL Script from your CA1 as shown below.



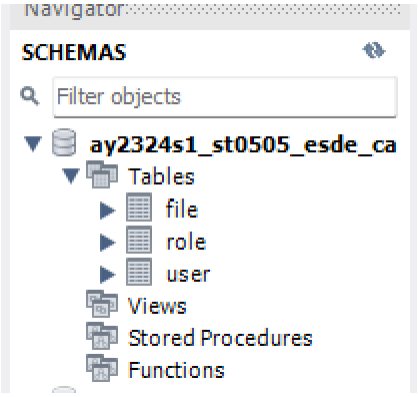
Step 2:

Run the script by clicking on the Lightning Icon as shown below.



Step 3:

You should be able to see the tables being created in your schema.

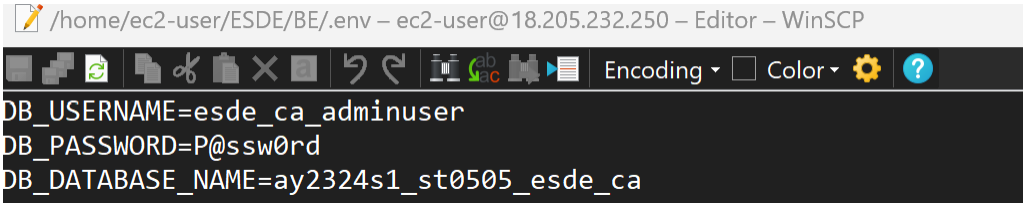


3.5 Using seeddata.js to seed the database

Step 1:

Go into WinSCP and go to the Backend Directory and look for the .env file under ESDE/BE/.env.

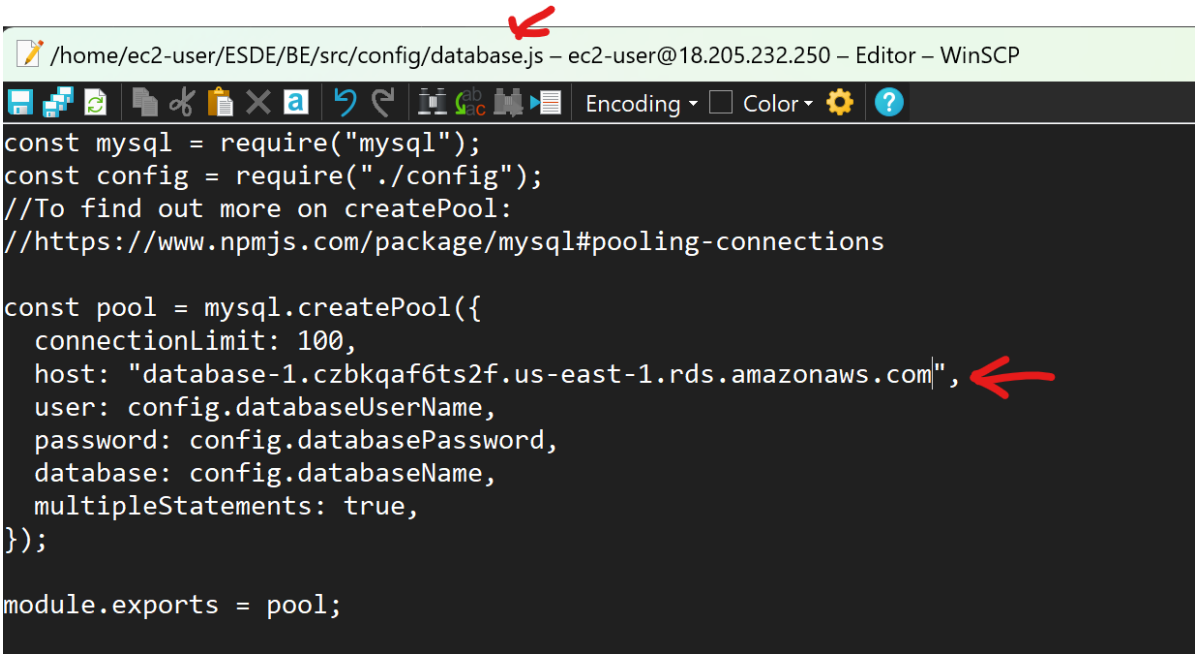
*Note: Make sure that the DB_USERNAME, DB_PASSWORD and DB_DATABASE_NAME matches the one in your MySQL Workbench.




```
DROP USER IF EXISTS 'esde_ca_adminuser'@'%';
CREATE USER 'esde_ca_adminuser'@'%' IDENTIFIED WITH mysql_native_password BY 'P@ssw0rd';
GRANT ALL PRIVILEGES ON ay2324s1_st0505_esde_ca.* TO 'esde_ca_adminuser'@'%';
```

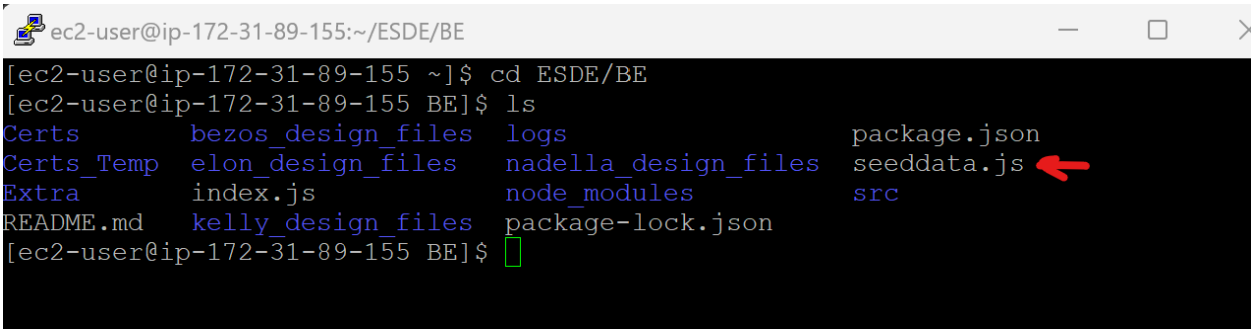
Step 2:

Go into database.js under ESDE/BE/src/config/database.js and change the host to the database endpoint from AWS RDS.



Step 3:

Open the terminal and go to the Backend Directory and make sure that you have seeddata.js in that BE Directory as shown below.

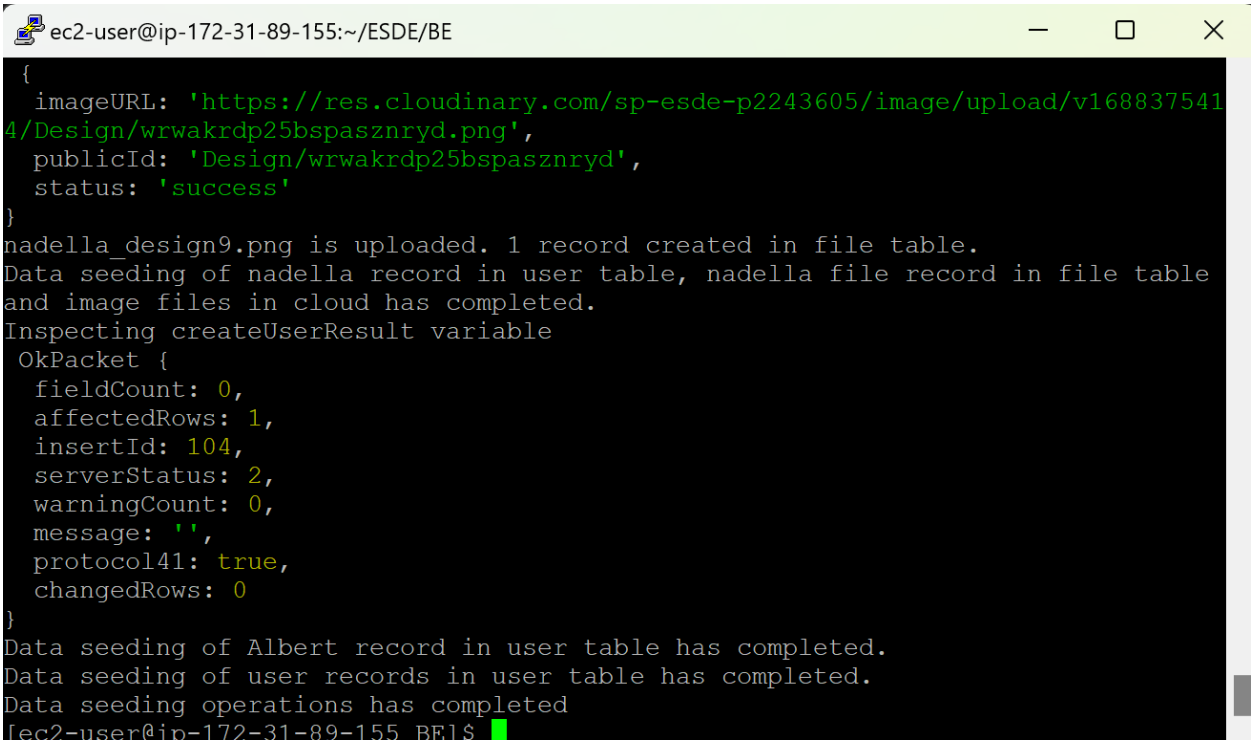


Step 4:

Use seeddata.js to populate the database in the ESDE/BE directory.

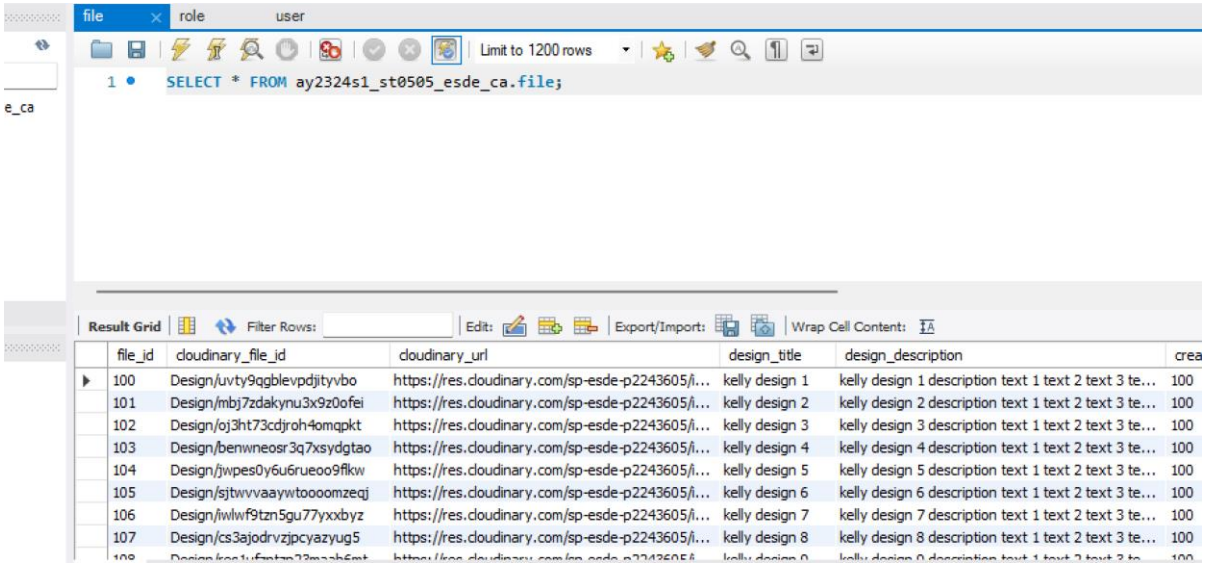
Run the following command: **node seeddata.js**

You should be able to successfully run the command without any errors.



Step 5:

Go into MySQL Workbench and make sure that all the tables have been seeded correctly.



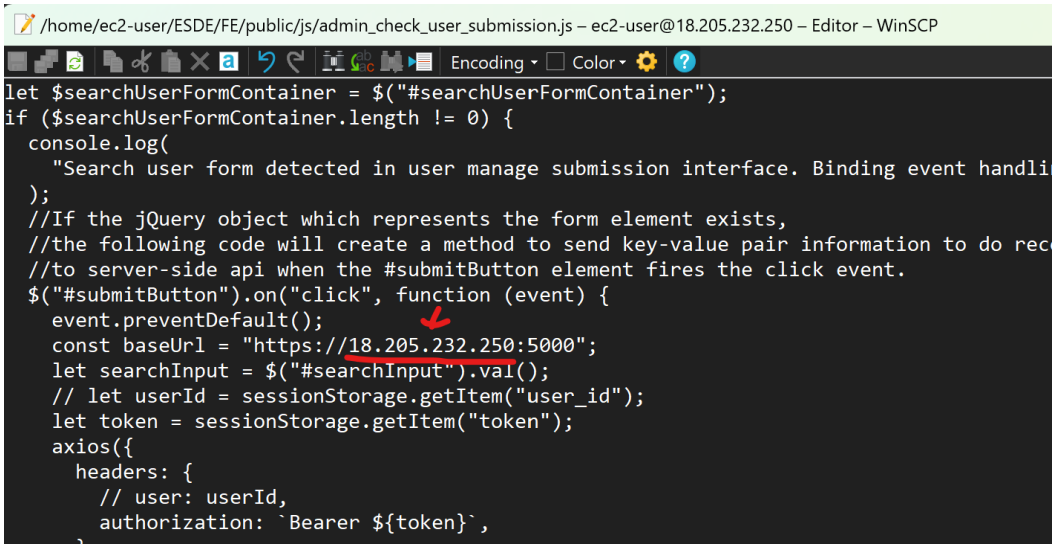
file_id	cloudinary_file_id	cloudinary_url	design_title	design_description	crea
100	Design/uvty9qgblevpdjityvbo	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 1	kelly design 1 description text 1 text 2 text 3 te...	100
101	Design/mbj7zdakynu3x9z0ofei	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 2	kelly design 2 description text 1 text 2 text 3 te...	100
102	Design/oj3ht73cdjroh4omqpkt	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 3	kelly design 3 description text 1 text 2 text 3 te...	100
103	Design/benwneosr3q7xsydgtao	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 4	kelly design 4 description text 1 text 2 text 3 te...	100
104	Design/jwpes0y6u6rueoo9fkw	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 5	kelly design 5 description text 1 text 2 text 3 te...	100
105	Design/sjtwvvaaywtoooomzeqj	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 6	kelly design 6 description text 1 text 2 text 3 te...	100
106	Design/iwlwf9tzn5gu77yxxbyz	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 7	kelly design 7 description text 1 text 2 text 3 te...	100
107	Design/cs3ajodrvzjpcyazyug5	https://res.cloudinary.com/sp-esde-p2243605/i...	kelly design 8	kelly design 8 description text 1 text 2 text 3 te...	100

3.6 Changing necessary codes in Frontend

Step 1:

Go into ESDE/FE/public/js in WinSCP and change all the following .js files that uses “localhost” to your Elastic IP Address which in my case is “18.205.232.250”.

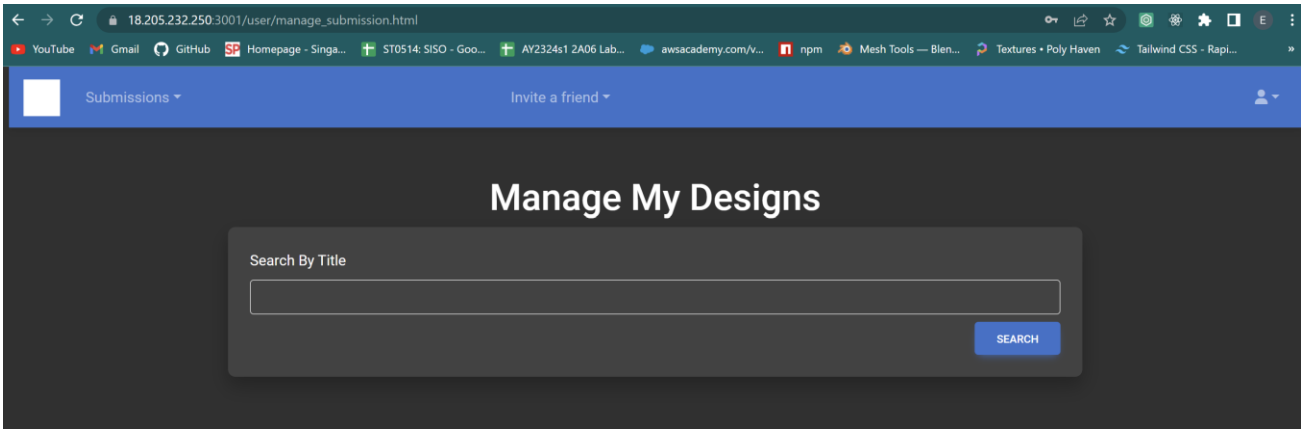
An example as shown below.




```
let $searchUserFormContainer = $("#searchUserFormContainer");
if ($searchUserFormContainer.length != 0) {
  console.log(
    "Search user form detected in user manage submission interface. Binding event handli
  );
  //If the jQuery object which represents the form element exists,
  //the following code will create a method to send key-value pair information to do rec
  //to server-side api when the #submitButton element fires the click event.
  $("#submitButton").on("click", function (event) {
    event.preventDefault();
    const baseUrl = "https://18.205.232.250:5000";
    let searchInput = $("#searchInput").val();
    // let userId = sessionStorage.getItem("user_id");
    let token = sessionStorage.getItem("token");
    axios({
      headers: {
        // user: userId,
        authorization: `Bearer ${token}`,
```

Step 2:

Finally, after changing all localhost to your Elastic IP, run the respective index.js files and see if you can login as shown below.

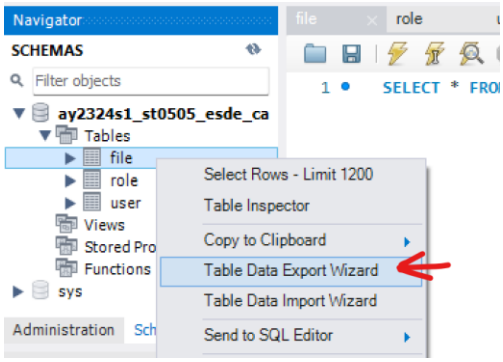


Part 4 – DynamoDB

4.1 Exporting the table data from MySQL Workbench

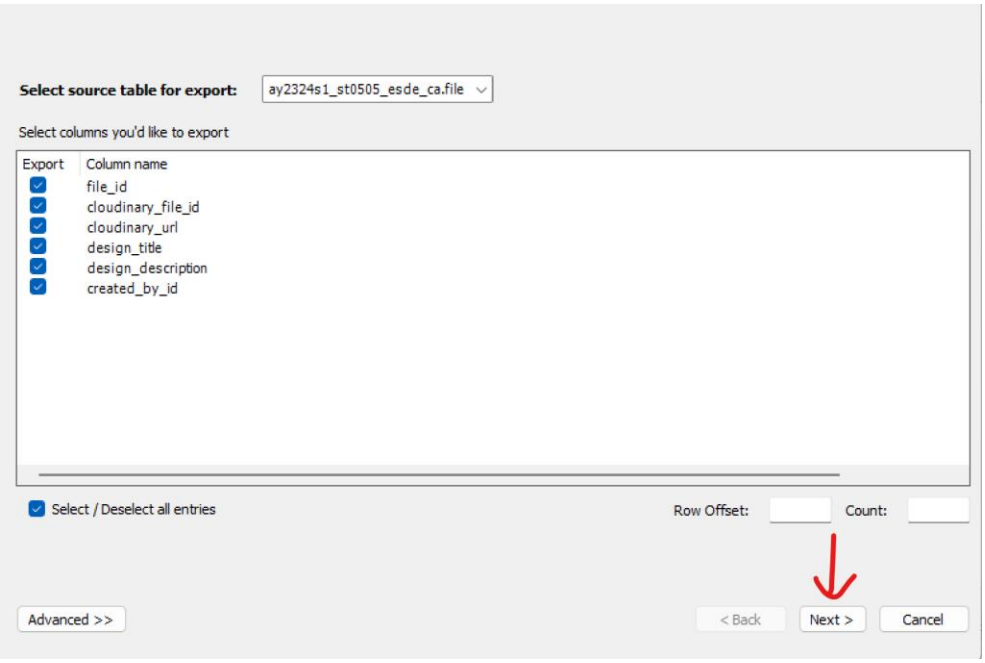
Step 1:

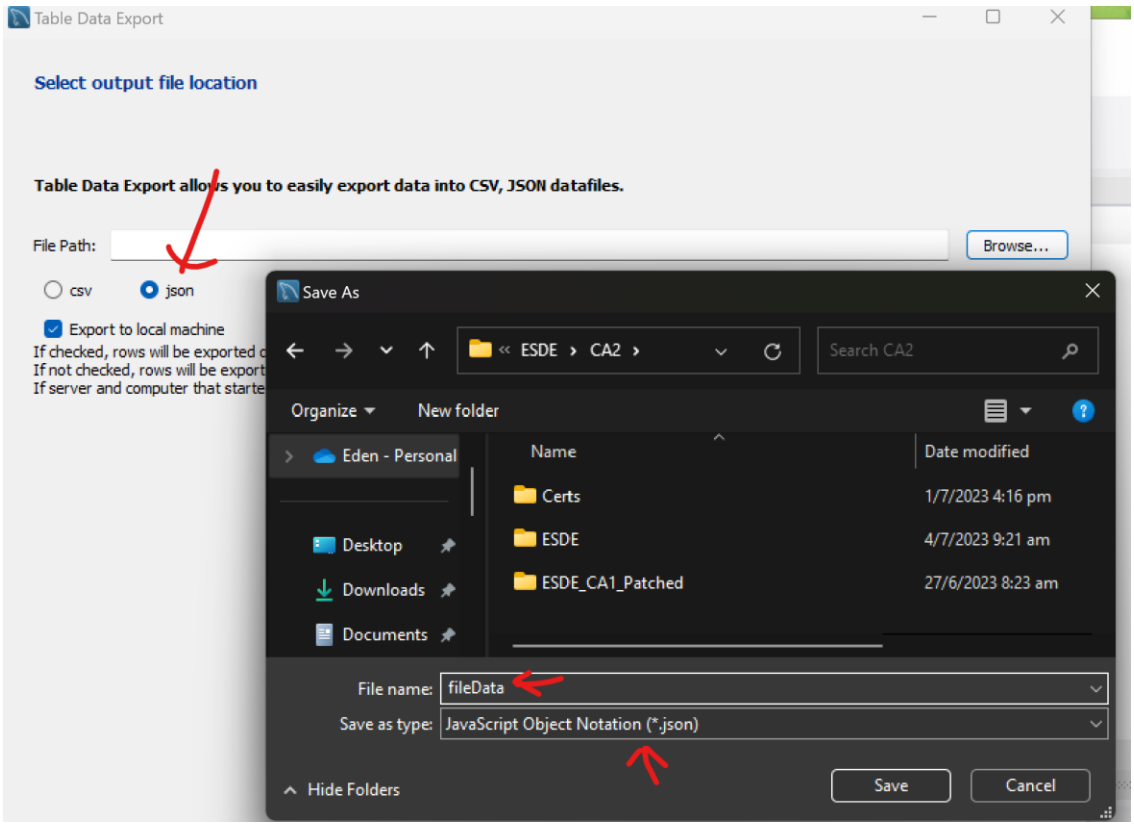
Go into MySQL Workbench and select the file table, right click, and select the "Table Data Export Wizard".



Step 2:

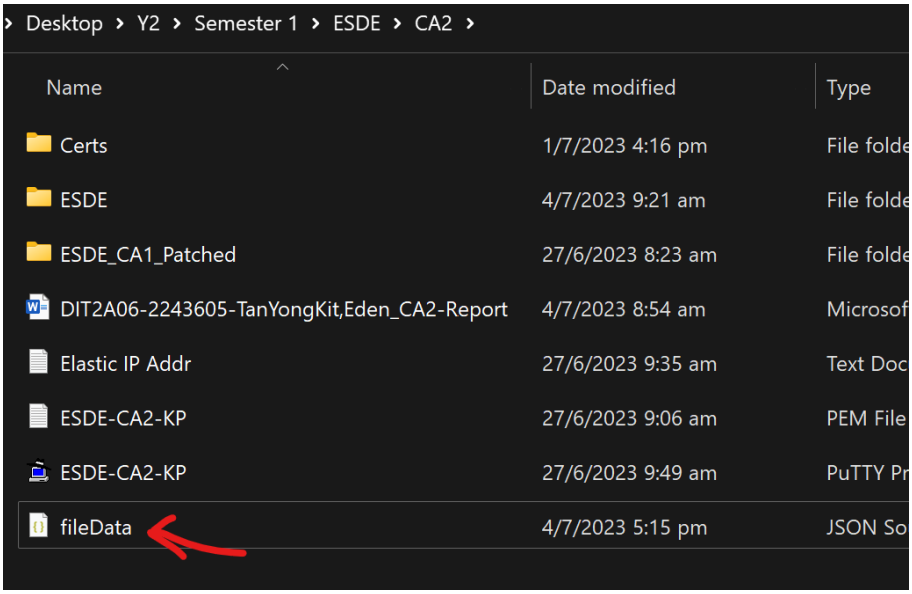
Click on Next and make sure that the file type is JSON. For the file name you can name it as fileData and save it.





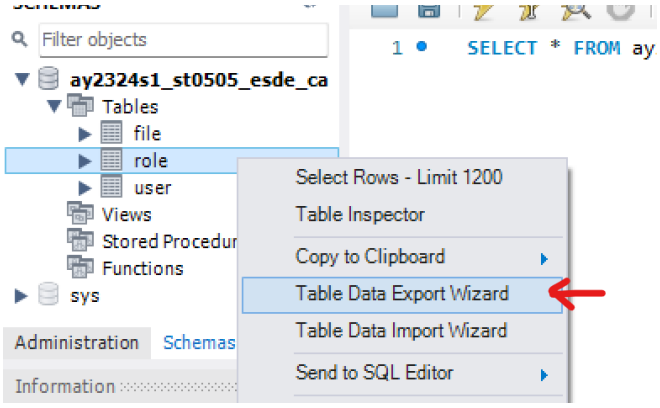
Step 3:

Afterwards, click Next and Next and Next and Finish and you should see the fileData in the directory you saved at.



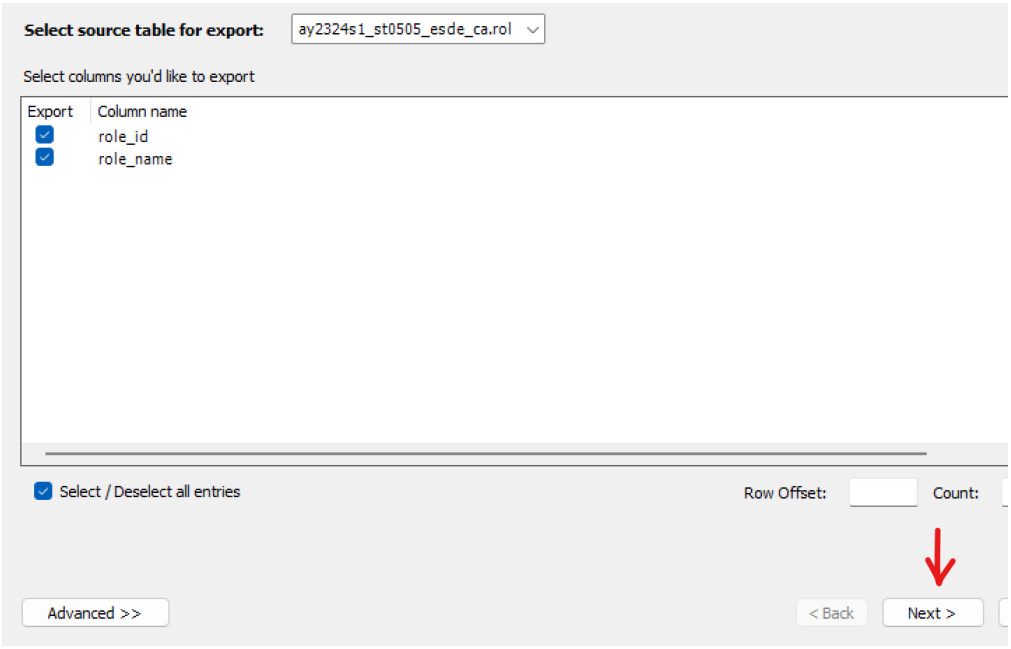
Step 4:

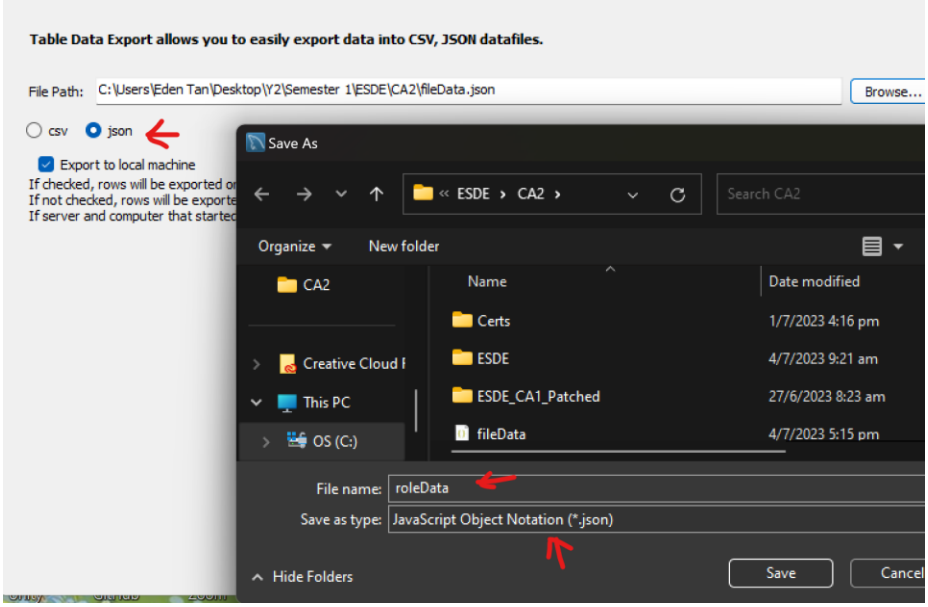
Now, let's do the same thing for the role table. Go to MySQL Workbench and right click and choose the "Table Data Export Wizard".



Step 5:

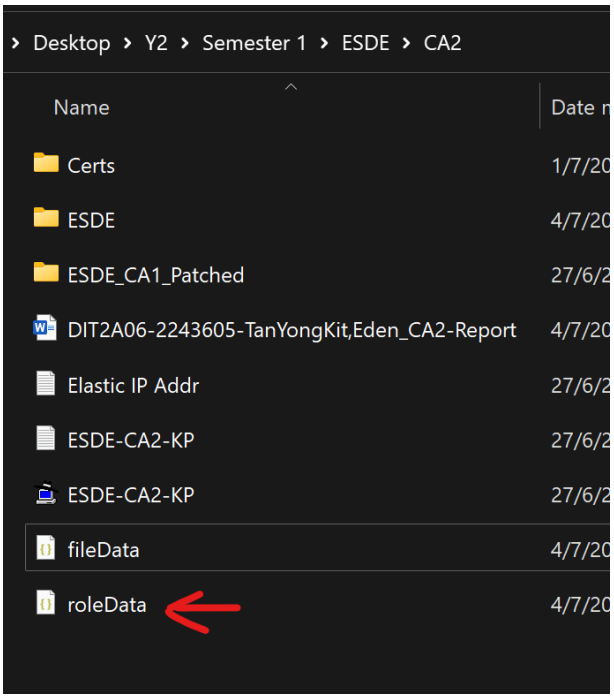
Click Next, and make sure that the file type JSON and you can name the file as roleData.





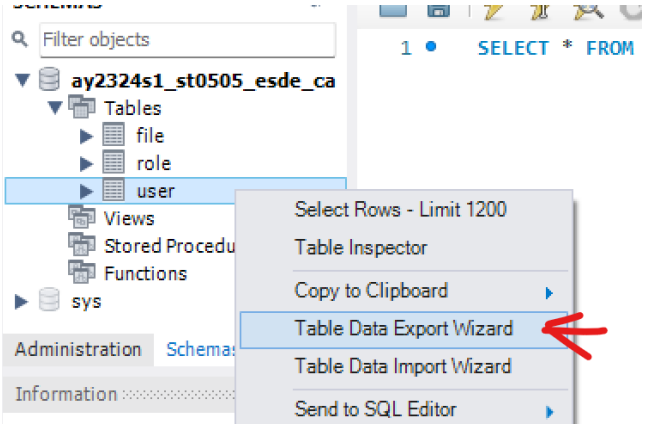
Step 6:

Afterwards, click Next and Next and Next and Finish and you should see the fileData in the directory you saved at.



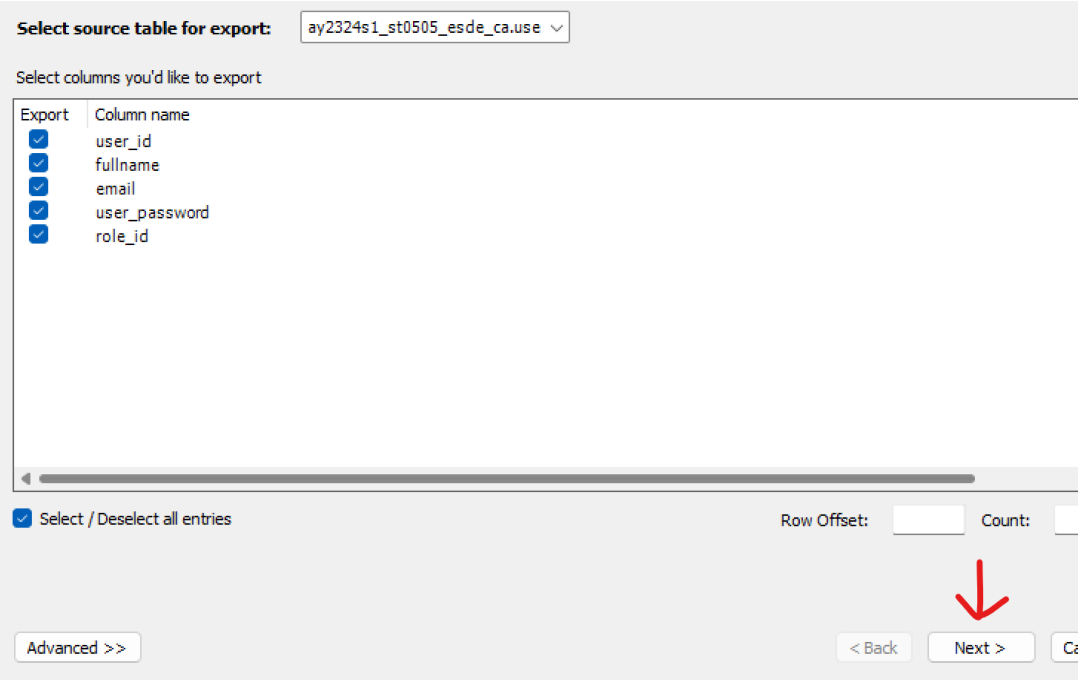
Step 7:

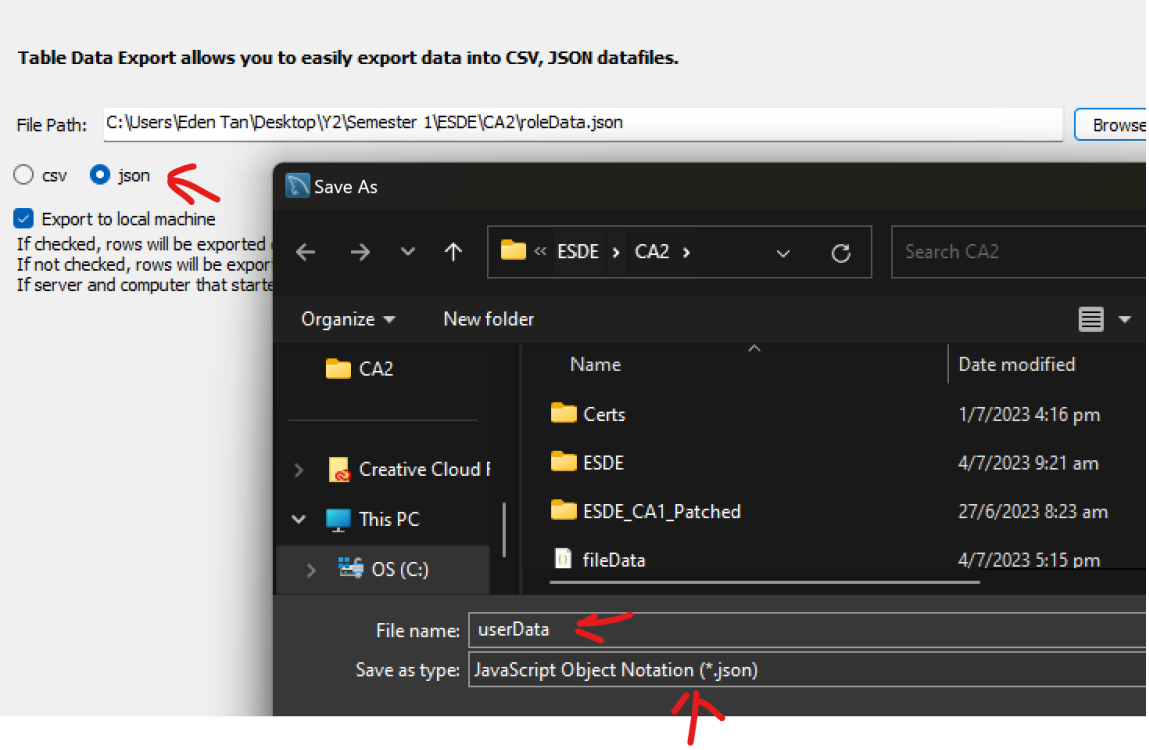
Finally, let's do the same thing for the user table. Go to MySQL Workbench and right click and choose the "Table Data Export Wizard".



Step 8:

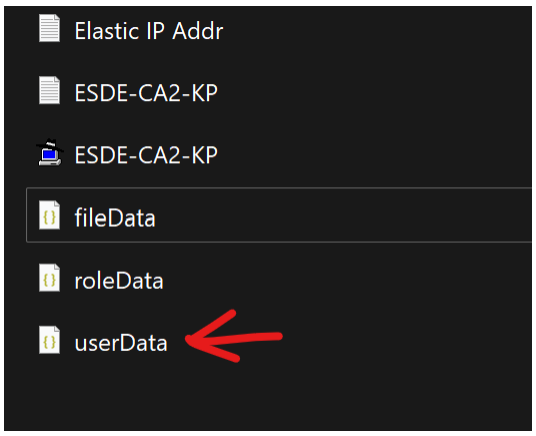
Click Next, and make sure that the file type JSON and you can name the file as userData.





Step 9:

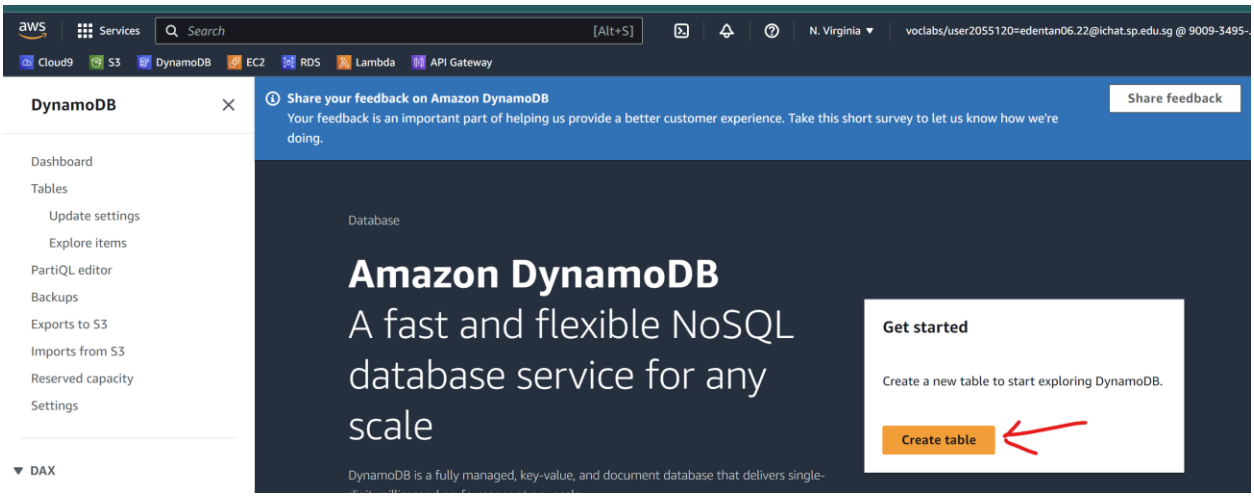
Afterwards, click Next and Next and Next and Finish and you should see the fileData in the directory you saved at.



4.2 Create a DynamoDB table for each table in MySQL Workbench

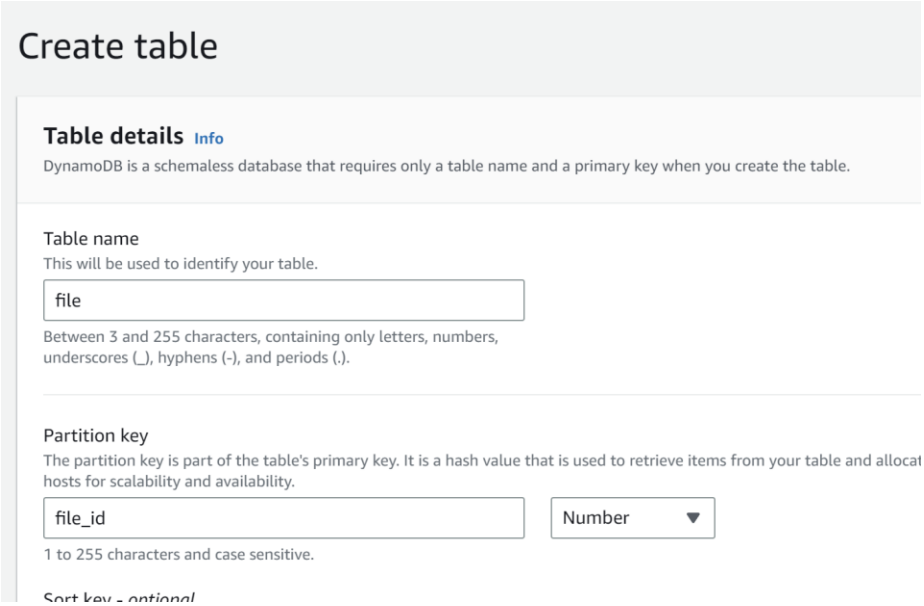
Step 1:

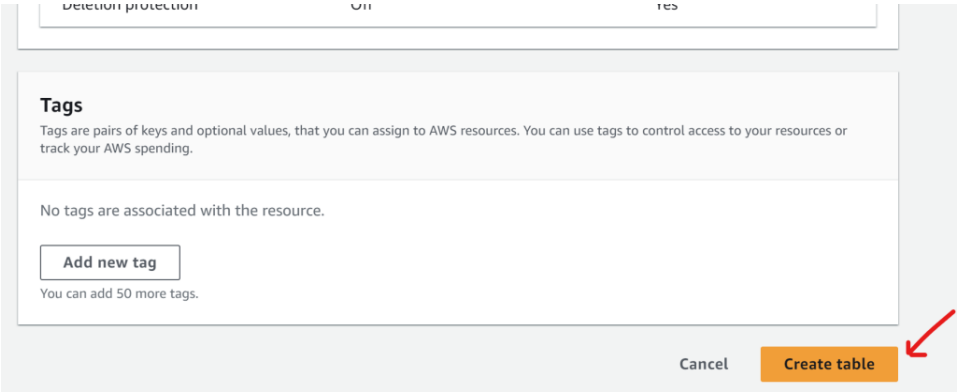
Go to DynamoDB in AWS and click on the “Create table” button.



Step 2:

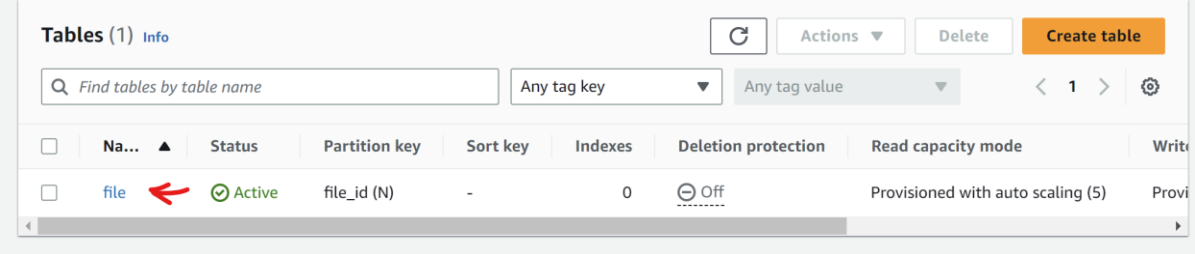
We will create a DynamoDB table called file and a Partition Key called file_id with Number selected and click on “Create table” button as shown below.





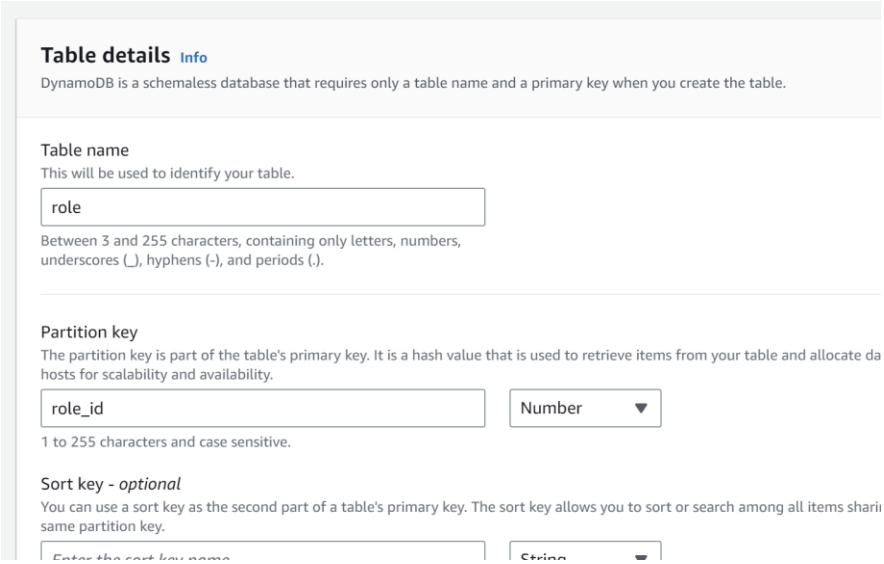
Step 3:

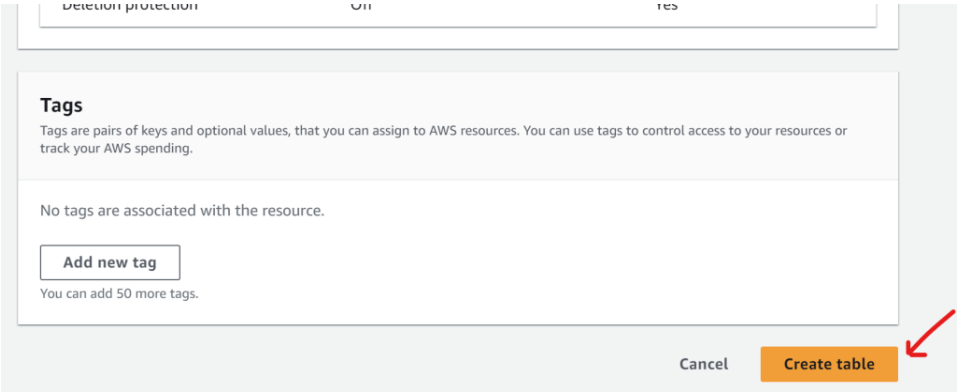
You should be able to see a file table created.



Step 4:

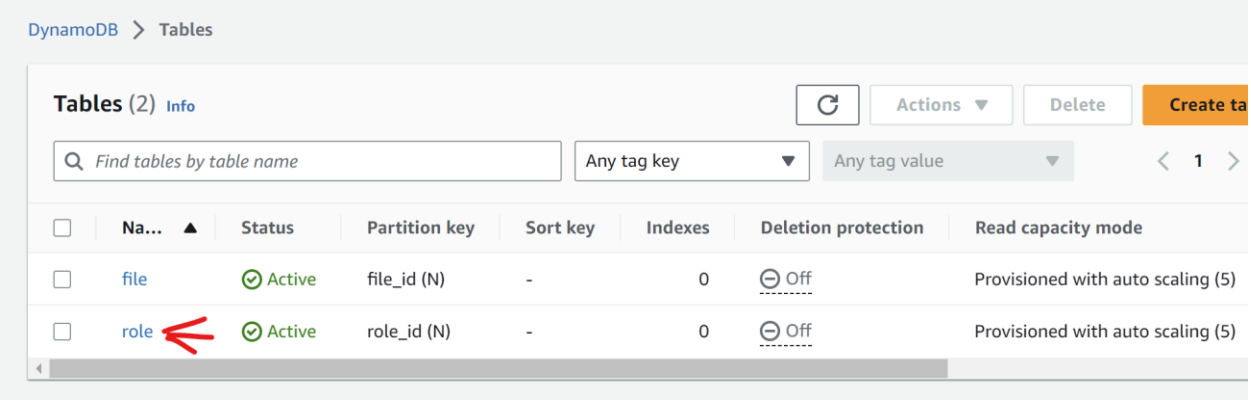
We will create another DynamoDB table called role and a Partition Key called role_id with Number selected and click on “Create table” button as shown below.





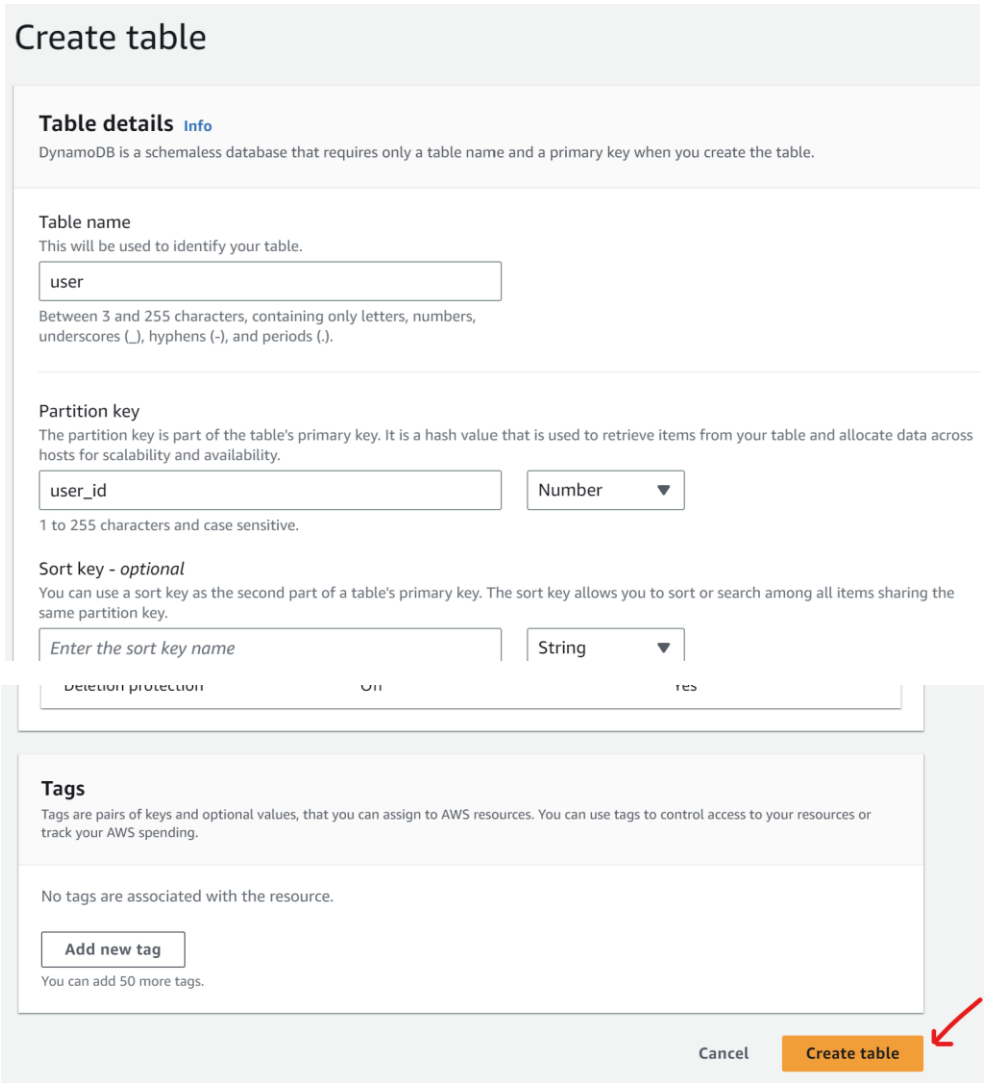
Step 5:

You should be able to see a role table created.



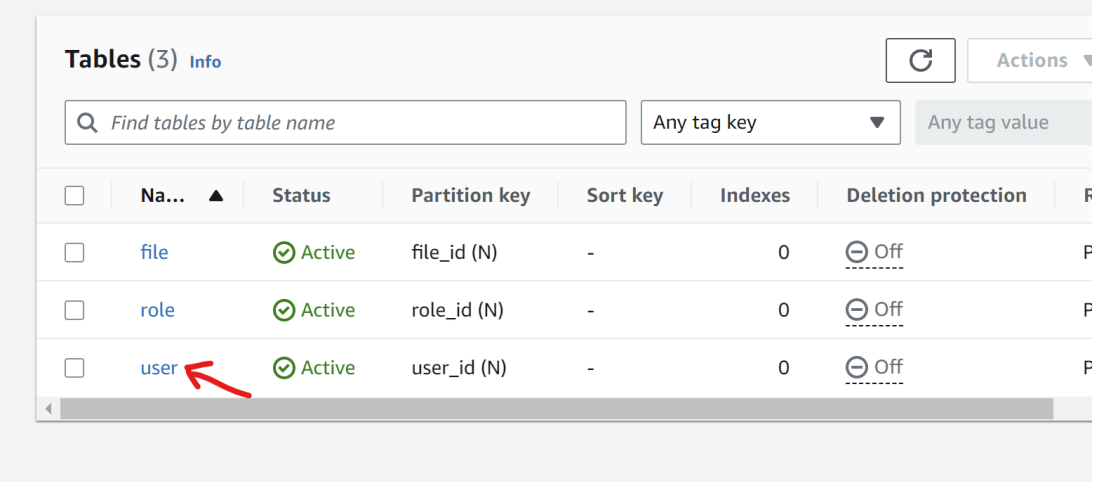
Step 6:

Finally, we will create another DynamoDB table called user and a Partition Key called user_id with Number selected and click on “Create table” button as shown below.



Step 7:

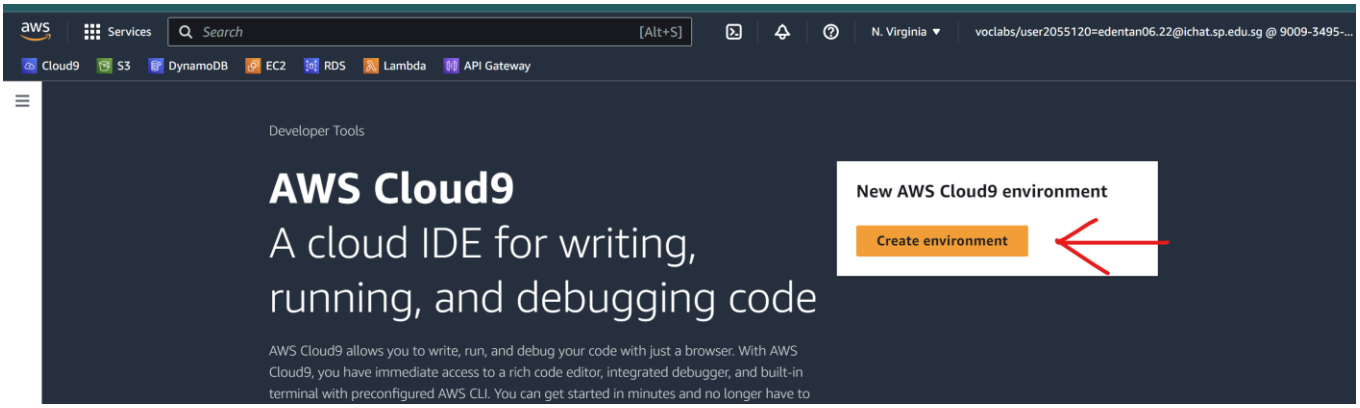
You should be able to see a user table created.



4.3 Adding the MySQL Workbench data to DynamoDB using Cloud9

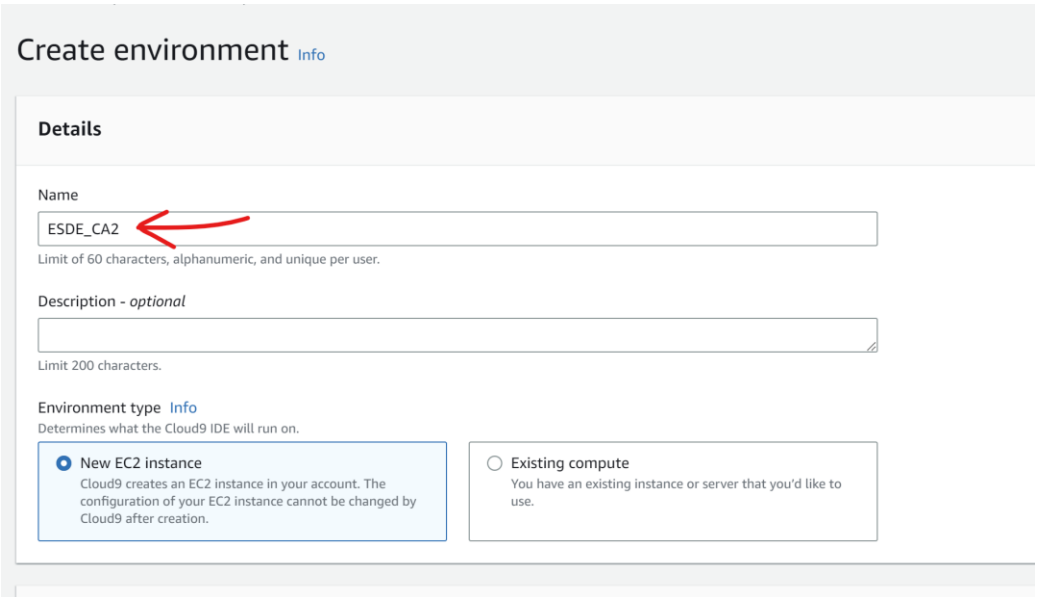
Step 1:

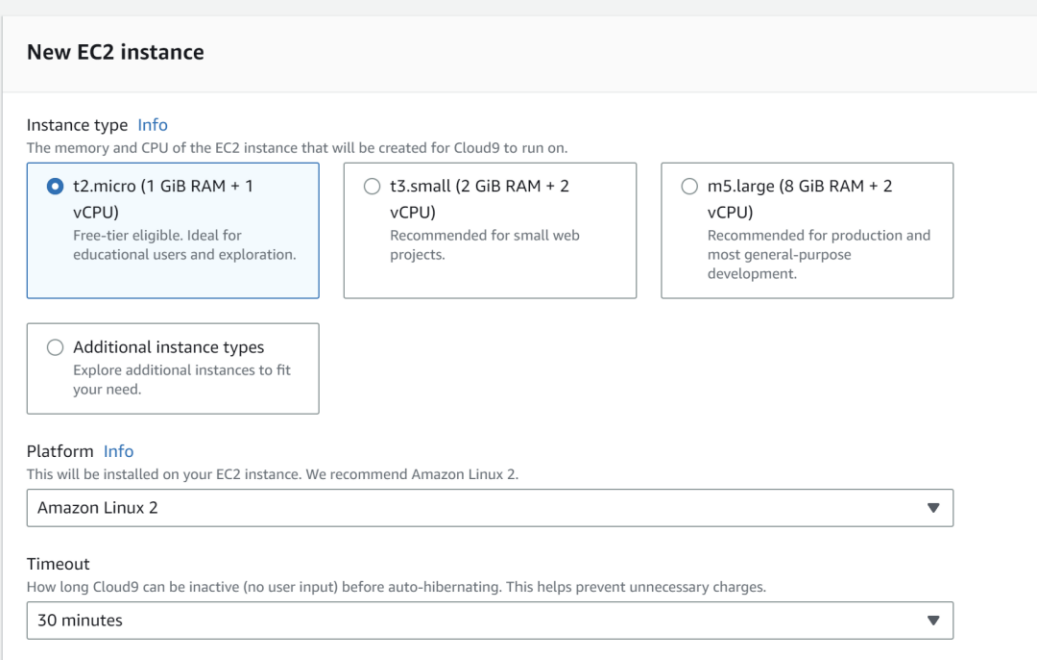
Go to Cloud9 in AWS and click on the “Create environment” button to create a new Cloud9 Environment.



Step 2:

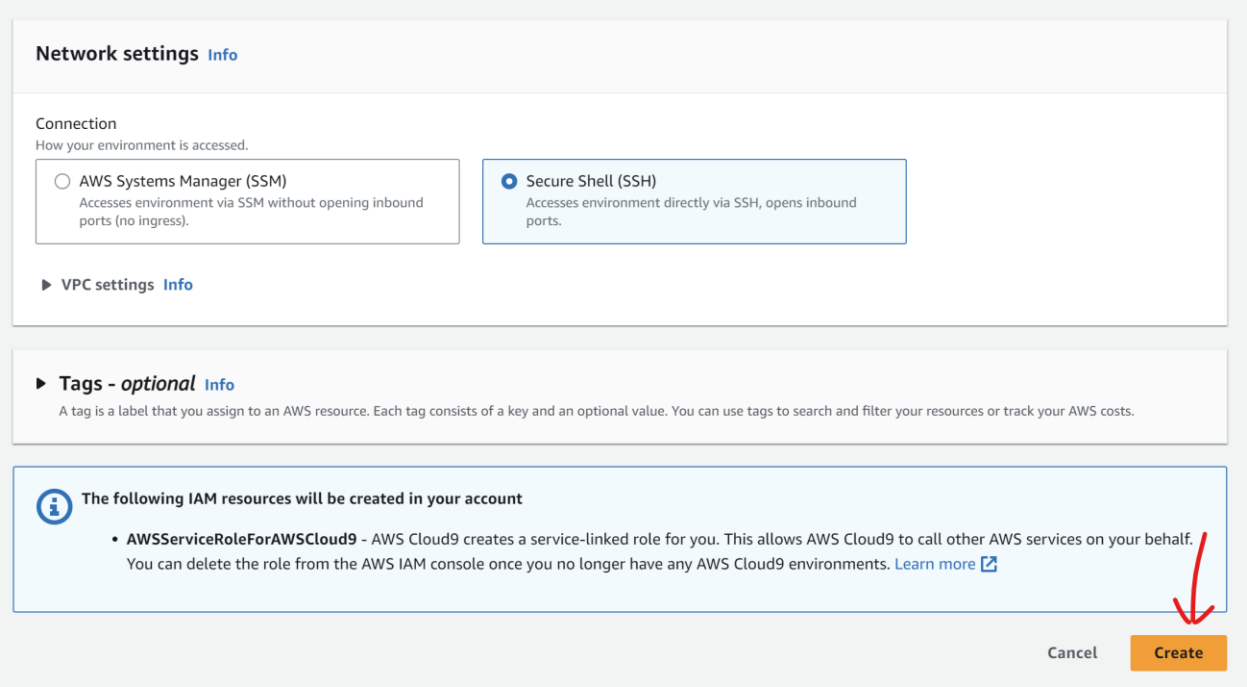
Give a name for your Environment and make sure that the Environment and Instance type, Platform is the default as shown below.





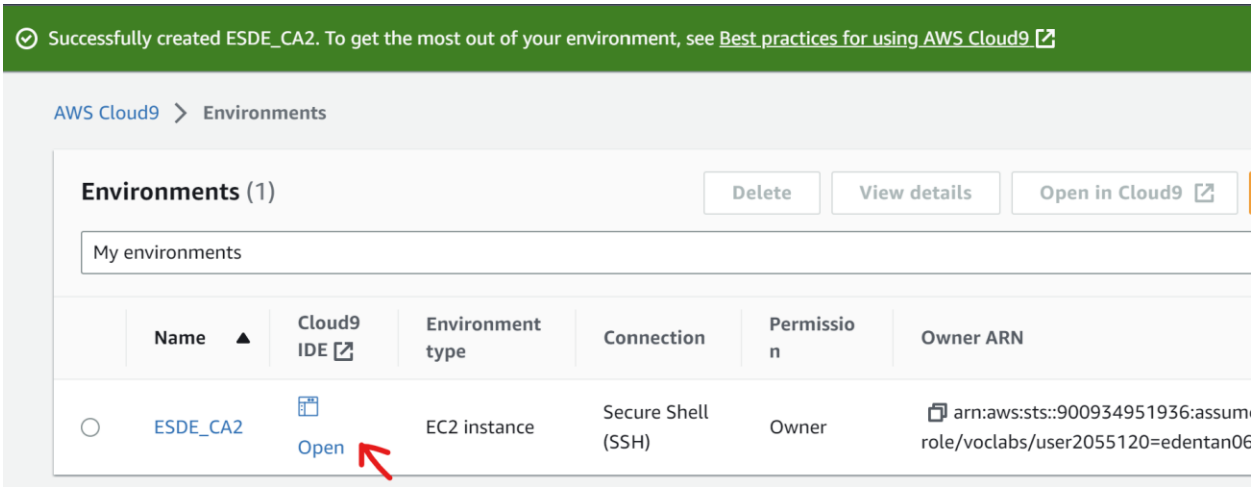
Step 3:

Select Secure Shell (SSH) under Network Settings and click on Create.



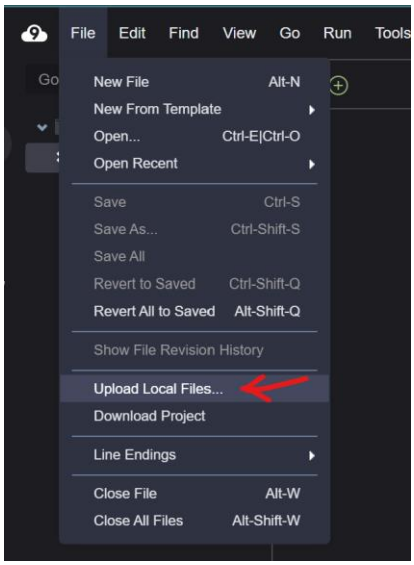
Step 4:

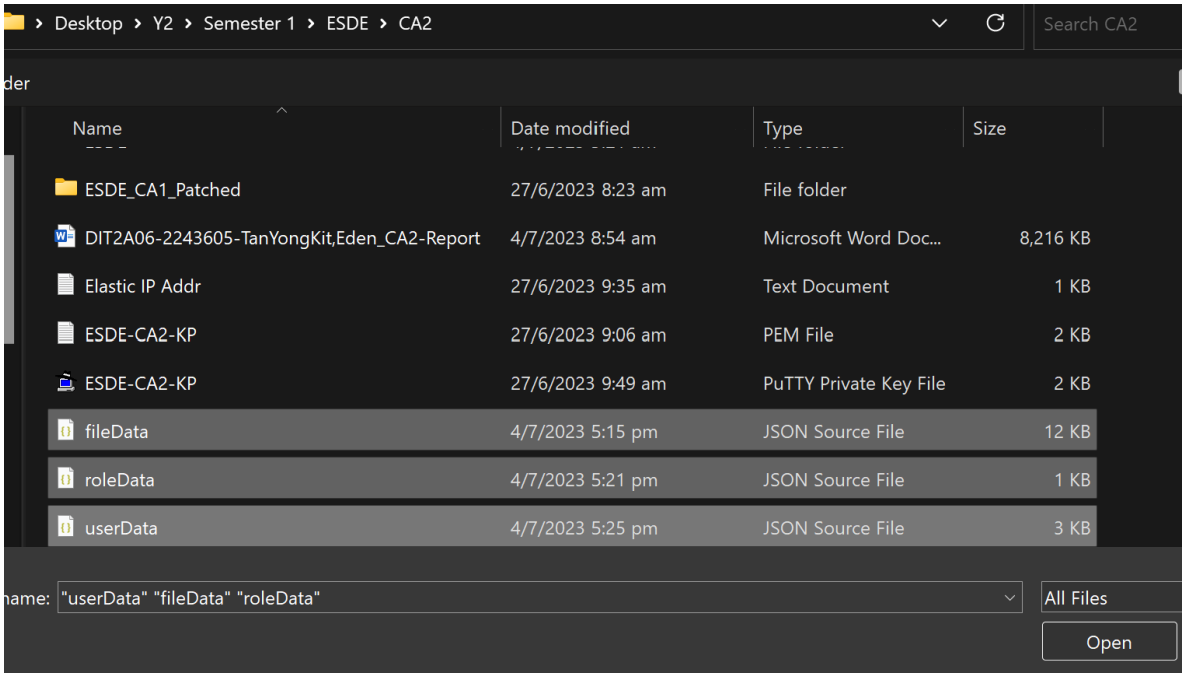
You should be able to see an Environment being created as shown below, click on Open to go to the Cloud9 IDE.



Step 5:

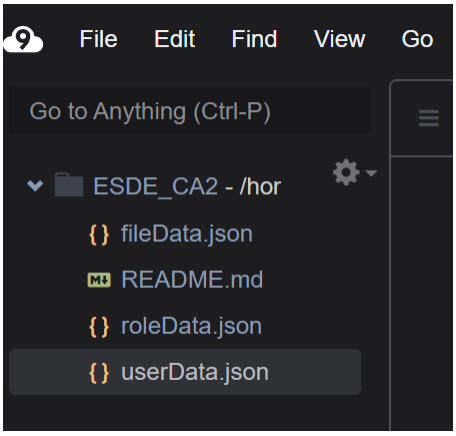
Click on File > Upload Local Files… to import the 3 JSON files that we got from just now.





Step 6:

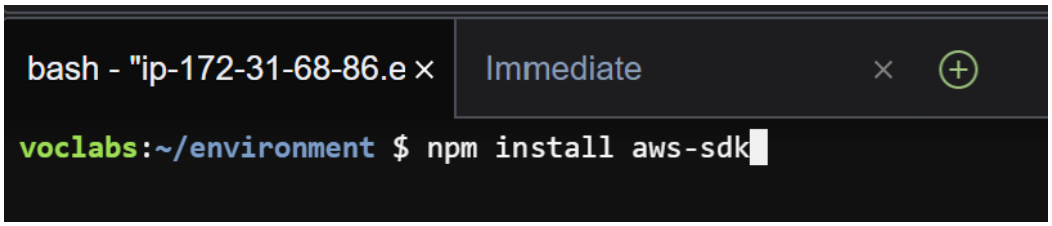
You should be able to see the 3 files in your Cloud9 Environment as shown below.



Step 7:

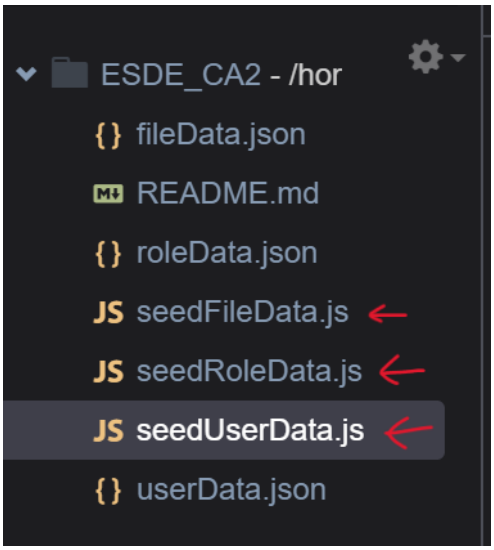
In the terminal run the following command.

Command: npm install aws-sdk



Step 8:

Create a new file in your environment called seedFileData.js, seedRoleData.js and seedUserData.js as shown below.



Step 9:

Under seedFileData.js input the following code to seed the File DynamoDB Table.

```
var
   AWS = require("aws-sdk"),
   DDB = new AWS.DynamoDB({
      apiVersion: "2012-08-10",
      region: "us-east-1"
   }),
   file_DATA_ARR = require("./fileData.json");

function addNewItemsFromJSON(){
        console.log("All items now removed, re-seeding now");
        var
                file = {},
                num_items_left = file_DATA_ARR.length,
                offset_index = 0;
        console.log('num_items_left initial', num_items_left)

        while(num_items_left > 0){
```

```
            var
                file_formatted_arr = [],
                params = {},
                item_added_count = 0;
            for(var i_int = offset_index; i_int < offset_index + Math.min(25,
num_items_left); i_int += 1){
                file = {
                PutRequest: {
                    Item: {
                        file_id: {
                            "N":
file_DATA_ARR[i_int].file_id.toString()
                        },
                        cloudinary_file_id: {
                            "S":
file_DATA_ARR[i_int].cloudinary_file_id
                        },
                        cloudinary_url: {
                            "S": file_DATA_ARR[i_int].cloudinary_url
                        },
                        design_title: {
                            "S": file_DATA_ARR[i_int].design_title
                        },
                        design_description: {
                            "S":
file_DATA_ARR[i_int].design_description
                        },
                        created_by_id: {
                            "N":
file_DATA_ARR[i_int].created_by_id.toString()
                        }
                    }
                }
              };
            file_formatted_arr.push(file);
            item_added_count = item_added_count + 1
            }
```

```
                params = {
                        RequestItems: {
                                "file": file_formatted_arr.reverse()
                        }
                };
                DDB.batchWriteItem(params, function(err, data){
                        if(err){
                                throw err;
                        }
                        console.log("OK");
                });
                num_items_left = num_items_left-item_added_count;
                offset_index = offset_index+item_added_count;
                console.log('num_items_left', num_items_left)
        }
}

(function init(){
        addNewItemsFromJSON();
})();
```

Step 10:

Under seedRoleData.js input the following code to seed the Role DynamoDB Table.

```
var
   AWS = require("aws-sdk"),
   DDB = new AWS.DynamoDB({
      apiVersion: "2012-08-10",
      region: "us-east-1"
   }),
   file_DATA_ARR = require("./roleData.json");

function addNewItemsFromJSON(){
        console.log("All items now removed, re-seeding now");
        var
                file = {},
```

```
            num_items_left = file_DATA_ARR.length,
            offset_index = 0;
        console.log('num_items_left initial', num_items_left)

        while(num_items_left > 0){
            var
                file_formatted_arr = [],
                params = {},
                item_added_count = 0;
            for(var i_int = offset_index; i_int < offset_index+Math.min(25,
num_items_left); i_int += 1){
                file = {
                PutRequest: {
                    Item: {
            role_id: {
              "N": file_DATA_ARR[i_int].role_id.toString()
            },
            role_name: {
              "S": file_DATA_ARR[i_int].role_name
            }
          }
                }
              };
            file_formatted_arr.push(file);
            item_added_count = item_added_count + 1
            }
            params = {
                RequestItems: {
                    "role": file_formatted_arr.reverse()
                }
            };
            DDB.batchWriteItem(params, function(err, data){
                if(err){
                    throw err;
                }
                console.log("OK");
            });
```

```
                num_items_left = num_items_left-item_added_count;
                offset_index = offset_index+item_added_count;
                console.log('num_items_left', num_items_left)
        }
}

(function init(){
        addNewItemsFromJSON();
})();
```

Step 11:

Under seedUserData.js input the following code to seed the User DynamoDB Table.

```
var
   AWS = require("aws-sdk"),
   DDB = new AWS.DynamoDB({
      apiVersion: "2012-08-10",
      region: "us-east-1"
   }),
   file_DATA_ARR = require("./userData.json");

function addNewItemsFromJSON(){
        console.log("All items now removed, re-seeding now");
        var
                file = {},
                num_items_left = file_DATA_ARR.length,
                offset_index = 0;
        console.log('num_items_left initial', num_items_left)

        while(num_items_left > 0){
                var
                        file_formatted_arr = [],
                        params = {},
                        item_added_count = 0;
                for(var i_int = offset_index; i_int < offset_index+Math.min(25,
num_items_left); i_int += 1){
```

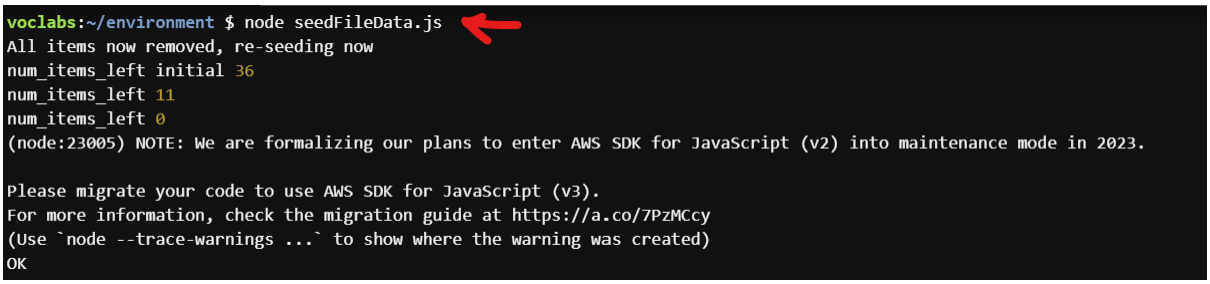
```
                file = {
                PutRequest: {
                        Item: {
    user_id: {
       "N": file_DATA_ARR[i_int].user_id.toString()
    },
    fullname: {
       "S": file_DATA_ARR[i_int].fullname
    },
    email: {
       "S": file_DATA_ARR[i_int].email
    },
    user_password: {
       "S": file_DATA_ARR[i_int].user_password
    },
    role_id: {
       "N": file_DATA_ARR[i_int].role_id.toString()
    }
}
                }
              };
  file_formatted_arr.push(file);
  item_added_count = item_added_count + 1
  }
  params = {
                RequestItems: {
                        "user": file_formatted_arr.reverse()
                }
  };
  DDB.batchWriteItem(params, function(err, data){
                if(err){
                        throw err;
                }
                console.log("OK");
  });
  num_items_left = num_items_left-item_added_count;
  offset_index = offset_index+item_added_count;
```

```
                console.log('num_items_left', num_items_left)
        }
}

(function init(){
        addNewItemsFromJSON();
})();
```

Step 12:

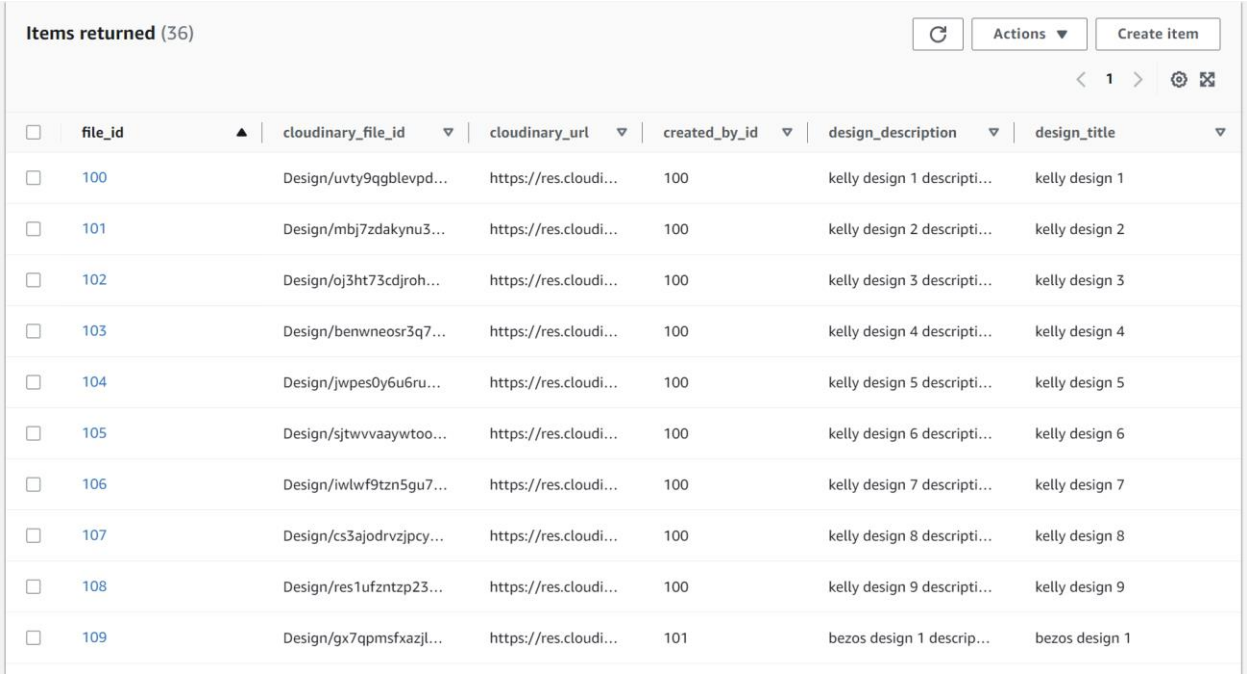
In the bash terminal run the following command to seed the file table in DynamoDB.

Command: node seedFileData.js



Step 13:

You should be able to see the file JSON data being populated into the DynamoDB file table.



Items returned (36)

Actions | Create item

1

file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
100	Design/uvty9qgblevpd...	https://res.cloudi...	100	kelly design 1 descripti...	kelly design 1
101	Design/mbj7zdakynu3...	https://res.cloudi...	100	kelly design 2 descripti...	kelly design 2
102	Design/oj3ht73cdjroh...	https://res.cloudi...	100	kelly design 3 descripti...	kelly design 3
103	Design/benwneosr3q7...	https://res.cloudi...	100	kelly design 4 descripti...	kelly design 4
104	Design/jwpes0y6u6ru...	https://res.cloudi...	100	kelly design 5 descripti...	kelly design 5
105	Design/sjtwvvaaywtoo...	https://res.cloudi...	100	kelly design 6 descripti...	kelly design 6
106	Design/iwlwf9tzn5gu7...	https://res.cloudi...	100	kelly design 7 descripti...	kelly design 7
107	Design/cs3ajodrvzjpcy...	https://res.cloudi...	100	kelly design 8 descripti...	kelly design 8
108	Design/res1ufzntzp23...	https://res.cloudi...	100	kelly design 9 descripti...	kelly design 9
109	Design/gx7qpmsfxazjl...	https://res.cloudi...	101	bezos design 1 descrip...	bezos design 1

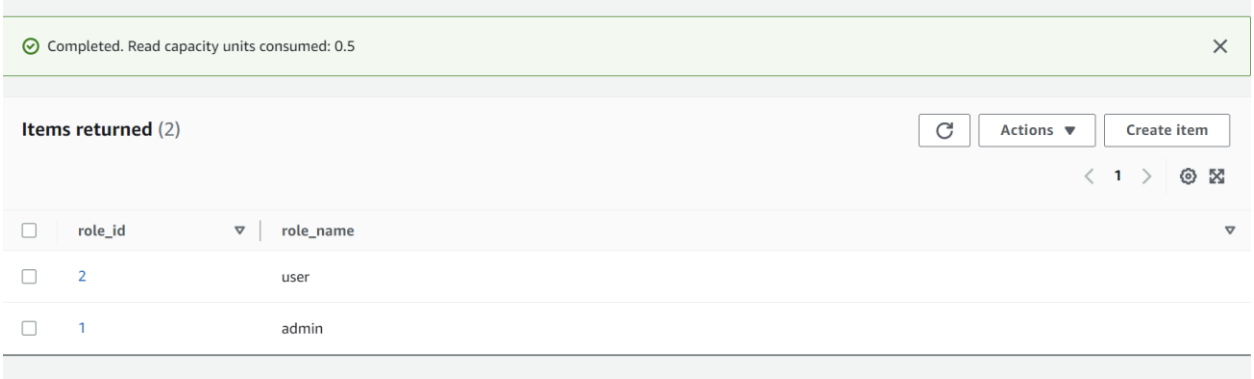
Step 14:

Next, run the following command in the bash terminal to seed the DynamoDB role table.

```
voclabs:~/environment $ node seedRoleData.js
All items now removed, re-seeding now
num_items_left initial 2
num_items_left 0
(node:24171) NOTE: We are formalizing our plans to enter AWS SDK for JavaScript (v2) into maintenance mode in 2023.

Please migrate your code to use AWS SDK for JavaScript (v3).
For more information, check the migration guide at https://a.co/7PzMCcy
(Use `node --trace-warnings ...` to show where the warning was created)
OK
```

Step 15:

You should be able to see the role JSON data being populated into the DynamoDB role table.



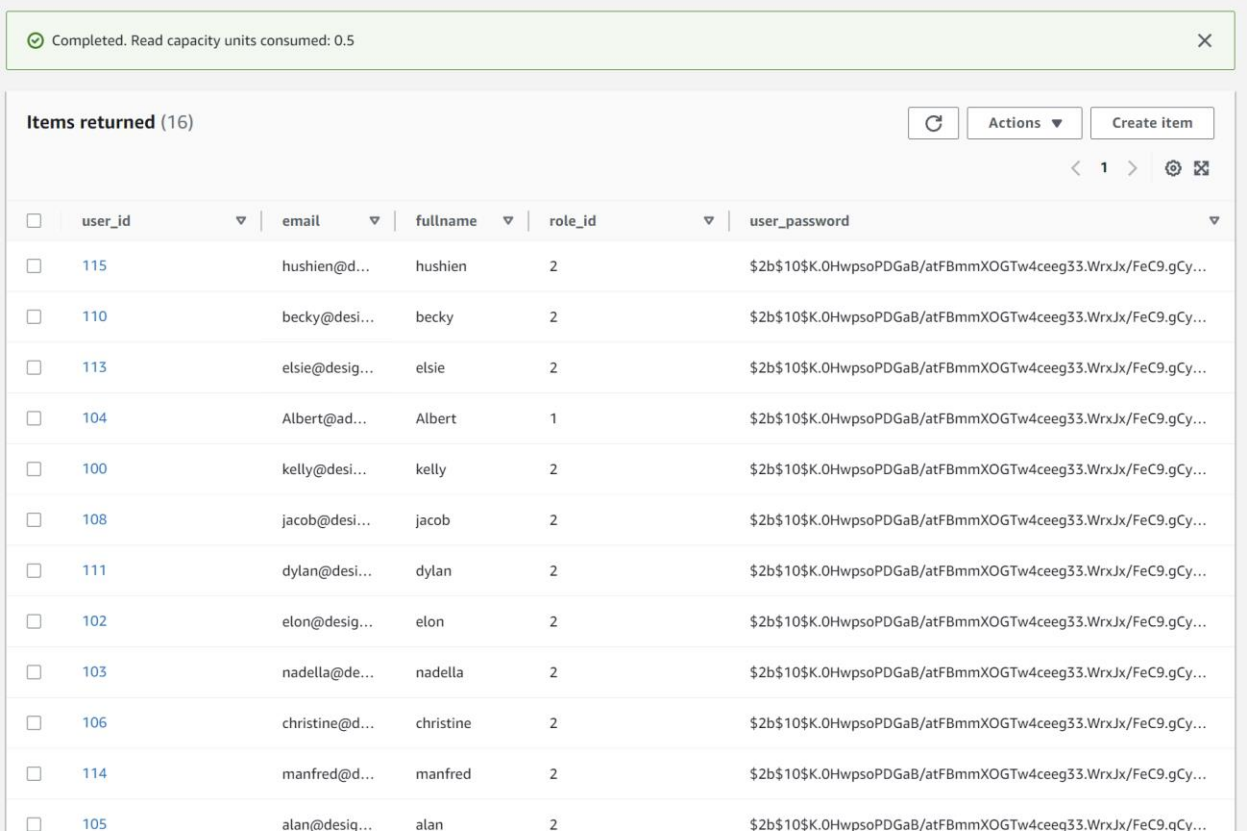
Step 16:

Finally, run the following command in the bash terminal to seed the DynamoDB user table.

```
voclabs:~/environment $ node seedUserData.js
All items now removed, re-seeding now
num_items_left initial 16
num_items_left 0
(node:24884) NOTE: We are formalizing our plans to enter AWS SDK for JavaScript (v2) into maintenance mode in 2023.

Please migrate your code to use AWS SDK for JavaScript (v3).
For more information, check the migration guide at https://a.co/7PzMCcy
(Use `node --trace-warnings ...` to show where the warning was created)
OK
```

Step 17:

You should be able to see the user JSON data being populated into the DynamoDB user table.

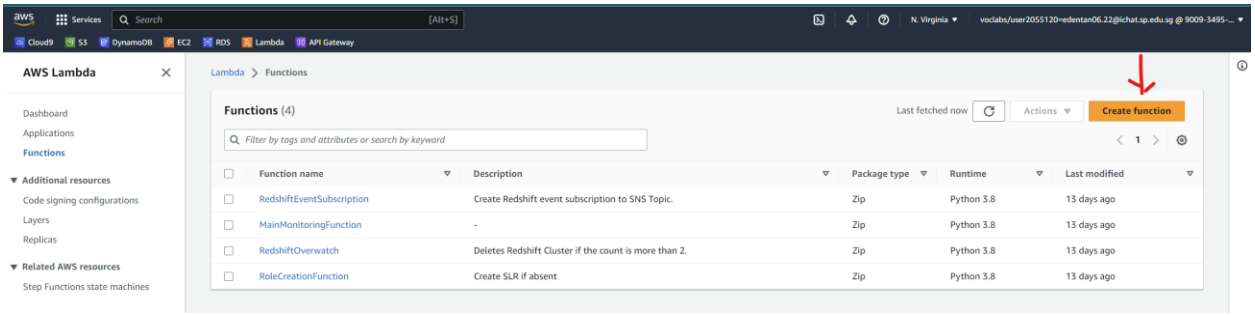


Part 5 – Set up Lambda

5.1 Creating Lamba Function getDesigns

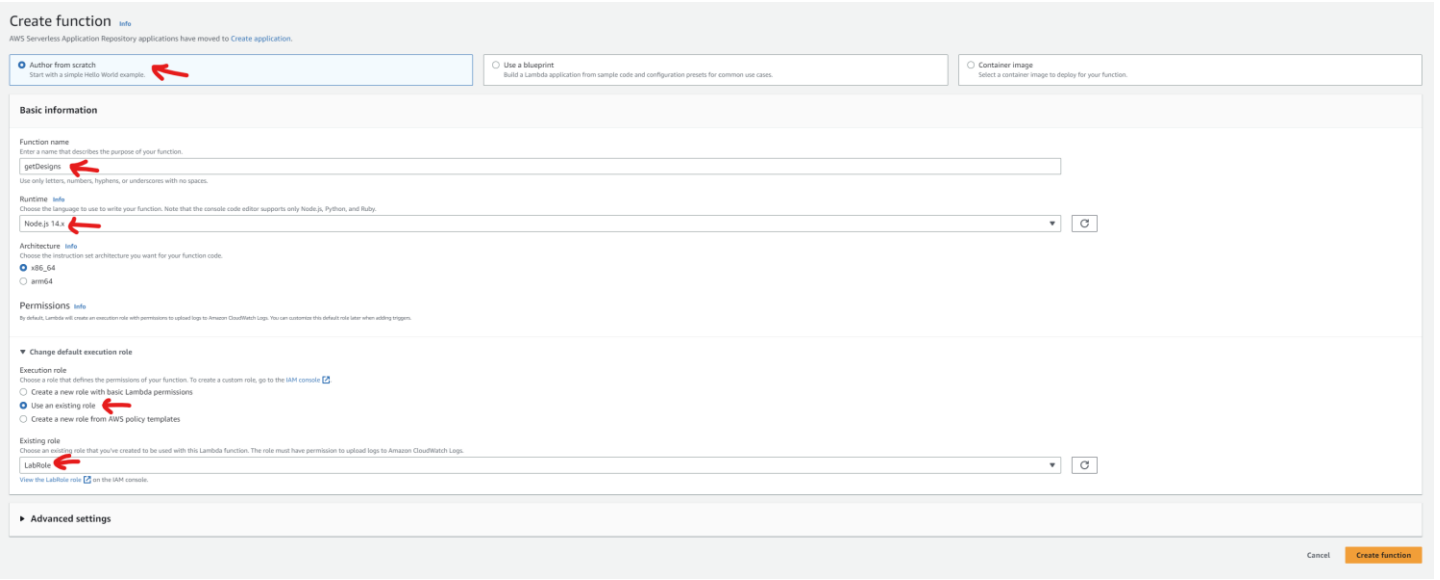
Step 1:

Go into AWS Lambda and click on “Create function” to create a new Lambda Function getDesigns.



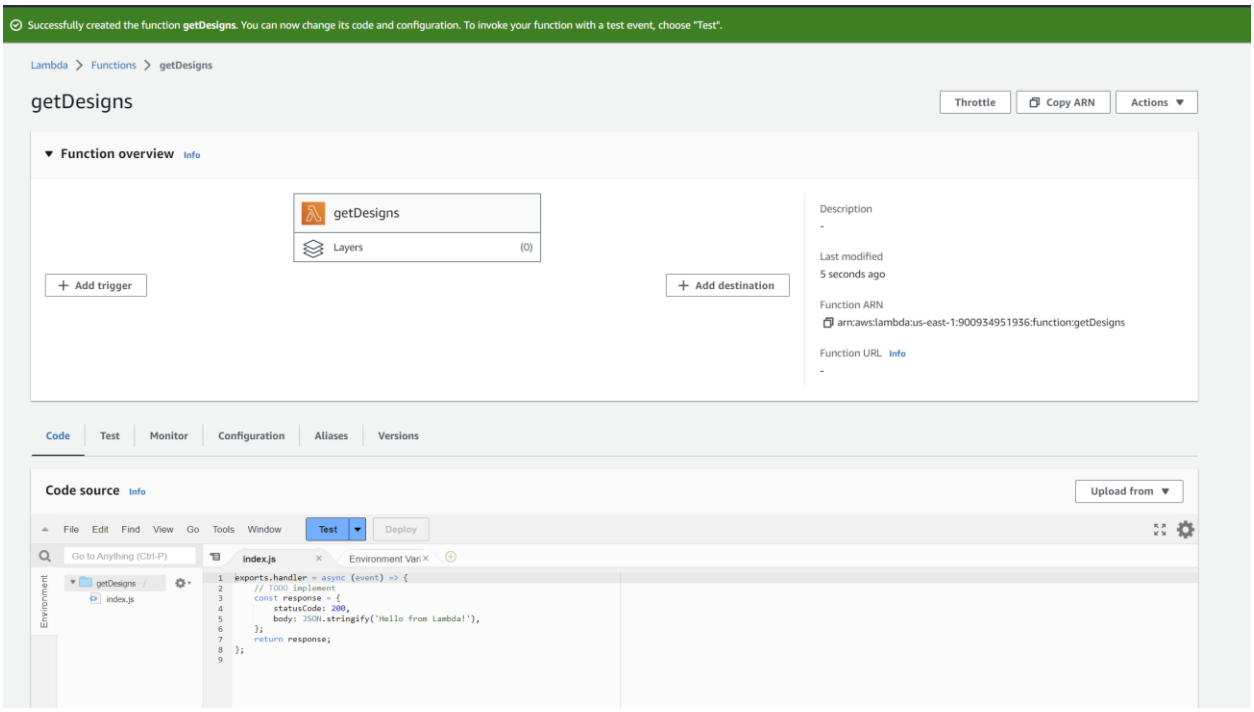
Step 2:

Give the function a name of “getDesigns”, Select “Node.js 14.x” as your runtime, Select “Use an existing role” and choose “LabRole” under Execution role and click on “Create function”.



Step 3:

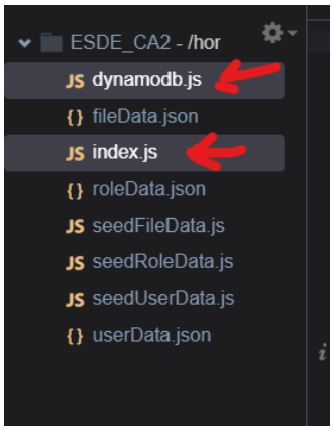
You should be able to see that the function has been successfully created as shown below.



5.2 Creating files in Cloud9 for Lamba Function getDesigns

Step 1:

Go into Cloud9 and bring in 2 files called dynamodb.js and index.js as shown below.



dynamodb.js codes:

```
var AWS = require('aws-sdk');

async function queryitems_dynamodb(region,
table_name,expr_attr_values,key_cond_expr,proj_expr) {
   console.log("In the queryitems_dynamodb method...")
   var dynamodb = new AWS.DynamoDB.DocumentClient({region: region});

   try{
      var params = {  TableName: table_name,
                  ExpressionAttributeValues: expr_attr_values,
                  KeyConditionExpression: key_cond_expr ,
                  ProjectionExpression: proj_expr}

      var items = []

      const results = await dynamodb.query(params).promise()
      console.log("Printing results from  queryitems_dynamodb " + results)

      return results;
   }
   catch(tryerror) {
      console.log("Error occurred in dynamodb.query..")
      console.log(tryerror, tryerror.stack); // an error occurred
   }
} //end function

module.exports = queryitems_dynamodb
```

index.js codes:

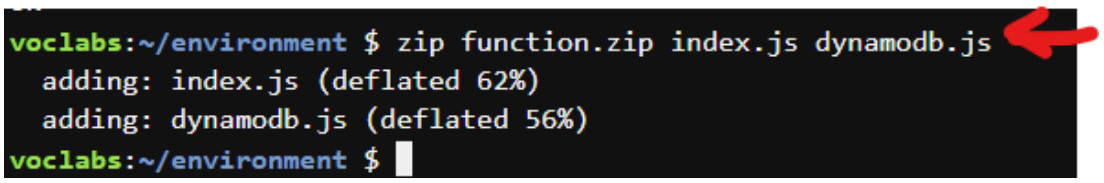
```
var dynamodbQuery = require("dynamodb");

exports.handler = async function (event, context, callback) {
  if (
    event.fid ||
    (event.queryStringParameters && event.queryStringParameters.fid)
  ) {
    if (event.fid) var fileId = parseInt(event.fid);
    else var fileId = parseInt(event.queryStringParameters.fid);
    var region = "us-east-1";
    var table_name = "file";
    var expr_attr_values = { ":fileid": fileId };
    var key_cond_expr = "file_id=:fileid";
    var proj_expr = "file_id,cloudinary_url,design_title,design_description";
    await dynamodbQuery(
      region,
      table_name,
      expr_attr_values,
      key_cond_expr,
      proj_expr
    )
      .then((data) => {
        console.log("Successfully got items from dynamodb.query");
        var responseCode = 200;
        var jsonResult = { filedata: data.Items[0] };
        let response = {
          statusCode: responseCode,
          body: JSON.stringify(jsonResult),
          headers: {
            "Access-Control-Allow-Headers": "Content-Type,user",
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Methods": "OPTIONS,POST,GET",
          },
        };
        console.log("response: " + JSON.stringify(response));
        callback(null, response);
```

```
    })
    .catch((error) => {
      console.log(
        "There has been a problem with your fetch operation: " + error.message
      );
      var responseCode = 500;

      let response = {
        statusCode: responseCode,
        body: JSON.stringify(error),
      };

      console.log("response: " + JSON.stringify(response));
      callback(null, response);
    });
  } //end if
};
```

Step 2:

Run the following command as shown below to zip up the index.js and dynamodb.js into a function.zip.

Command: zip function.zip index.js dynamodb.js



Step 3:

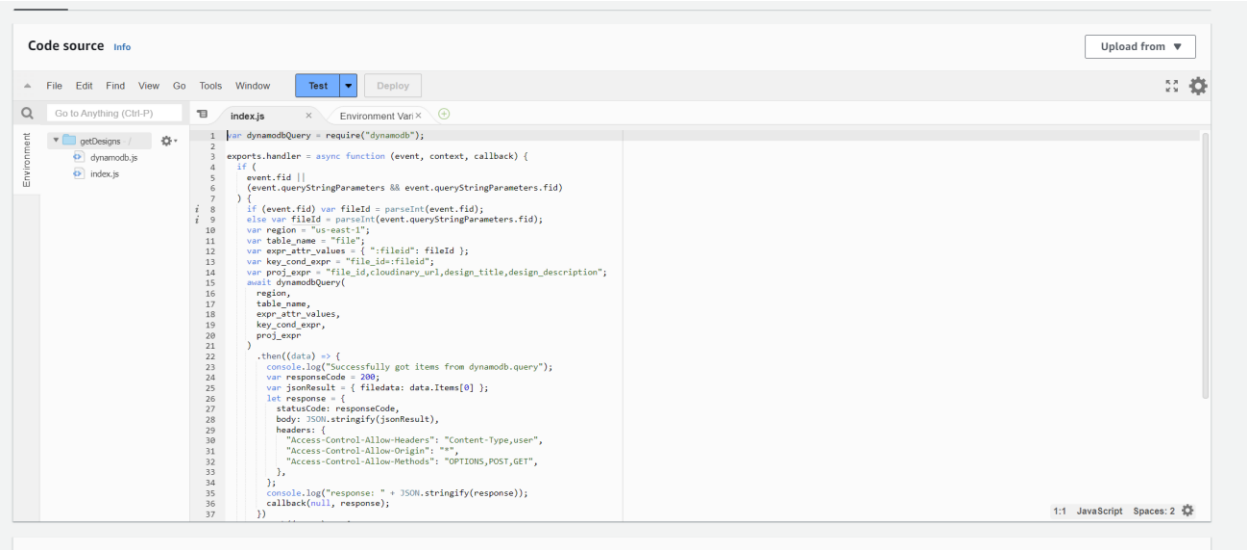
Run the following command to bring your Cloud9 dynamodb.js and index.js files into the Lamba function called getDesigns.

Command: aws lambda update-function-code --function-name getDesigns --zip-file fileb://function.zip

```
voclabs:~/environment $ aws lambda update-function-code --function-name getDesigns --zip-file fileb://function.zip
{
    "LastUpdateStatus": "InProgress",
    "FunctionName": "getDesigns",
    "LastModified": "2023-07-10T06:26:13.000+0000",
    "RevisionId": "d9e4610d-d4f6-48b6-9dc9-e09fb75b9deb",
    "LastUpdateStatusReason": "The function is being created.",
    "MemorySize": 128,
    "State": "Active",
    "Version": "$LATEST",
```

Step 4:

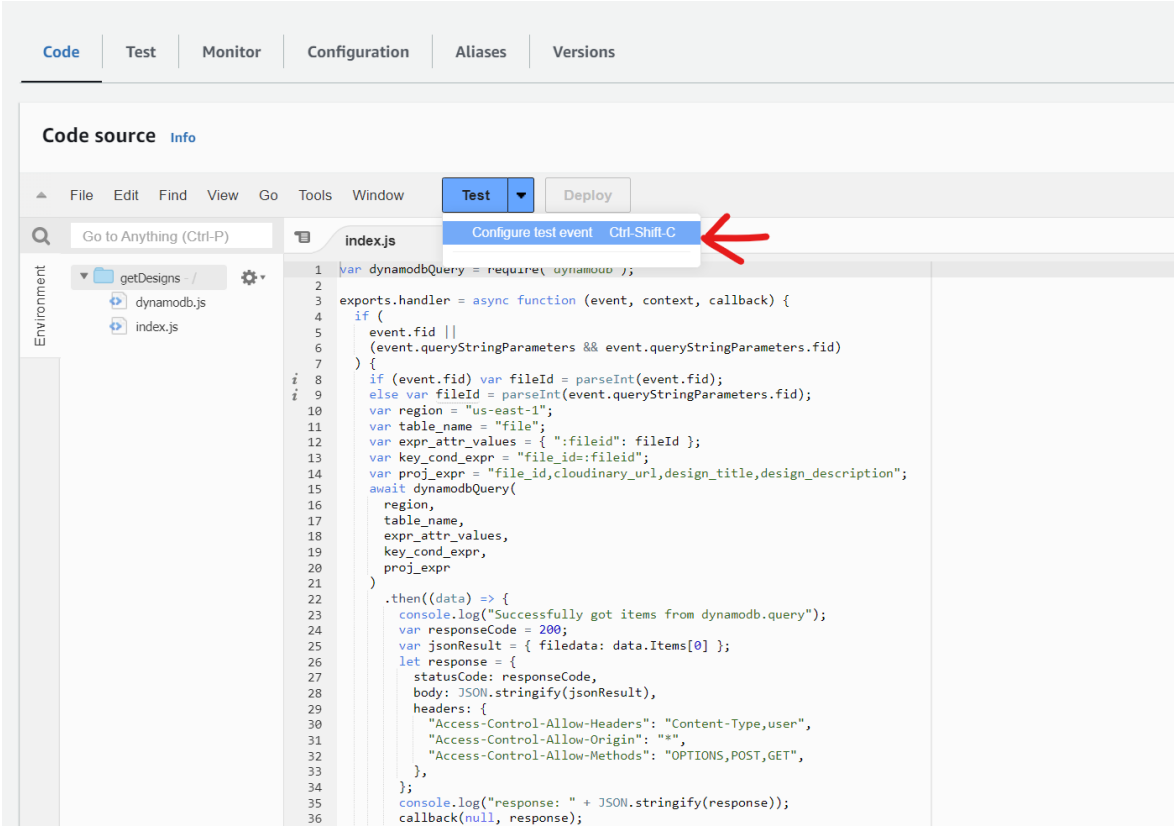
When you go back to your getDesigns Lamba function you should see the dynamodb.js and index.js files filled with the respective codes.



5.3 Configure Test Event for Lambda function getDesigns

Step 1:

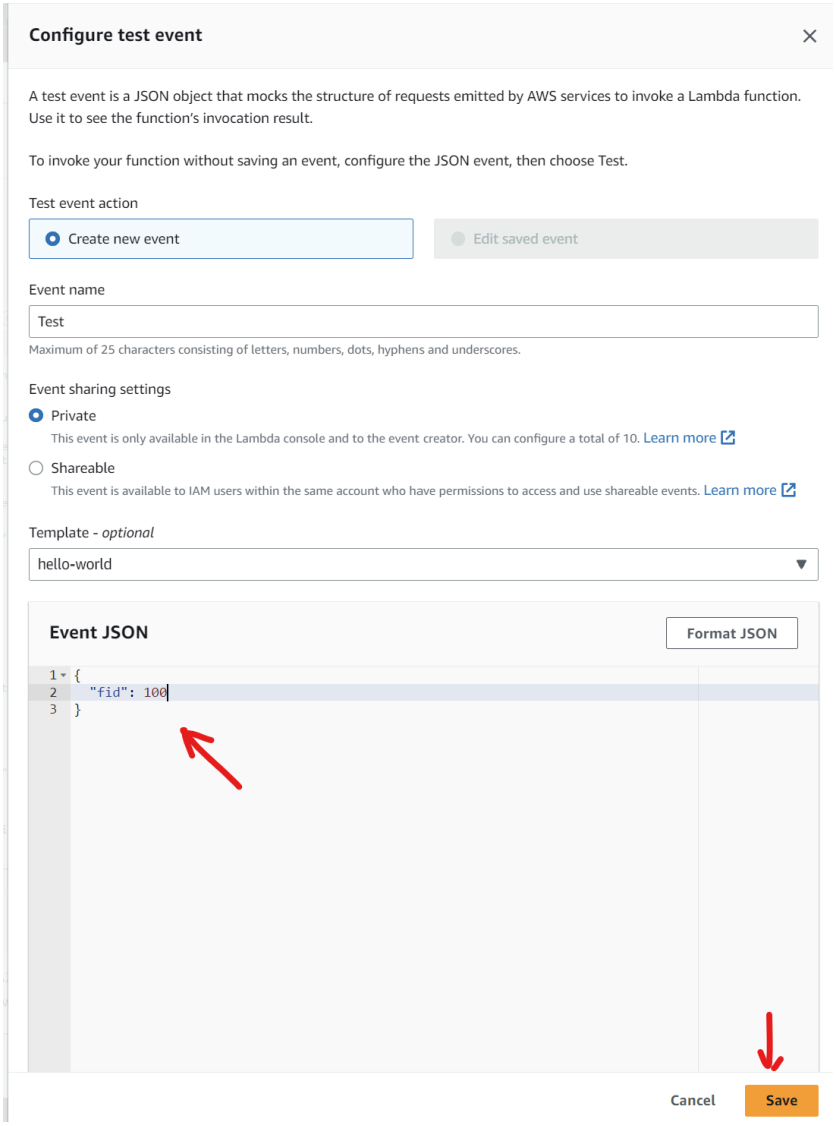
Under Test, click on “Configure test event” to create a new event to test the Lambda function getDesigns.



Step 2:

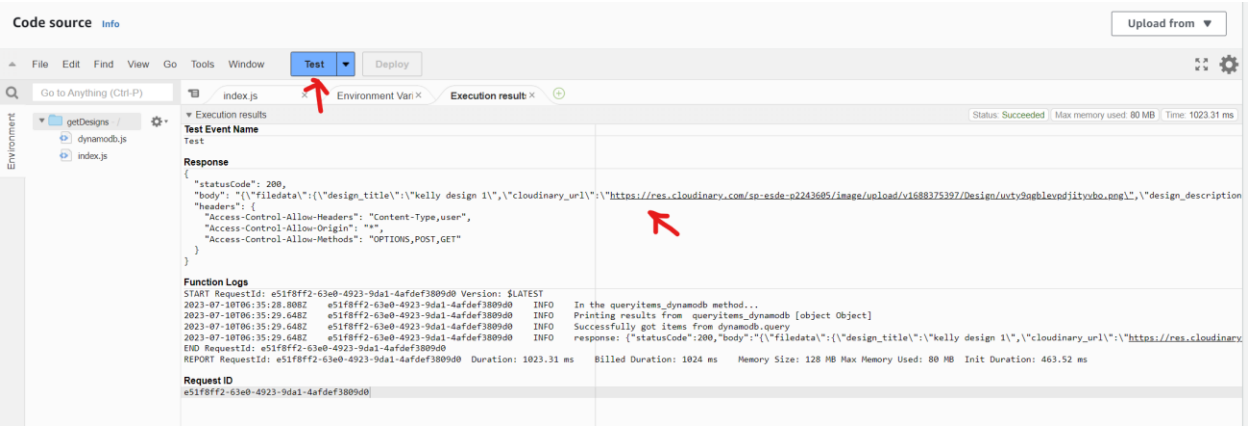
Give the test event a name of “Test” and replace the Event JSON with the following as shown below.

Note: The following is to retrieve the Kelly Design 1.



Step 3:

Finally, run the Test and you should see that the response shows Kelly's Design 1.

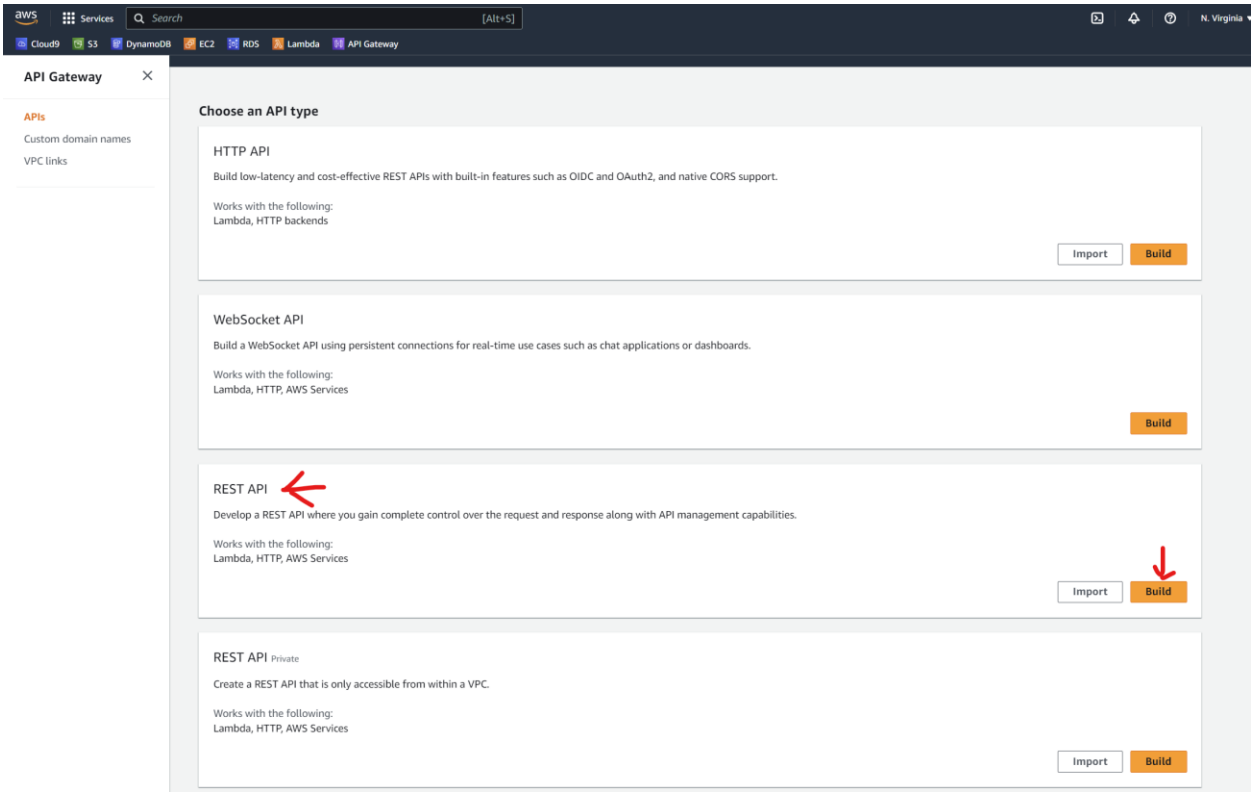


Part 6 – API Gateway

6.1 Creating an API Gateway for Lambda Function getDesigns

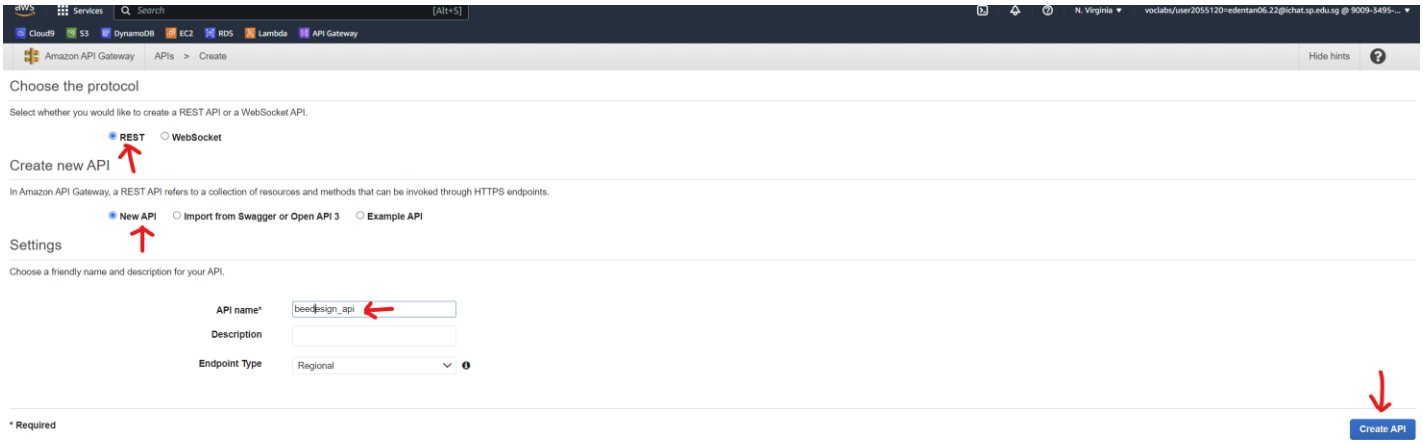
Step 1:

Go into AWS API Gateway Dashboard and click on the “Build” under REST API.



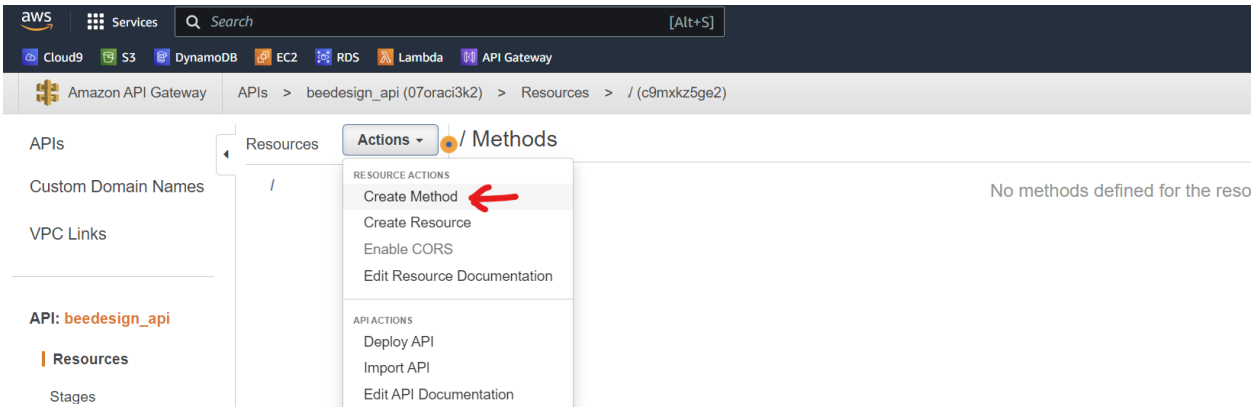
Step 2:

Make sure that you selected REST, and New API. For the API Name, name it as beedesign_api, and click on “Create API”.



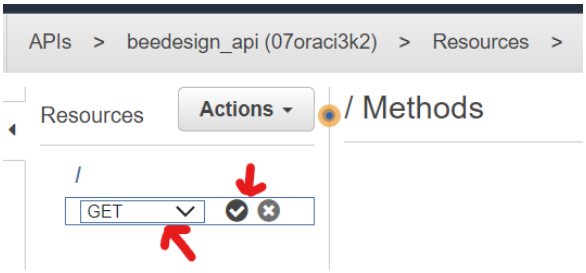
Step 3:

Click on the / (Root) and Under the Actions click on Create Method.



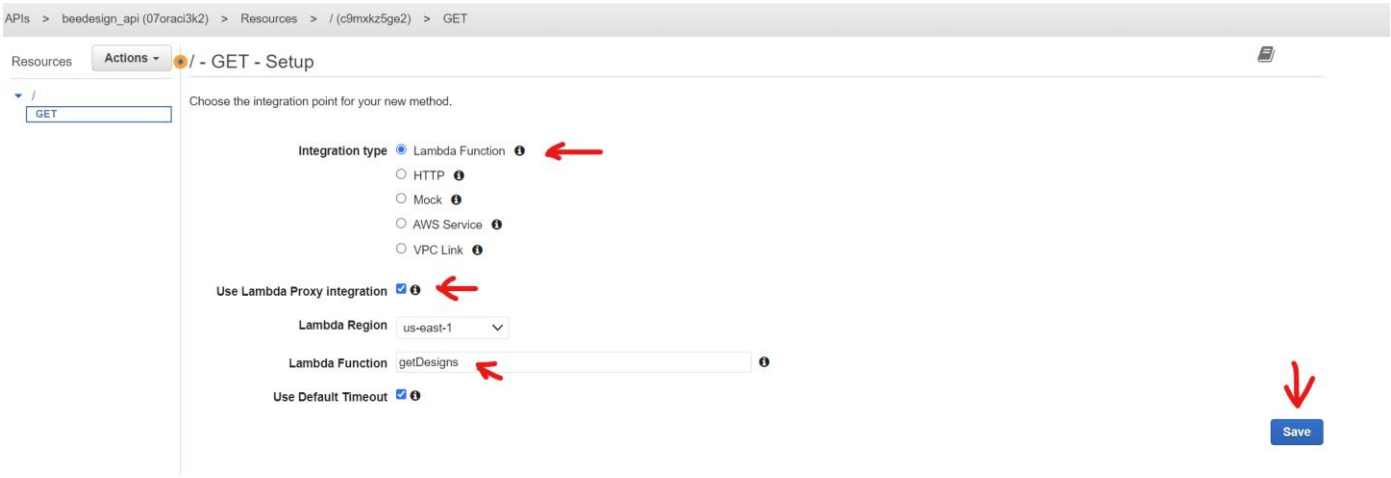
Step 4:

Under the Dropdown list select “GET” Method and click on the tick icon as shown below.



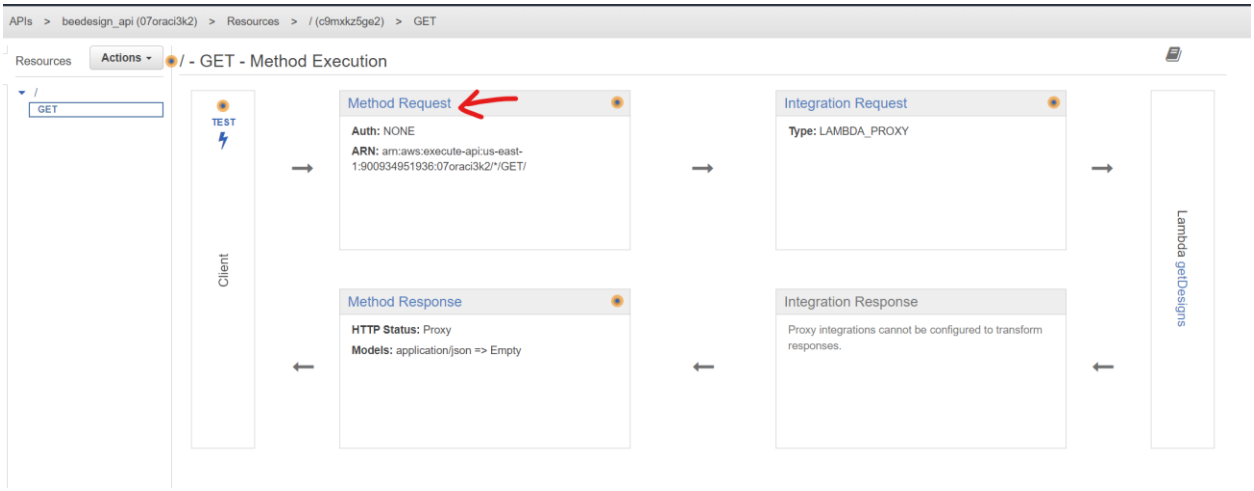
Step 5:

Make sure that you have selected the following as shown below for the GET Method. Select Lambda Function, Check “Use Lambda Proxy Integration” and for the Lambda Function name it as “getDesigns” and then click on the “Save” button.



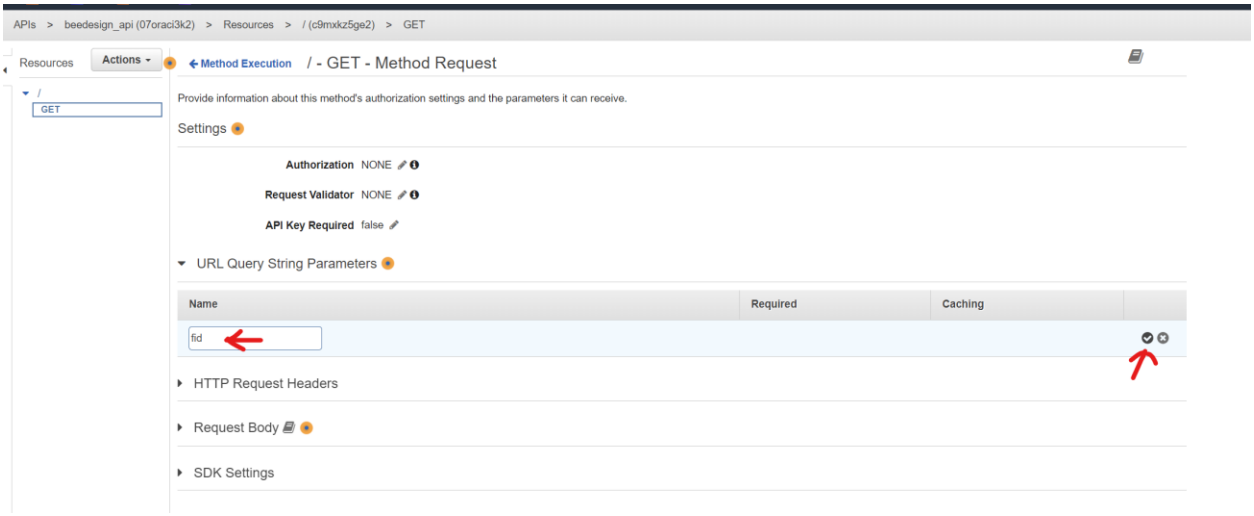
Step 6:

You will see the same as shown below, Click on the Method Request.



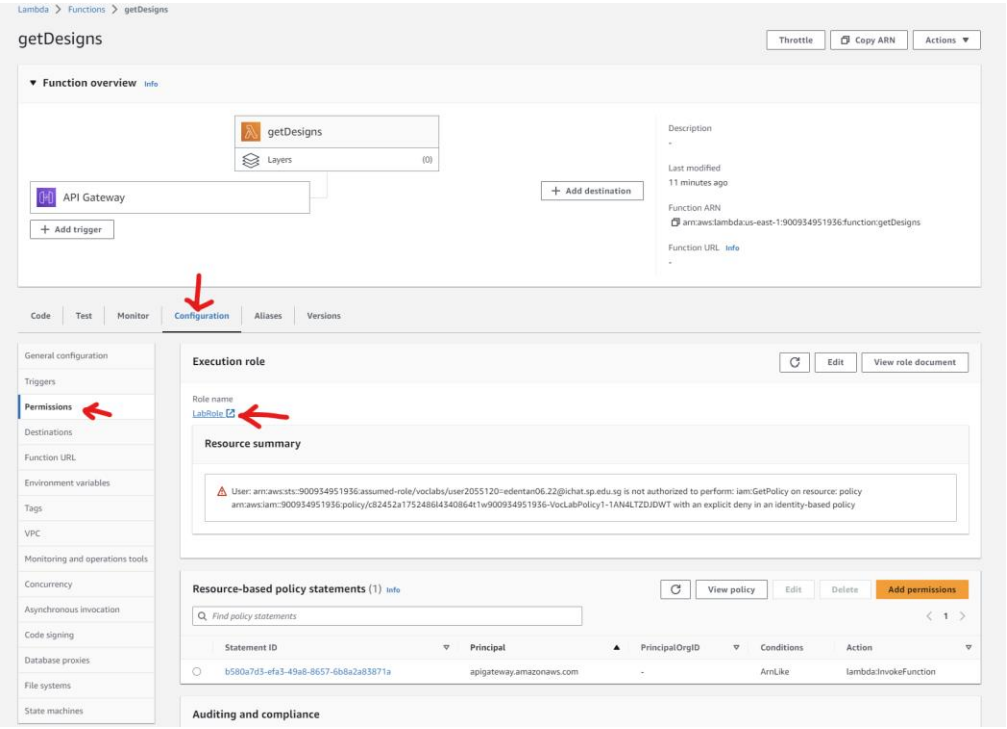
Step 7:

Under URL Query String Parameters add a new query string called “fid” and click on the tick icon to save it.



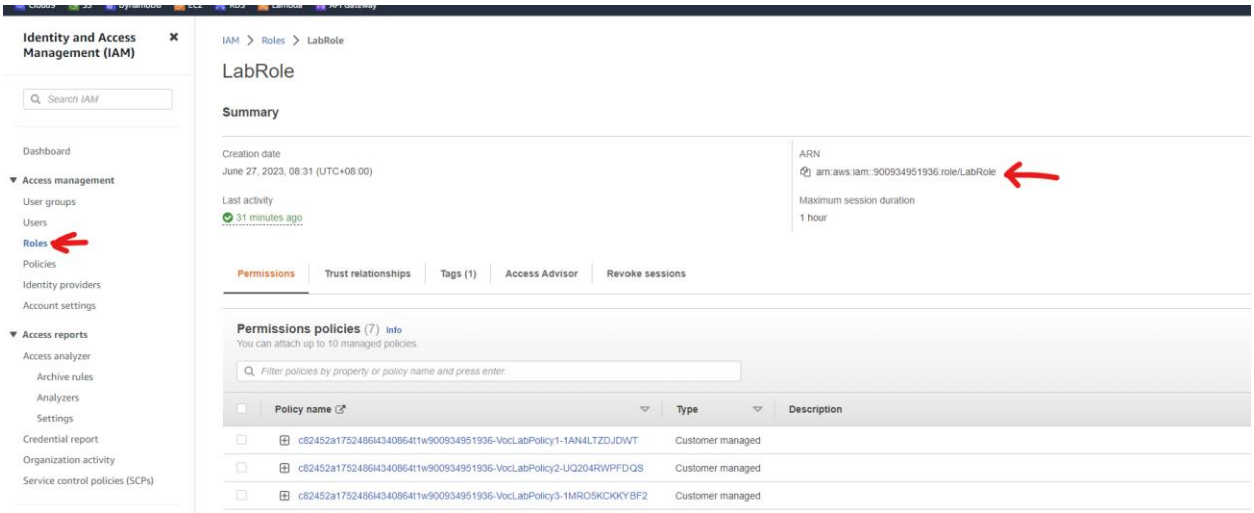
Step 8:

Next, go to your getDesigns Lambda function and click on “Configuration” and then “Permissions” as shown below and click on “LabRole” as shown below.



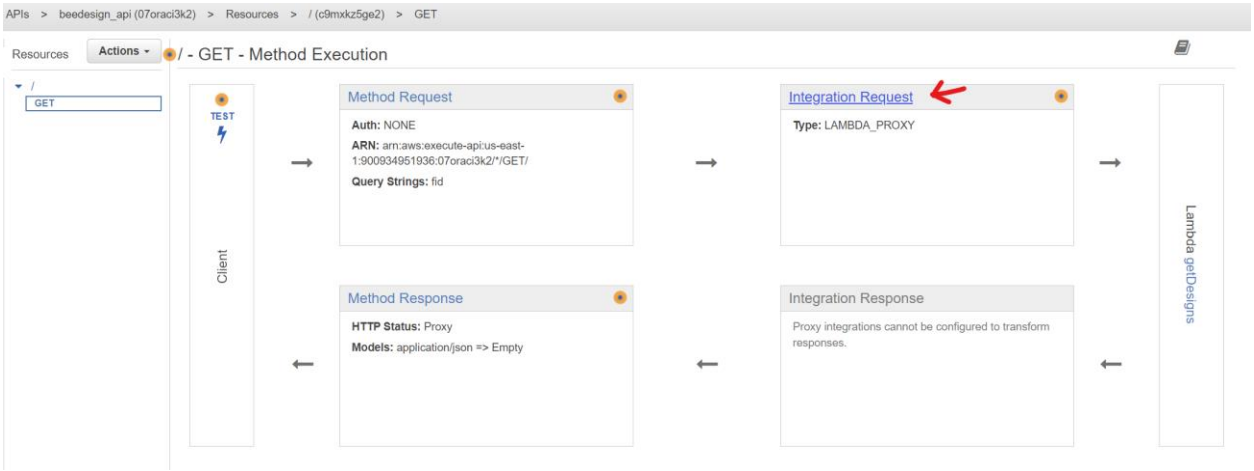
Step 9:

Under the IAM > Roles, copy the ARN and keep it somewhere where we can retrieve it later.



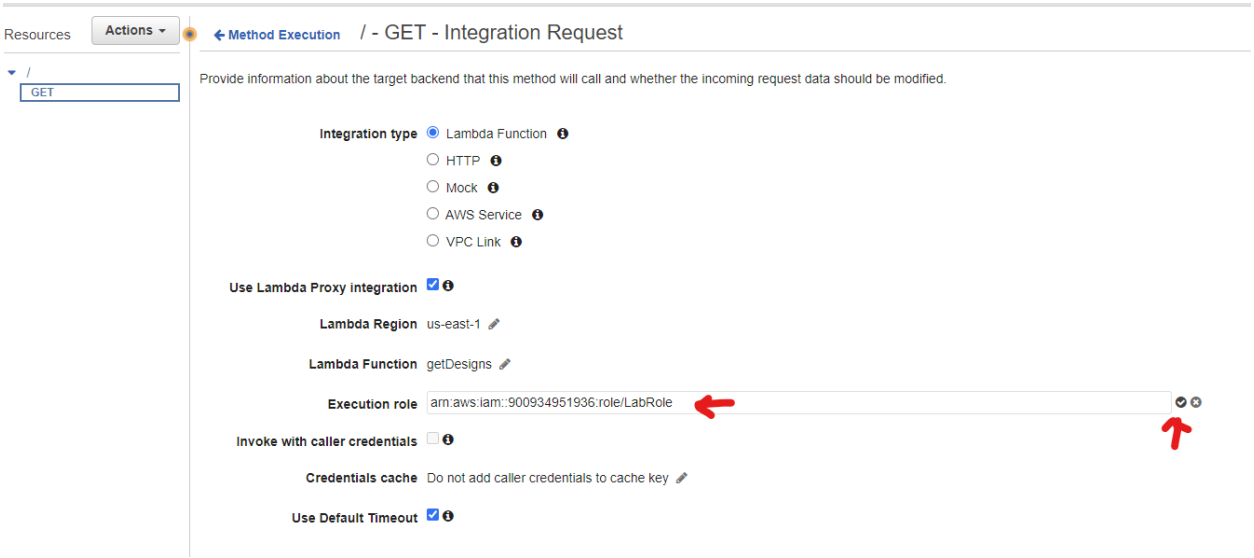
Step 10:

Next, go back to the GET Method in your API Gateway and click on Integration Request.



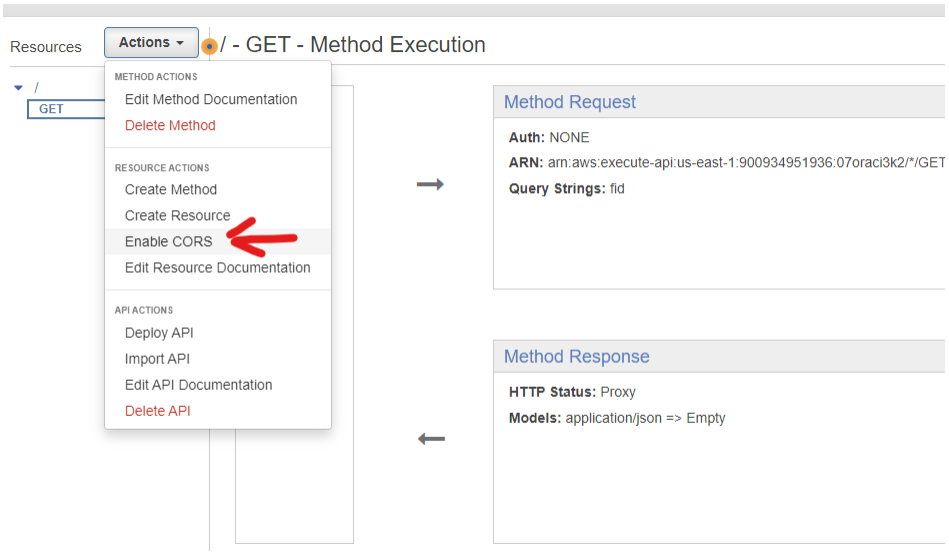
Step 11:

Paste the ARN Role in the Execution role as shown below and click on the Tick Icon to save it.



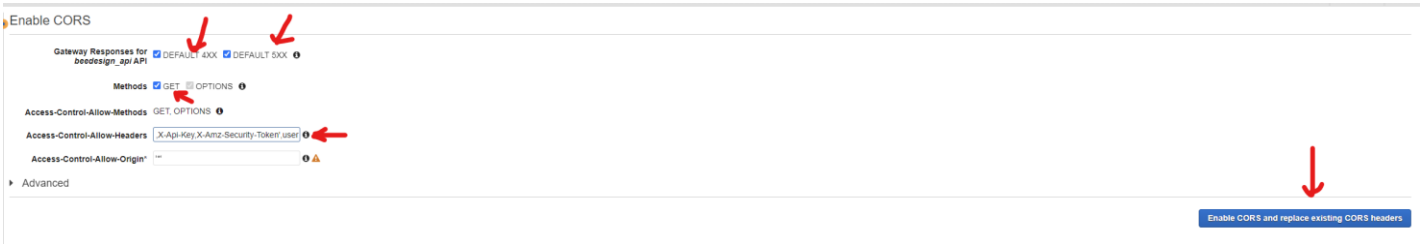
Step 12:

Under Actions, click on Enable CORS.

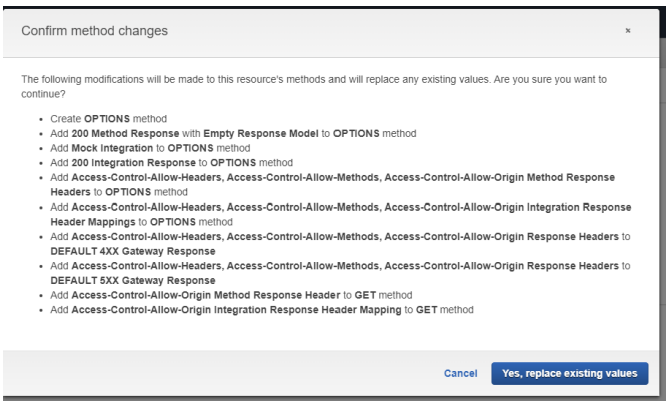


Step 13:

Select DEFAULT 4XX and 5XX, and add a ",user" at the end of Access-Control-Allow-Headers before the closing quotations and click on "Enable CORS" button as shown below.



Click on "Yes, replace existing values".

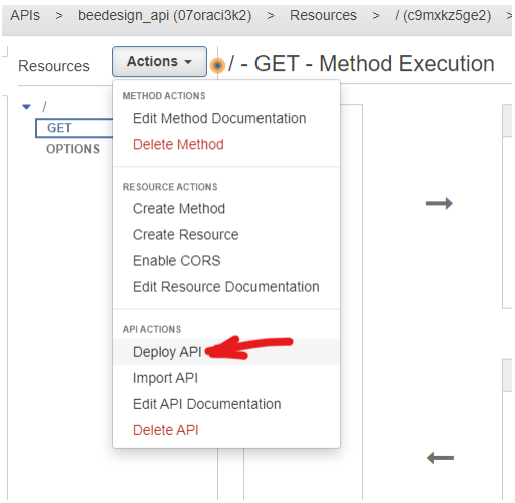


Enable CORS

- ✓ Create **OPTIONS** method
- ✓ Add **200 Method Response** with **Empty Response Model** to **OPTIONS** method
- ✓ Add **Mock Integration** to **OPTIONS** method
- ✓ Add **200 Integration Response** to **OPTIONS** method
- ✓ Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Method Response Headers** to **OPTIONS** method
- ✓ Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Integration Response Header Mappings** to **OPTIONS** method
- ✓ Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Response Headers** to **DEFAULT 4XX Gateway Response**
- ✓ Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Response Headers** to **DEFAULT 5XX Gateway Response**
- ✓ Add **Access-Control-Allow-Origin Method Response Header** to **GET** method
- ✓ Add **Access-Control-Allow-Origin Integration Response Header Mapping** to **GET** method

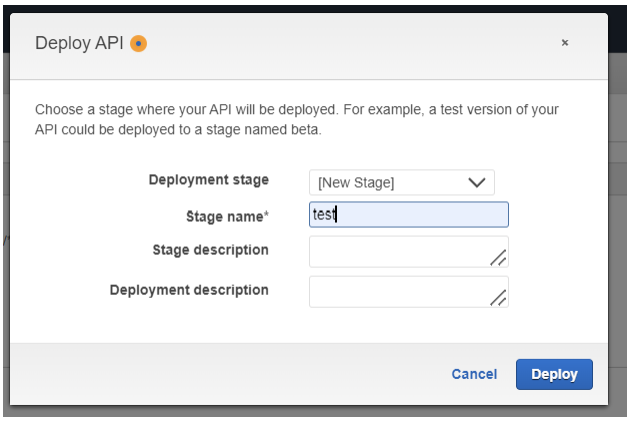
Step 14:

Click on “Deploy API” under Actions.



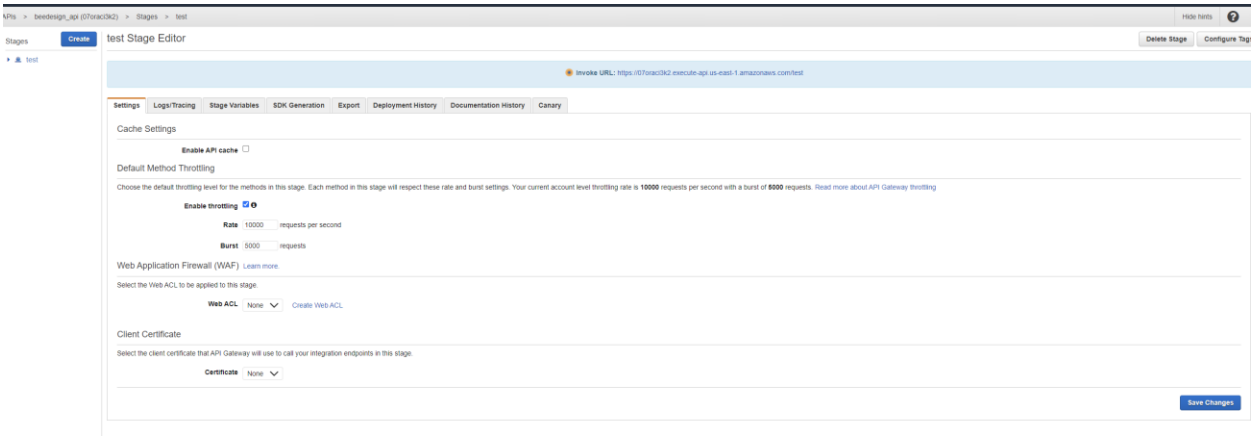
Step 15:

Select “New Stage” and Enter “test” as the Stage name and click on “Deploy” button.



Step 16:

You should be able to see the Invoke URL. Copy it and save it somewhere for us to use later.



6.2 Setting up the update design file in WinSCP

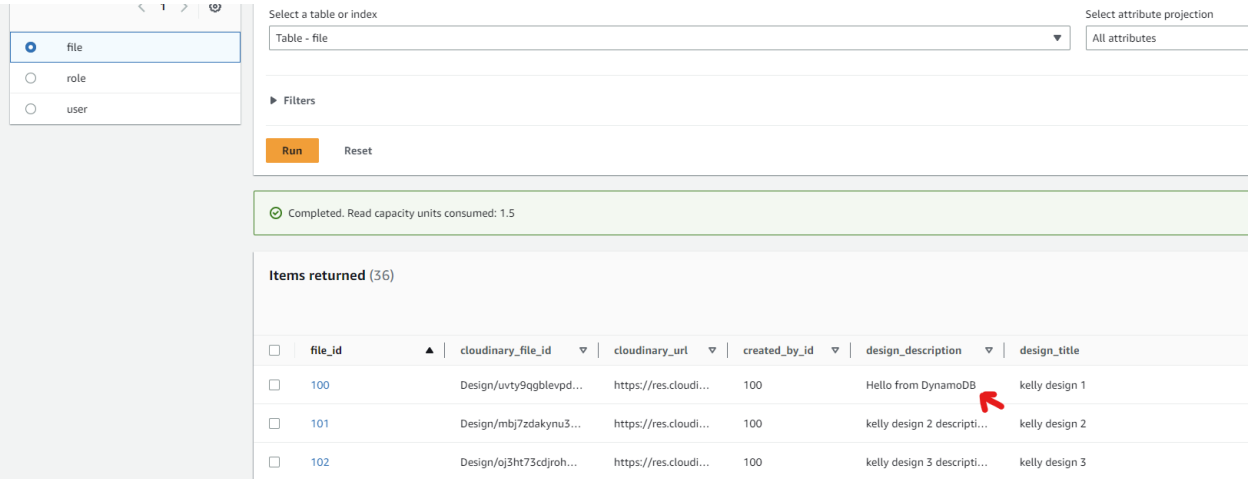
Step 1:

Go to the update_design.js under ESDE/FE/public/js/update_design.js under WinSCP and change the getOneData() function url to the following below.

```
function getOneData() {
  const baseUrl = "https://18.205.232.250:5000";
  //Get the fileId information from the web browser URL textbox
  let query = window.location.search.substring(1);
  let arrayData = query.split("=");
  let fileId = arrayData[1];
  let token = sessionStorage.getItem("token");
  console.dir("Obtained file id from URL : ", fileId);
  let userId = sessionStorage.getItem("user_id");
  axios({
    headers: {
      user: userId,
      authorization: `Bearer ${token}`,
    },
    method: "get",
    url: 'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test?fid=' + fileId,
  })
    .then(function (response) {
```

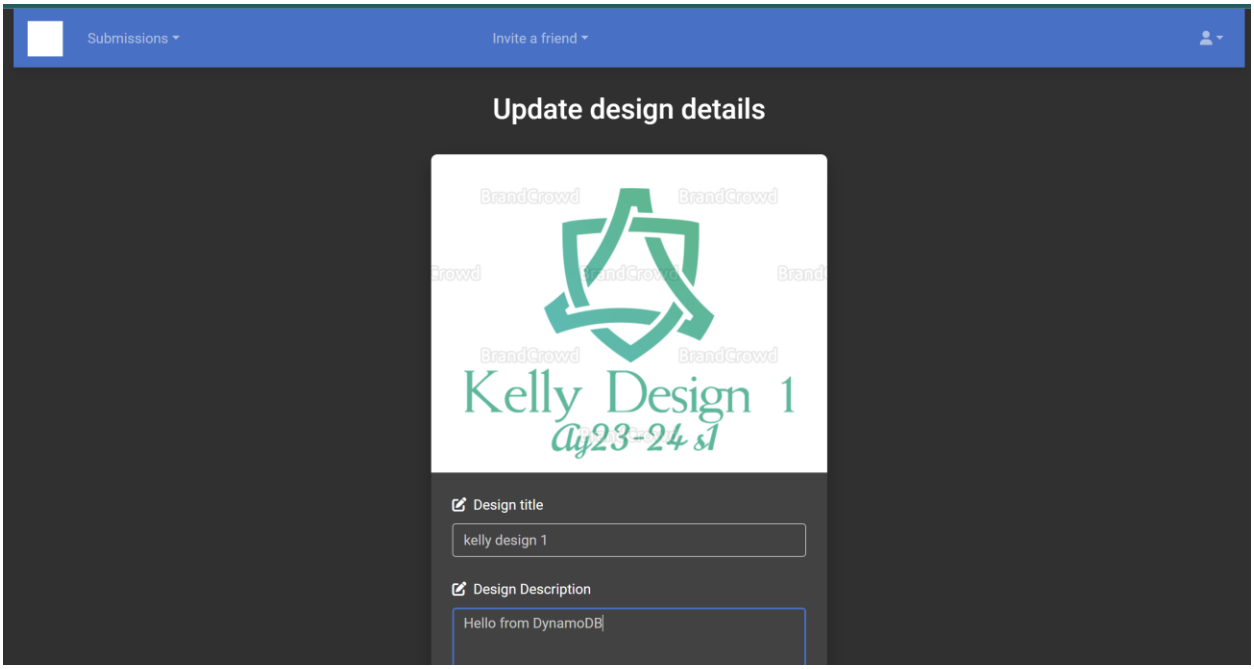
Step 2:

Go into DynamoDB and change the file_id 100 design description to “Hello from DynamoDB” to check if it is reflected in our website.



Step 3:

Go into your website and login as Kelly and view the 1st design. Click on Update and you should see that you are able to view the “Hello from DynamoDB”.

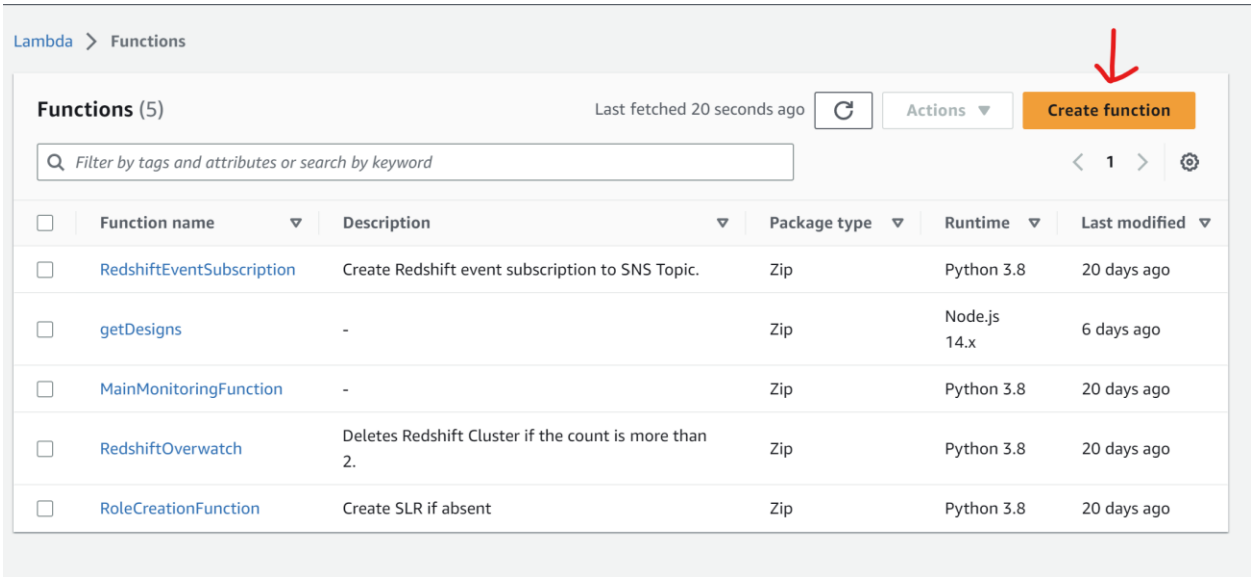


7. Additional Features

7.1 Creating Update Design Lambda Function

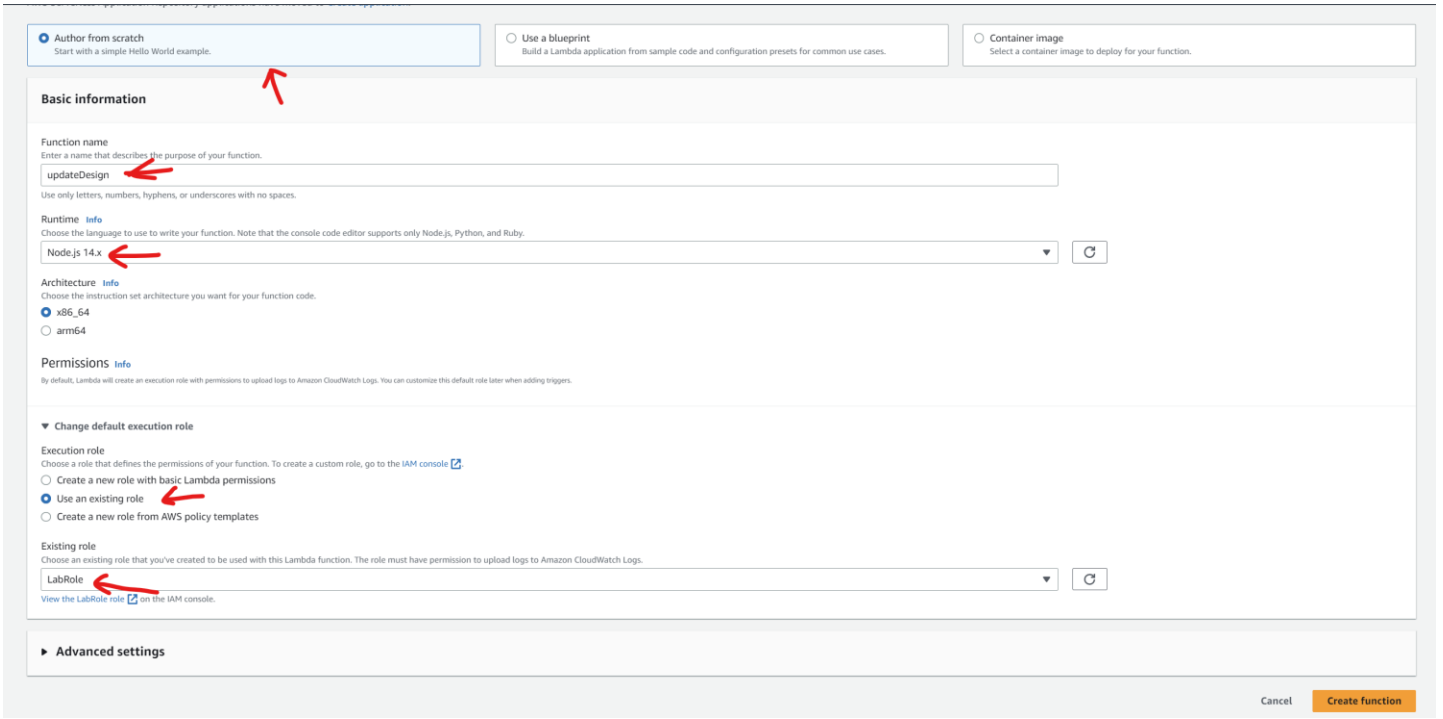
Step 1:

Go into AWS Lambda and click on “Create function” to create a new Lambda function.



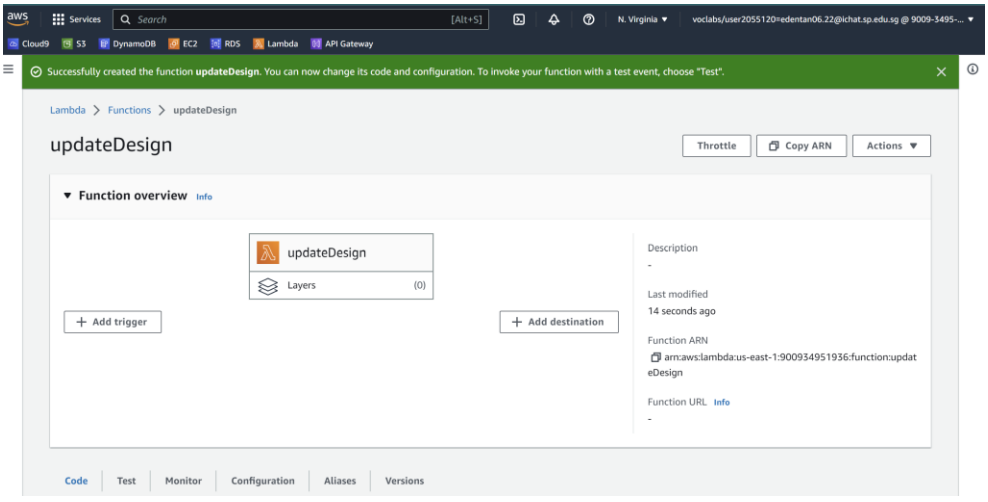
Step 2:

Select “Author from scratch”, and type “updateDesign” as the function name, and for the runtime make sure that you select “Node.js 14.x” and for default execution role select “Use an existing role” and select LabRole and click on the “Create function” button as shown below.



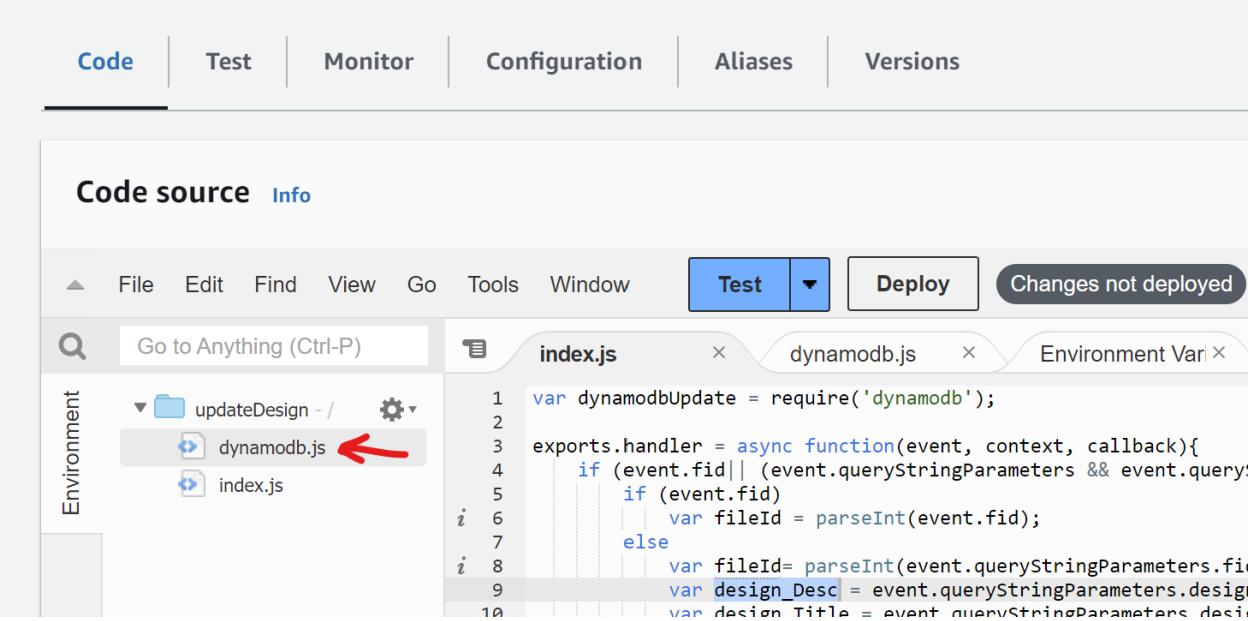
Step 3:

You should be able to see a successfully created message as shown below.



Step 4:

At the bottom under Code Source create a file called dynamodb.js.



Step 5:

Type in the following codes for index.js file and dynamodb.js file as shown below.

dynamodb.js codes:

```
var AWS = require('aws-sdk');

async function updateitems_dynamodb(region,
table_name,expr_attr_values,key_cond_expr,proj_expr,design_Desc,design_Titl
e) {
   console.log("In the queryitems_dynamodb method...");
   var dynamodb = new AWS.DynamoDB({region: region});
   try{
      var params = {
         Key:{
            "file_id": {
               N: String(expr_attr_values)
            }
         },
         ExpressionAttributeValues:{
```

```
            ":dD": {
               S: design_Desc
            },
            ":dT": {
               S: design_Title
            },
         },
         TableName: table_name,
         UpdateExpression: `SET design_description=:dD, design_title=:dT`
      };

      return dynamodb.updateItem(params, function(err, data) {
         console.log(err, data);
      });

   } catch(tryerror) {
      console.log("Error occurred!");
      console.log(tryerror, tryerror.stack); // an error occurred
   }
} //end function

module.exports = updateitems_dynamodb;
```

index.js codes

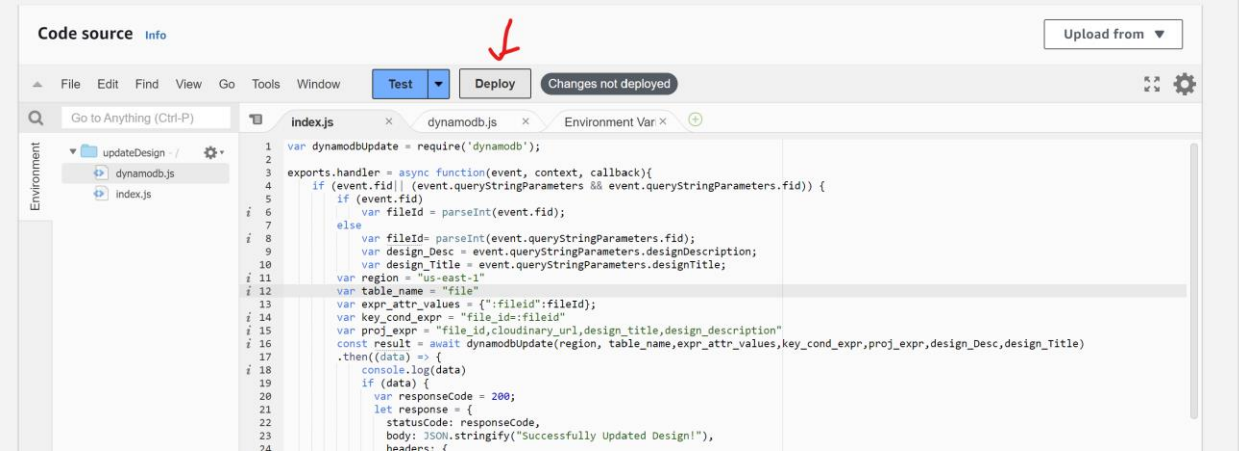
```
var dynamodbUpdate = require('dynamodb');

exports.handler = async function(event, context, callback){
   if (event.fid|| (event.queryStringParameters &&
event.queryStringParameters.fid)) {
      if (event.fid)
         var fileId = parseInt(event.fid);
      else
         var fileId= parseInt(event.queryStringParameters.fid);

      if (event.designTitle)
         var design_Title = event.designTitle;
```

```
    else
      var design_Title = event.queryStringParameters.designTitle;

    if (event.designDescription)
      var design_Desc = event.designDescription;
    else
      var design_Desc = event.queryStringParameters.designDescription;

    var region = "us-east-1"
    var table_name = "file"
    var expr_attr_values = fileId;
    var key_cond_expr = "file_id=:fileid"
    var proj_expr = "file_id,cloudinary_url,design_title,design_description"
    const result = await dynamodbUpdate(region,
table_name,expr_attr_values,key_cond_expr,proj_expr,design_Desc,design_Titl
e)
    .then((data) => {
      console.log(data)
      if (data) {
       var responseCode = 200;
       let response = {
        statusCode: responseCode,
        body: JSON.stringify("Successfully Updated Design!"),
        headers: {
          "Access-Control-Allow-Headers" : "Content-Type,user",
          "Access-Control-Allow-Origin": "*",
          "Access-Control-Allow-Methods": "OPTIONS,POST,GET,PUT"
        },
        "isBase64Encoded": true
       }
       console.log("response: " + JSON.stringify(response))
       callback(null, response);
      }
      else {
       var responseCode = 500;
       let response = {
        statusCode: responseCode,
```

```
            body: JSON.stringify("Failed to Update Design."),
            headers: {
               "Access-Control-Allow-Headers" : "Content-Type,user",
               "Access-Control-Allow-Origin": "*",
               "Access-Control-Allow-Methods": "OPTIONS,POST,GET,PUT"
            },
            "isBase64Encoded": true
          }
          console.log("response: " + JSON.stringify(response))
          callback(null, response);
        }
      })
      .catch(error => {
        console.log(error)
        console.log('Error: ' + error.message);
        var responseCode = 500;

        let response = {
           statusCode: responseCode,
           body: JSON.stringify(error)
        }

        console.log("response: " + JSON.stringify(response))
        callback(null, response);
      });
   } //end if
}
```

Step 6:

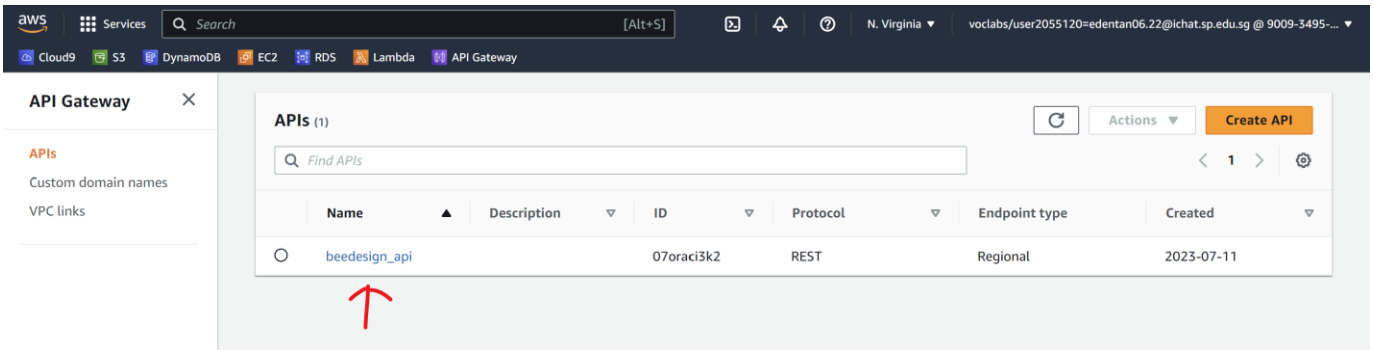
After adding the codes to the respective files click on Deploy as shown below.



7.2 Creating API Gateway for updateDesign Lambda Function

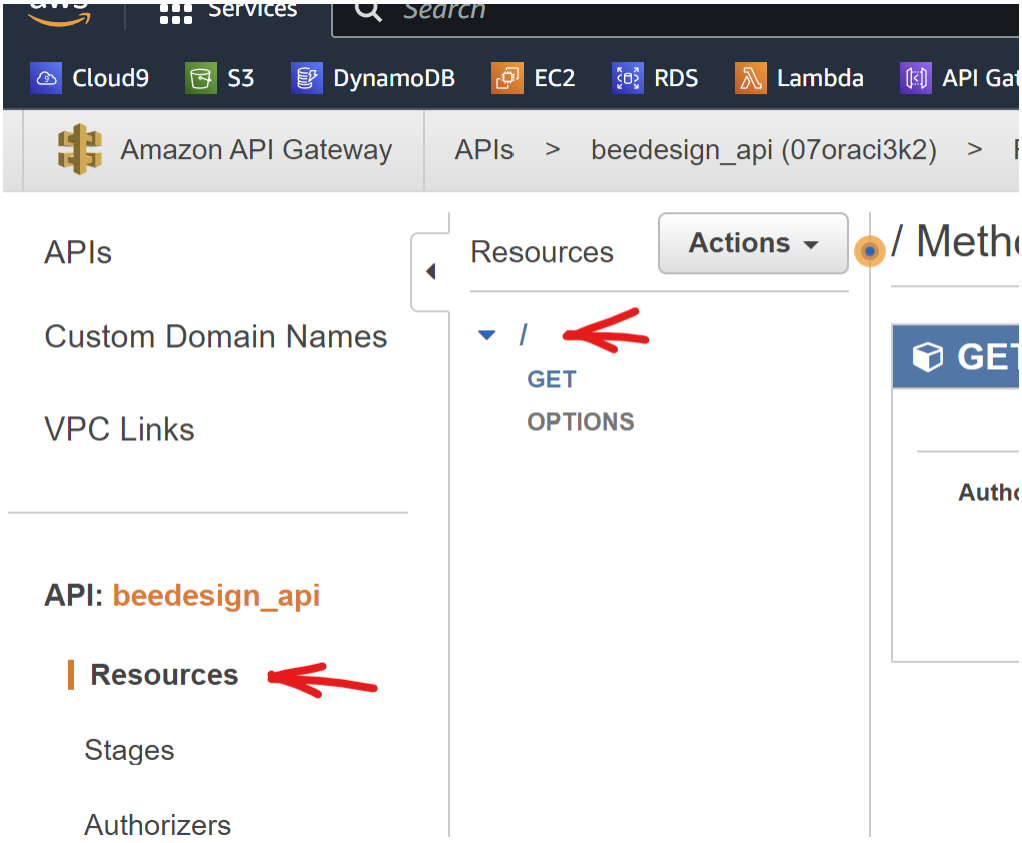
Step 1:

Go to AWS API Gateway and click on beedesign_api that we created above under Part 6 API Gateway.



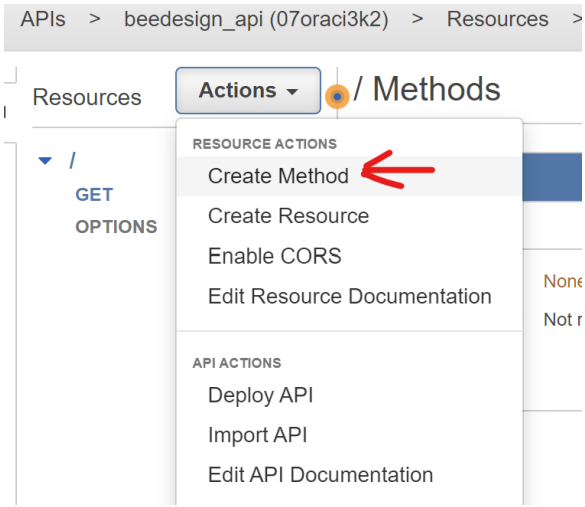
Step 2:

Go to Resources and Click on the / as shown below.



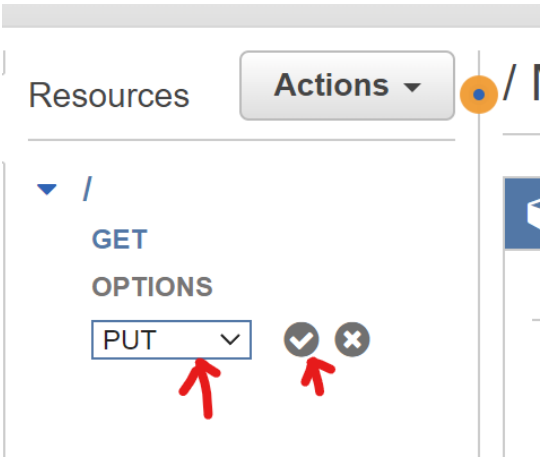
Step 3:

Under Actions, click on “Create Method”.



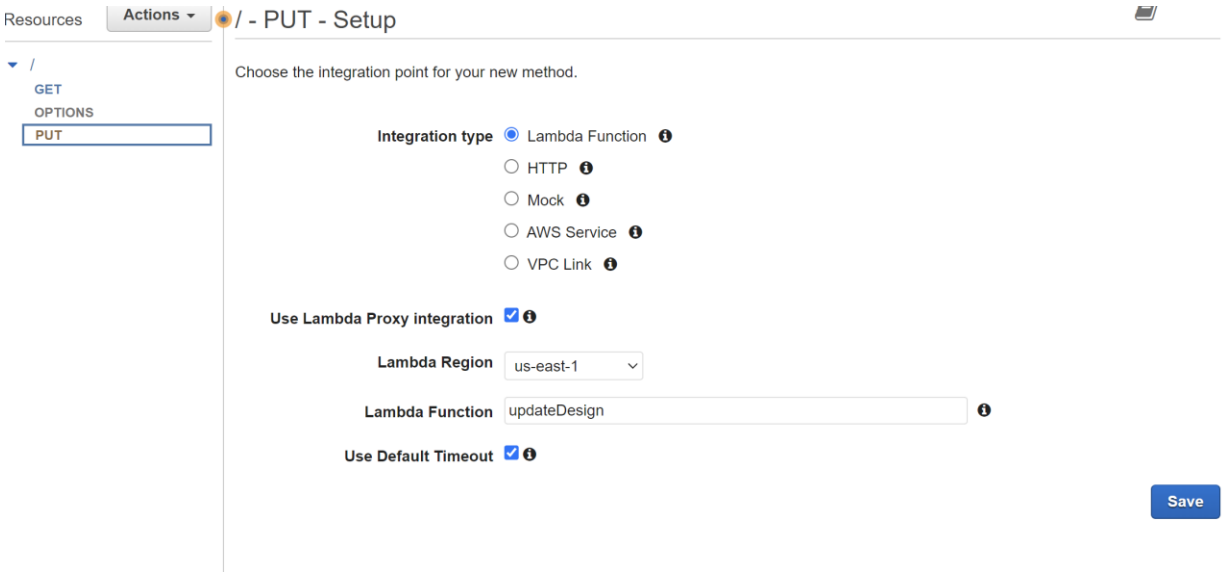
Step 4:

Select PUT from the dropdown list and click on the tick icon to confirm.



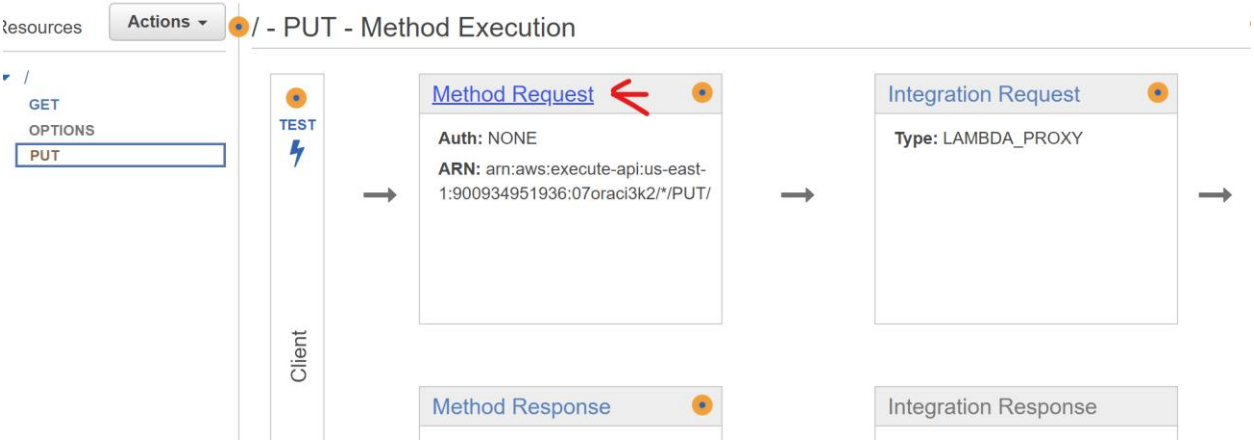
Step 5:

Next, after clicking on the tick icon a setup pane will appear. Select “Lambda Function” for the integration type, and check the use Lambda Proxy integration and for Lambda Function type in “updateDesign” and click on the “Save” button.



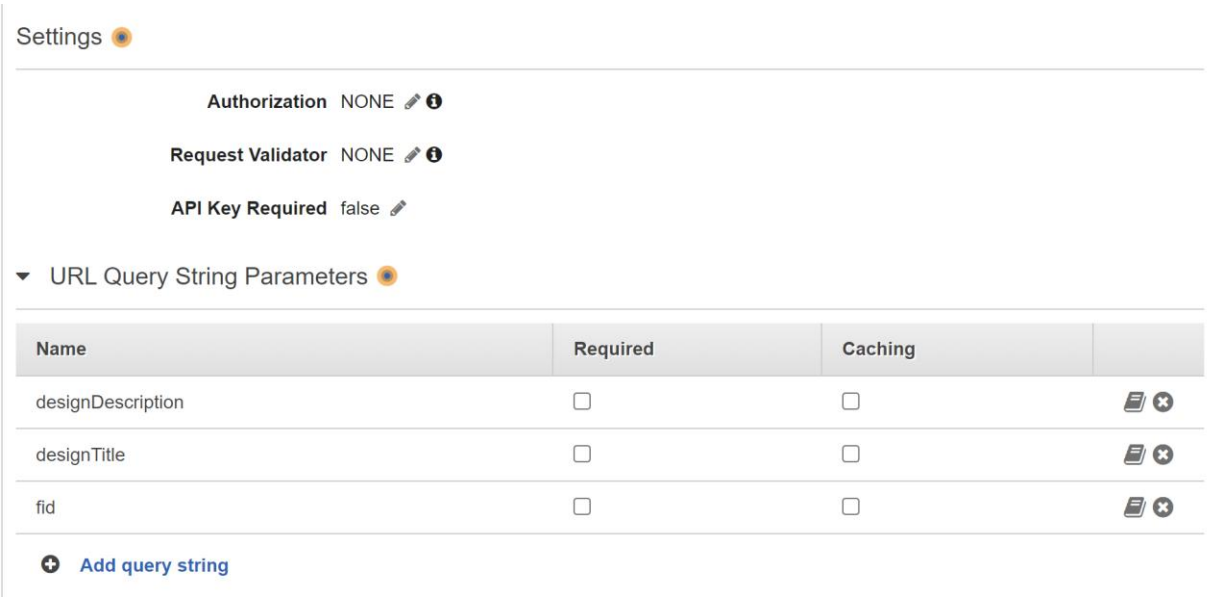
Step 6:

Go to “Method Request” as shown below.



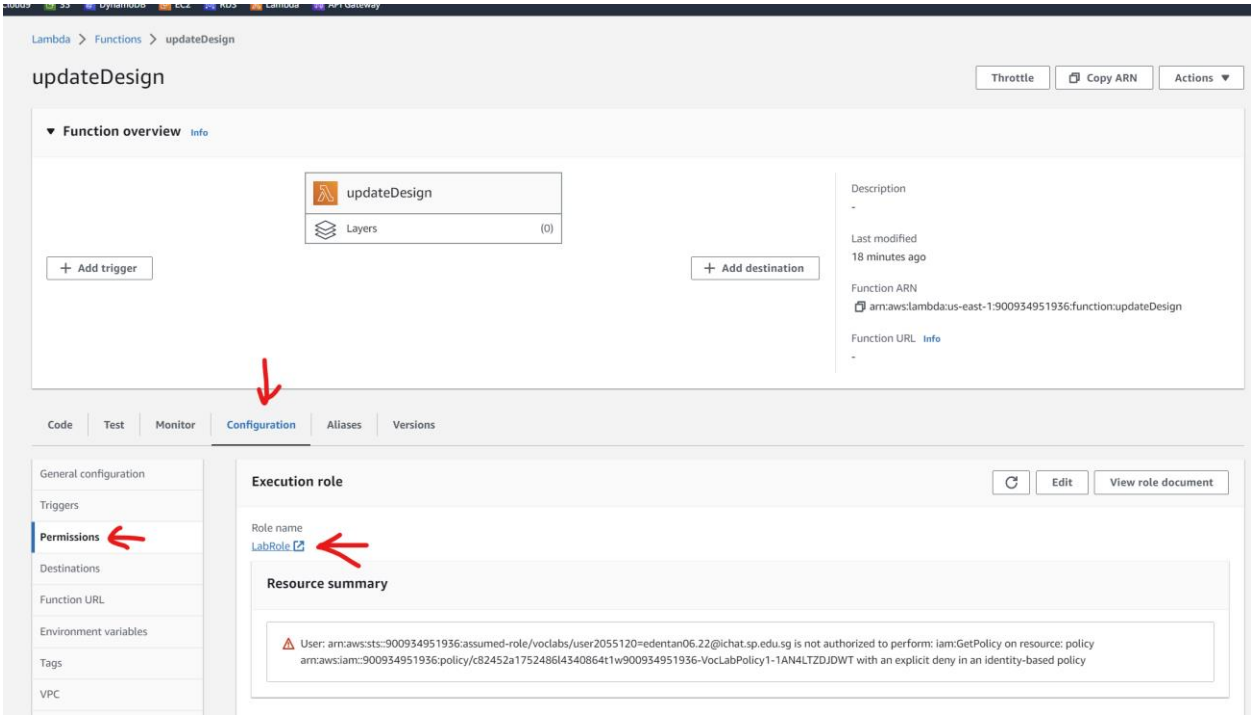
Step 7:

Under URL Query String Parameters add the following as shown below.



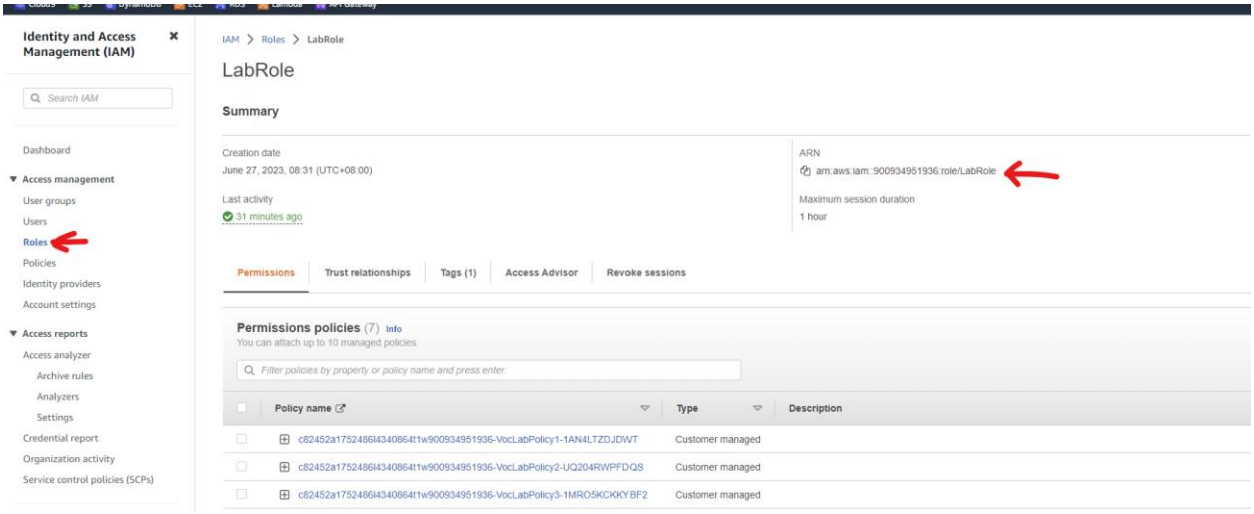
Step 8:

Next, go to your updateDesign Lambda function and click on “Configuration” and then “Permissions” as shown below and click on “LabRole” as shown below.



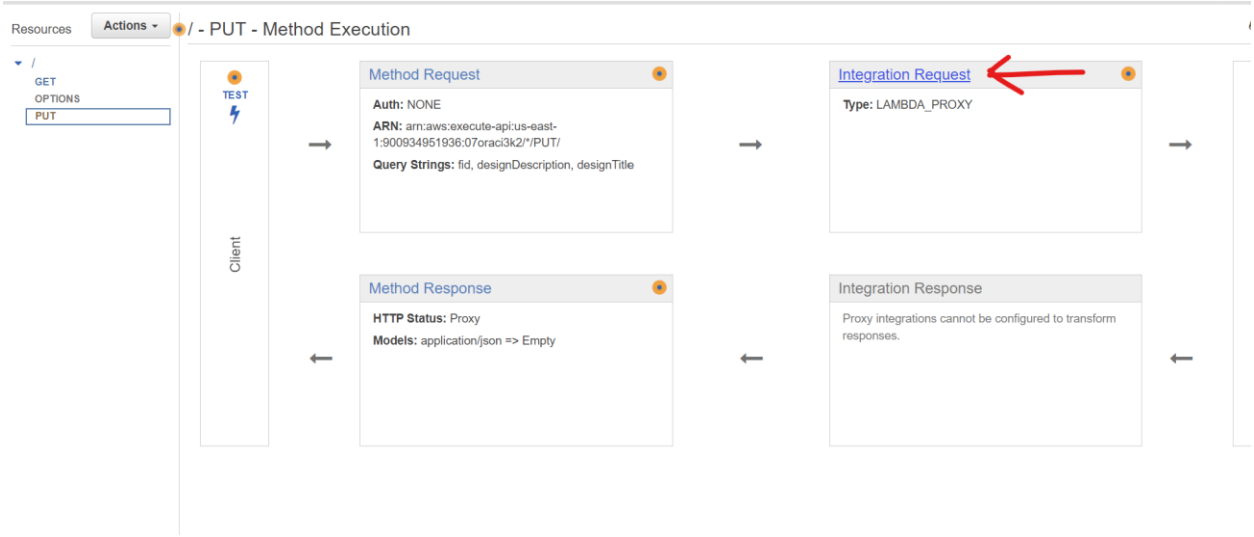
Step 9:

Under the IAM > Roles, copy the ARN and keep it somewhere where we can retrieve it later.



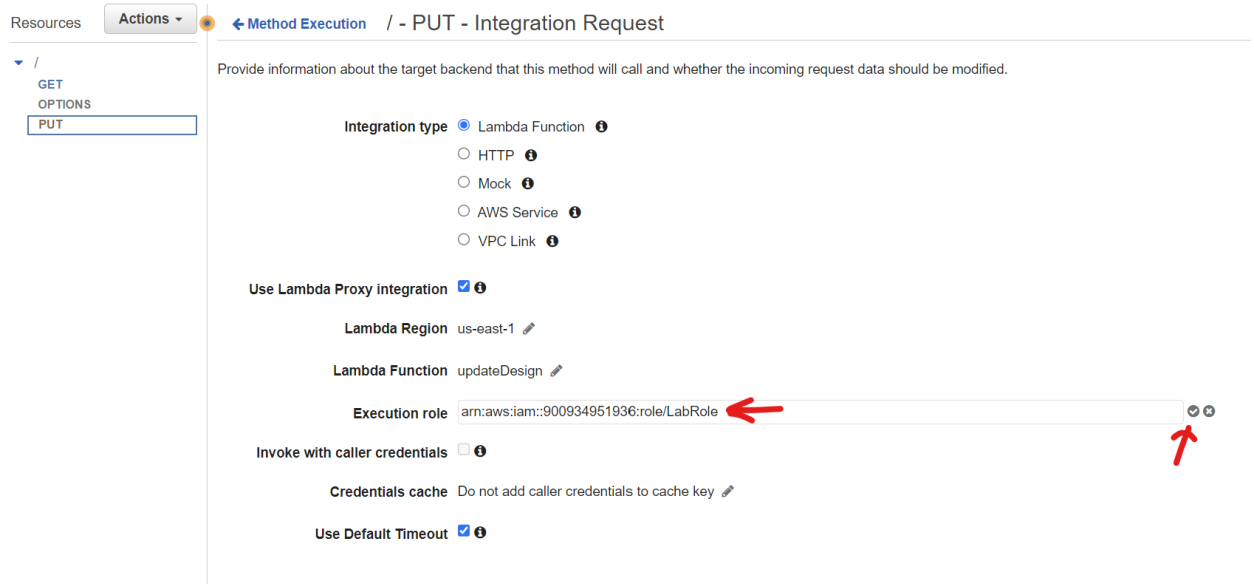
Step 10:

Next, go back to the GET Method in your API Gateway and click on Integration Request.



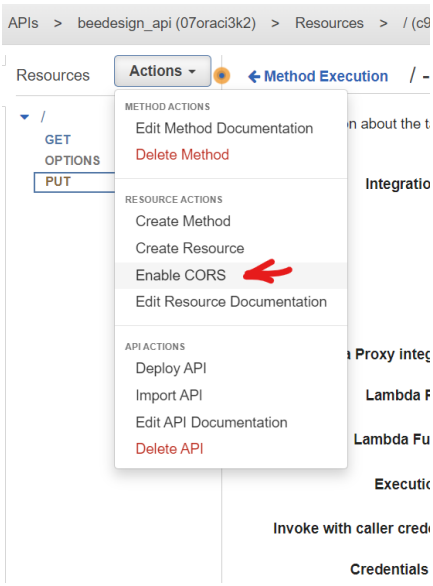
Step 11:

Paste the ARN Role in the Execution role as shown below and click on the Tick Icon to save it.



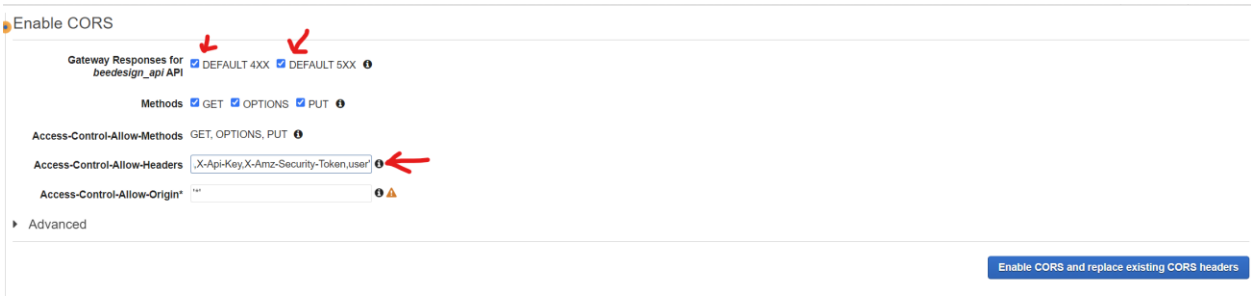
Step 12:

Next, under Actions click on Enable CORS.

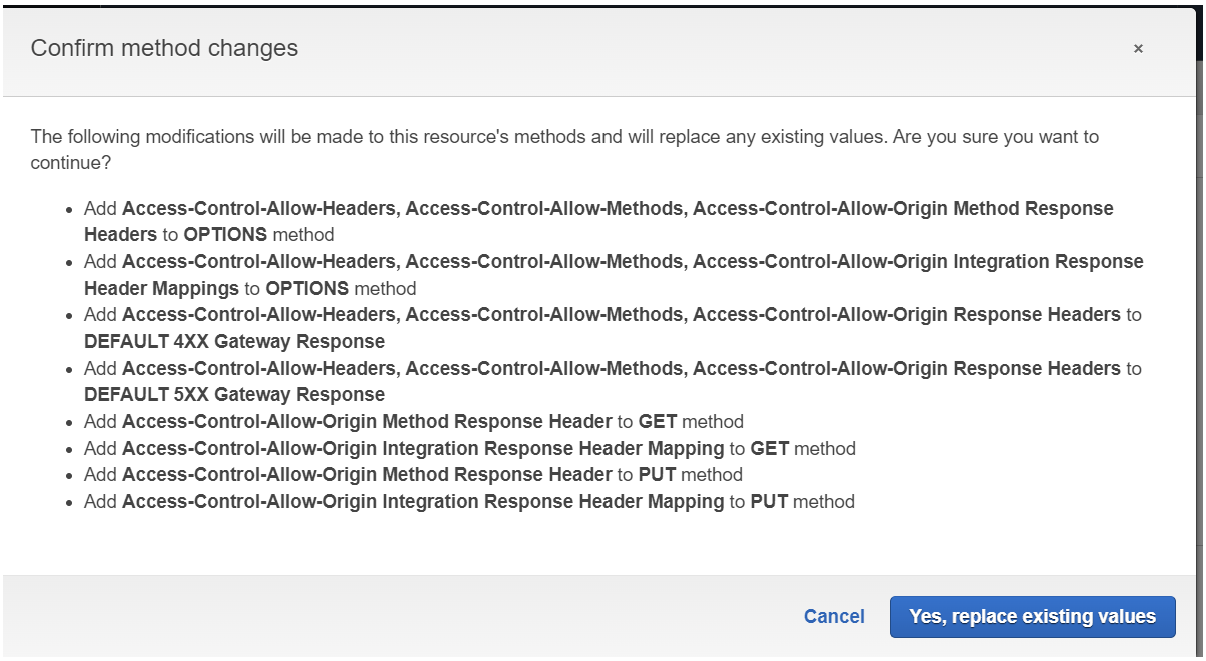


Step 13:

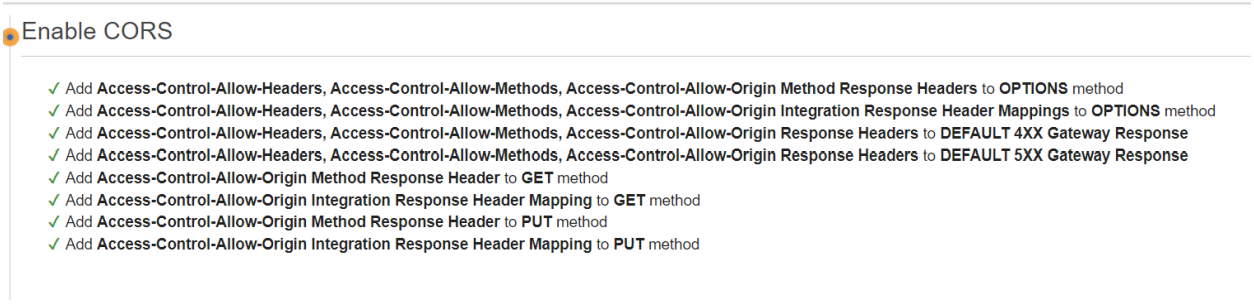
Check on the Default 4XX and 5XX and add a ",user" at the end right before the closing quotation and then click on "Enable CORS" button.



Click on Yes, replace existing values.

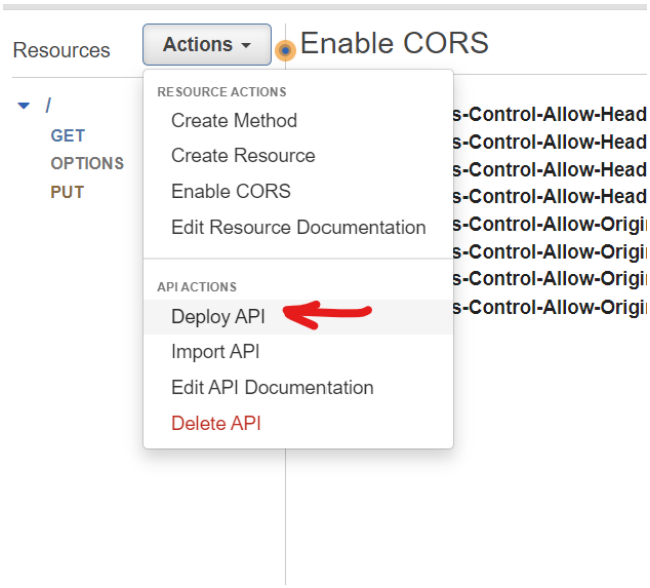


You should see the following.



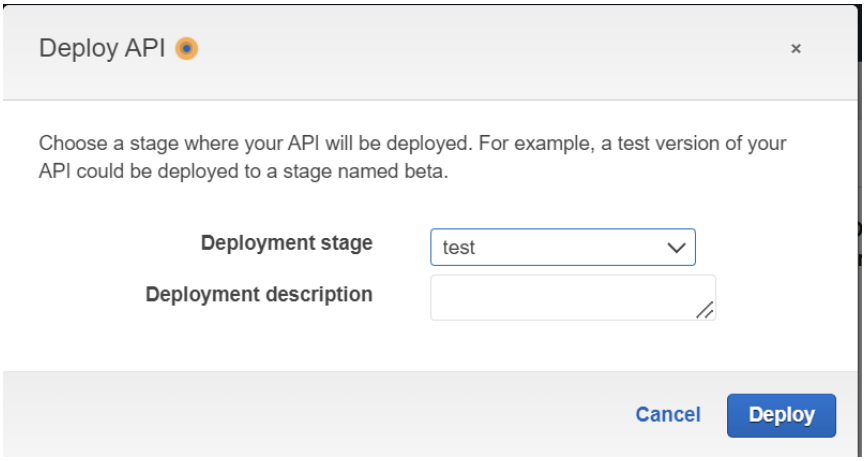
Step 14:

Under Actions, click on Deploy API.

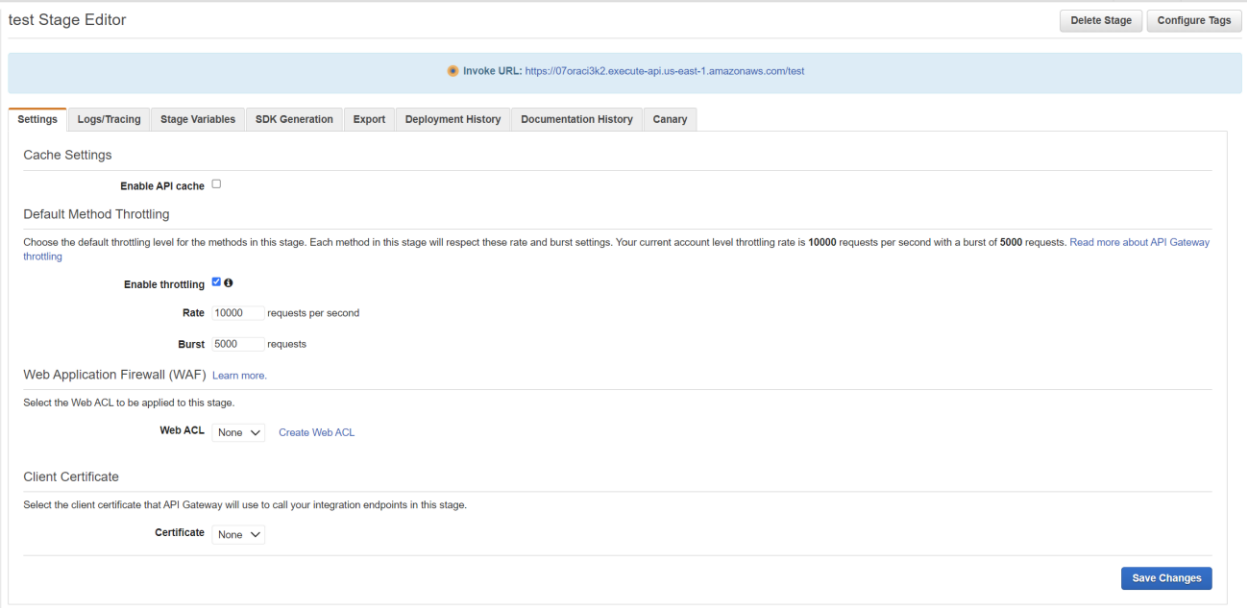


Step 15:

Under Deployment stage select “test” and click on Deploy.



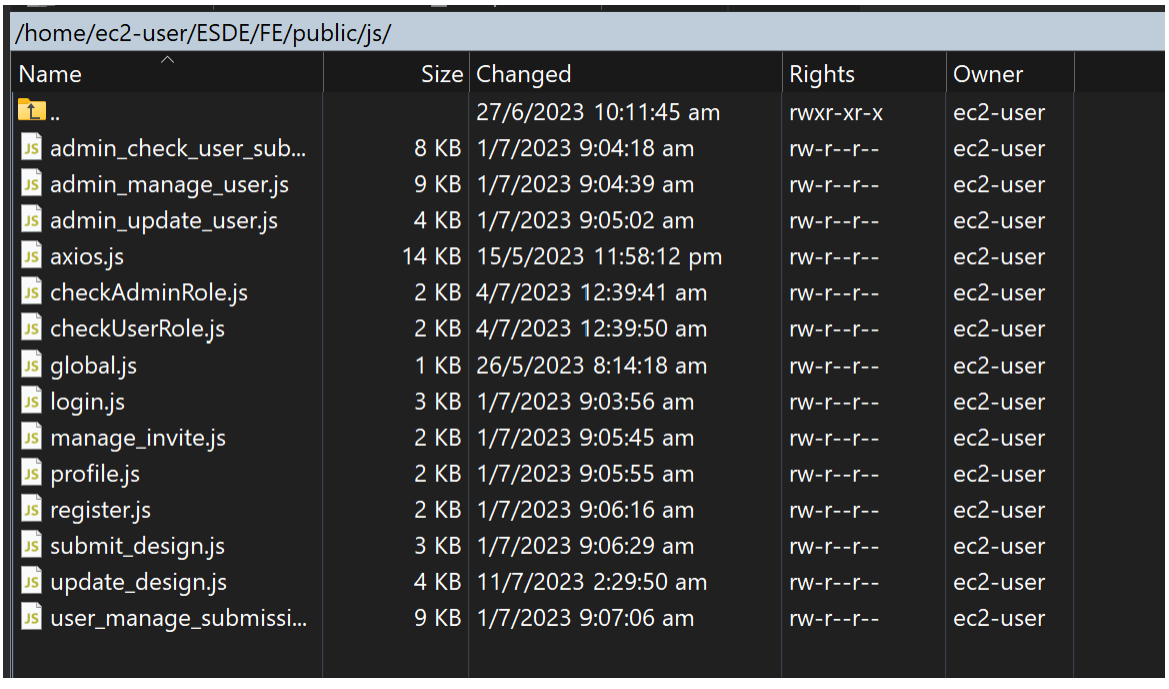
You should be able to see the following.



7.3 Adding the updateDesign URL to WinSCP

Step 1:

Go into WinSCP and go to the directory "/home/ec2-user/ESDE/FE/public/js".



Step 2:

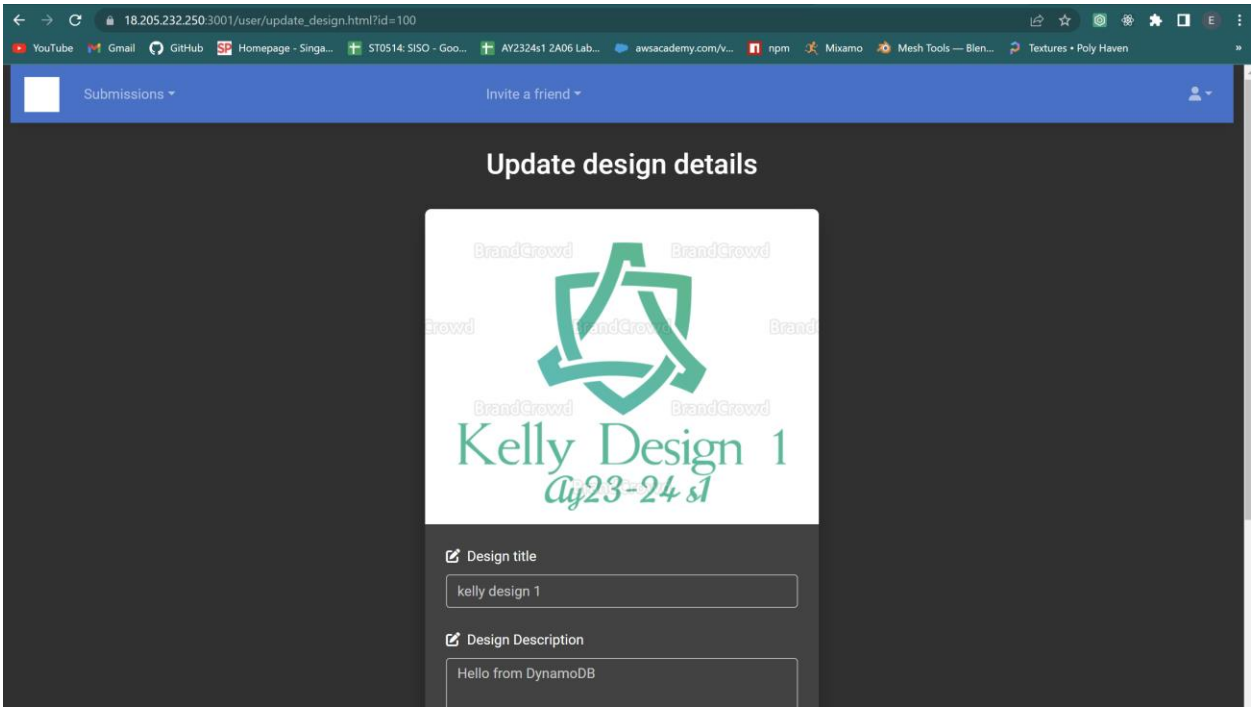
Click on update_design.js file and paste the following into the URL and save the file.

'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test?fid='+fileId+'&designTitle='+designTitle+'&designDescription='+designDescription

```
//making a PUT HTTP request.
let webFormData = new FormData();
webFormData.append("designTitle", designTitle);
webFormData.append("designDescription", designDescription);
webFormData.append("fileId", fileId);
axios({
  method: "put",
  url: 'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test?fid='+fileId+'&designTitle='+designTitle+'&designDescription='+designDescription,
  data: webFormData,
  headers: {
    "Content-Type": "multipart/form-data",
```

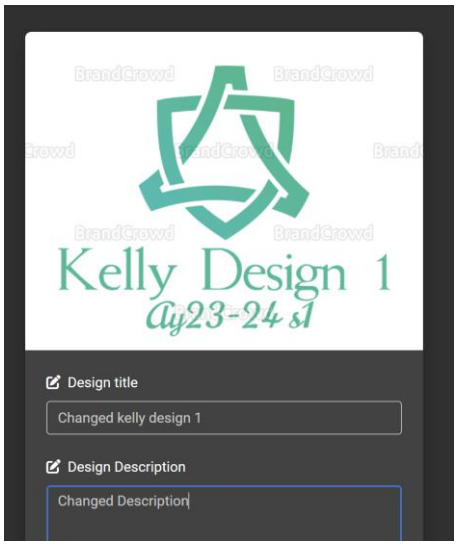
Step 3:

Run the frontend and backend server and go to Kelly's Account and go to her designs page and click on Kelly Design 1.



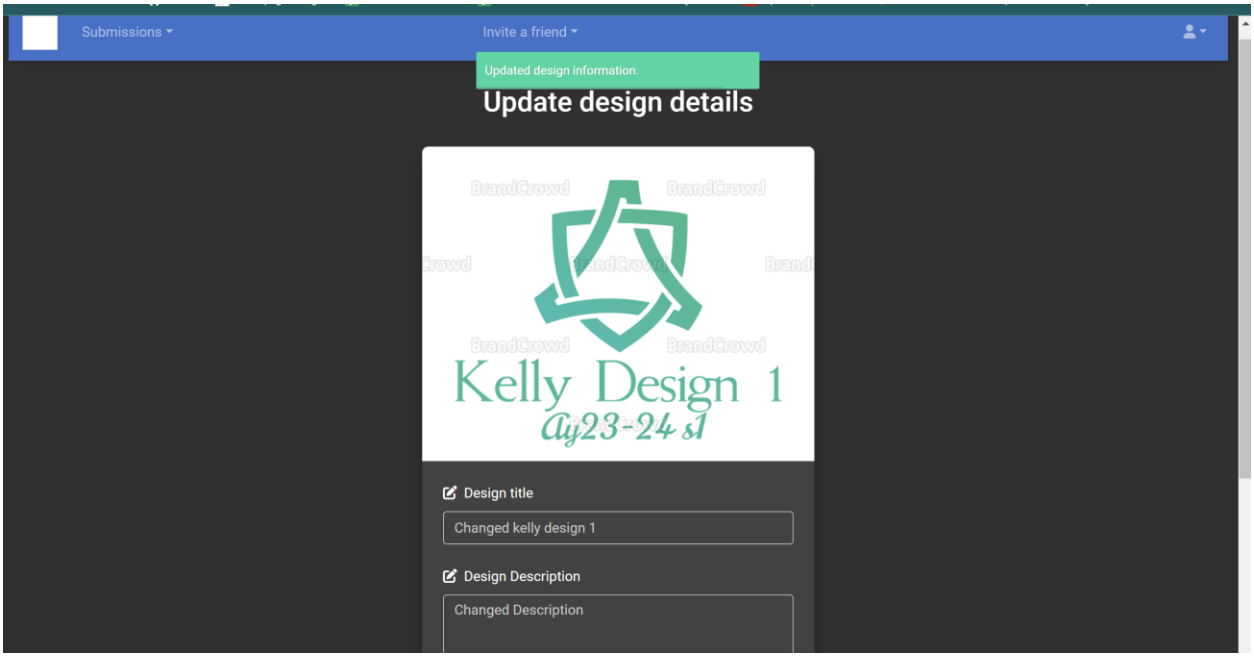
Step 4:

Change the Design Title to “Changed Kelly design 1” and Design Description to “Changed Description” and click on the submit button.



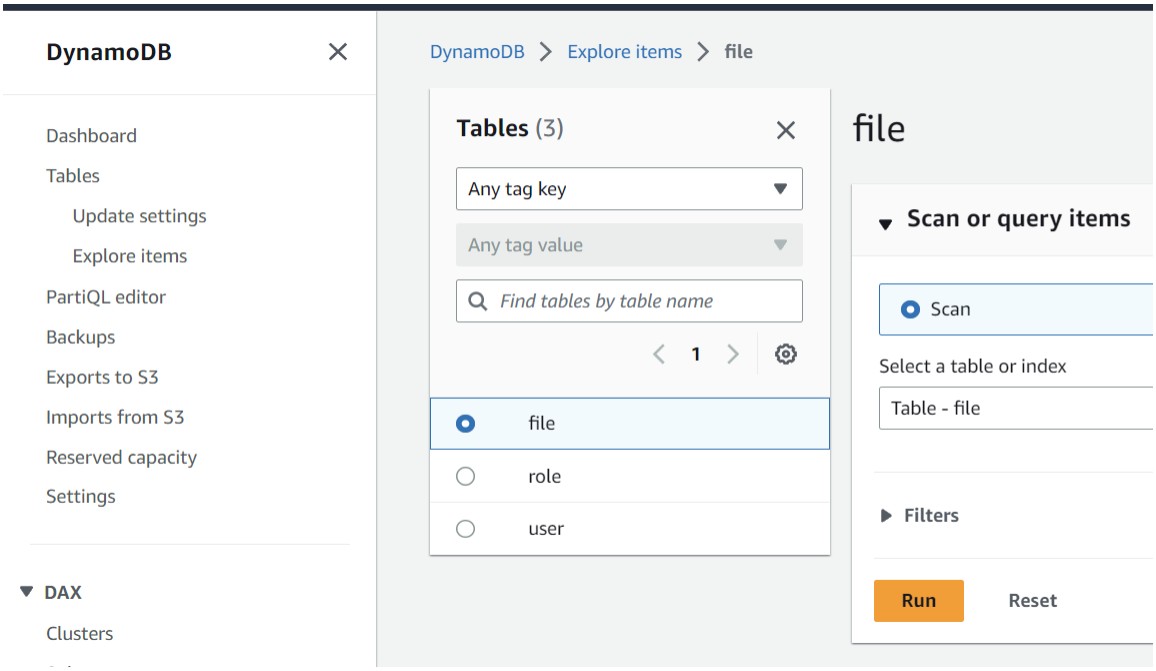
Step 5:

You should be able to see the following Successful Update Message.



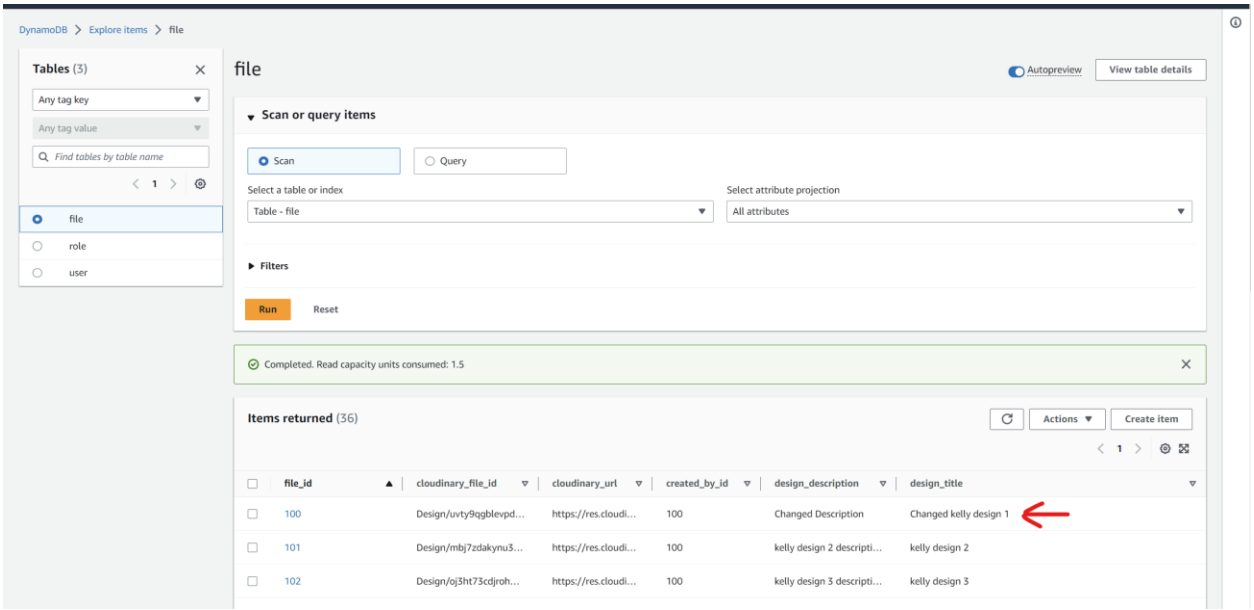
Step 6:

Go to AWS dynamoDB and click on file table as shown below.



Step 7:

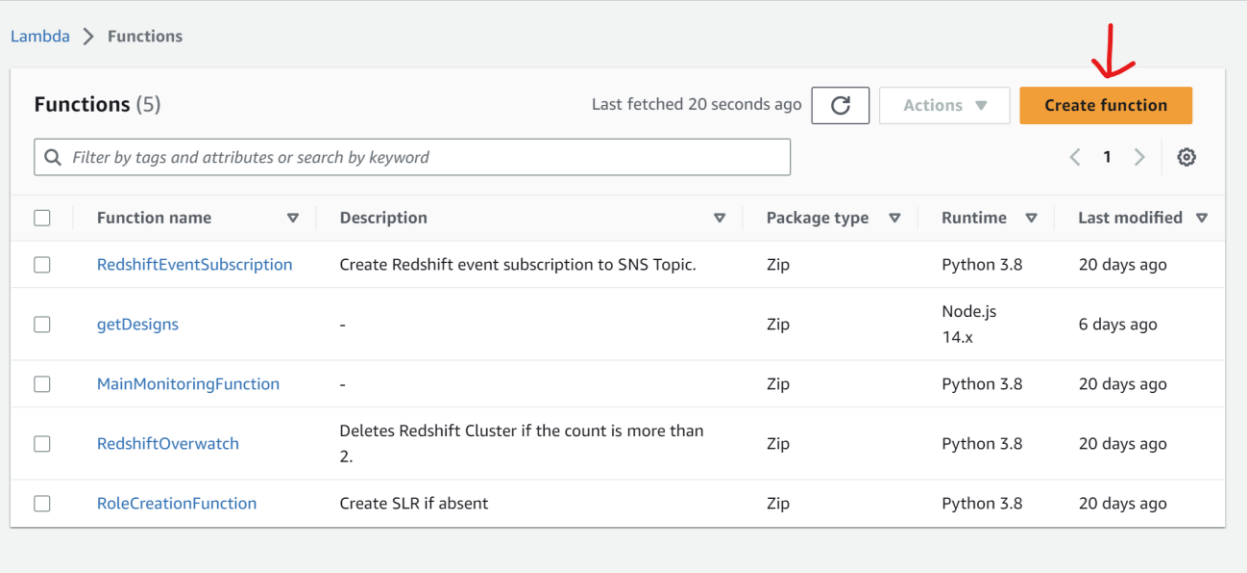
You should be able to see the changes reflected in the dynamoDB file table.



7.4 Creating manageSubmissions Lambda Function and Index Name for File Table

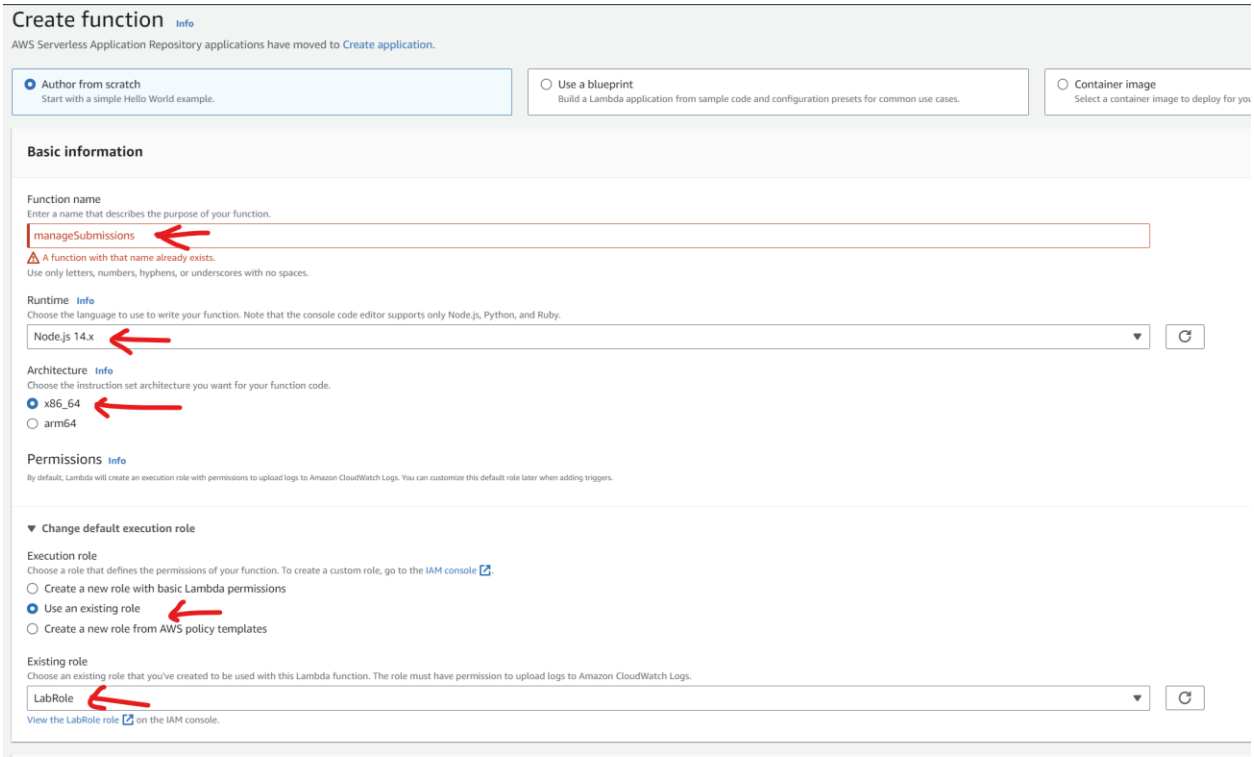
Step 1:

Go into AWS Lambda and click on “Create function” to create a new Lambda function.



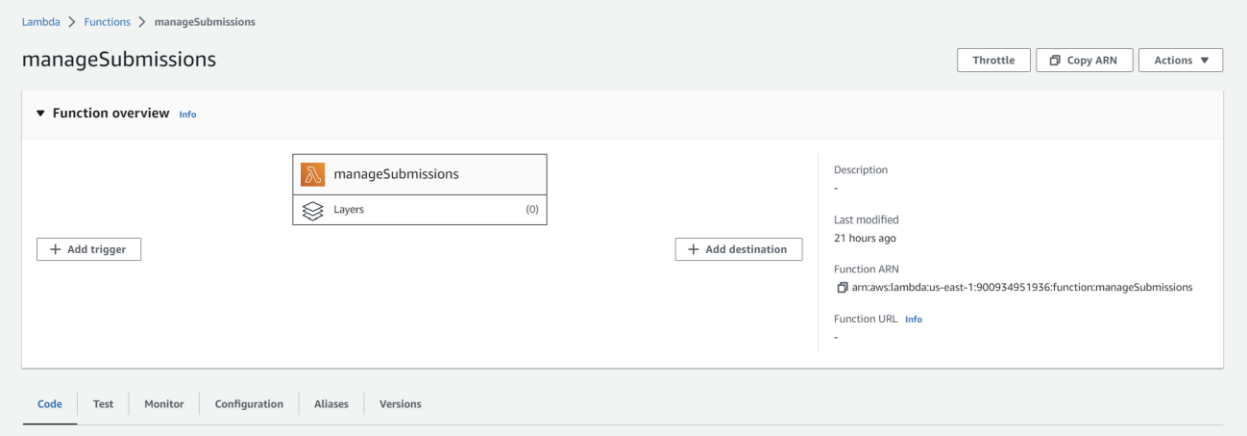
Step 2:

Select “Author from scratch”, and type “manageSubmissions” as the function name, and for the runtime make sure that you select “Node.js 14.x” and for default execution role select “Use an existing role” and select LabRole and click on the “Create function” button as shown below.



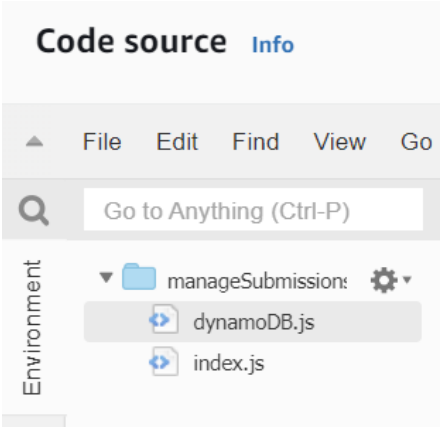
Step 3:

You should be able to see the following below.



Step 4:

Scroll down and make sure that you have a dynamoDB.js and index.js file created as shown below.



Step 5:

Add in the following codes to your index.js and dynamoDB.js files and click on the deploy button to save the codes.

dynamoDB.js

```
var AWS = require('aws-sdk');

async function queryitems_dynamodb(region, table_name, expr_attr_values,
key_cond_expr, proj_expr, creator, pageNumber, searchInput) {
   console.log("In the queryitems_dynamodb method...");
   var dynamodb = new AWS.DynamoDB.DocumentClient({ region: region });
   var index = 0;
   var count = 0;

   async function search(err, data) {
      console.log(data);
      if (err) {
         console.log("Search Error");
      } else {
         console.log("Successfully Searched");
         console.log(params);
         data.Items.forEach(function (itemData, index) {
```

```
            console.log("Item :", ++count, JSON.stringify(itemData));
        });

        if (data.LastEvaluatedKey) {
            console.log("Scanning for more...");
            params.ExclusiveStartKey = data.LastEvaluatedKey;
            return dynamodb.query(params, search);
        }
    }
}

try {
    var params = {
        TableName: "file",
        IndexName: "created_by_id-file_id-index",
        KeyConditionExpression: "#createdBy = :creator",
        ExpressionAttributeNames: {
            "#createdBy": "created_by_id"
        },
        ExpressionAttributeValues: {
            ":creator": creator,
        },
        ScanIndexForward: true,
    };

    // Conditionally add the FilterExpression if searchInput is not empty
    if (searchInput && searchInput.trim().length > 0) {
        params.FilterExpression = "contains(design_title, :searchInput)";
        params.ExpressionAttributeValues[":searchInput"] = searchInput;
    }

    const results = await dynamodb.query(params, search).promise();
    return results;
} catch (tryerror) {
    console.log("Error occurred in dynamodb.query..");
    console.log(tryerror, tryerror.stack); // an error occurred
}
```

```
} //end function

module.exports = queryitems_dynamodb;
```

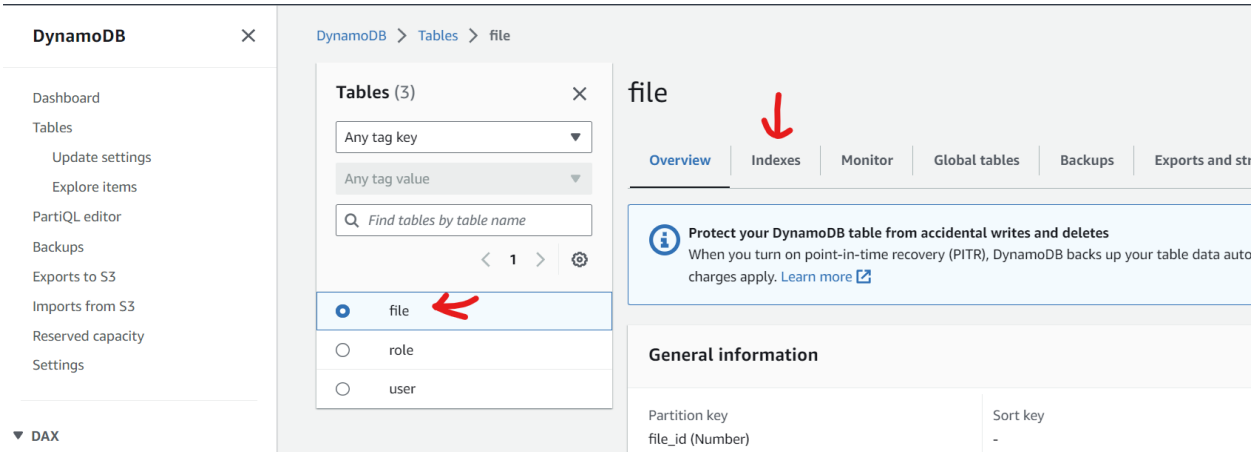
index.js

```
var dynamodbQuery = require('dynamoDB');

exports.handler = async function(event, context, callback){
   if (event.userId || event.queryStringParameters.userId) {
      if (event.userId)
         var creator = parseInt(event.userId)
      else
         var creator = parseInt(event.queryStringParameters.userId)

      var region = "us-east-1";
      var table_name = "file";
      var expr_attr_values = { ":creator": creator };
      var key_cond_expr = "created_by_id=:creator";
      var proj_expr =
"file_id,cloudinary_url,design_title,design_description,created_by_id";
      var pageNum = parseInt(event.pageNumber);

      var searchInput; // Initialize searchInput with an empty string
      if (event.queryStringParameters &&
event.queryStringParameters.searchInput) {
         searchInput = event.queryStringParameters.searchInput || null;
      }
      else {
         searchInput = event.searchInput || null;
      }

      await dynamodbQuery(region,
table_name,expr_attr_values,key_cond_expr,proj_expr,creator,pageNum,
searchInput)
      .then(data => {
            console.log("Successfully got items from dynamodb.query");
```

```
            console.log(data)
            var responseCode = 200;
            var totalNumberOfRecords = data.Count;
            var jsonResult = {
               'filedata': data.Items,
               'total_number_of_records': totalNumberOfRecords
            };
            let response = {
               statusCode: responseCode,
               body: JSON.stringify(jsonResult),
               headers: {
                  "Access-Control-Allow-Headers" : "Content-Type,user",
                  "Access-Control-Allow-Origin": "*",
                  "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
               }
            };
            console.log("response: " + JSON.stringify(response));
            callback(null, response);
      })
      .catch(error => {
         console.log('There has been a problem with your fetch operation: ' +
error.message);
         var responseCode = 500;

         let response = {
            statusCode: responseCode,
            body: JSON.stringify(error)
         }

         console.log("response: " + JSON.stringify(response))
         callback(null, response);
       });
   } //end if
}
```

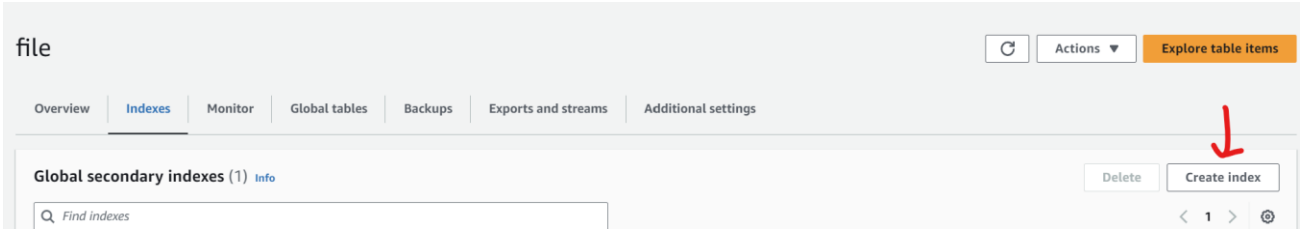
Step 6:

Next, go to dynamoDB and look for the file table and click on the Indexes Tab.



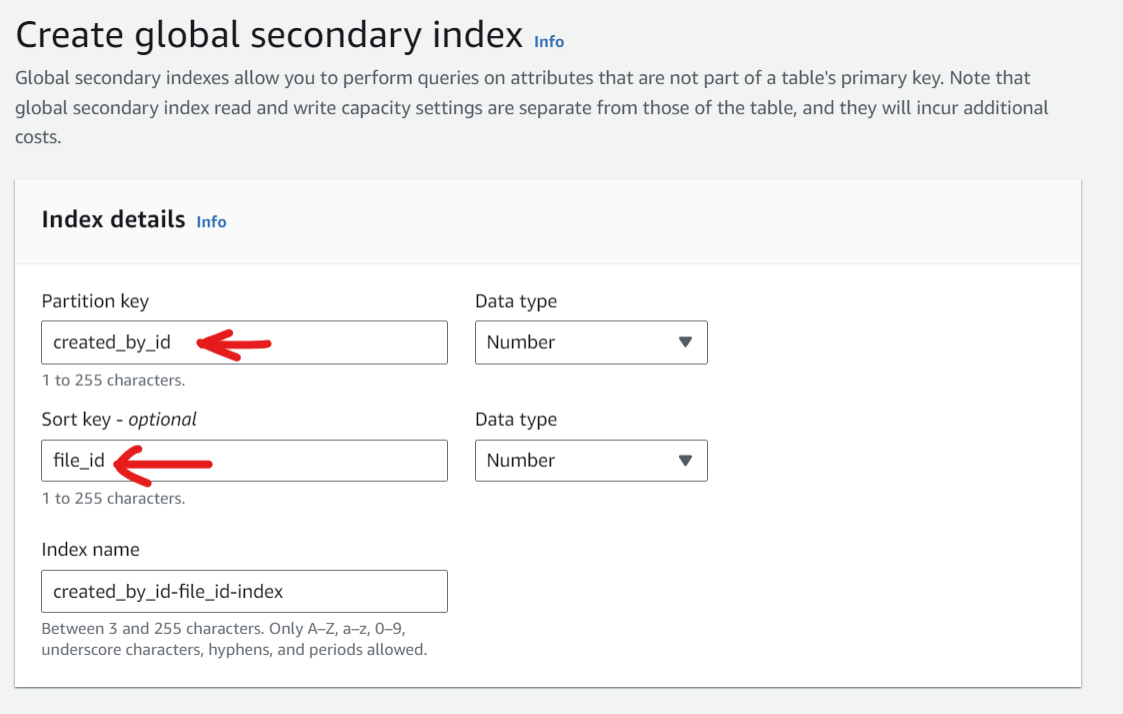
Step 7:

Click on Create Index to create a new index for us to use in the index.js.



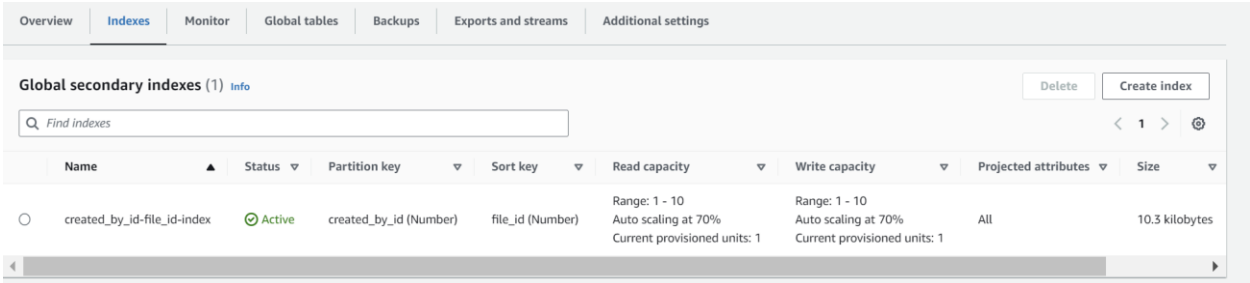
Step 8:

Input “created_by_id” as the Partition Key and “file_id” as the Sort Key. Ignore the Index Name and leave it as it is since it is auto generated and click on “Create Index” at the bottom.



Step 9:

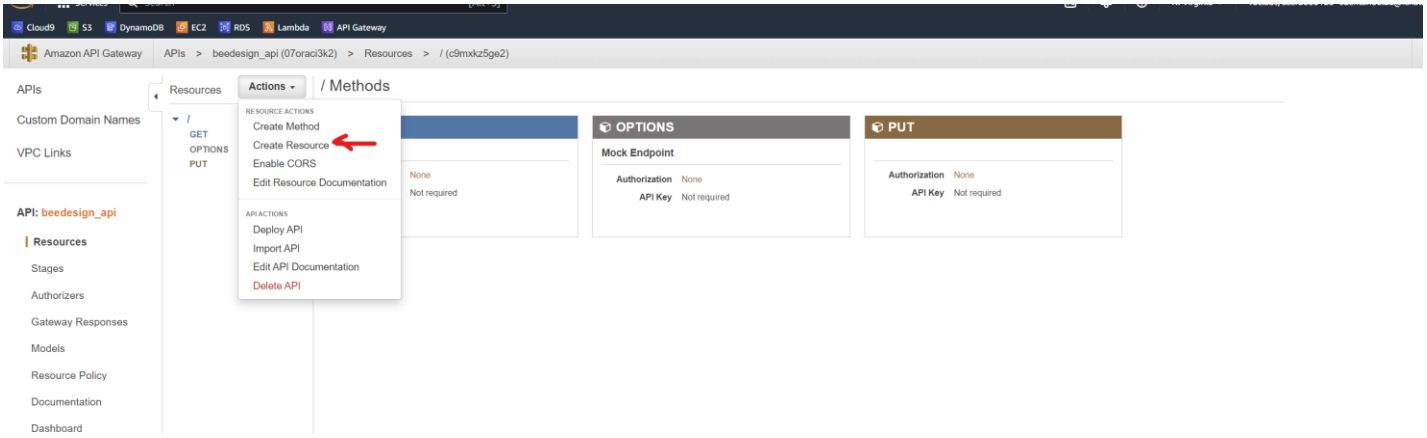
Upon successful creation, you should be able to see the successfully created index name as shown below.



7.5 Creating manageSubmissions API Gateway

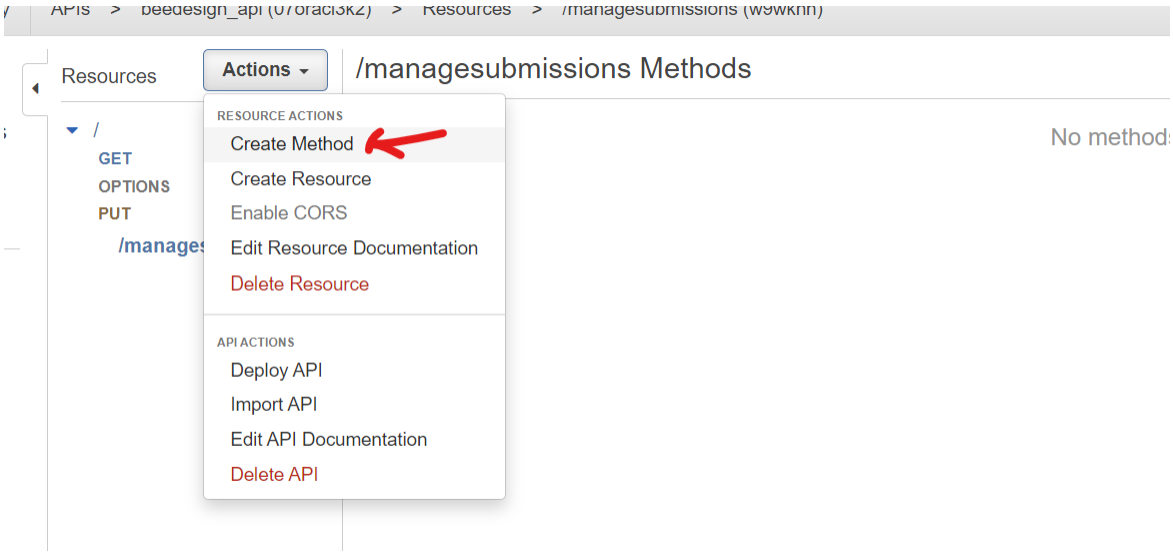
Step 1:

Go to AWS API Gateway and click on Actions and click on Create Resource and name the resource as manageSubmissions.



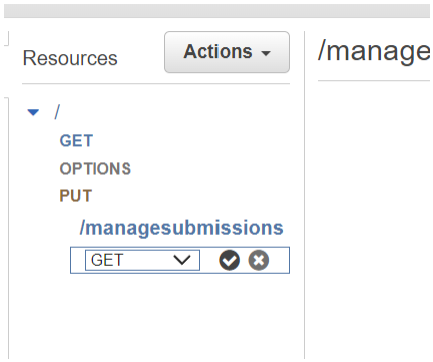
Step 2:

Click on the newly created resource and click on Actions followed by Create Method.



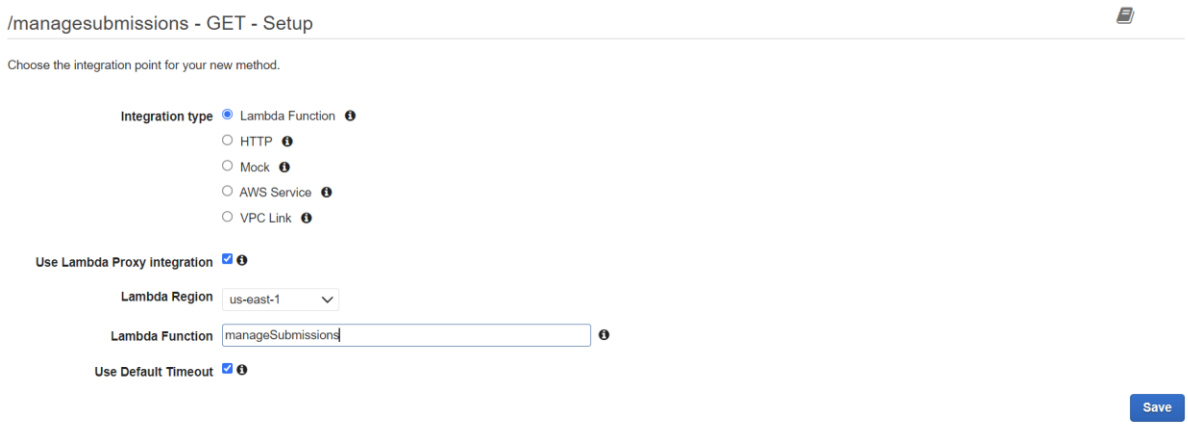
Step 3:

Next select the GET Method and click on the tick icon to confirm.



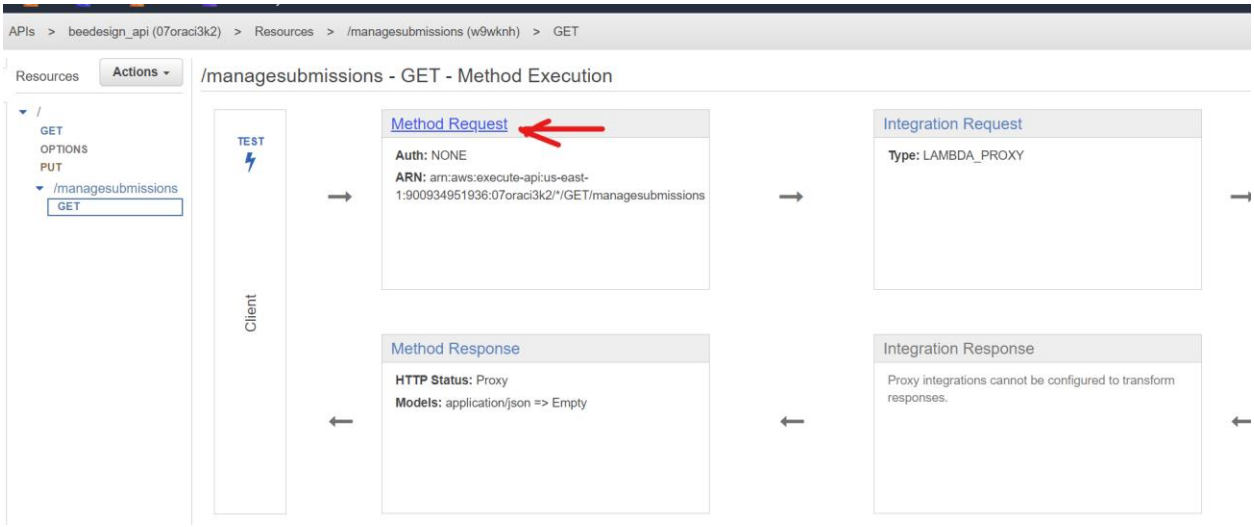
Step 4:

A Setup pane will appear, make sure that you follow the following setup as shown below where Lambda Function is manageSubmissions, click on save and ok.



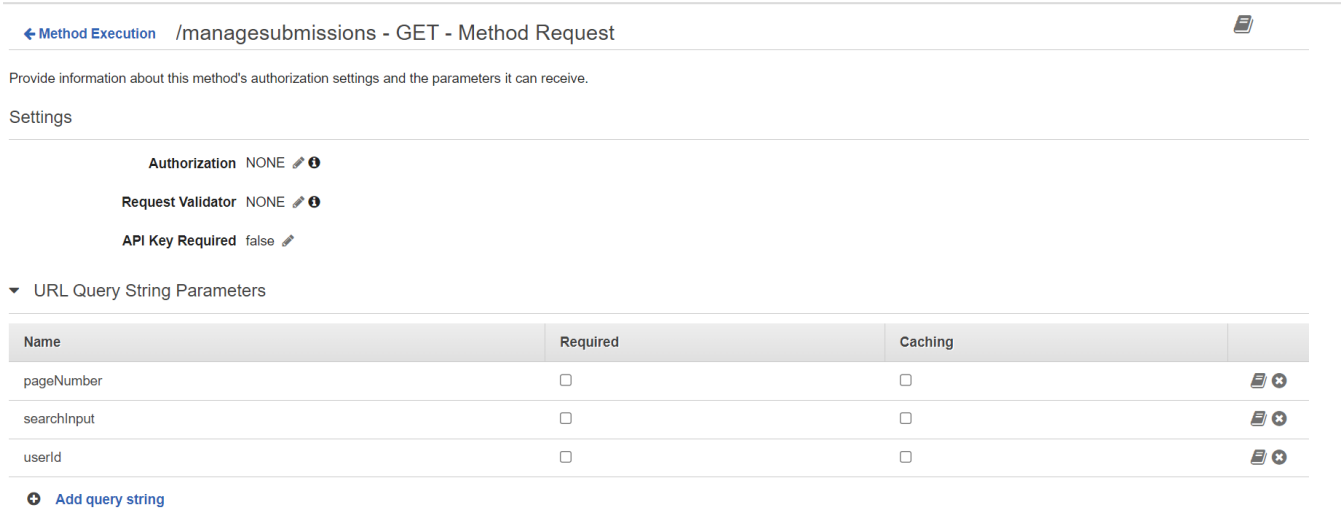
Step 5:

Click on Method Request.



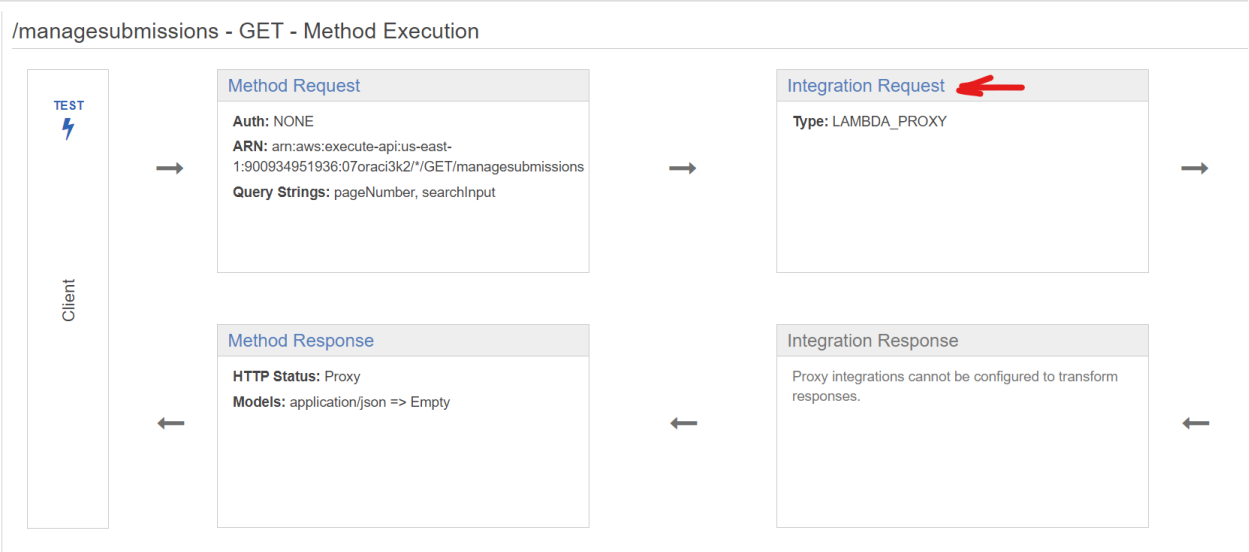
Step 6:

Under URL Query String Parameters make sure that you have userId, pageNumber and searchInput being added and for request headers make sure you have user added.



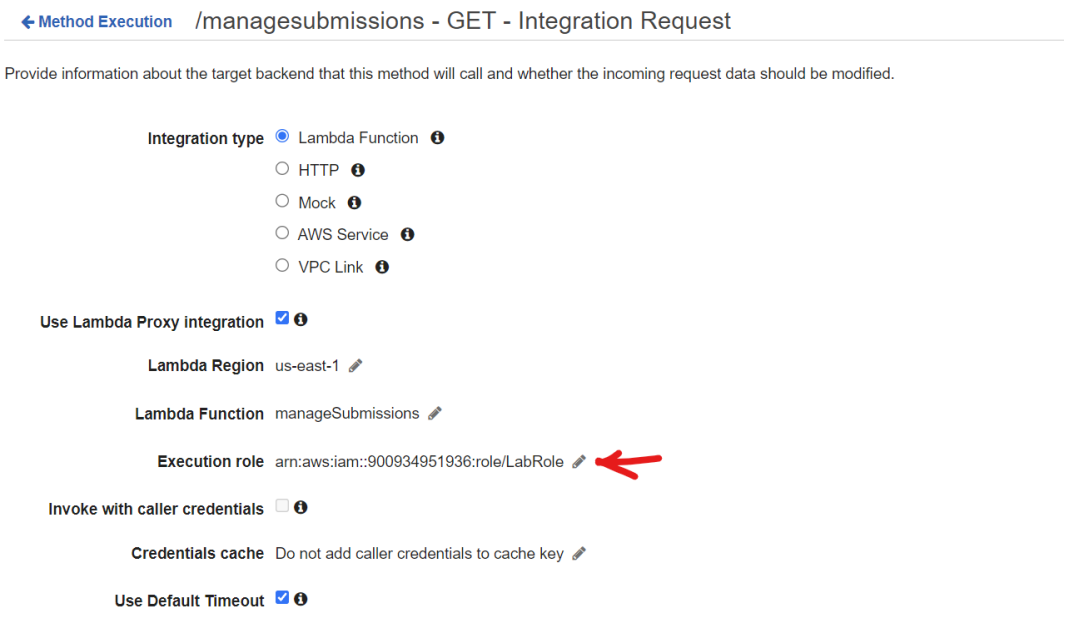
Step 7:

Go back and click on Integration request.



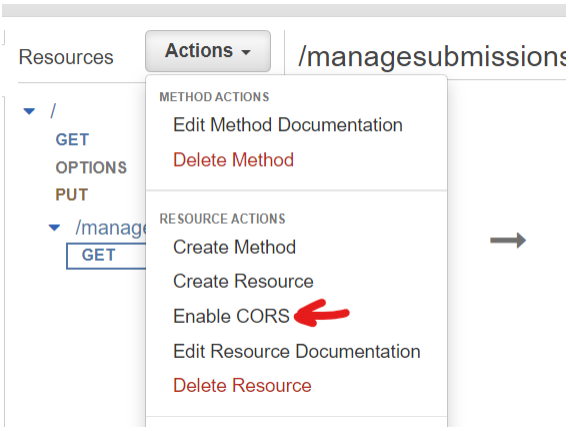
Step 8:

We will use the same execution role that we used from 7.2 Step 9.



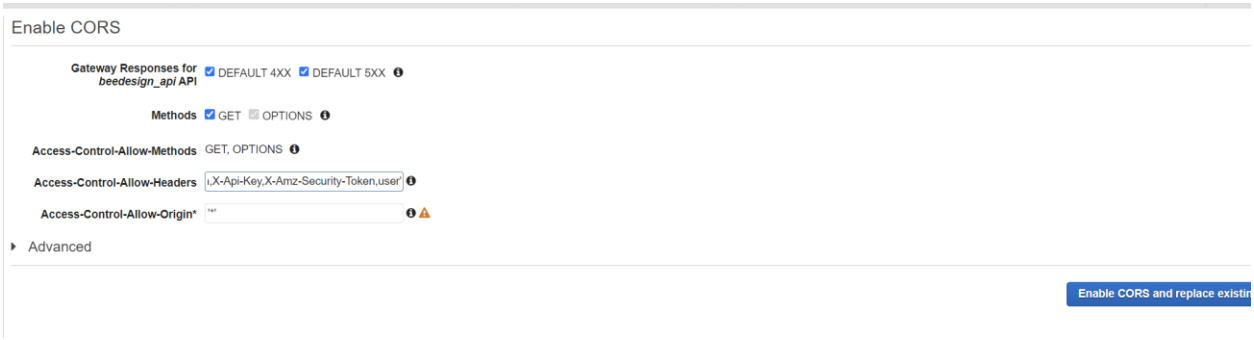
Step 9:

Go back and click on Actions and click on Enable CORS.

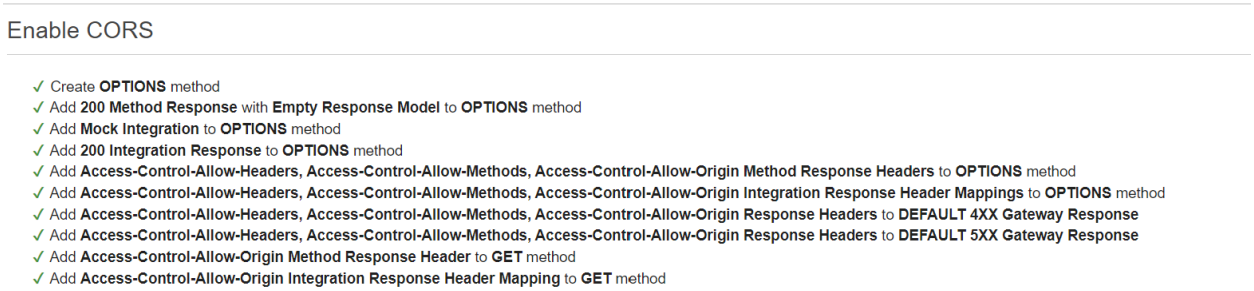


Step 10:

Make sure that you have the following configurations as shown below, where at the end of Headers add a ",user" and check the necessary things. Once done, click on Enable CORS.

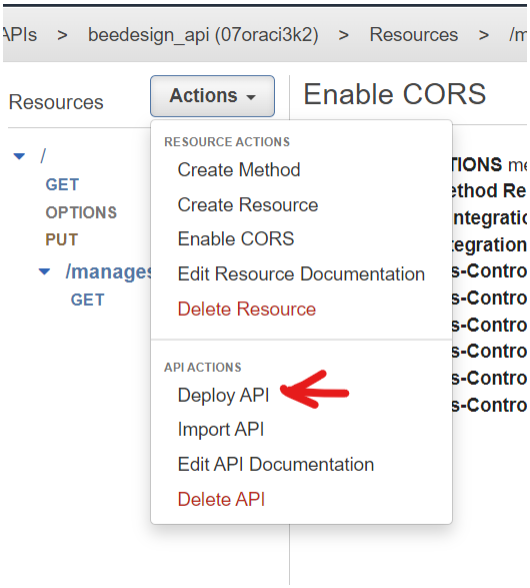


You should see the following.

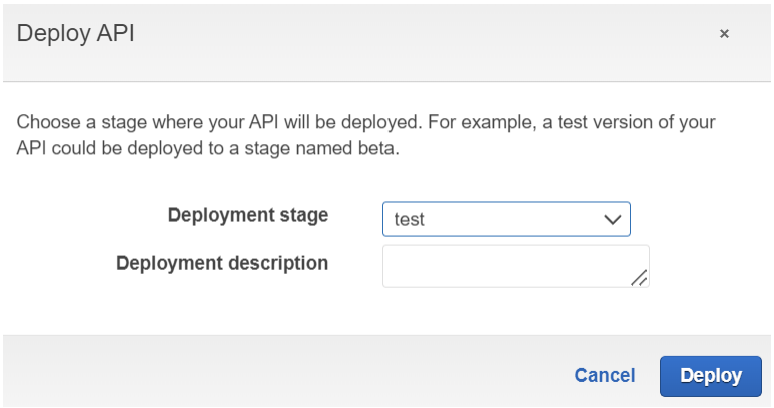


Step 11:

Click on Actions and click on Deploy API.



Make sure to use test for deployment stage and click on Deploy.



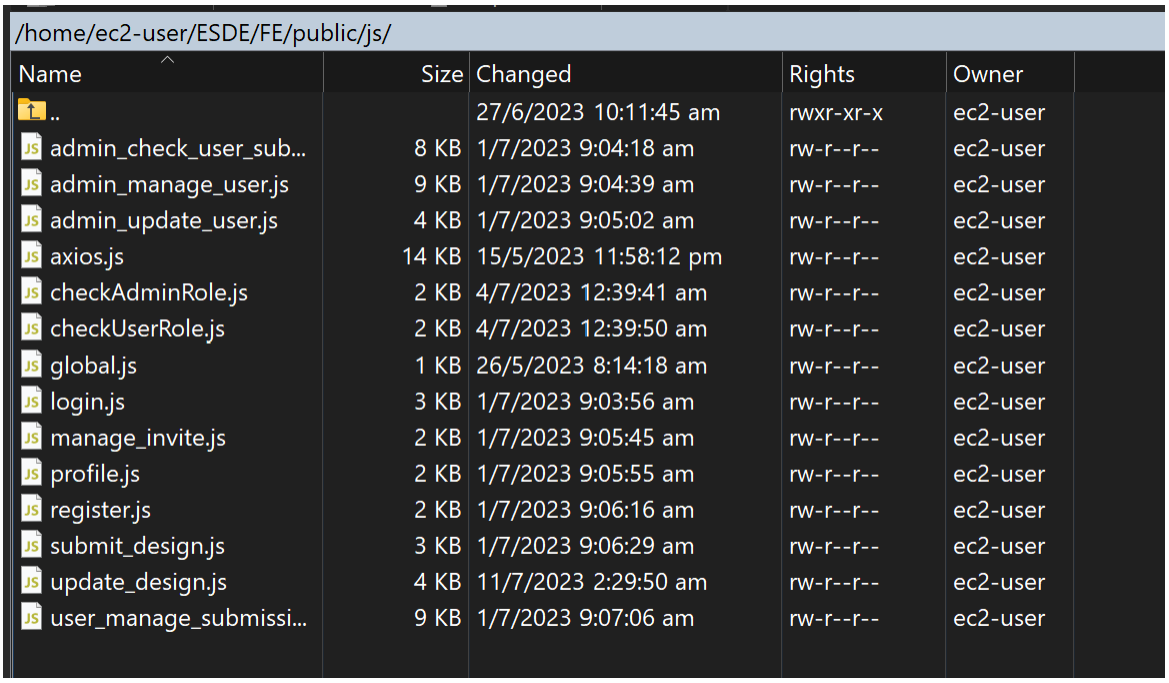
Step 12:

You should be able to see the following. Showing that you have successfully created the API Gateway.

7.6 Adding manageSubmissions API to WinSCP

Step 1:

Go into WinSCP and go to the directory "/home/ec2-user/ESDE/FE/public/js".



Name	Size	Changed	Rights	Owner
..		27/6/2023 10:11:45 am	rwxr-xr-x	ec2-user
admin_check_user_sub...	8 KB	1/7/2023 9:04:18 am	rw-r--r--	ec2-user
admin_manage_user.js	9 KB	1/7/2023 9:04:39 am	rw-r--r--	ec2-user
admin_update_user.js	4 KB	1/7/2023 9:05:02 am	rw-r--r--	ec2-user
axios.js	14 KB	15/5/2023 11:58:12 pm	rw-r--r--	ec2-user
checkAdminRole.js	2 KB	4/7/2023 12:39:41 am	rw-r--r--	ec2-user
checkUserRole.js	2 KB	4/7/2023 12:39:50 am	rw-r--r--	ec2-user
global.js	1 KB	26/5/2023 8:14:18 am	rw-r--r--	ec2-user
login.js	3 KB	1/7/2023 9:03:56 am	rw-r--r--	ec2-user
manage_invite.js	2 KB	1/7/2023 9:05:45 am	rw-r--r--	ec2-user
profile.js	2 KB	1/7/2023 9:05:55 am	rw-r--r--	ec2-user
register.js	2 KB	1/7/2023 9:06:16 am	rw-r--r--	ec2-user
submit_design.js	3 KB	1/7/2023 9:06:29 am	rw-r--r--	ec2-user
update_design.js	4 KB	11/7/2023 2:29:50 am	rw-r--r--	ec2-user
user_manage_submissi...	9 KB	1/7/2023 9:07:06 am	rw-r--r--	ec2-user

Step 2:

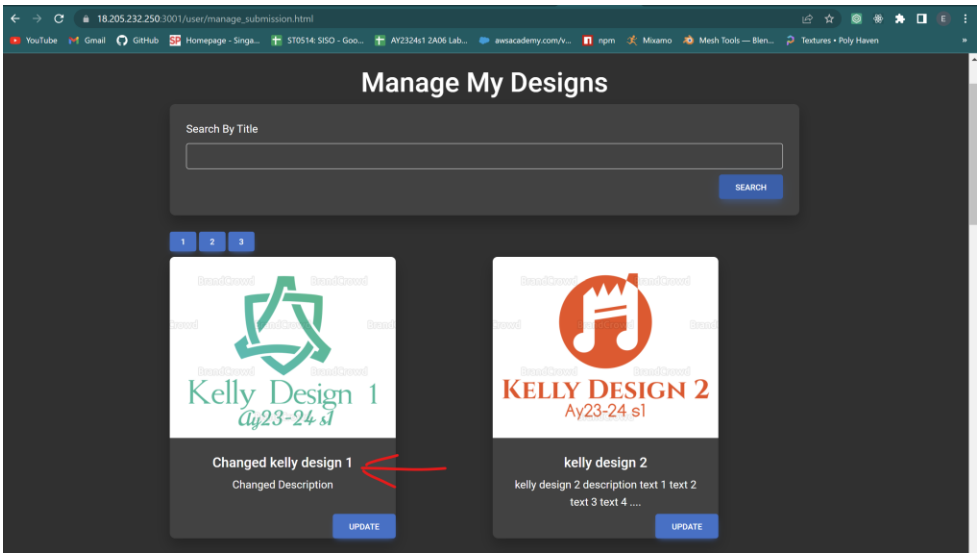
Click on user_manage_submission.js and add in the following url for the first endpoint and save it.

"'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test/managesubmissions?pageNumber=' + pageNumber + '&searchInput=' + searchInput"

```
let $searchDesignFormContainer = $("#searchDesignFormContainer");
if ($searchDesignFormContainer.length != 0) {
  console.log(
    "Search design form detected in user manage submission interface. Binding event handling logic to form elements."
  );
  //If the jQuery object which represents the form element exists,
  //the following code will create a method to send key-value pair information to do record searching
  //to server-side api when the #submitButton element fires the click event.
  $("#submitButton").on("click", function (event) {
    event.preventDefault();
    const baseUrl = "https://18.205.232.250:5000";
    let searchInput = $("#searchInput").val();
    let userId = sessionStorage.getItem("user_id");
    let token = sessionStorage.getItem("token");
    axios({
      headers: {
        user: userId,
        authorization: `Bearer ${token}`,
      },
      method: "get",
      url: 'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test/managesubmissions?userId='+userId+'&pageNumber=1&searchInput='+searchInput,
    })
      .then(function (response) {
        //console.log(response)
        //Using the following to inspect the response.data data structure
```

Step 3:

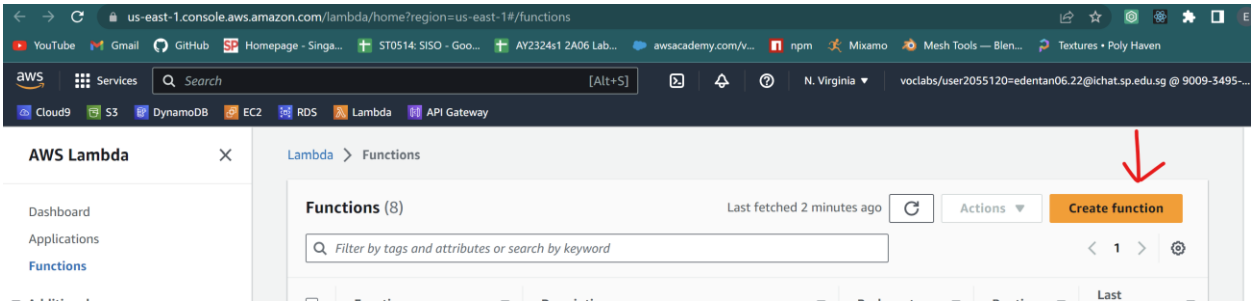
Now, run the frontend and backend server and see if you can view the updated Kelly Design 1 without going to the update page.



7.8 Creating getProfile Lambda Function

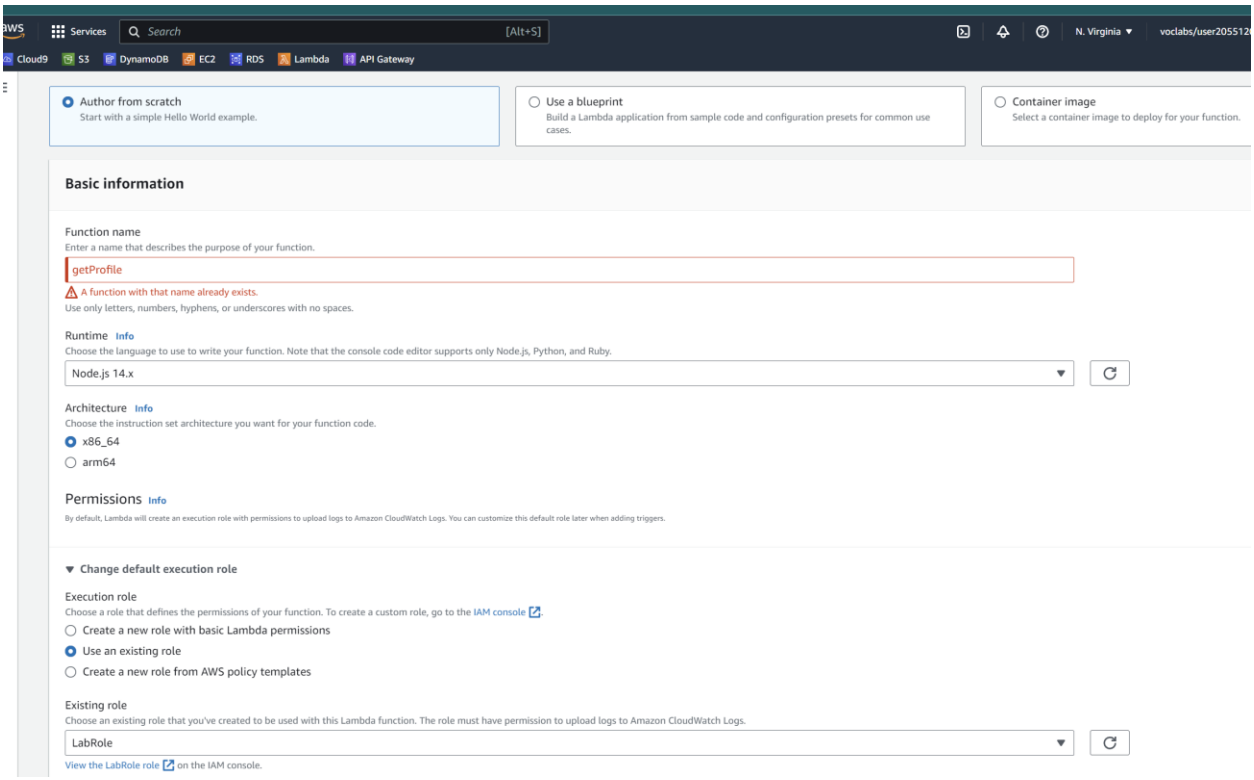
Step 1:

Go to AWS Lambda and create a new function by clicking on the “Create function” button.



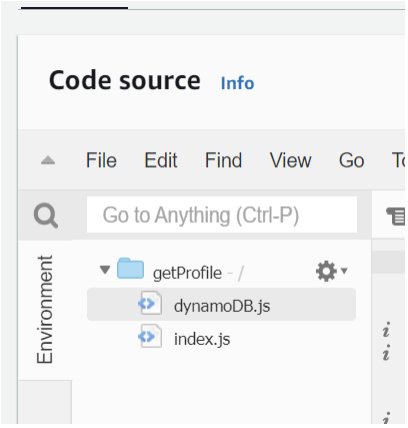
Step 2:

Fill in the function name as getProfile and choose Node.js 14.x for the Runtime and choose LabRole for the Role and click on “Create function” below.



Step 3:

Next, go into the getProfile Lambda function and go to the Code source at the bottom and make sure that you have a index.js and dynamoDB.js file created as shown below.



Step 4:

Input the following codes into eh index.js and dynamoDB.js files accordingly.

index.js

```
var dynamodbQuery = require('dynamoDB');

exports.handler = async (event, context, callback) => {
   if (event.headers['user']){
      console.log(event)
      var userId = parseInt(event.headers['user']);
      var region = "us-east-1";
      var table_name = "user";
      var proj_expr = "user_id,email,role_id,fullname"
      var expr_attr_values = { ":userid": userId }
      var key_cond_expr = "user_id=:userid"
      await dynamodbQuery(region,
table_name,expr_attr_values,key_cond_expr,proj_expr)
      .then(data => {
         console.log(data)
         console.log("Successfully got items from dynamodb.query")
         var responseCode = 200;
```

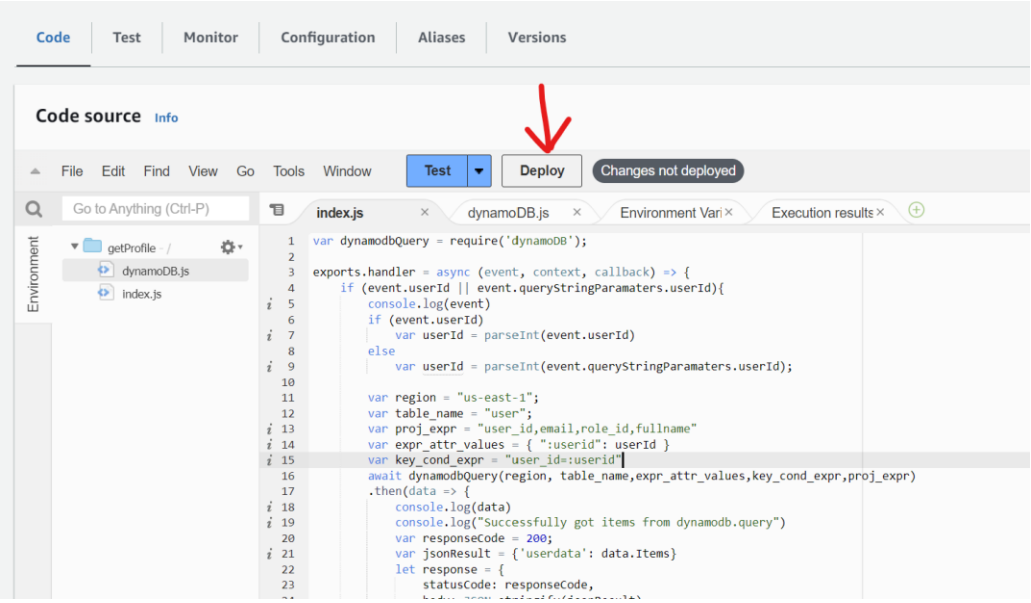
```
        var jsonResult = {'userdata': data.Items}
        let response = {
          statusCode: responseCode,
          body: JSON.stringify(jsonResult),
          headers: {
            "Access-Control-Allow-Headers" : "Content-Type,user",
            "Access-Control-Allow-Origin": "*",
            "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
          }
        }
        console.log("response: " + JSON.stringify(response))
        callback(null, response);
      })
      .catch(error => {
        console.log('There has been a problem with your fetch operation: ' +
error.message);
        var responseCode = 500;

        let response = {
          statusCode: responseCode,
          body: JSON.stringify(error)
        }

        console.log("response: " + JSON.stringify(response))
        callback(null, response);
      });
  }
};
```

dynamoDB.js

```
var AWS = require('aws-sdk');

async function queryitems_dynamodb(region,
table_name,expr_attr_values,key_cond_expr,proj_expr) {
   console.log("In the queryitems_dynamodb method...")
   var dynamodb = new AWS.DynamoDB.DocumentClient({region: region});

   try{
      var params = {
         TableName: table_name,
         ExpressionAttributeValues: expr_attr_values,
         KeyConditionExpression: key_cond_expr ,
         ProjectionExpression: proj_expr
      }
      const results = await dynamodb.query(params).promise()
      console.log("Printing results from  queryitems_dynamodb " + results)
      return results;
   }
   catch(tryerror) {
      console.log("Error occurred in dynamodb.query..")
      console.log(tryerror, tryerror.stack); // an error occurred
   }
} //end function

module.exports = queryitems_dynamodb
```

Step 5:

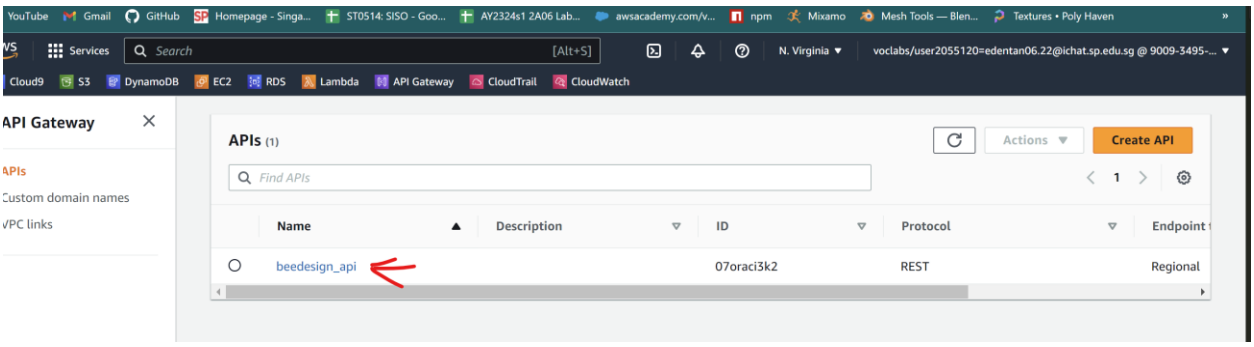
Click on the deploy button as shown below to save the codes that you have added.



7.9 Creating the getProfile API Gateway

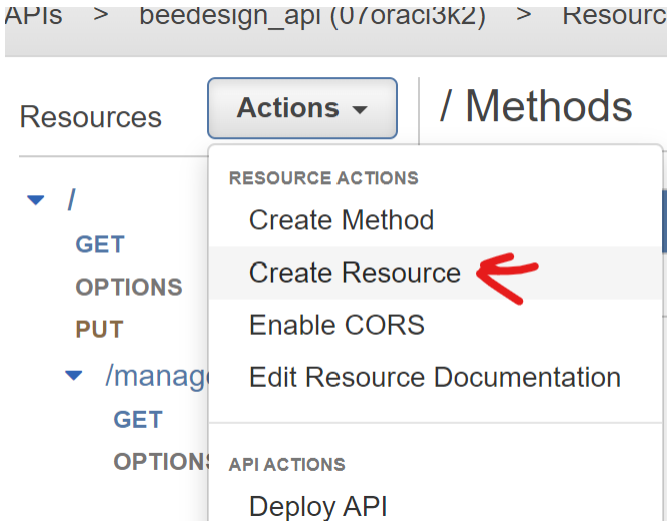
Step 1:

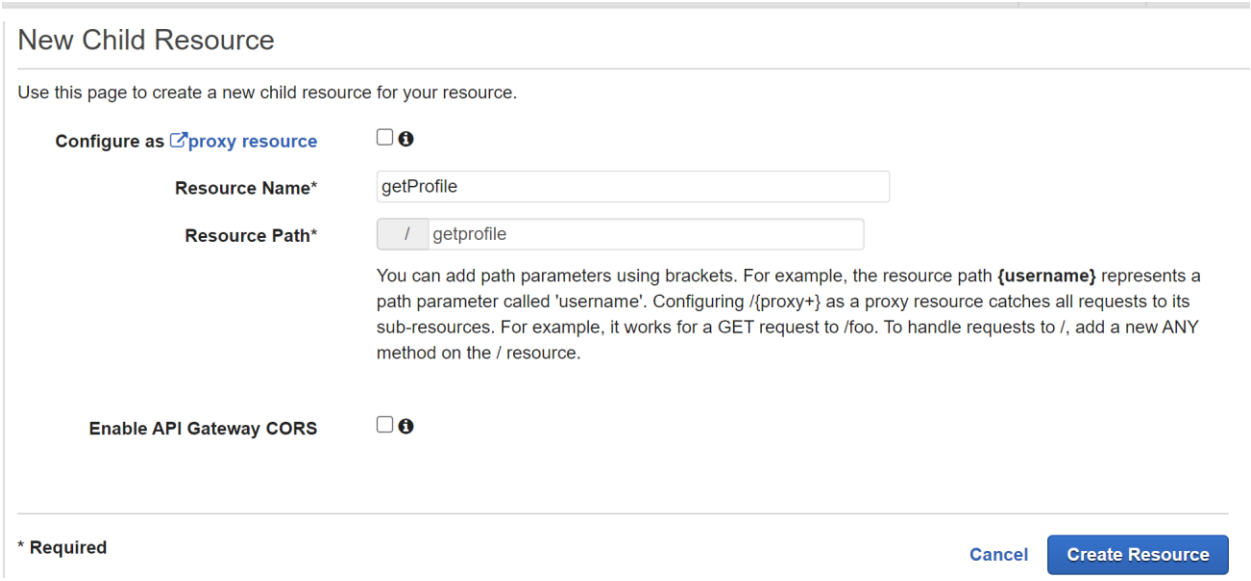
Go into AWS API Gateway and click on the beedesign_api.



Step 2:

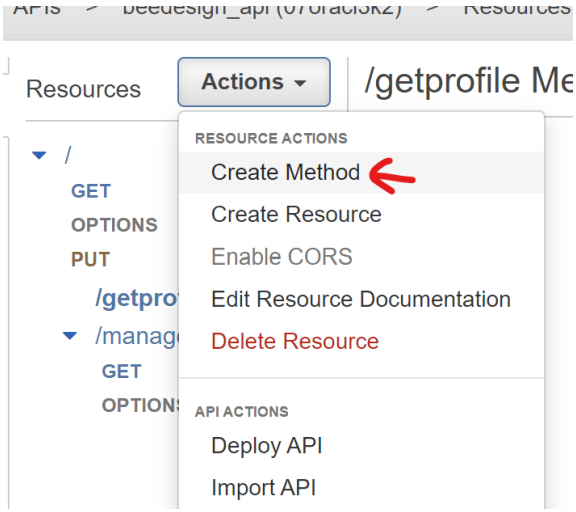
Under Actions, click on Create Resource and name it as getProfile as shown below and click on create resource.





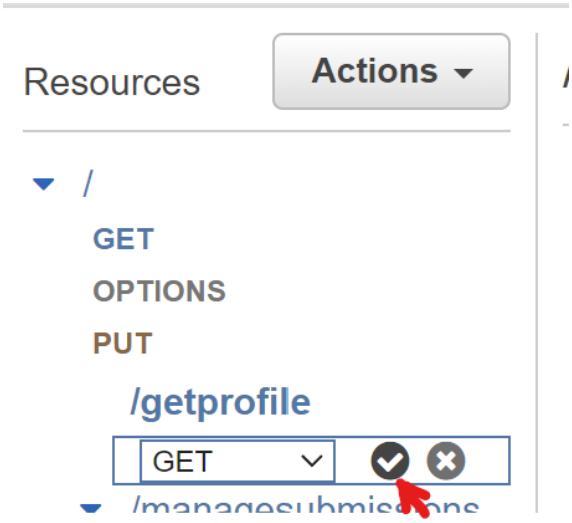
Step 3:

Next, click on Actions and select Created Method.



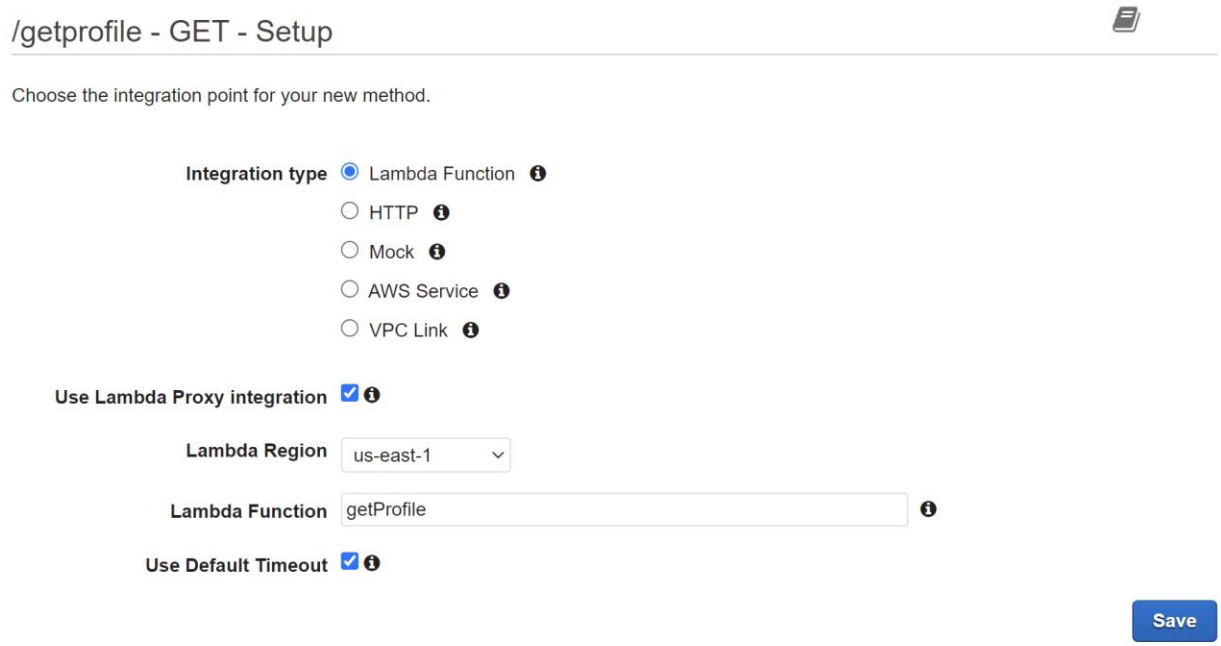
Step 4:

Select the GET request and click on the tick icon.

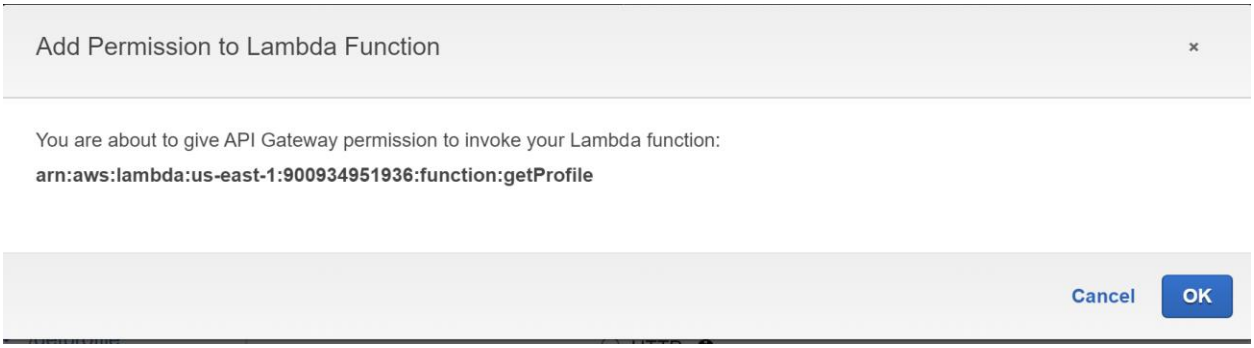


Step 5:

For the setup, follow the set up as shown below and click on the Save button.

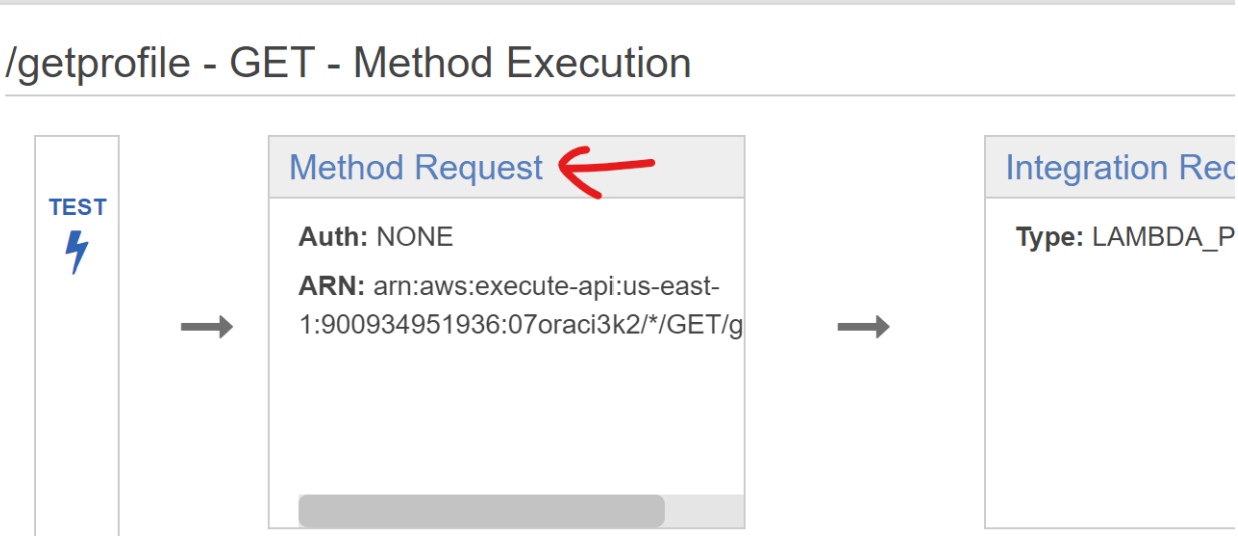


Click on the OK button.



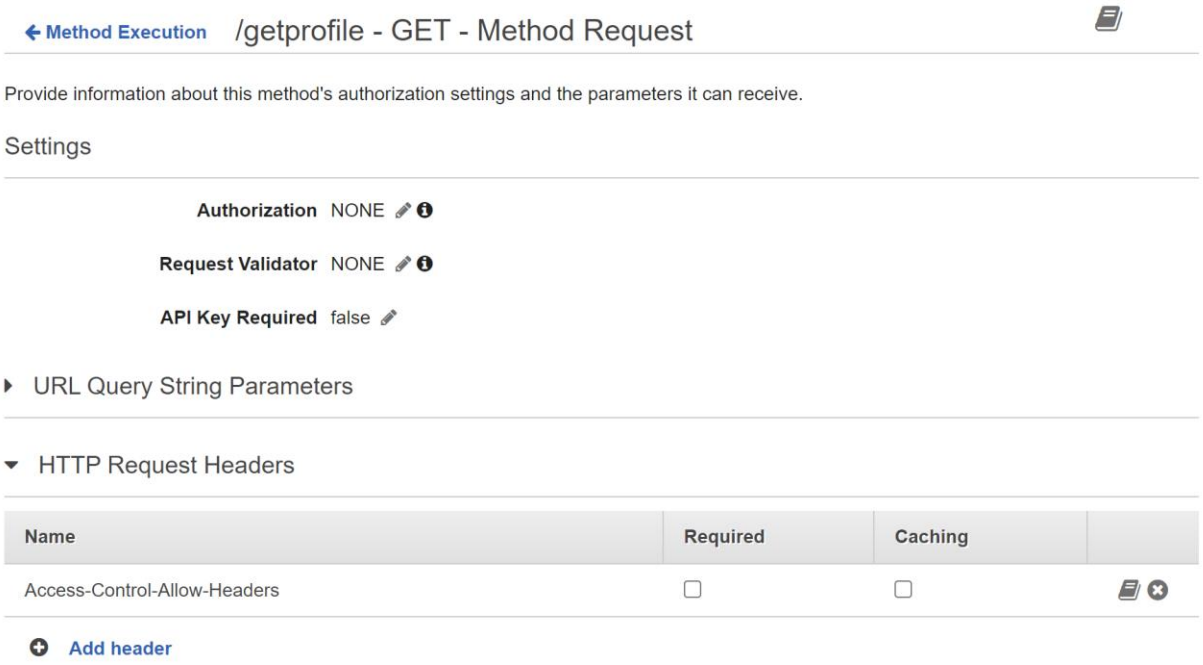
Step 6:

Click on the Method Request as shown below.



Step 7:

For the HTTP Request Headers add the following as shown below.

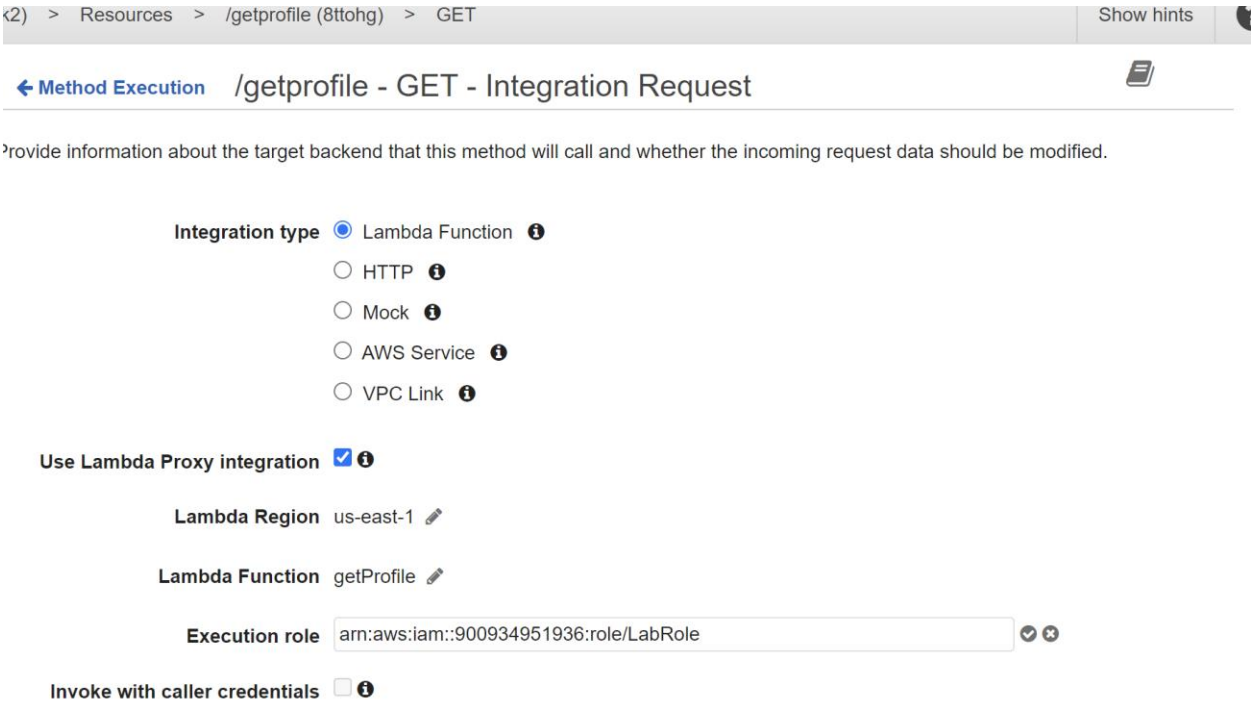


Step 8:

Next, go back and click on the Integration Request.

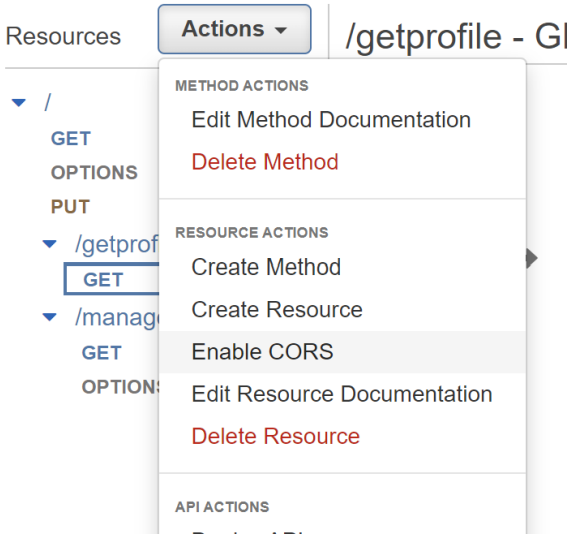
Step 9:

Fill in the Execution Role with LabRole and click on the tick icon, we will use the same execution role that we used from 7.2 Step 9.



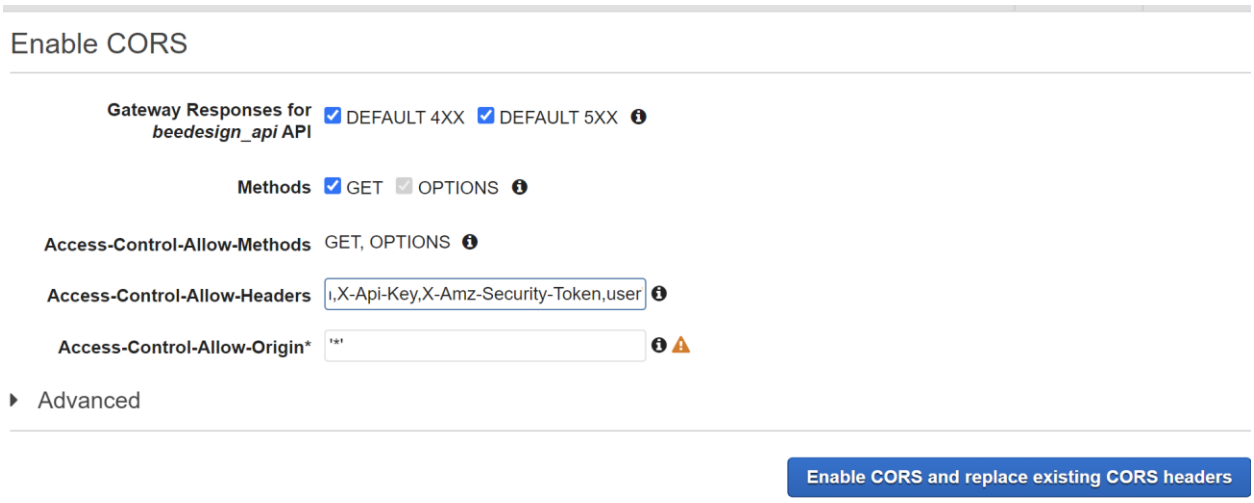
Step 10:

Click on Actions and click on Enable CORS.

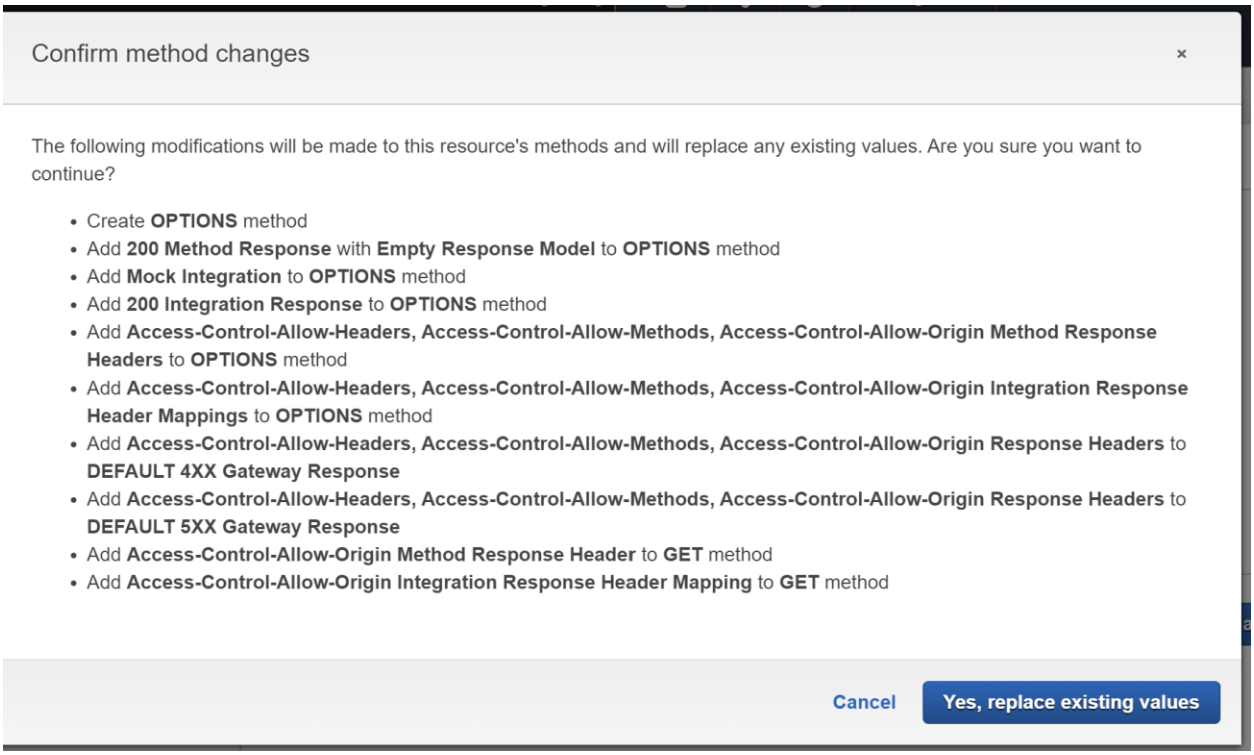


Step 11:

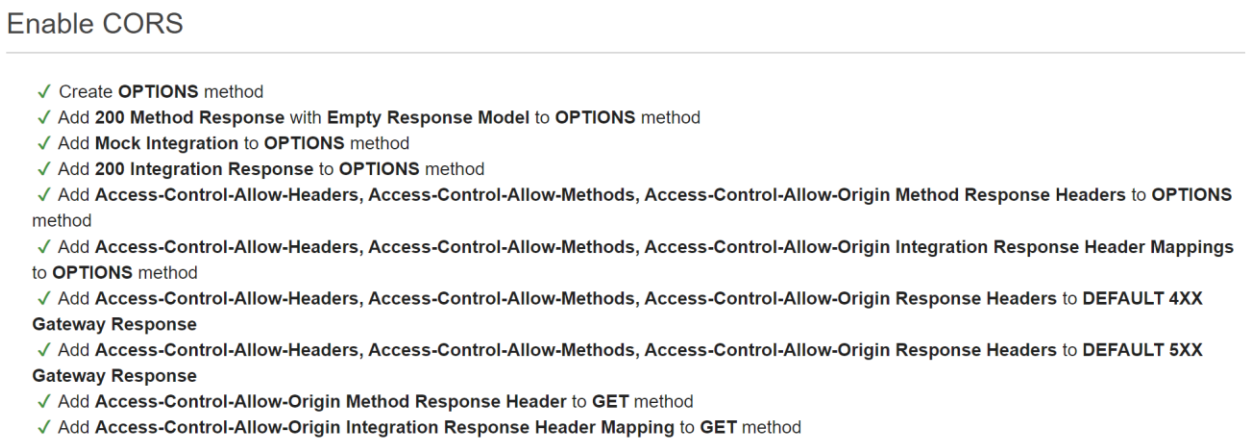
Remember to check the following as shown below and add a “,user” before the closing quote for Access-Control_Allow_Headers and click on Enable CORS.



Click on Yes, replace existing values.

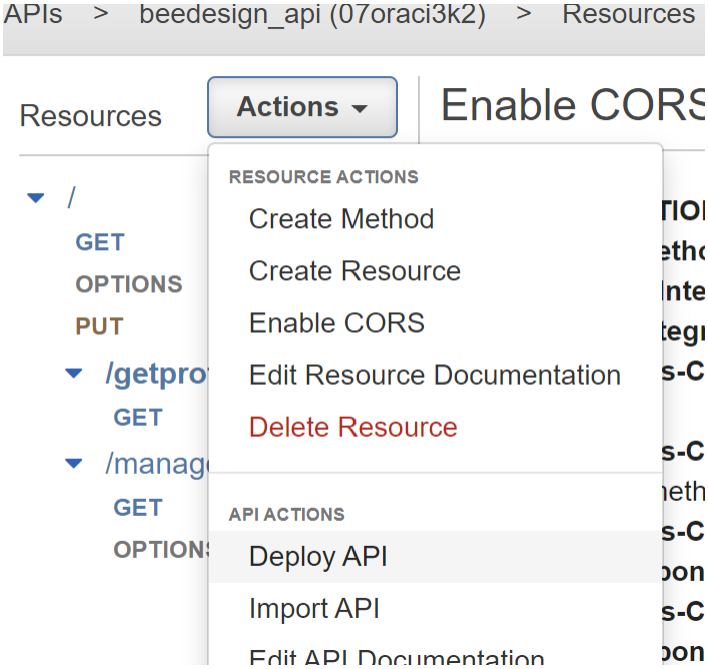


You should see the following.



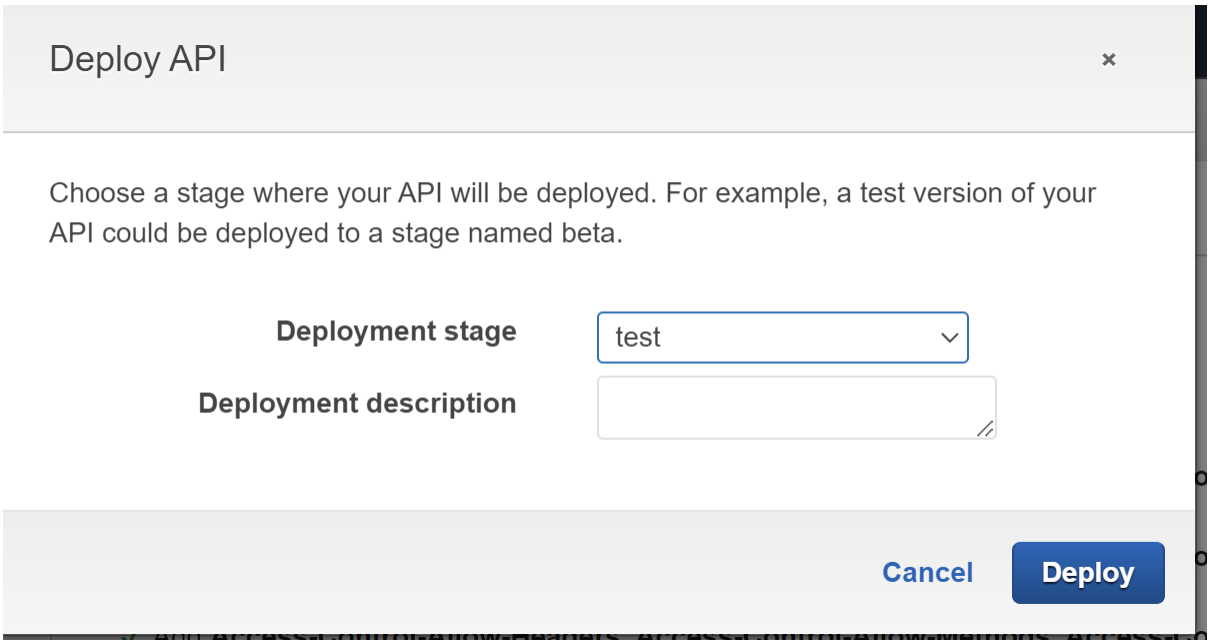
Step 12:

Click on Actions and then click on Deploy API.



Step 13:

Select test as deployment stage and click on Deploy button.



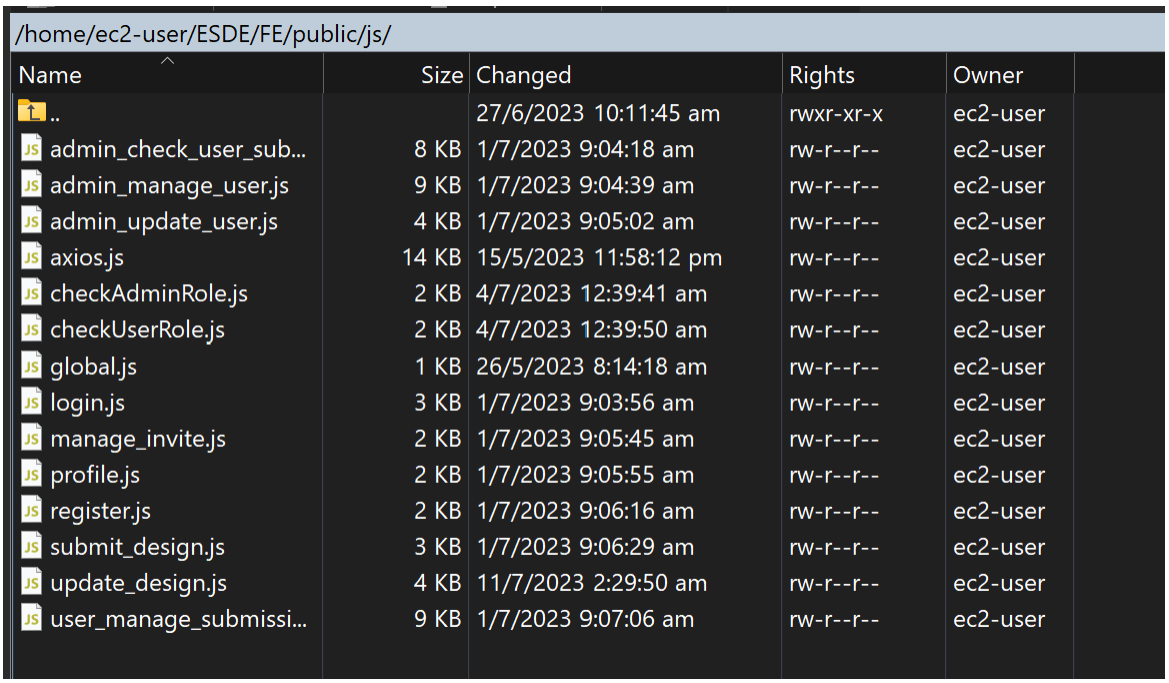
Step 14:

You should be able to see the following.

7.10 Adding the getProfile API Gateway to WinSCP

Step 1:

Go into WinSCP and go to the directory "/home/ec2-user/ESDE/FE/public/js".



Name	Size	Changed	Rights	Owner
..		27/6/2023 10:11:45 am	rwxr-xr-x	ec2-user
admin_check_user_sub...	8 KB	1/7/2023 9:04:18 am	rw-r--r--	ec2-user
admin_manage_user.js	9 KB	1/7/2023 9:04:39 am	rw-r--r--	ec2-user
admin_update_user.js	4 KB	1/7/2023 9:05:02 am	rw-r--r--	ec2-user
axios.js	14 KB	15/5/2023 11:58:12 pm	rw-r--r--	ec2-user
checkAdminRole.js	2 KB	4/7/2023 12:39:41 am	rw-r--r--	ec2-user
checkUserRole.js	2 KB	4/7/2023 12:39:50 am	rw-r--r--	ec2-user
global.js	1 KB	26/5/2023 8:14:18 am	rw-r--r--	ec2-user
login.js	3 KB	1/7/2023 9:03:56 am	rw-r--r--	ec2-user
manage_invite.js	2 KB	1/7/2023 9:05:45 am	rw-r--r--	ec2-user
profile.js	2 KB	1/7/2023 9:05:55 am	rw-r--r--	ec2-user
register.js	2 KB	1/7/2023 9:06:16 am	rw-r--r--	ec2-user
submit_design.js	3 KB	1/7/2023 9:06:29 am	rw-r--r--	ec2-user
update_design.js	4 KB	11/7/2023 2:29:50 am	rw-r--r--	ec2-user
user_manage_submissi...	9 KB	1/7/2023 9:07:06 am	rw-r--r--	ec2-user

Step 2:

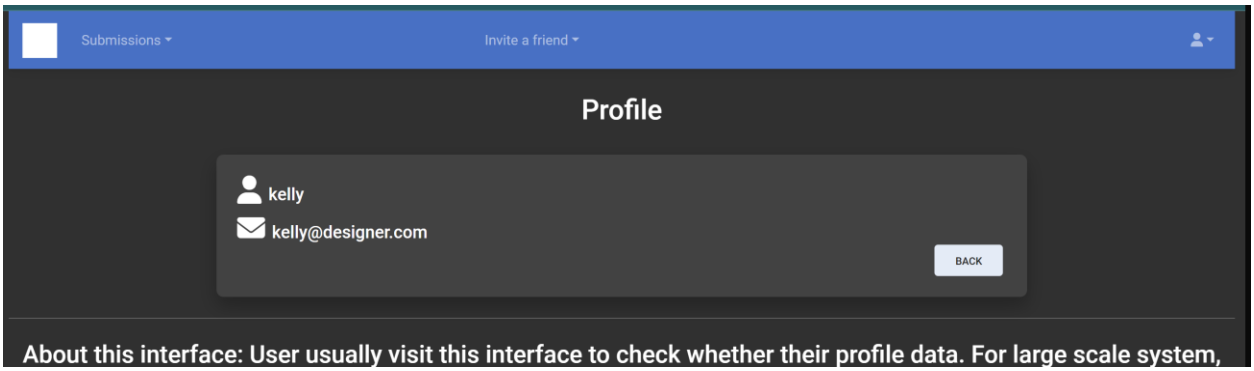
Click on the profile.js file and add the API Gateway Invoke URL to the URL as shown below and save the file.

URL Link: 'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test/getprofile?userId=' + userId

```
function getOneUser() {
  const baseUrl = "https://18.205.232.250:5000";
  let userId = sessionStorage.getItem("user_id");
  let token = sessionStorage.getItem("token");
  axios({
    headers: {
      user: userId,
      authorization: `Bearer ${token}`,
    },
    method: "get",
    url: 'https://07oraci3k2.execute-api.us-east-1.amazonaws.com/test/getprofile',
  })
    .then(function (response) {
      //Using the following to inspect the response.data data structure
      //before deciding the code which dynamically populate the elements with data.
      console.dir(response.data.userdata);
      const record = response.data.userdata;
      $("#fullNameOutput").append(record[0].fullname); //text(record.fullname);
      $("#emailOutput").append(record[0].email); //text(record.email);
    })
    .catch(function (response) {
      //Handle error
      console.dir(response);
```

Step 3:

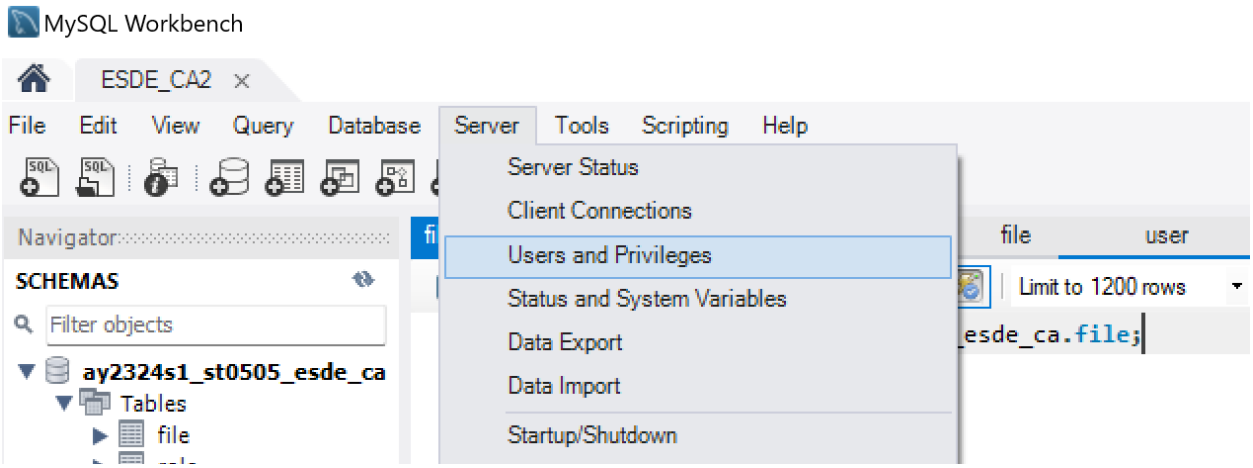
Now, run the frontend and backend server and see if you can view the current logged in user's profile.



7.11 Giving Least Privilege for Database credentials

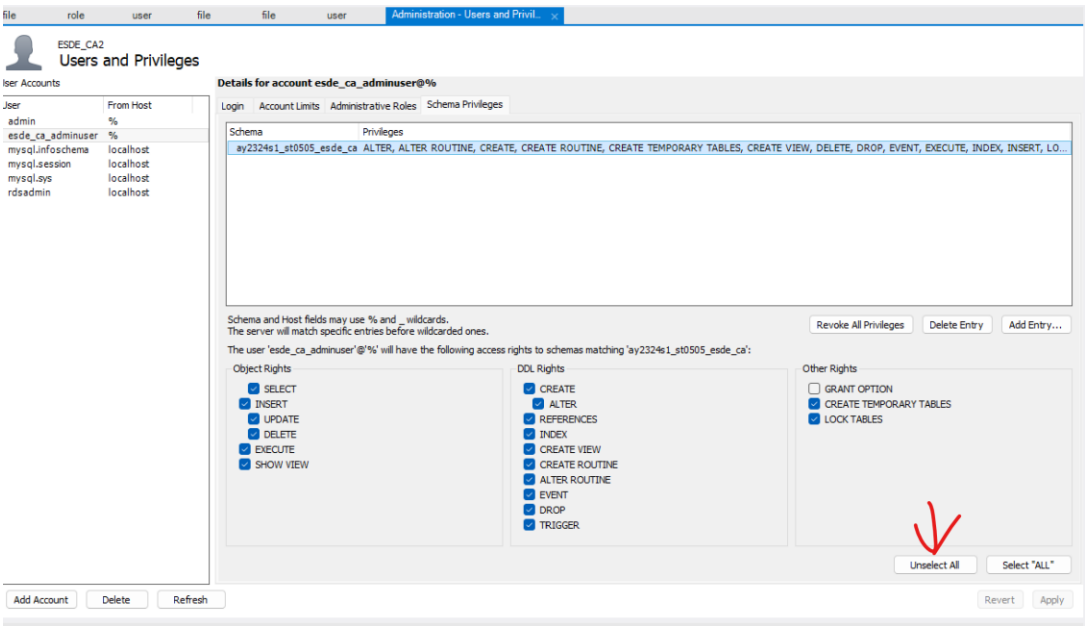
Step 1:

Go to MySQL Workbench and go to users and Privileges.



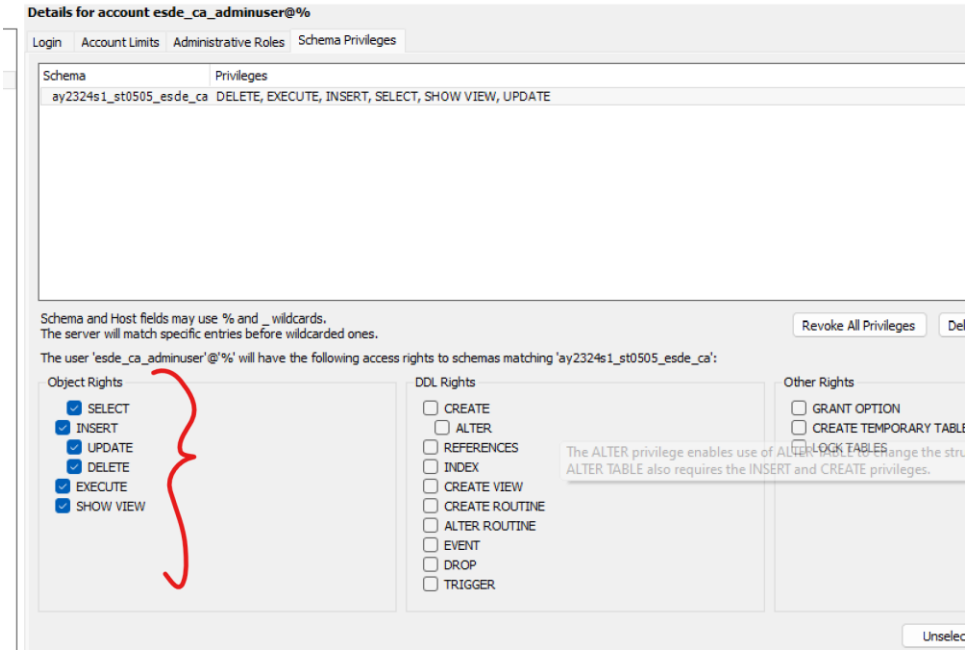
Step 2:

Go to Schema Privileges and click on unselect all.



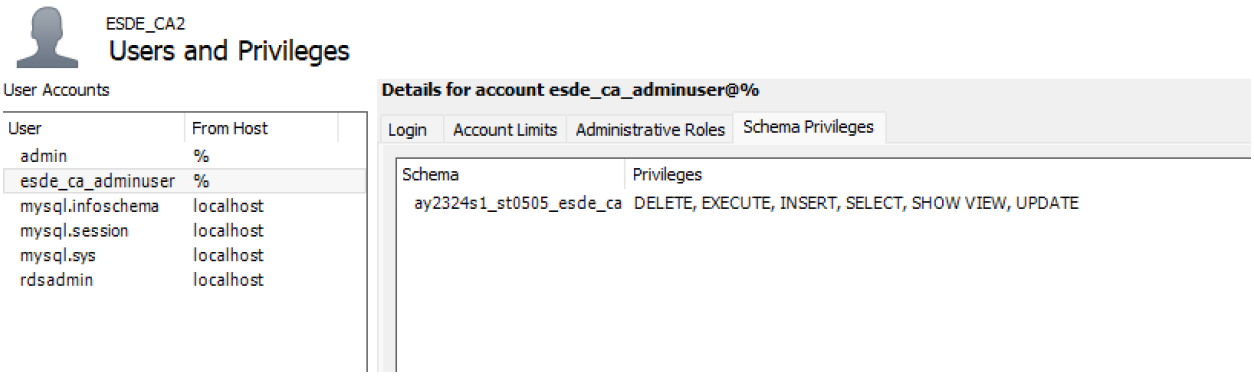
Step 3:

Select all under Object Rights and click on Apply.



Step 4:

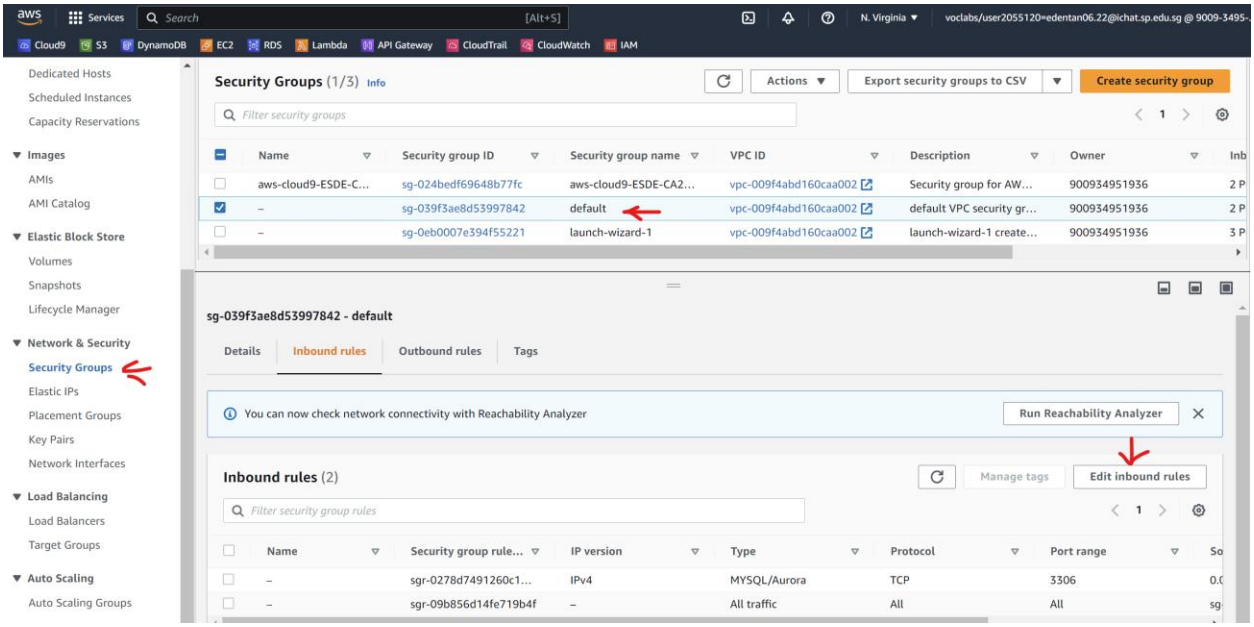
You should be able to see all the recent selected privileges for the User.



7.12 MySQL RDS Database Security, Encryption and Logging

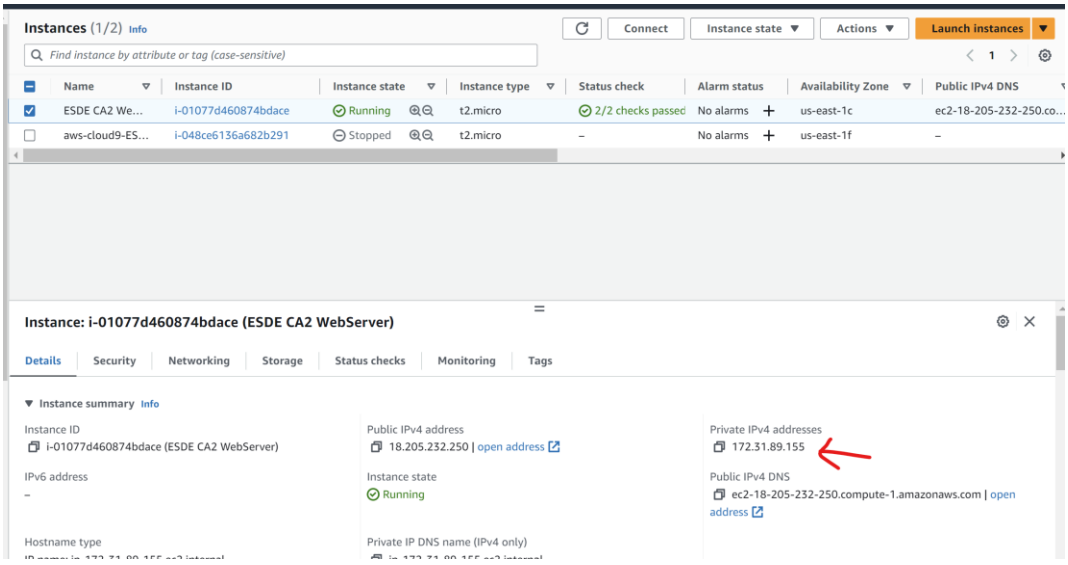
Step 1:

Go to AWS EC2 and go to Security Groups and select the default Security group name and click on Edit Inbound Rules as shown below.



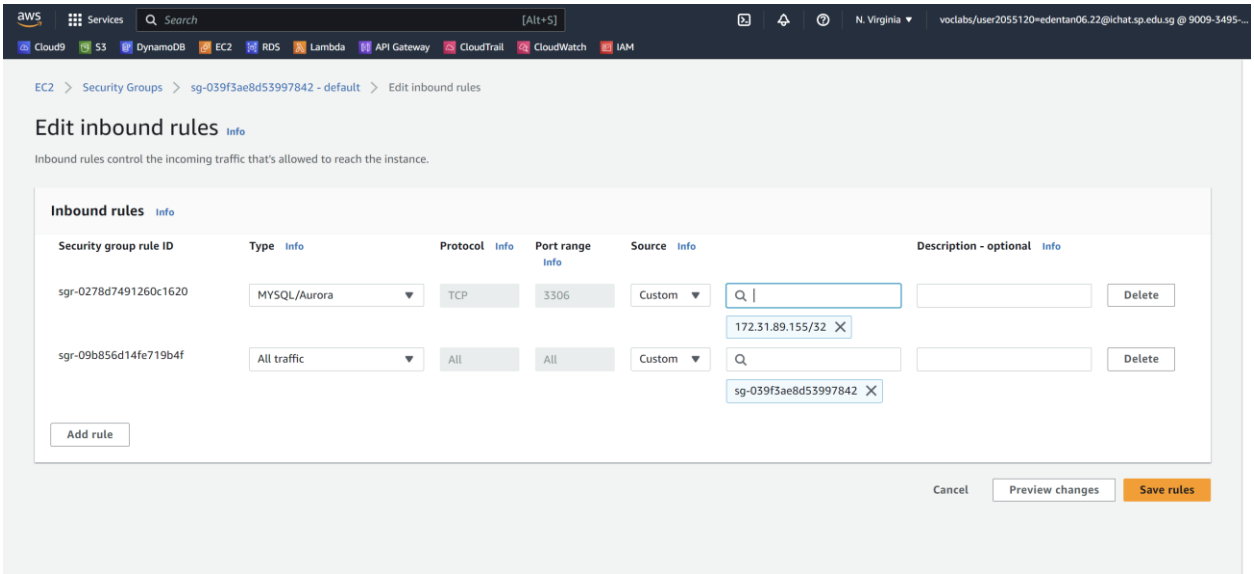
Step 2:

Open a separate tab and go to AWS EC2 Instances and choose the EC2 Instance using the RDS and copy the Private IP Address.



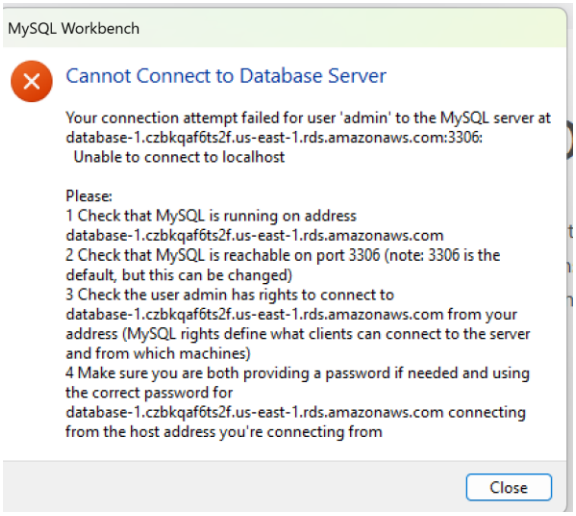
Step 3:

Next, paste the Private IP Address into the text box bedside the Source as 172.31.89.155/32 and save it.



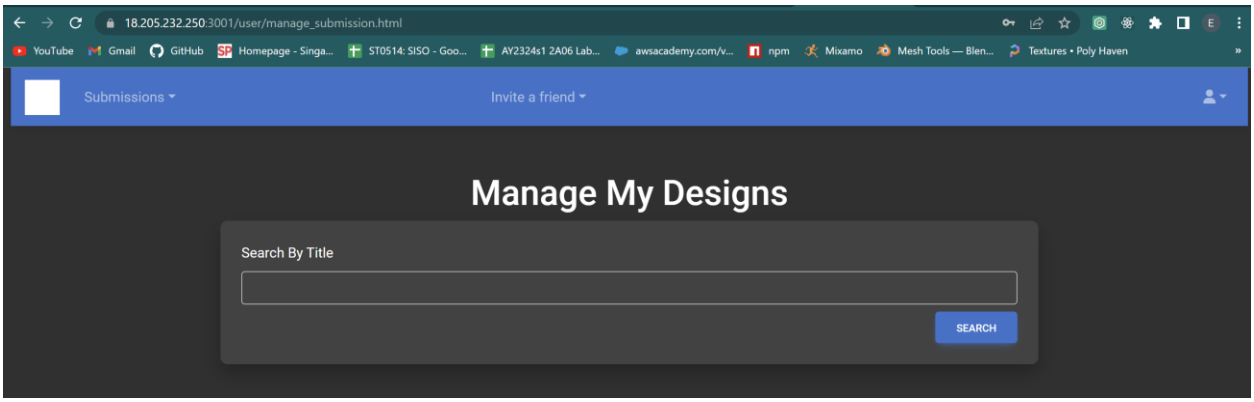
Step 4:

Now go to your MySQL Workbench and you should see that you are not able to view your database.



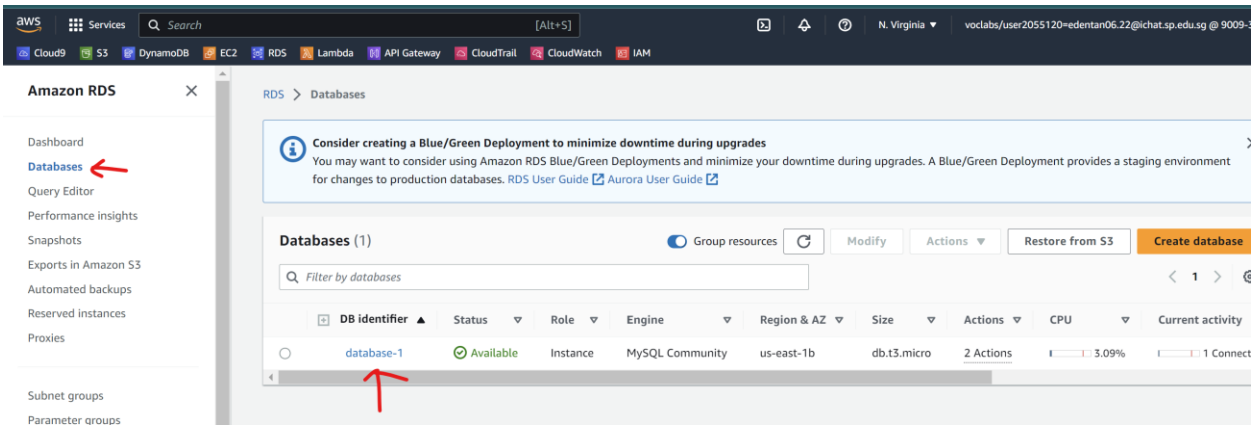
Step 5:

Check that you are still able to login to the Bee Design Website by logging in as Kelly.



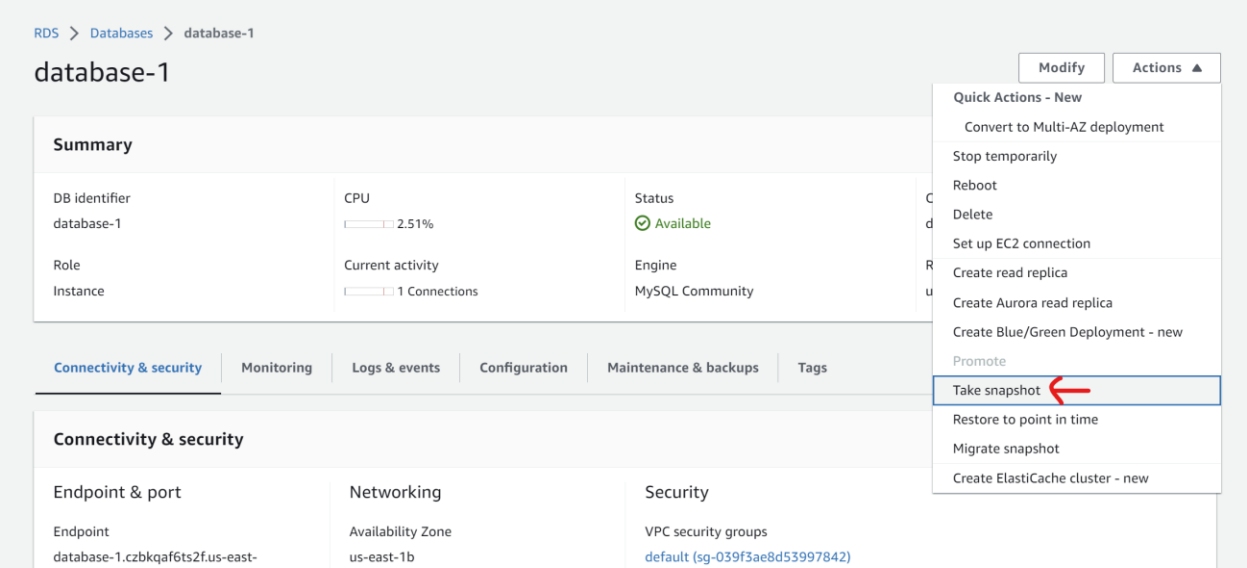
Step 6:

Now, go to AWS RDS and click on Databases and click on database-1.

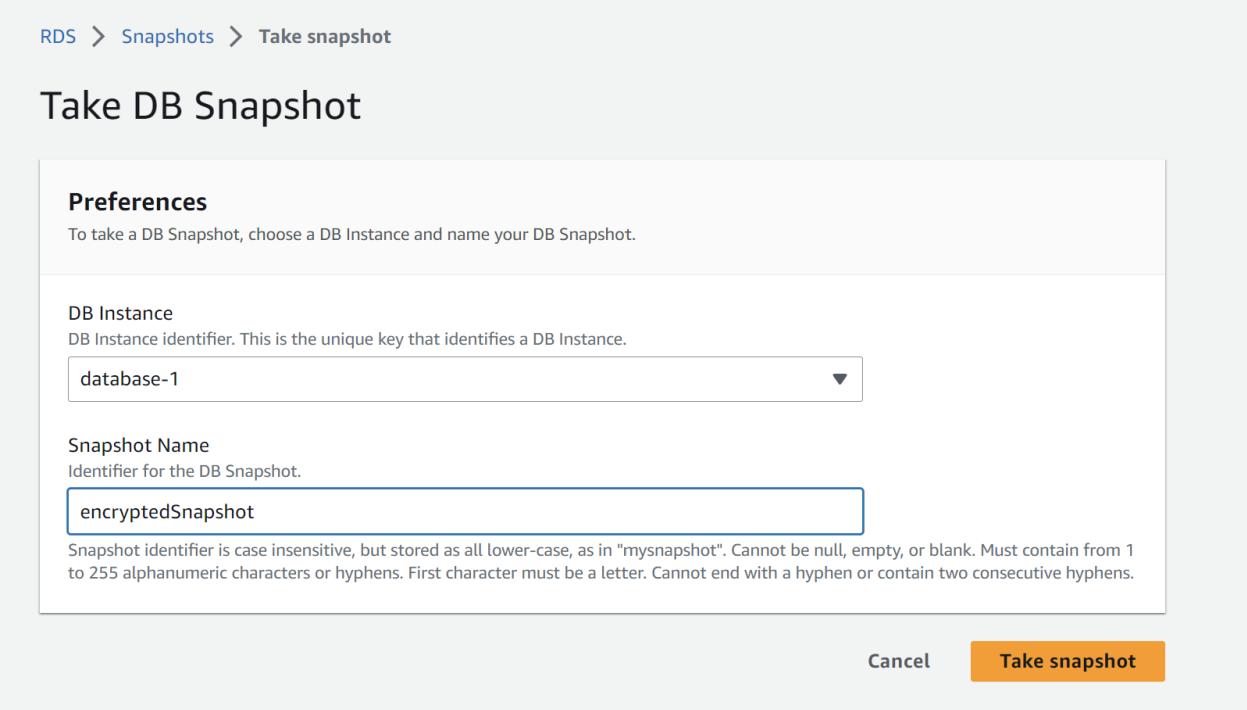


Step 7:

Under Actions, click on Take snapshot.

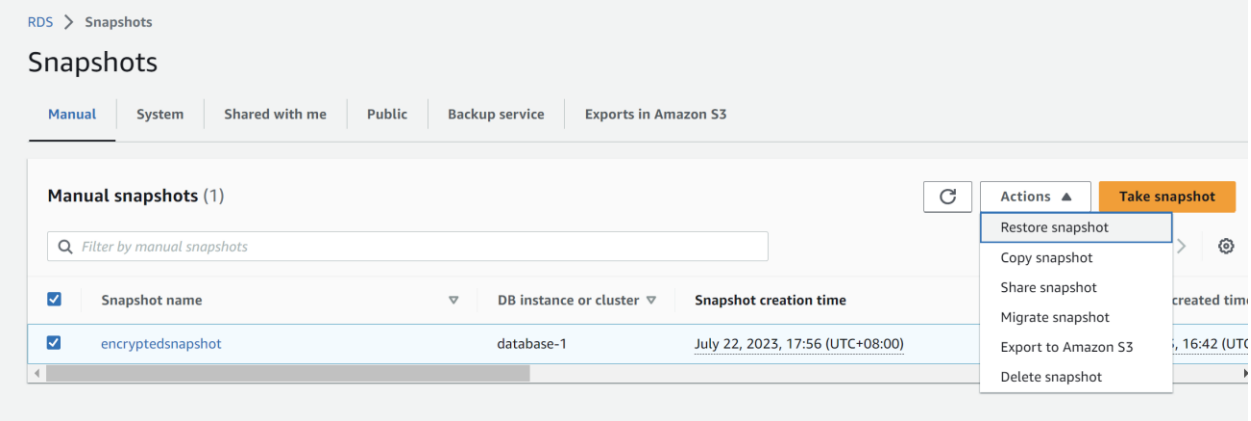


Give it a name of encryptedSnapshot and click on the Take snapshot button.



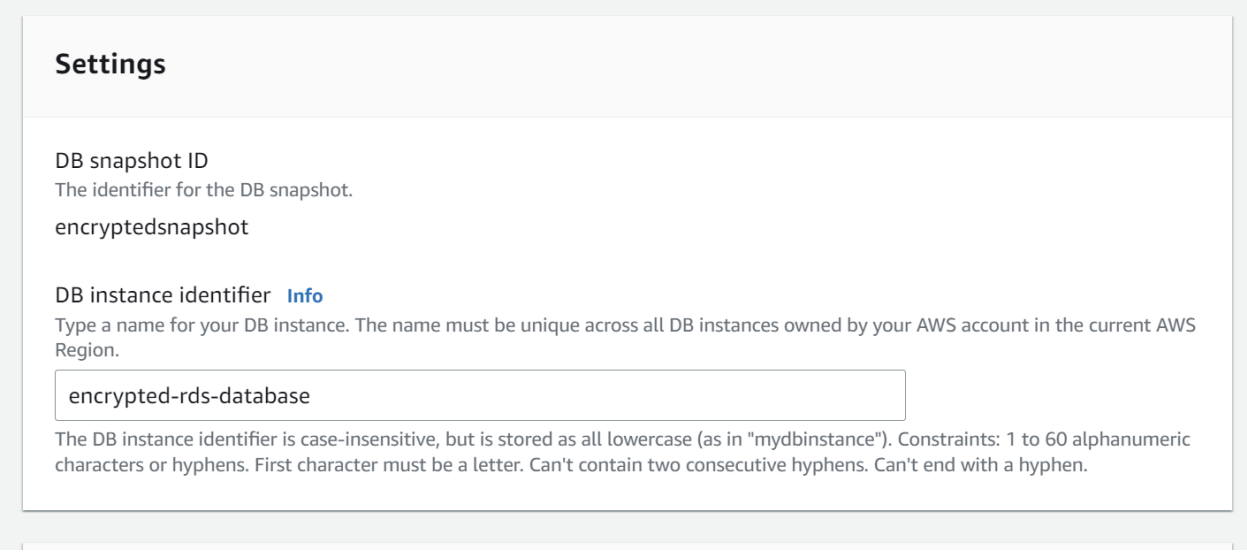
Step 8:

Make sure that your snapshot status is complete. Next, select the snapshot and click on Actions and select “Restore snapshot”.

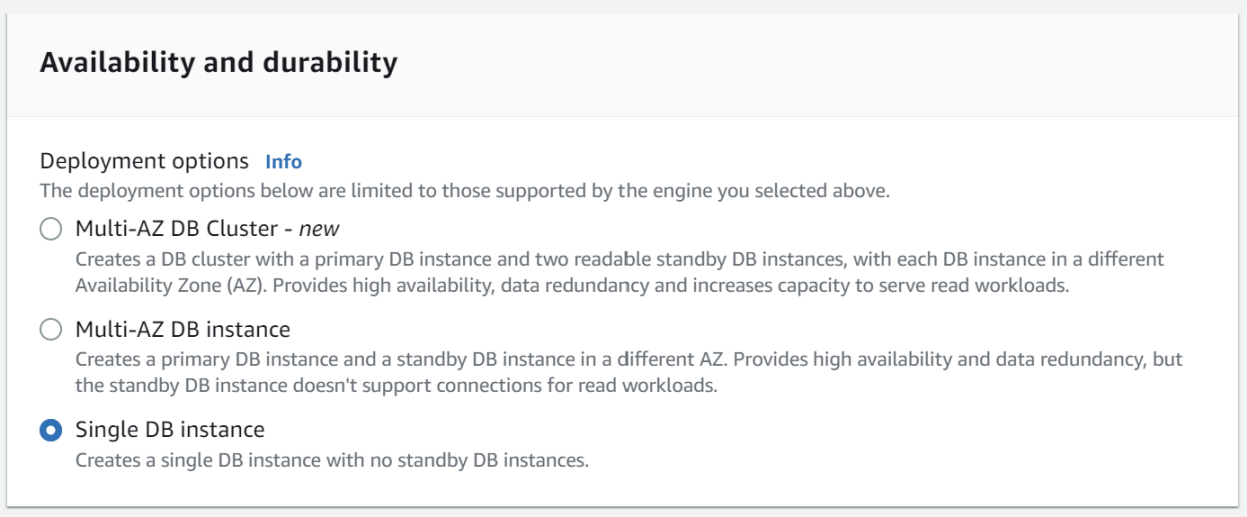


Step 9:

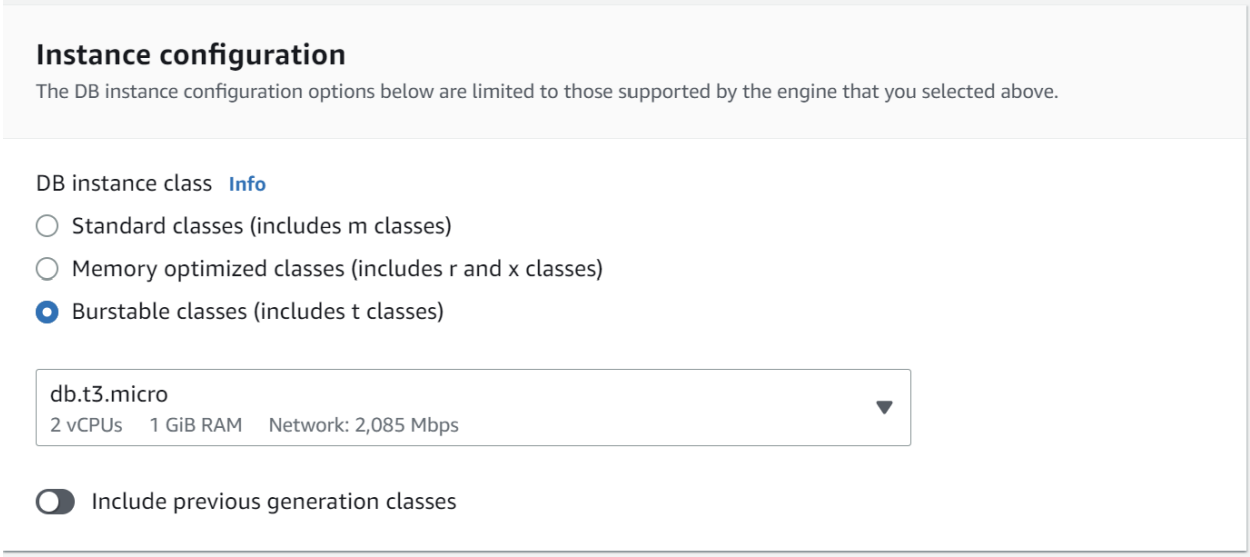
For the DB Instance identifier name, it as encrypted-rds-database as shown below.



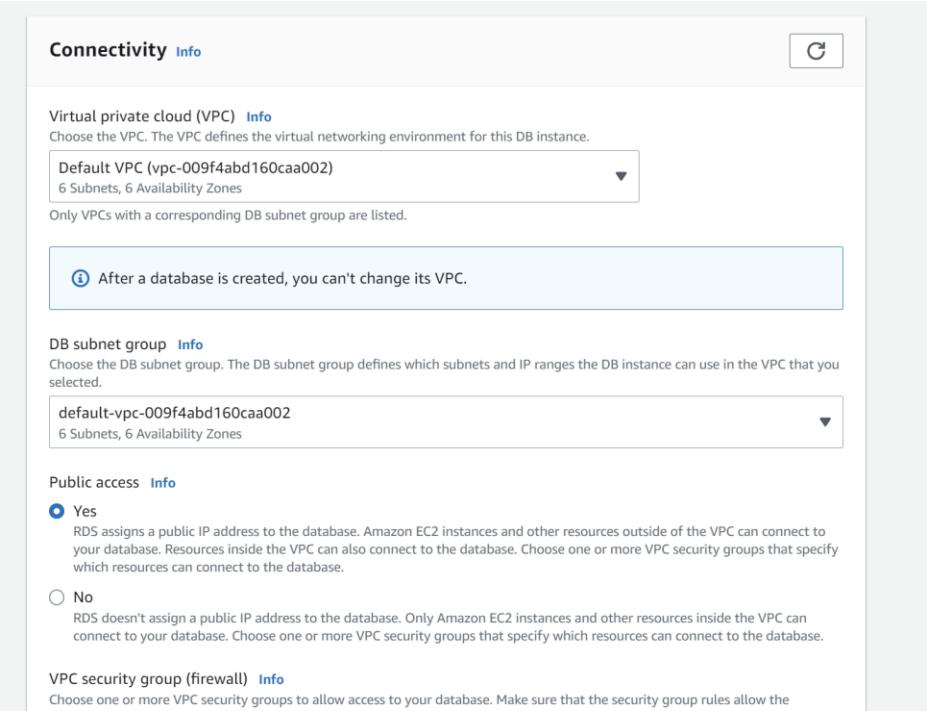
For Availability and durability select Single DB instance.



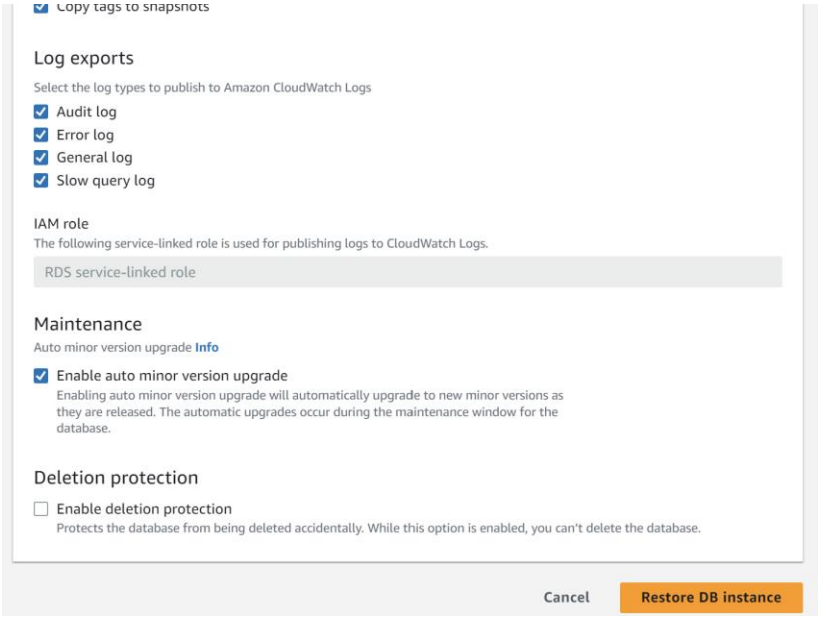
And for Instance, configuration select Burstable classes and then db.t3.micro.



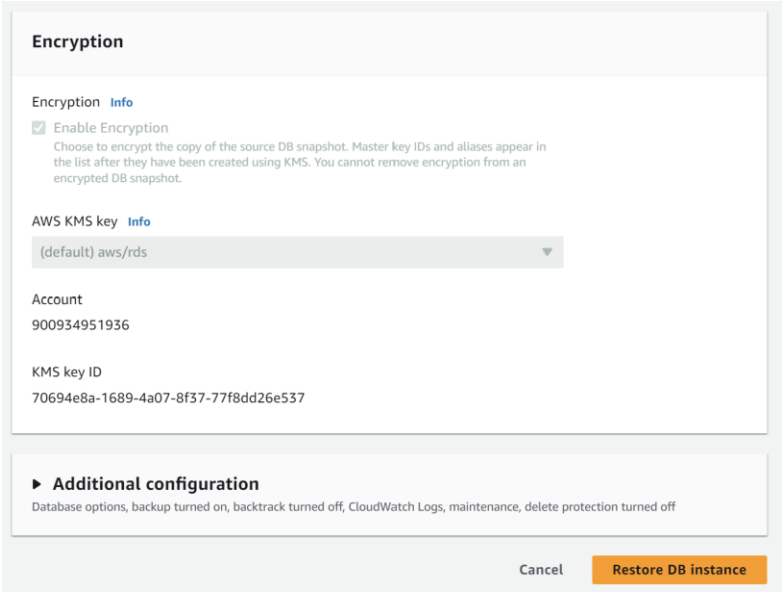
For Connectivity set public access to Yes.



For Additional configuration select all the Log exports as shown below.

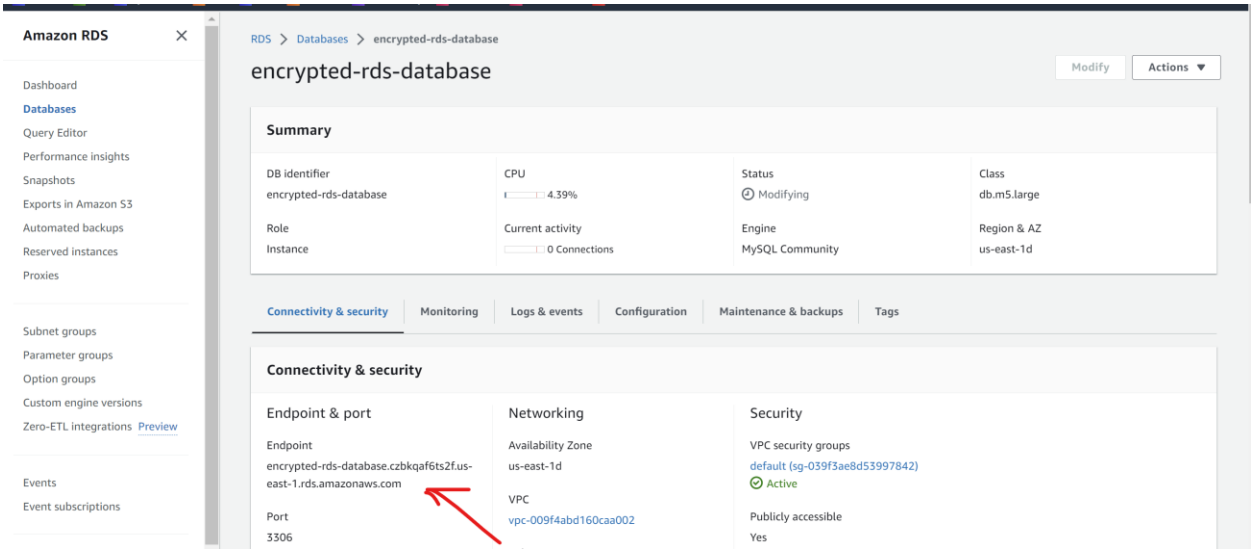


And for Encryption make sure that Enable Encryption is checked and then click on “Restore DB instance”.



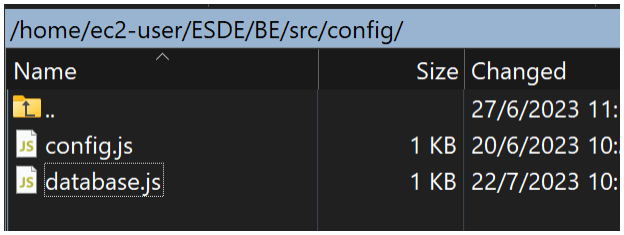
Step 9:

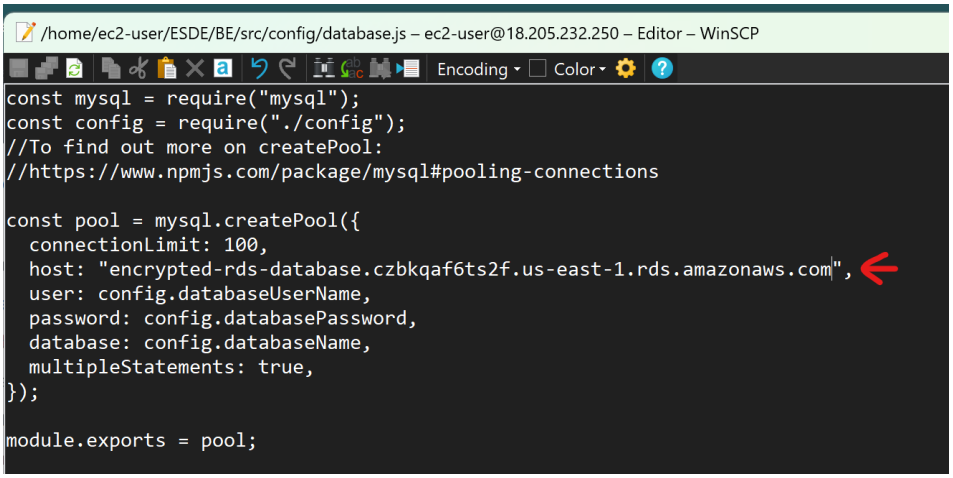
Go into the encrypted database and copy the Endpoint.



Step 10:

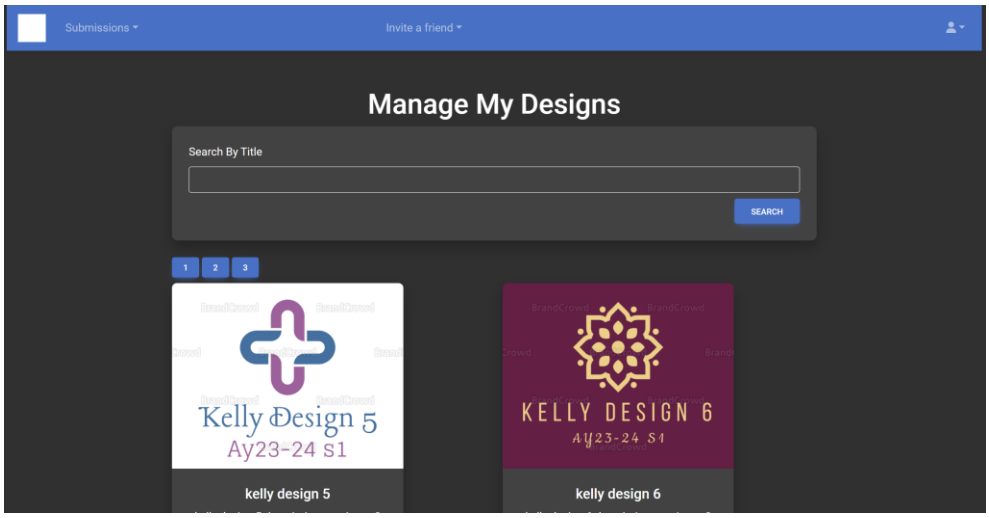
Go to WinSCP and go to the directory as shown below and add the encrypted rds database endpoint to the host.





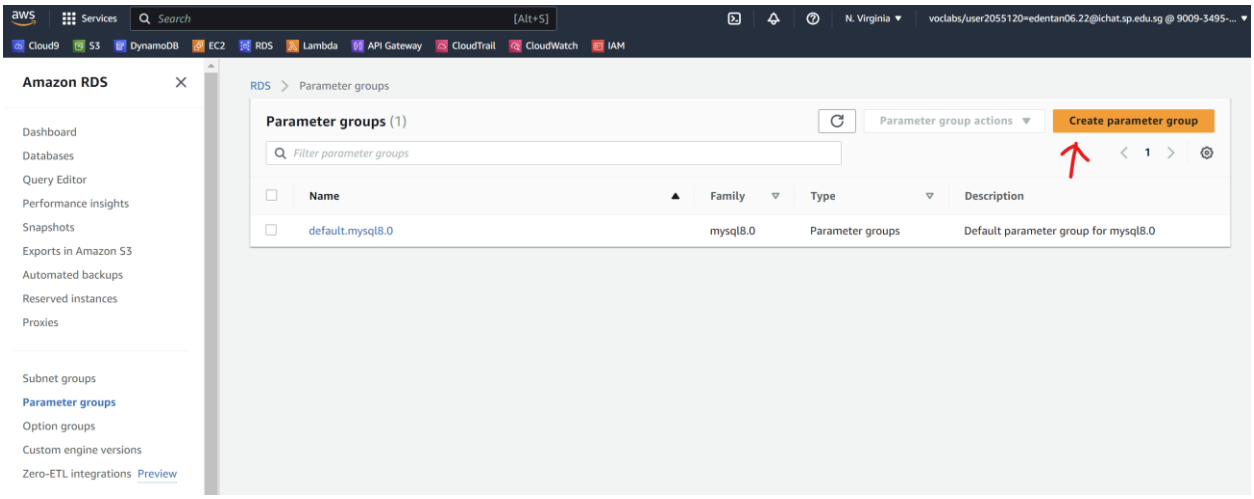
Step 11:

Now go to the Bee Design Website and try logging in if you are able to login then you have successfully encrypted the rds database.



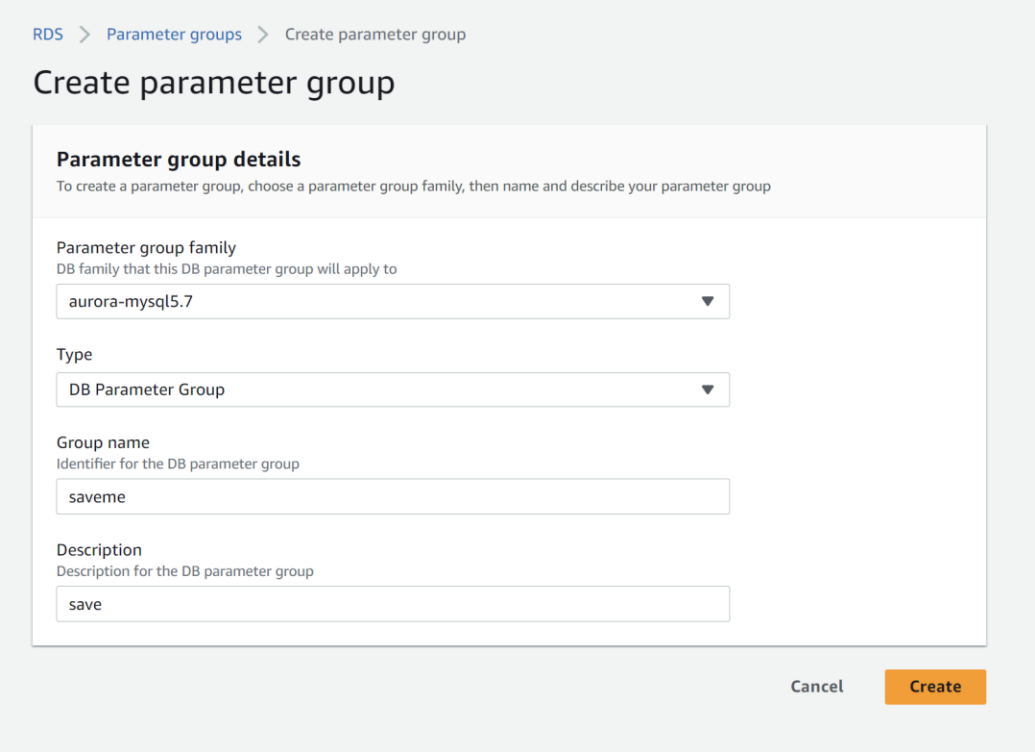
Step 12:

Go to AWS RDS and click on Parameter groups and click on Create parameter group.



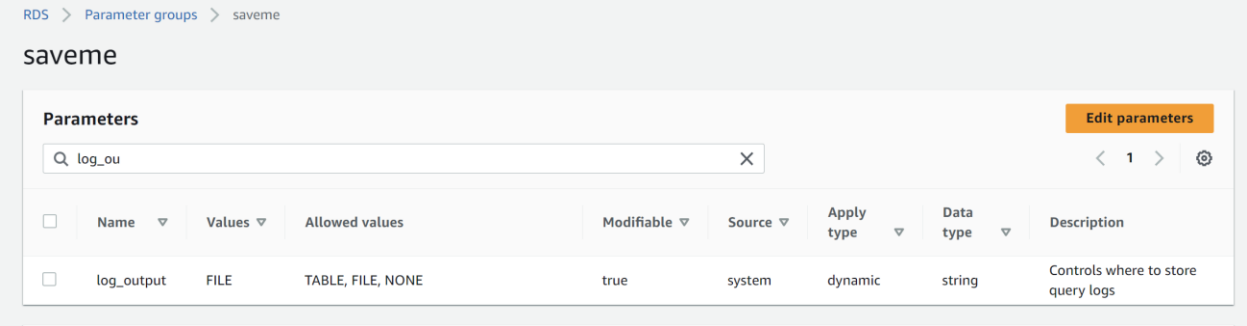
Step 13:

For the group name and description name it as shown below.



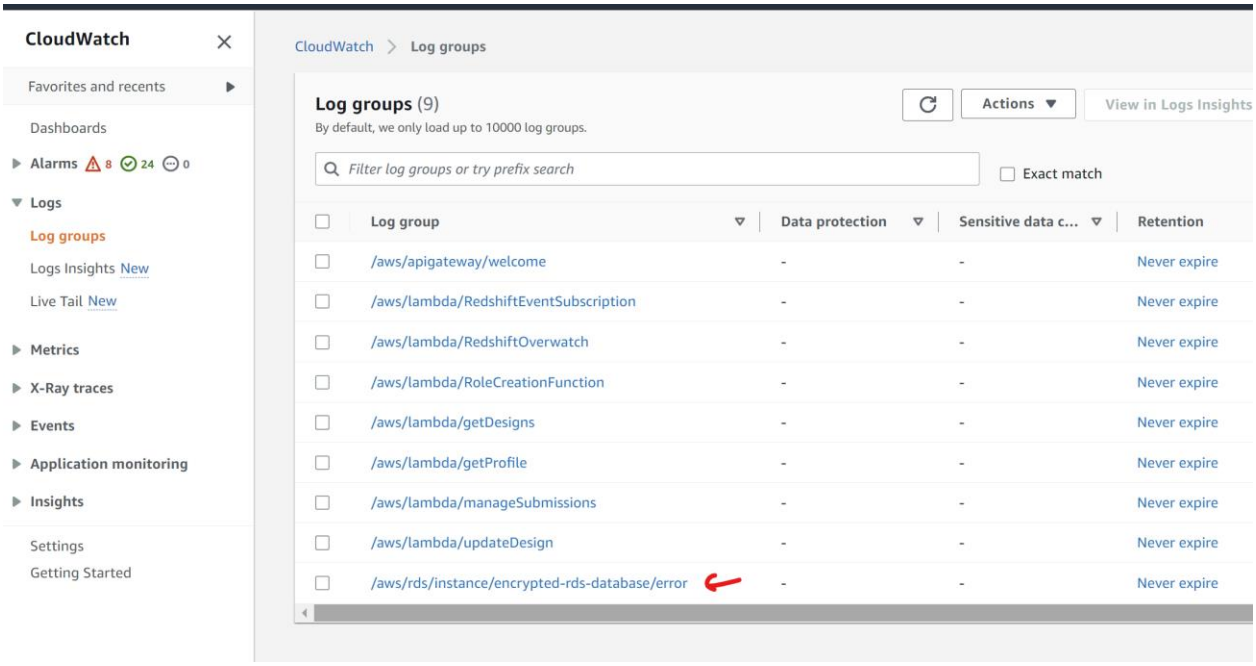
Step 14:

Go into the created Parameter group and make sure that log_output value is FILE.



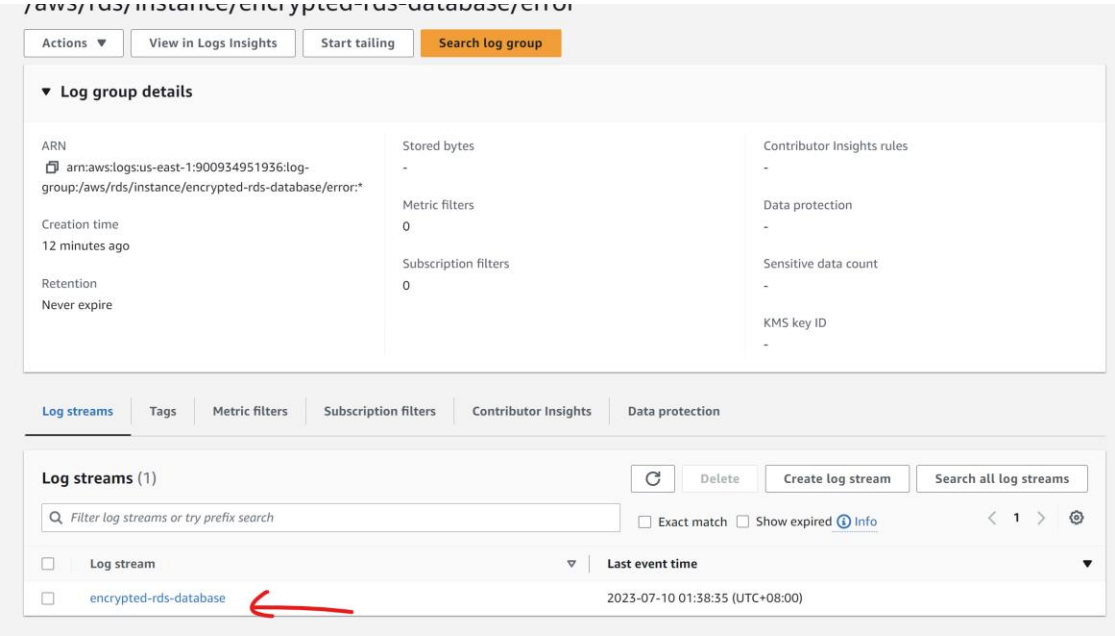
Step 15:

Head over to CloudWatch and under Log groups see if there is a /aws/rds/instance/encrypted-rds-database/error



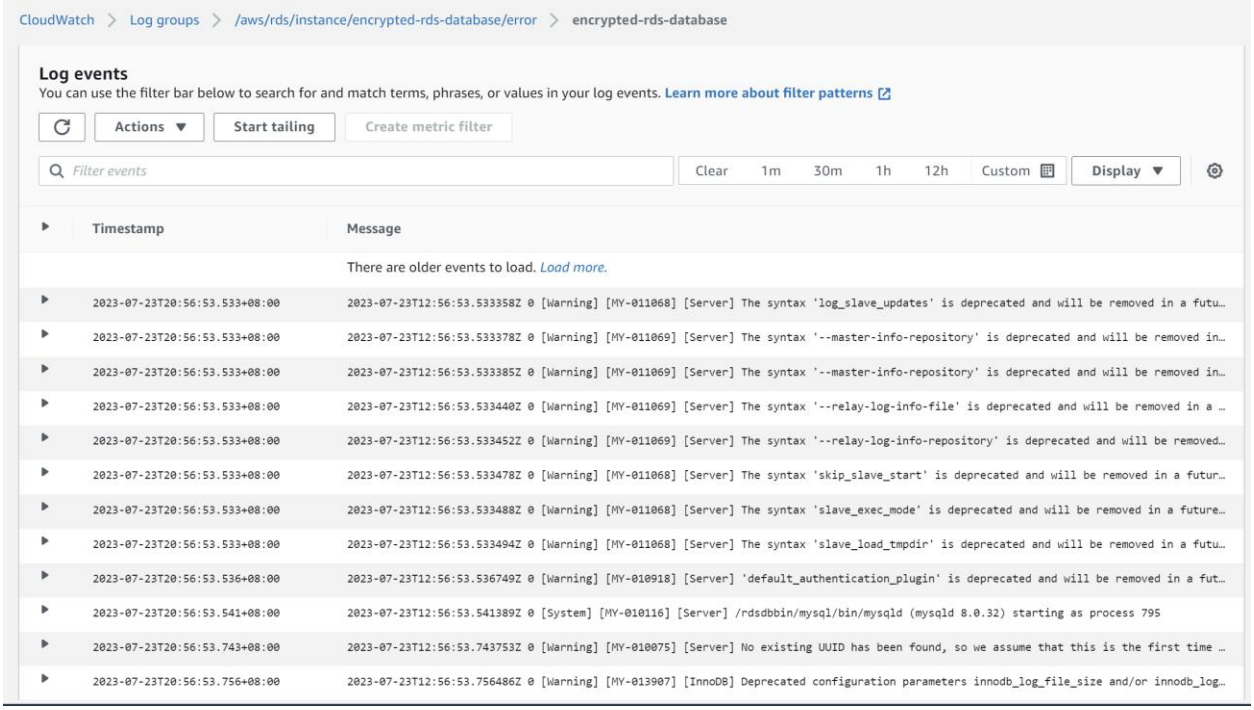
Step 16:

Click on the Log group and click on the encrypted-rds-database Log stream.



Step 17:

You should be able to see all the encrypted-rds-database logs as shown below.



References

- https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html