ESDE CA2

Name: Singh Sakshi

Student id : 2228479

Class: DIT/2a06

14 AUGUEST 2023

SINGAPORE POLYTECHNIC

SOC

Table of Contents

Part 1 - Set up EC2 instance on Aws. 4
What is Ec2 Functionality? 4
Access Ec2 Dashboard 4
Create new instance. 5
Configure Settings 6
Purpose of AMI: 6
Why t2.micro? 7
Create Key Pair 7
Network Settings 9
Create Elastic IP 11
Associate EIP with instance 14
Connect EIP to EC2 instance on WinSCP. 16
Download software. 16
Configure Settings 17
Part 2 - Install NodeJS and copy Bee Design app to EC2. 24
Installing Node js 24
Deploy Bee Design 26
Configuring security settings 31
Why configure security settings? 31
Launch website on EC2 33
Part 3 - Setup RDS 35
RDS 35
Configure settings to create RDS database. 35
Connect to RDS server by using MySQL workbench. 38
Added security group rule for MySQL. 38
In MySQL configuration 40
Configure Connection in MySQL Workbench: 40
Run Database Script 42
Modify database.js. 42
Run SeedData.js 44
Change URL 45
Launch website to check RDS configuration. 46
Part 4 – DynamoDB 48
DynamoDB in AWS 48
Export records from RDS MySQL to a Json file 48

Create DynamoDB Table. 52
Populate DynamoDB using Node js 53
Create Cloud 9 Environment 53
Uploaded file-data.json and seed_files_table.js to Cloud9 54
Part 5 - Setup Lambda 57
Lambda in AWS 57
Create Lambda function. 57
Prepare Lambda codes in cloud 9 58
Upload Lambda zip using AWS CLI 58
Index.js explanation 59
Dynamodbs.js Explanation 60
Create test event. 61
Part 6 - API Gateway 63
Api Gateway in Aws 63
Create Bee Design Api 63
Create and configure get Designs Method 65
Method Request 66
Integration Request 67
Enable Cors 69
Cors Importance 69
Deploy API 72
Modify App code to use API endpoint. 73
Modify DynamoDB 74
Testing the application 75
Additional Features 77
Submit Design (Post) 77
Create Lambda Function 77
Create FOLDER in Visual Studio Code 77
Environment Variables 79
Dynamodb.js Explanation 80
Index.js Explanation 81
Modify app code 82
Testing the Application 83
Update Design (Put) 85
Create Lambda Fucntion 85
Dynamodb.js 85

Index.js 86
Deploy API 86
Modified Code for App Update_Design.js 87
Testing Application 87
Manage User Submission (Get) 88
Create Lambda function. 88
Dynamodb.js 88
Index.js 89
API 90
Modified code for app 91
Testing Application 93
Get profile (Get) 94
Create Lambda function 94
Dyanamodb.js 94
Index.js 95
API 96
Modify app code 96
Testing application 96
MySQL RDS security 98
Ensuring only the EC2 instance running NODEJS code can have access to RDS 98
Increase Security Conclusion 101
Using the least privilege for database connection credentials 102
What does it mean? 102
VerifyAdmin function 105
In conclusion advantage to security 106

Part 1- Set up EC2 instance on Aws.

What is Ec2 Functionality?

Amazon Elastic Compute Cloud (Amazon EC2) is a web service offered by Amazon Web Services (AWS) that allows you to *create and manage virtual servers* in the cloud.

- In Ec2 you can create and configure instances of virtual machines, known as "instances," on-demand.
- These instances can be customized to meet your specific computing needs.

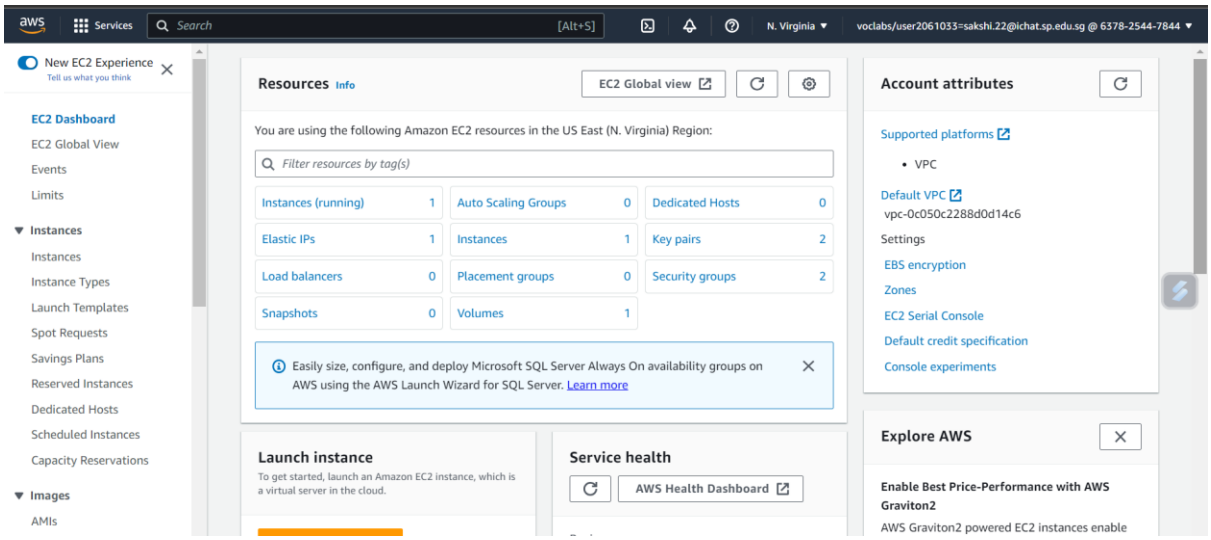
You will be **setting up and instance to host your Bee Design website** which will provide you with **more control** over the server environment **compared to hosting on localhost**, allowing you to configure the server exactly to your needs.

Access Ec2 Dashboard

Step 1: *Access the EC2 dashboard* by search up EC2 on AWS Management Console. You will reach the EC2 page.

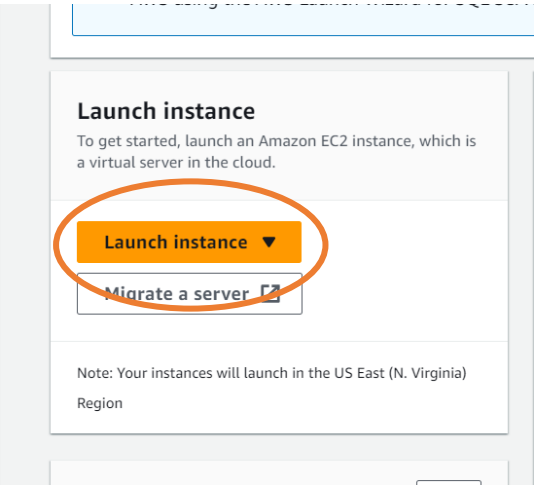
Step 2: Click on Ec2 dashboard to access it.

This is the dashboard.

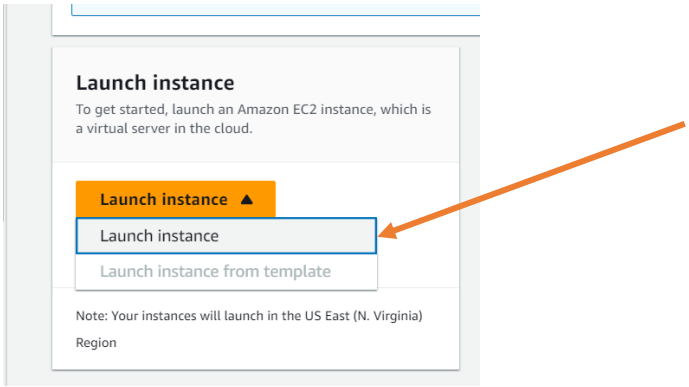


Create new instance.

Step 3: Scroll down and find “Launch instance: and click on the yellow button “**Launch Instance”.**

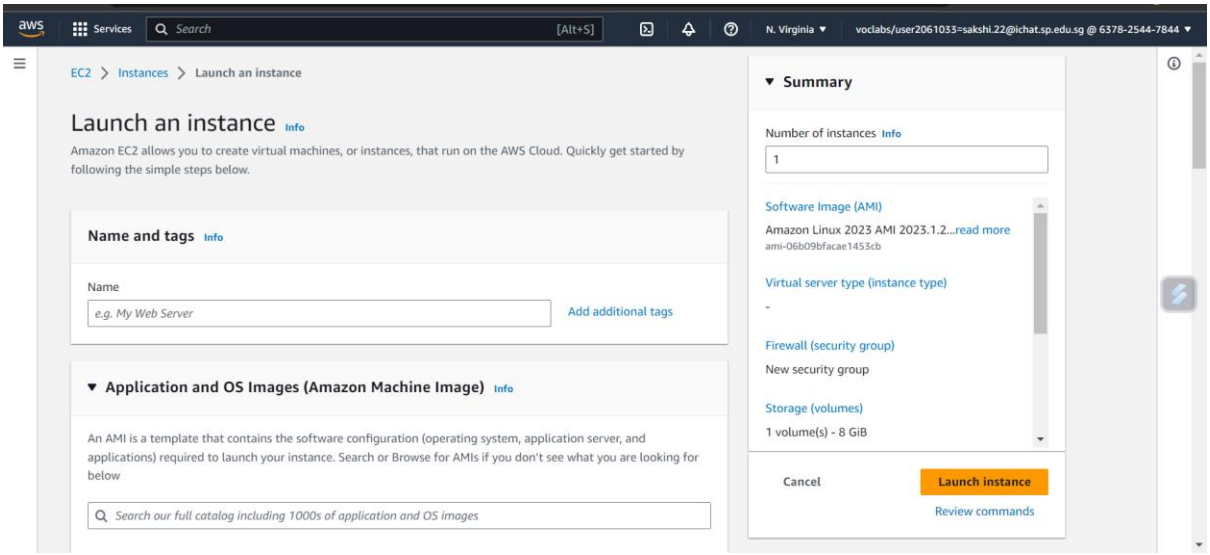


Step 4: Upon clicking, you will see a **dropdown with “Lauch Instance”**. *Click on it.*



Step 5: You will be directed to the page where you **can configure your instance settings** before creating it.

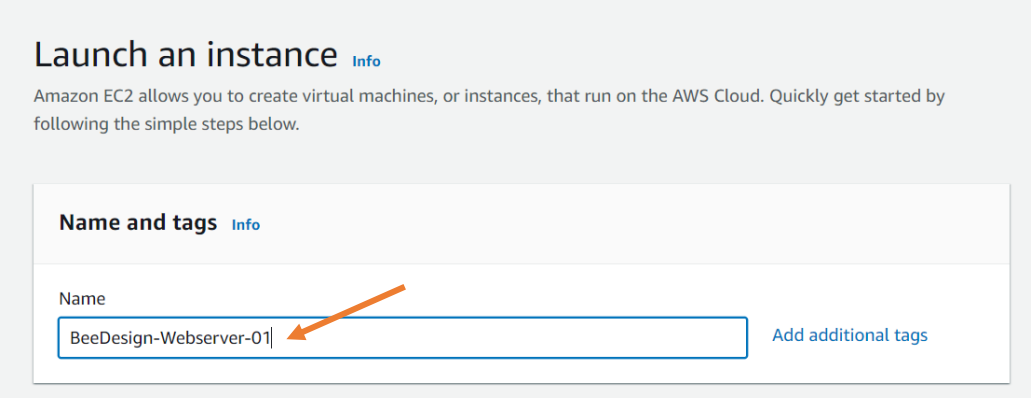
This is the page to configure your instance settings.



Configure Settings

Step 6: Name the instance as **“BeeDesign-Websever-01”.**

This will be the name of your instance for hosting your website later.



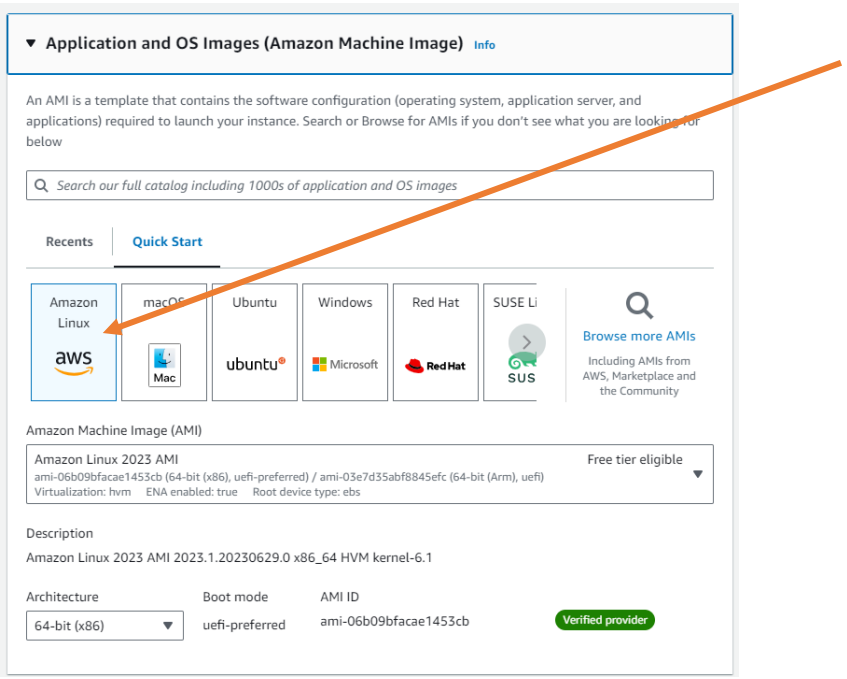
Purpose of AMI:

The AMI is a **pre-configured virtual machine image** that **contains the necessary information** to launch an instance, including the operating system, software, and other settings.

We will choose **Amazon Linux 2023** as the **AMI**

- Amazon Linux is a Linux distribution specifically designed and optimized for running on AWS infrastructure.
- It provides a **stable, secure, and lightweight environment** for hosting applications and services as for us would be our bee design website.

Step 7: **Choose Amazon Linux 2023** AMI 2023.0.20230614.0 x86_64 HVM kerne6.1l- as your Amazon Machine Image (AMI).

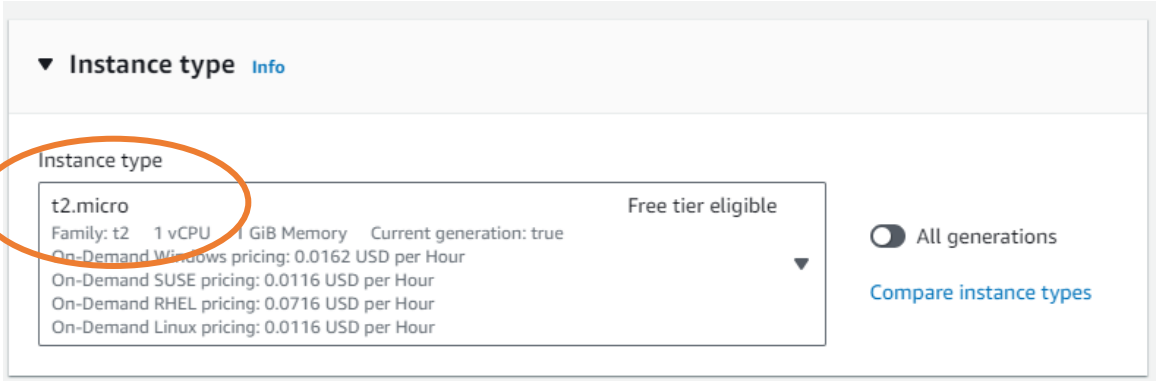


Why t2.micro?

Testing and Development:

- It's a good choice for developers who want to test applications, perform software development, and experiment with AWS services **without incurring high costs.**

Step 8: Choose instance type as t2. micro



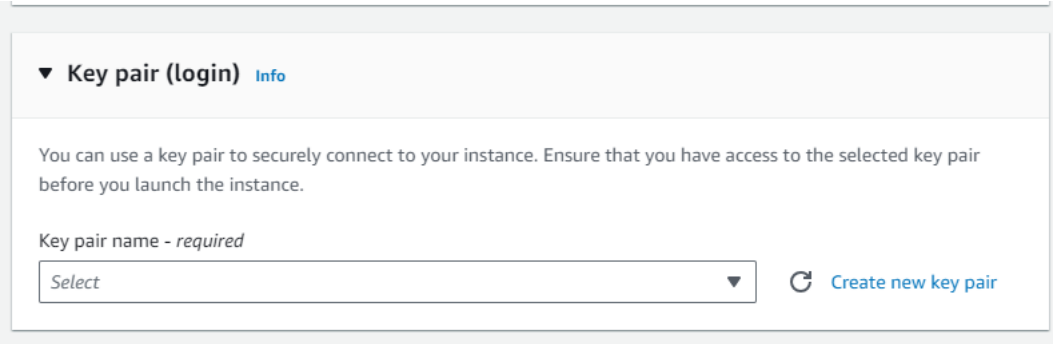
Create Key Pair

Now we will be creating a key pair.

Creating a key pair for an EC2 instance enhances security by enabling secure remote access.

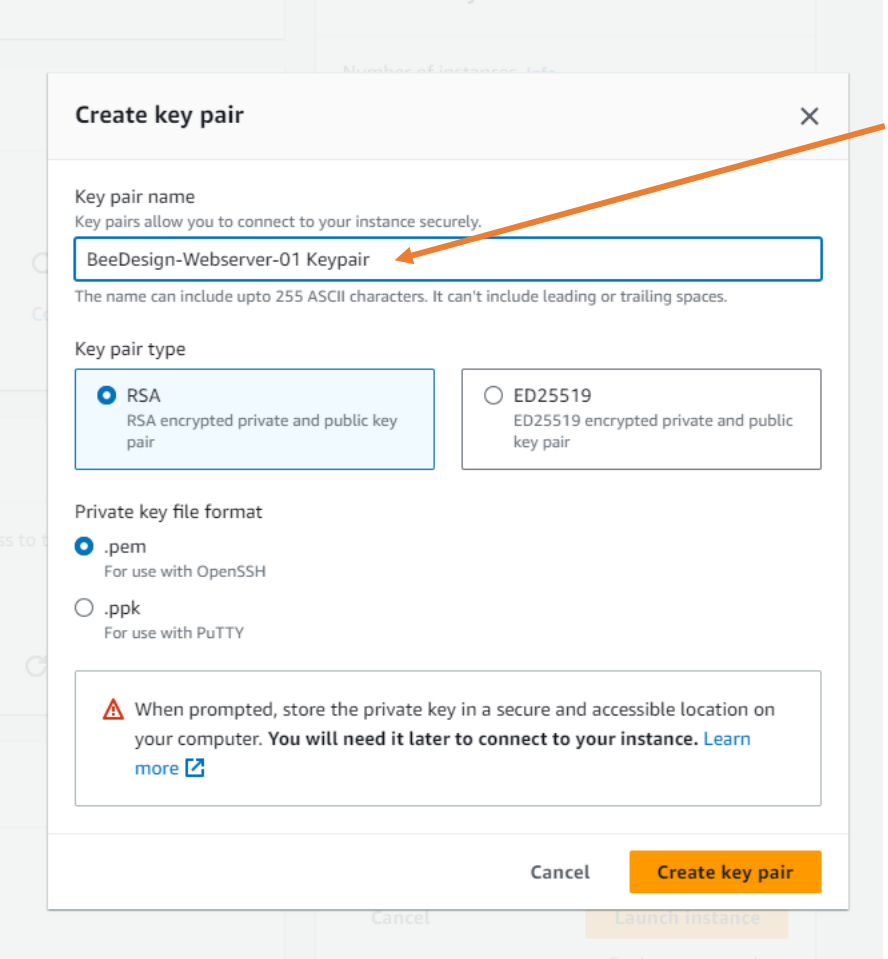
- The key pair consists of a public key on the instance and a private key on your local machine. This method is more secure than passwords, encrypts communication, and prevents unauthorized access.
- It integrates with IAM for access control, supports key rotation, and aligns with compliance requirements.

Step 9: Press create new keypair button.



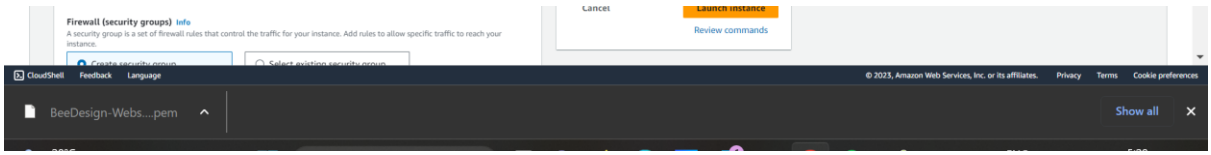
Step 10: Name key pair as **BeeDesign-Webserver-01** and leave the other settings as default.

Step 11: Press **“create key pair”.**

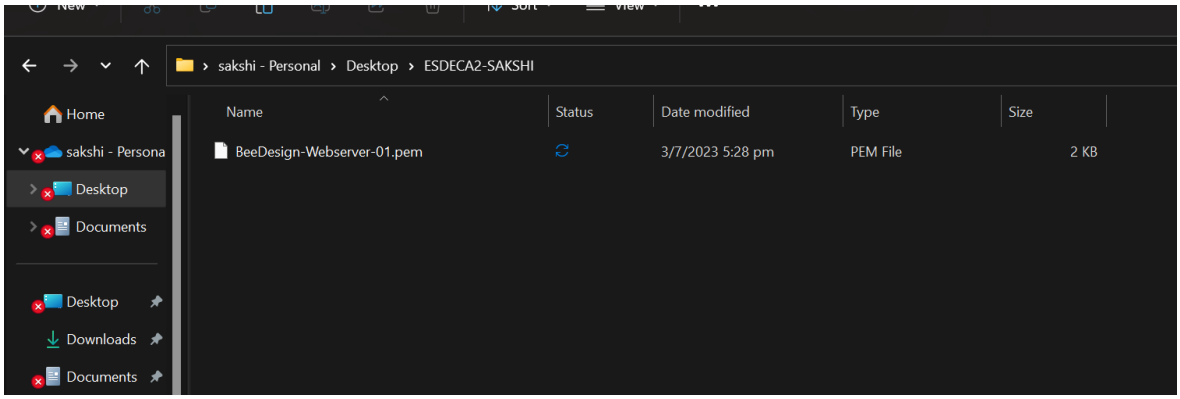


Step 12: **The .pem file** will be automatically downloaded.

The .pem file contains the private key portion of the key pair. You will use this private key to **authenticate yourself when connecting to your EC2 instance** using the SSH (Secure Shell) protocol.

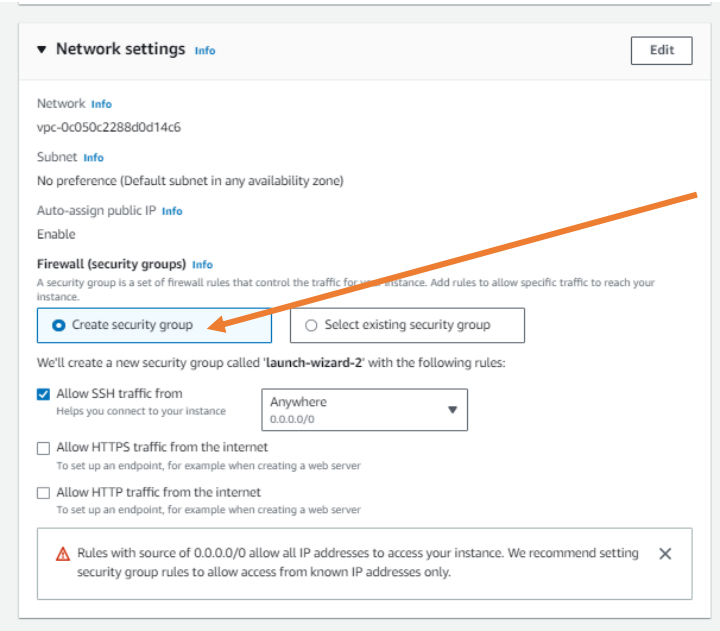


Step 13: Save the .pem file.

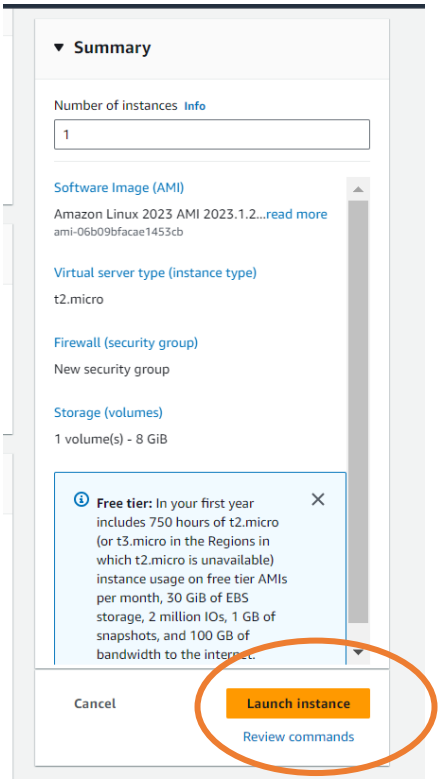


Network Settings

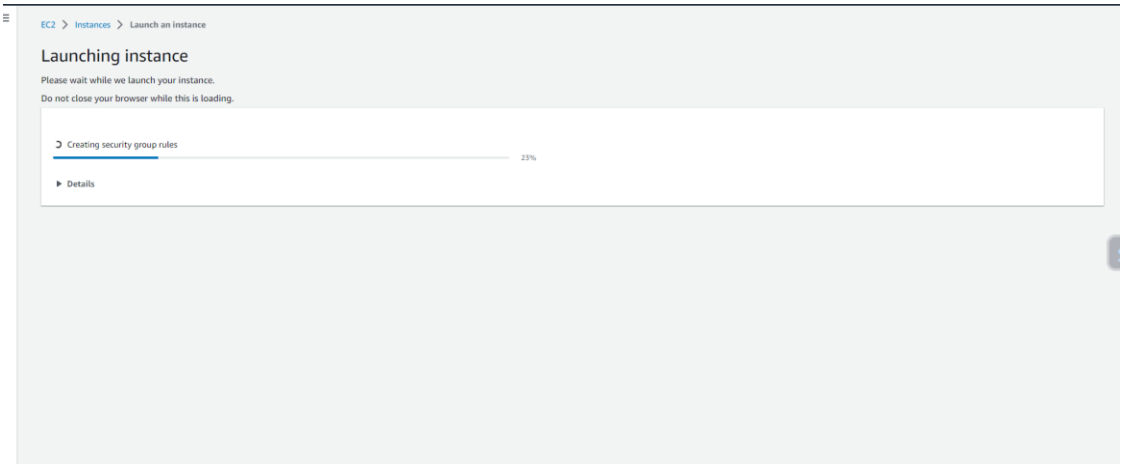
Step 10: Enable Auto-Ip to **Ensure security group allows SSH**.



Step 11: **Click launch instance** once all the settings are configured.

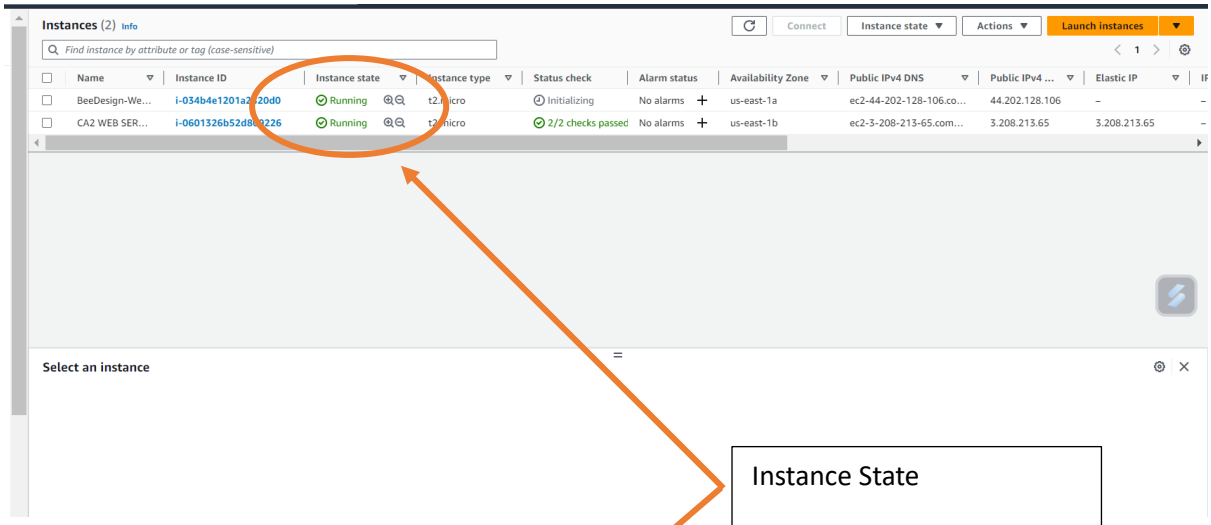


Step 12: The instance will be created.

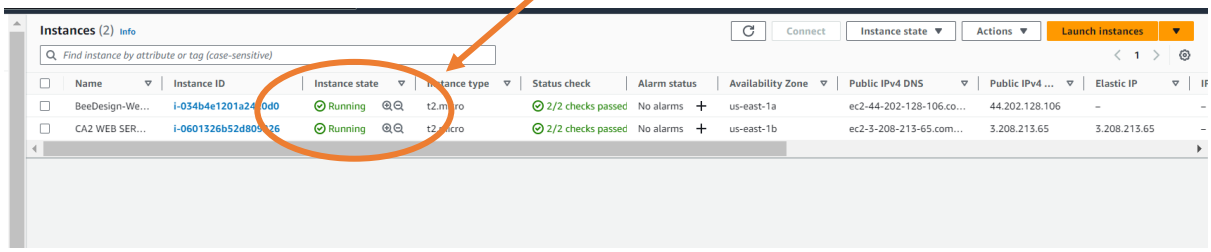


Step 13: Check that the instance has been created.

Before: The instance is **initialising** and is not fully created



After: The instance is **running** and successfully creates

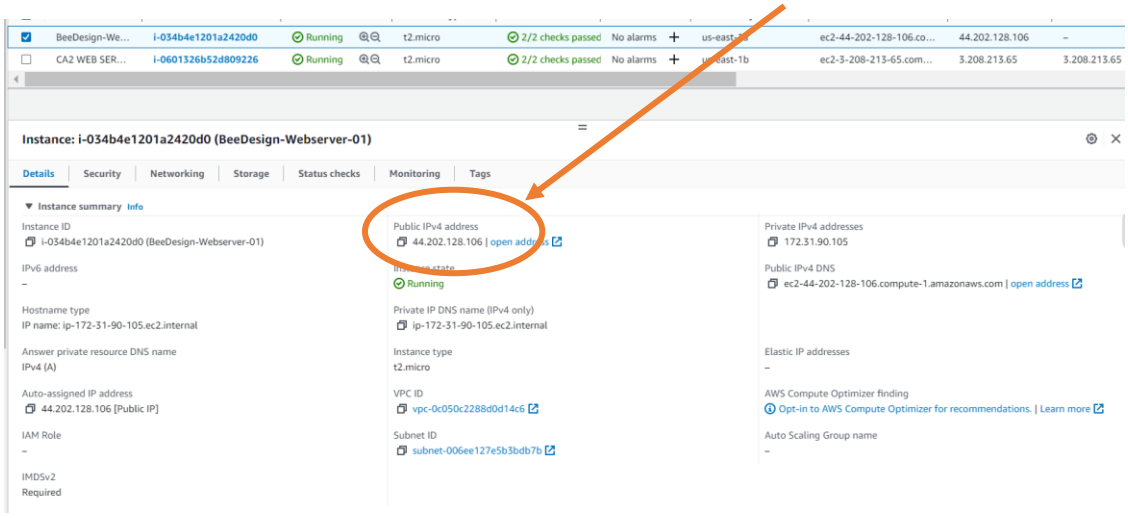


Create Elastic IP

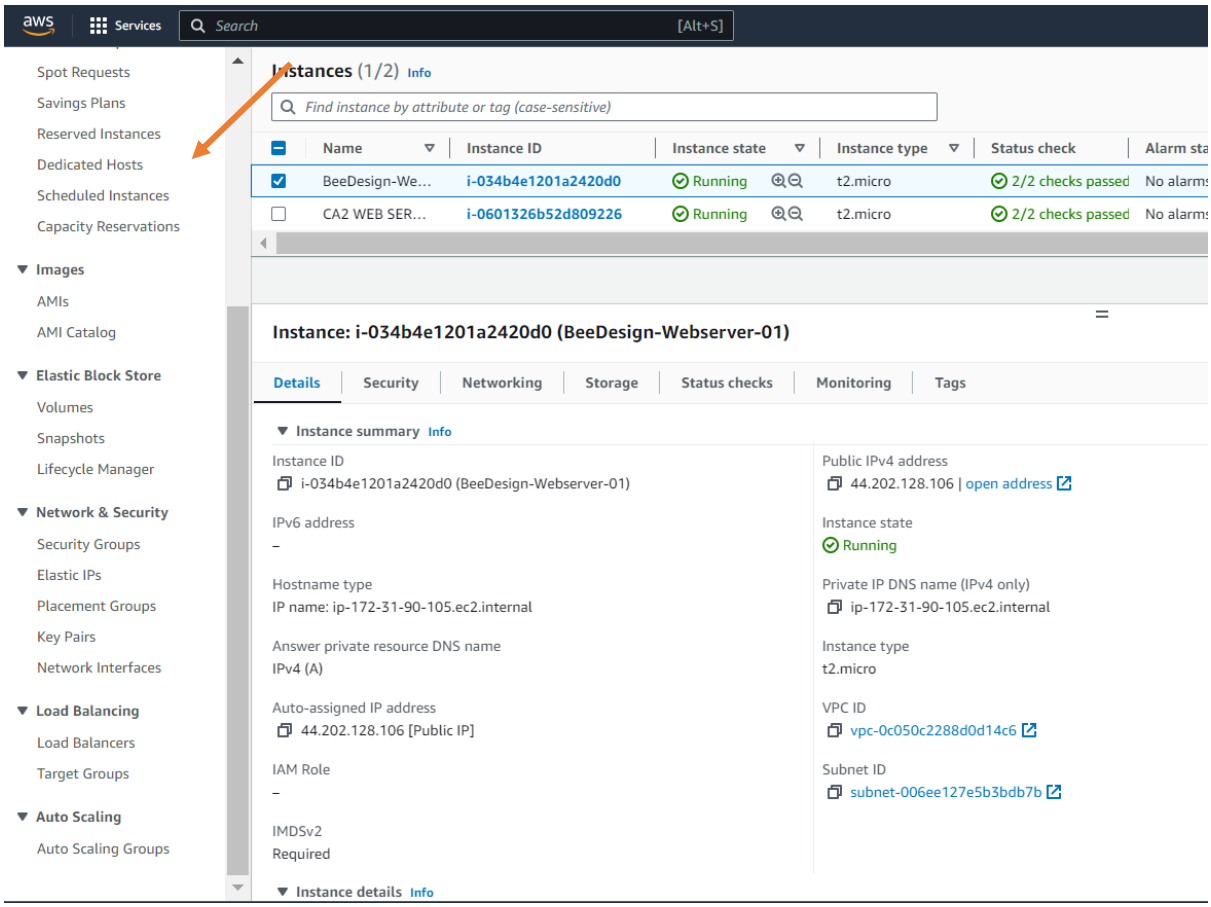
The ec2 instance is given a default public Ip address so we cannot use this Ip address as **it will keep changing** when we log into aws. This can be **problematic** for hosting a website, as **users won't reliably know the IP address to access your site. Hence we will create a elastic IP address so that**

As u can see the **public Ip now is 44.202.128.106**

Public Ip

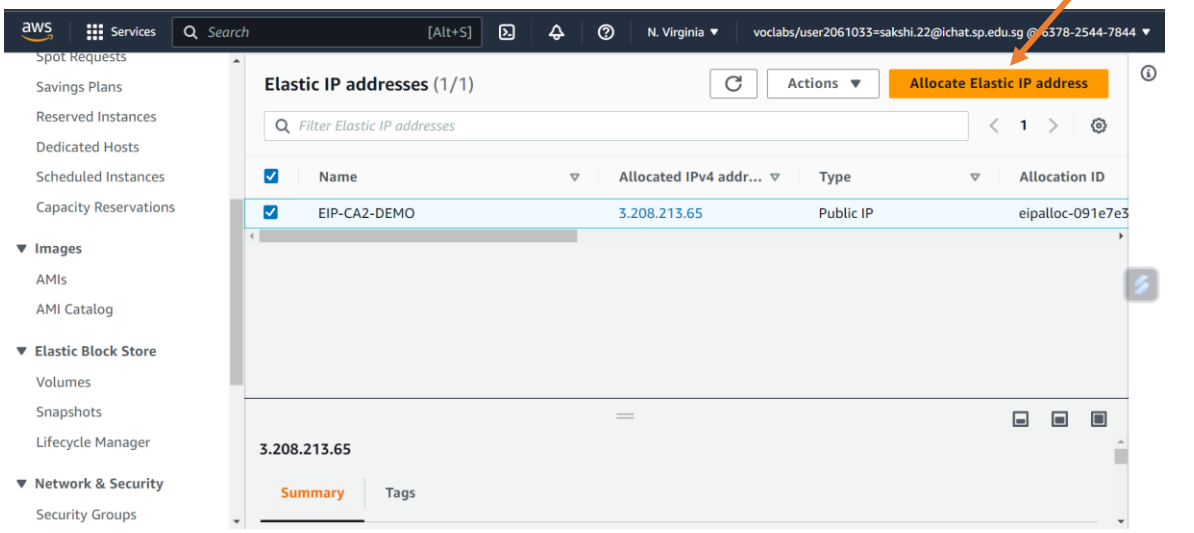


Step 14: Go the left navigation bar.



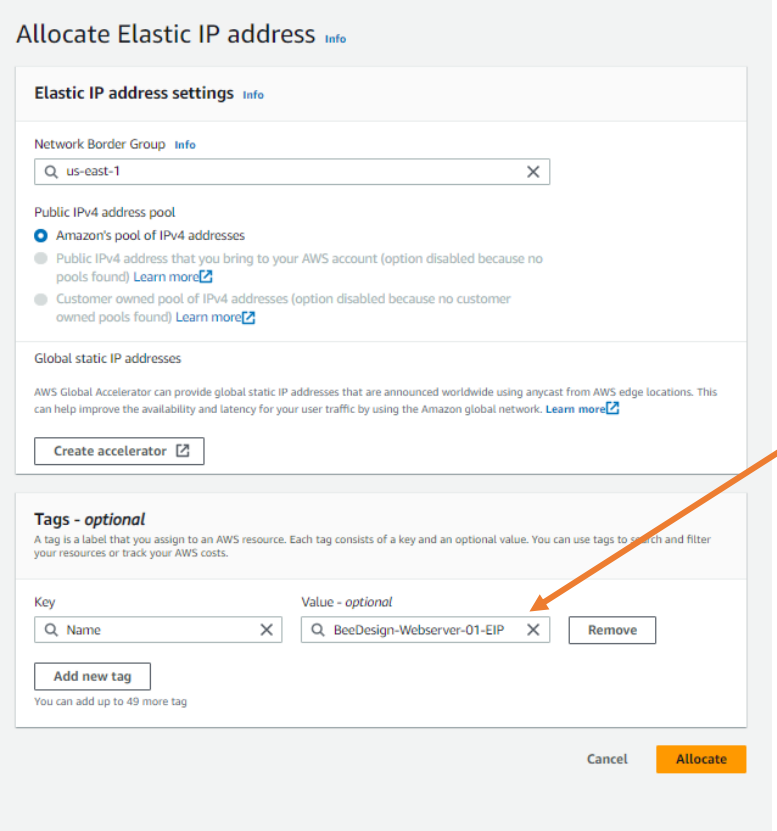
Step 15: Locate **Network and Security**

Step 16 : You will be directed to this page and press **“Allocate Elastic IP address” button.**



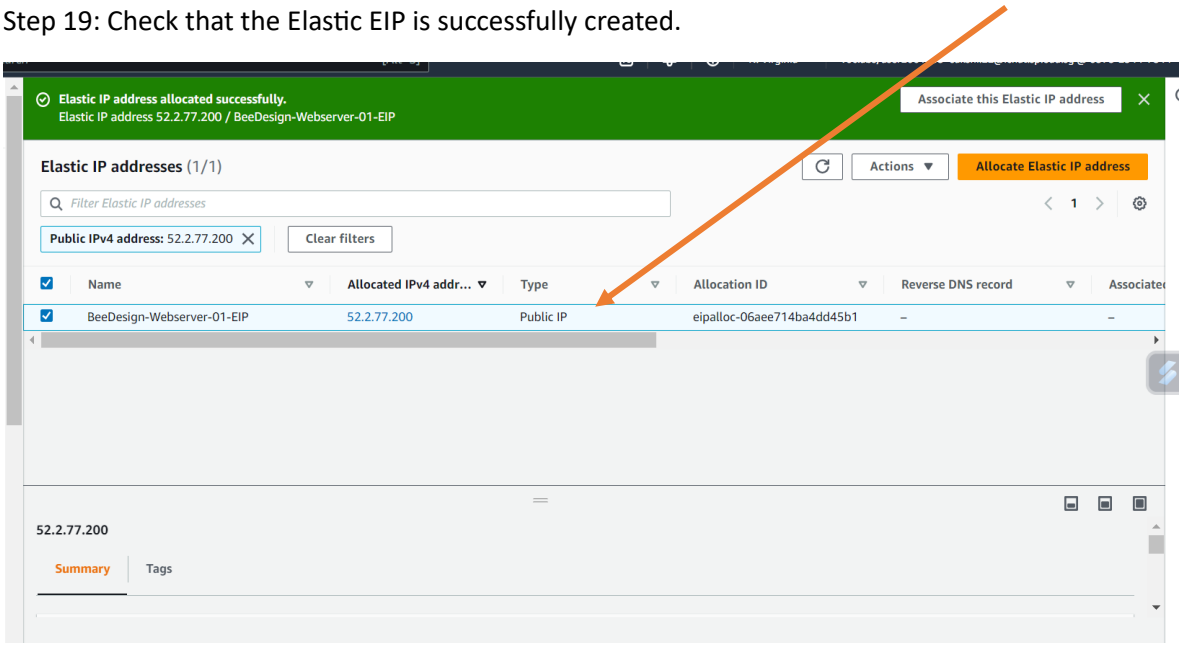
Step 17: You will be **directed to this page** to configure the settings of the EIP, **leave the settings at default** and **add a name tag** to identify to **identify the EIP address and click the allocate the items**.

Step 18: Click “**Allocate”** and now an elastic Ip address will be allocated to u as shown.



Name Tag for the EIP

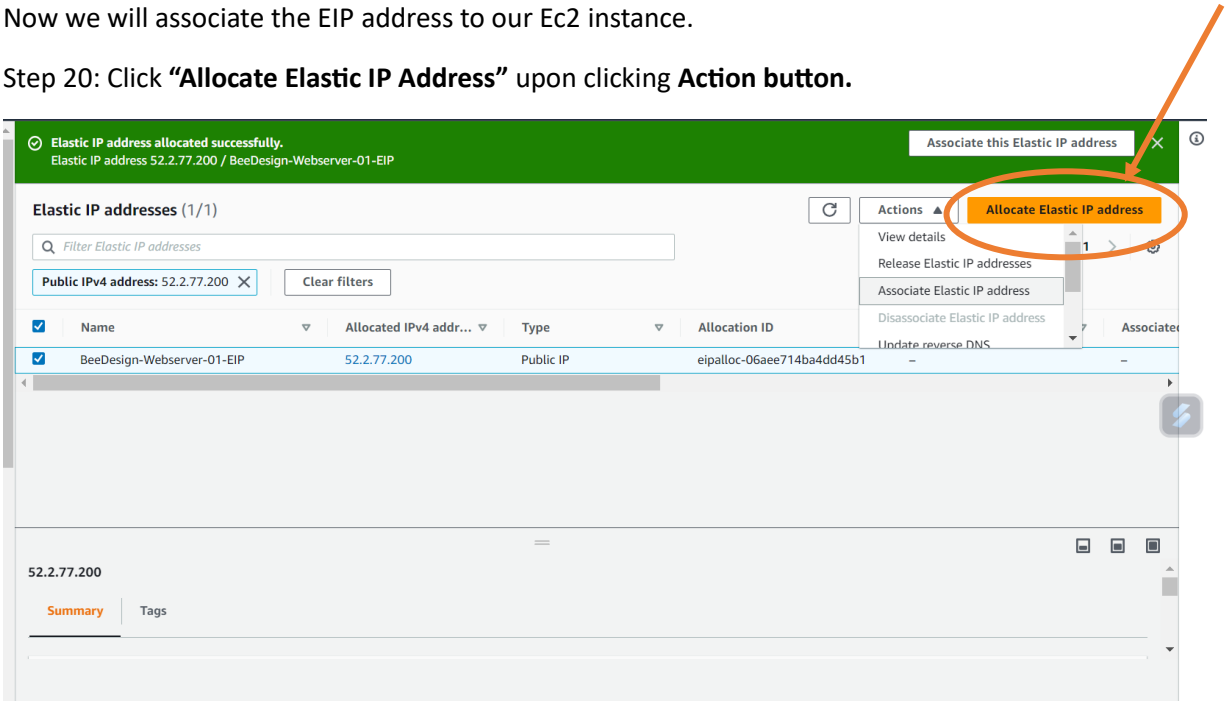
Step 19: Check that the Elastic EIP is successfully created.



Associate EIP with instance

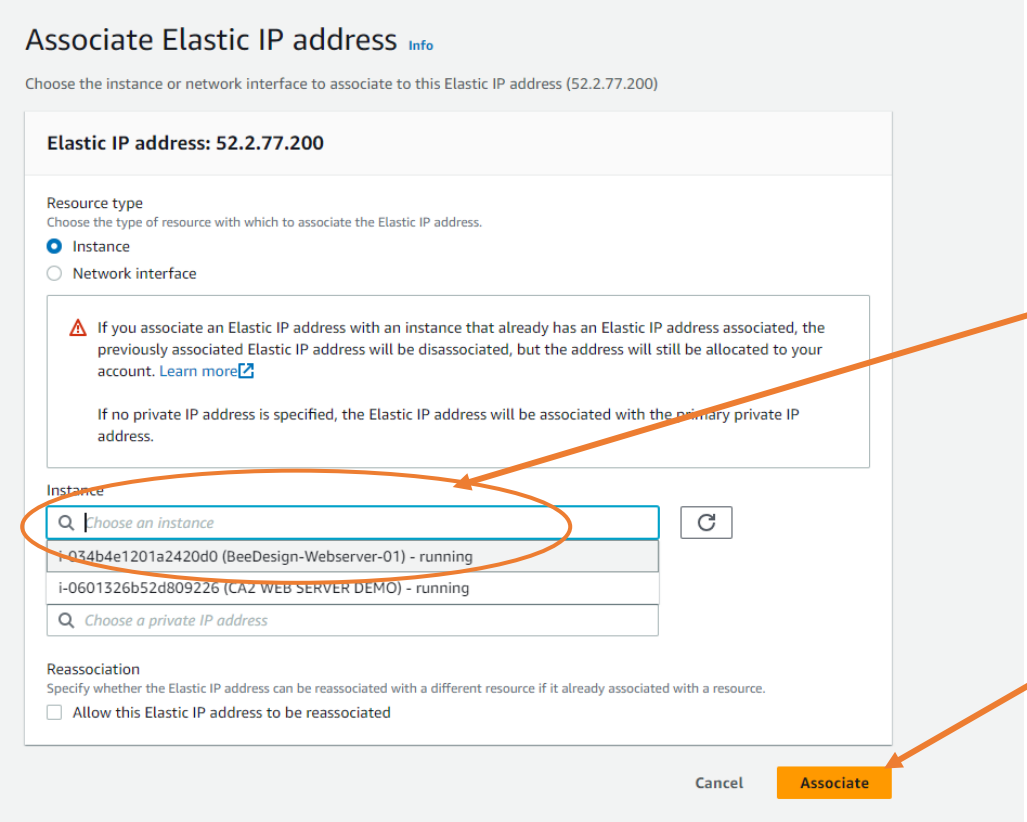
Now we will associate the EIP address to our Ec2 instance.

Step 20: Click **"Allocate Elastic IP Address"** upon clicking **Action button.**

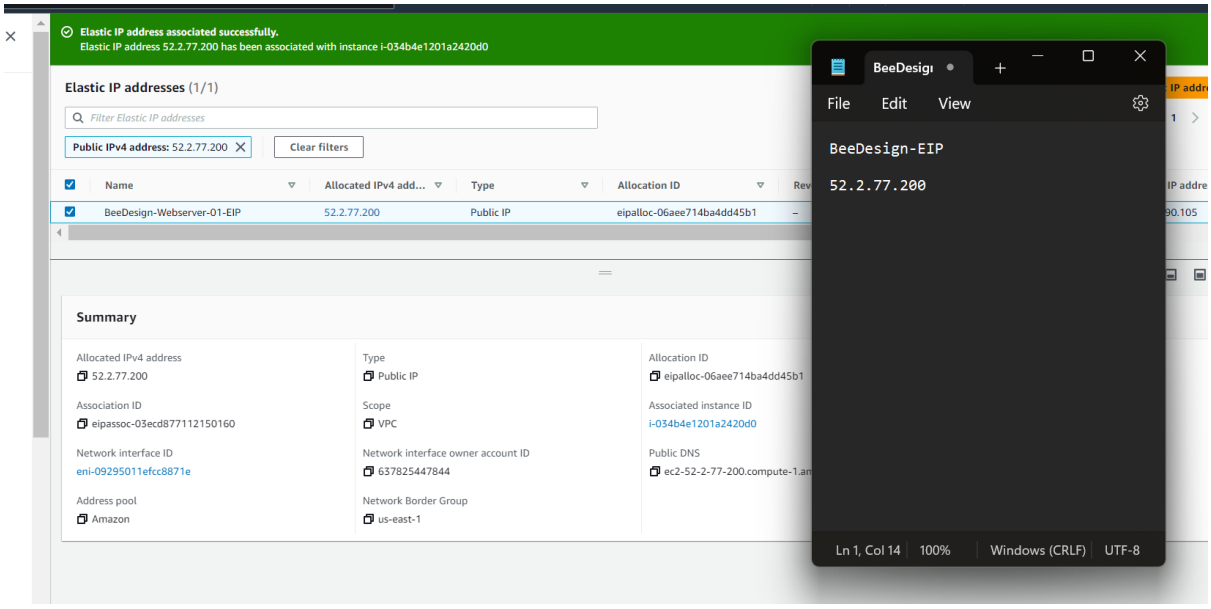


Step 21: **Choose the ec2 instance** in the dropdown menu and **click associate**.

Our instance is **BeeDesign-Webserver-01**

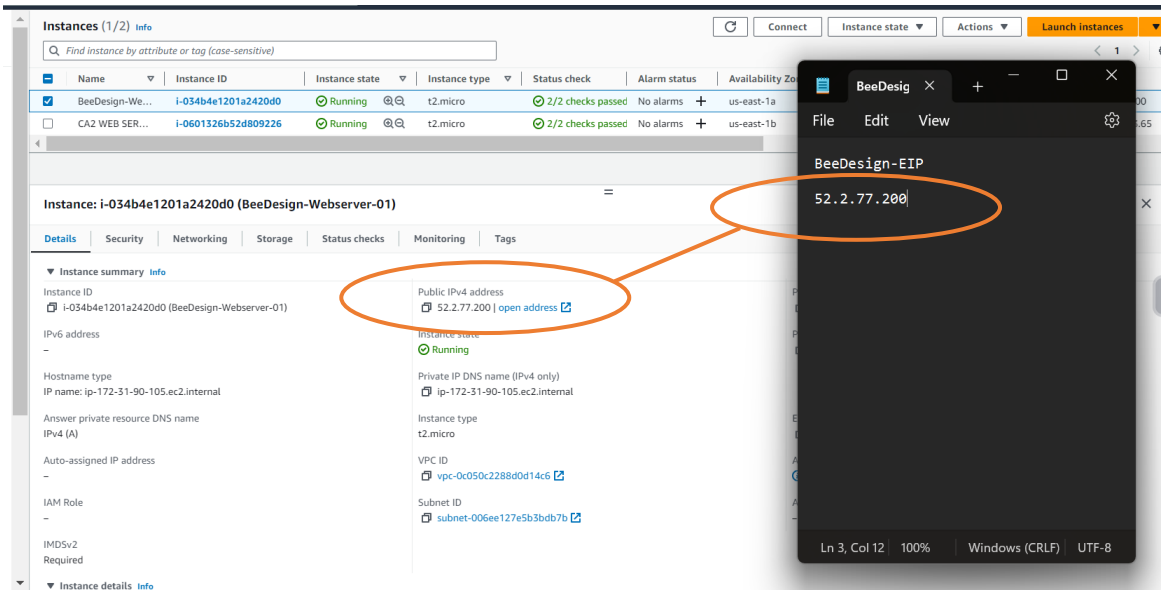


Step 22: Take note of your EIP address.



Step 23: Go back to you **EC2 instance**, **press the refresh button** and check the **public IP address** again, it will show you the **updated EIP address**.

Your instance now uses a EIP address that will remain the same every time you log into aws.

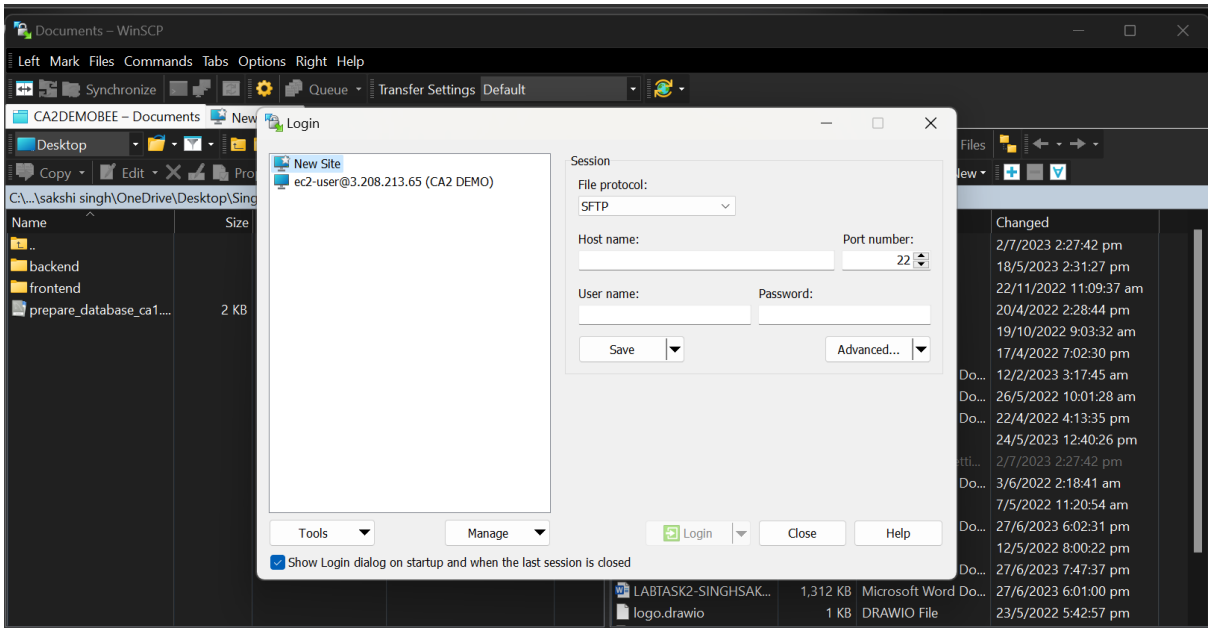


Connect EIP to EC2 instance on WinSCP.

Download software.

Step 24: Download **Putty** and **WinSCP**.

Step 25: Open WinSCP, it will show you a login screen.

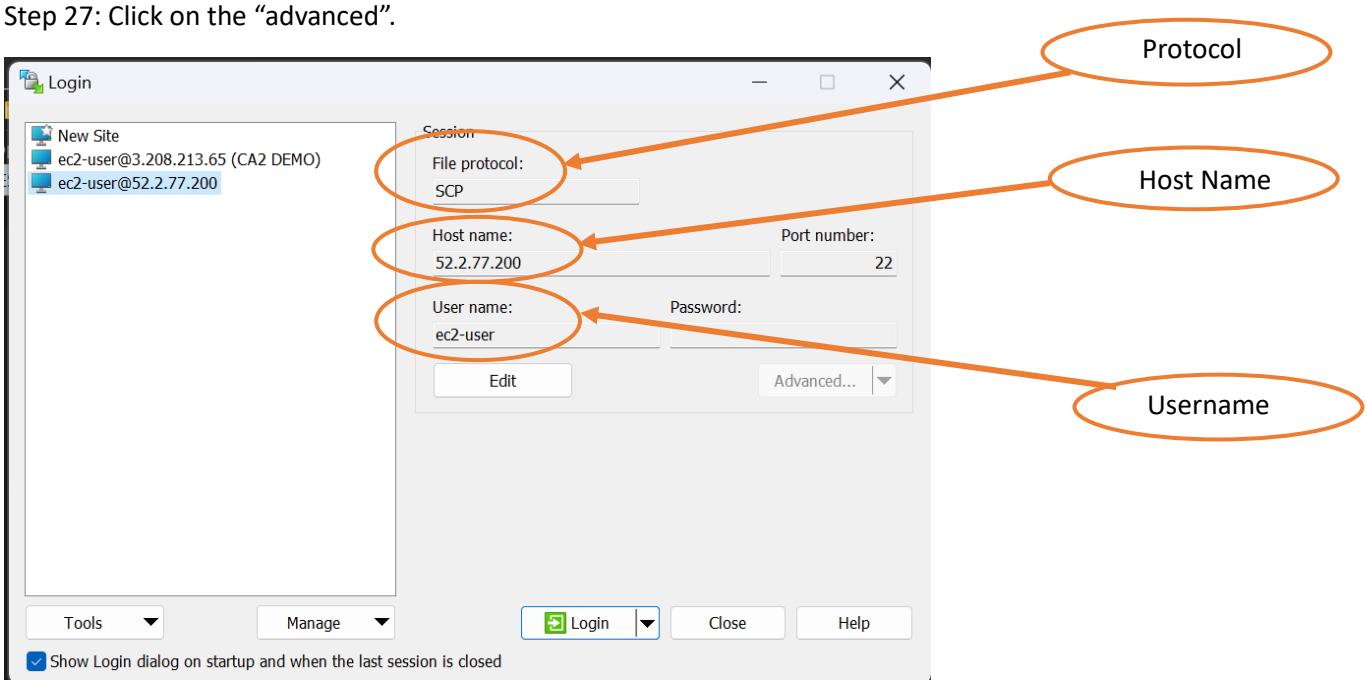


Configure Settings

Step 26: Now we must set the settings.

1. Set the **protocol** to SCP.
2. The **hostname** is the EIP address 52.2.77.200
3. Set the **name** to ec2-user.
4. Leave the **password** empty.

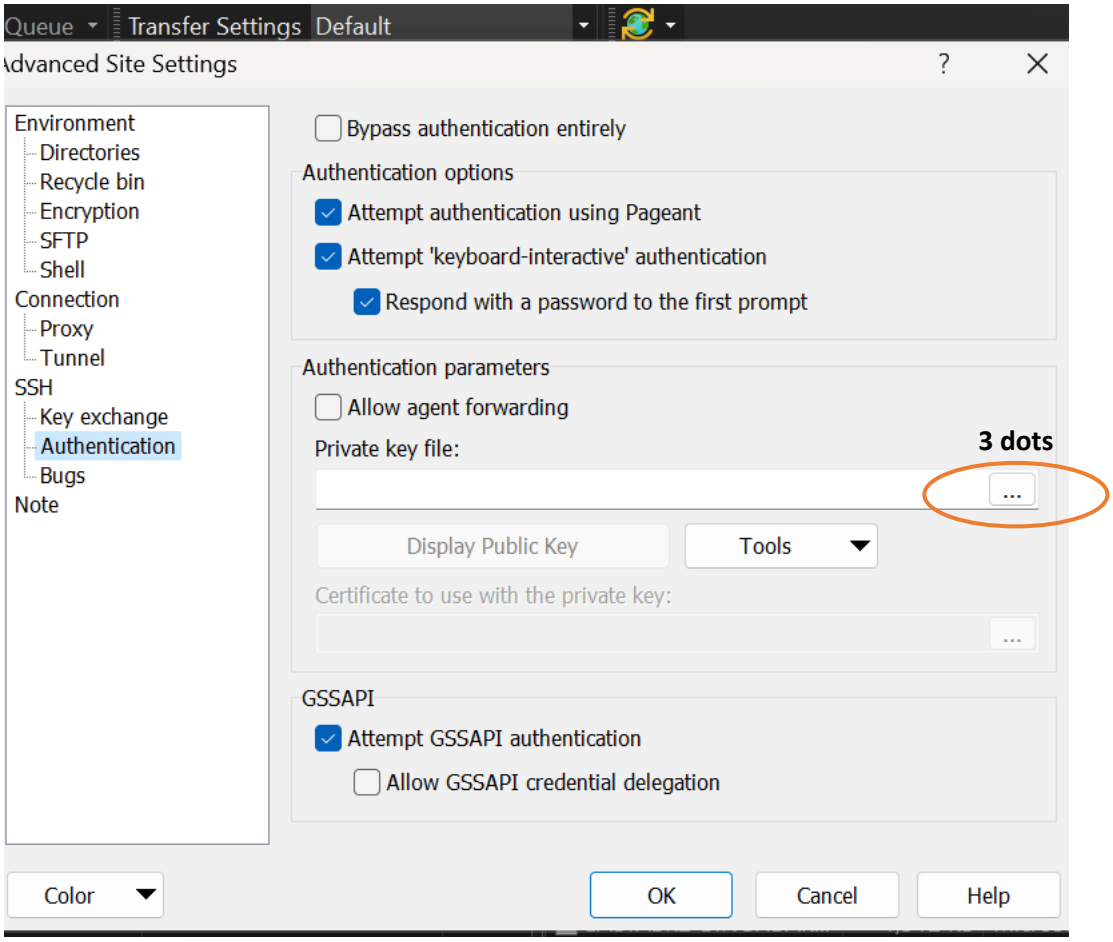
Step 27: Click on the “advanced”.



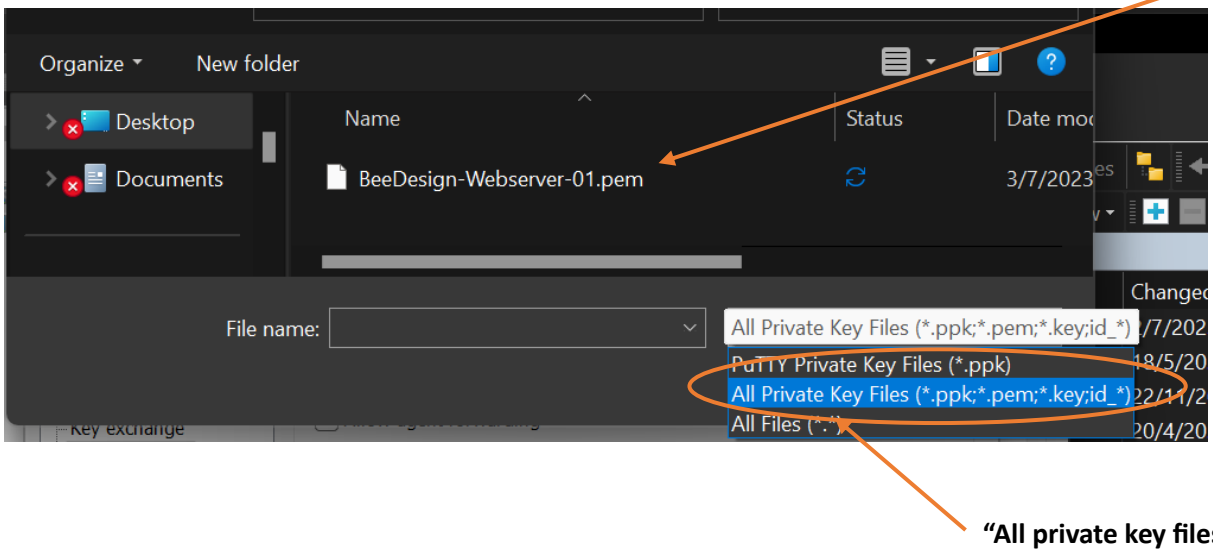
Step 28: Navigate to “Authentication”

1. Press the 3 dots.

1

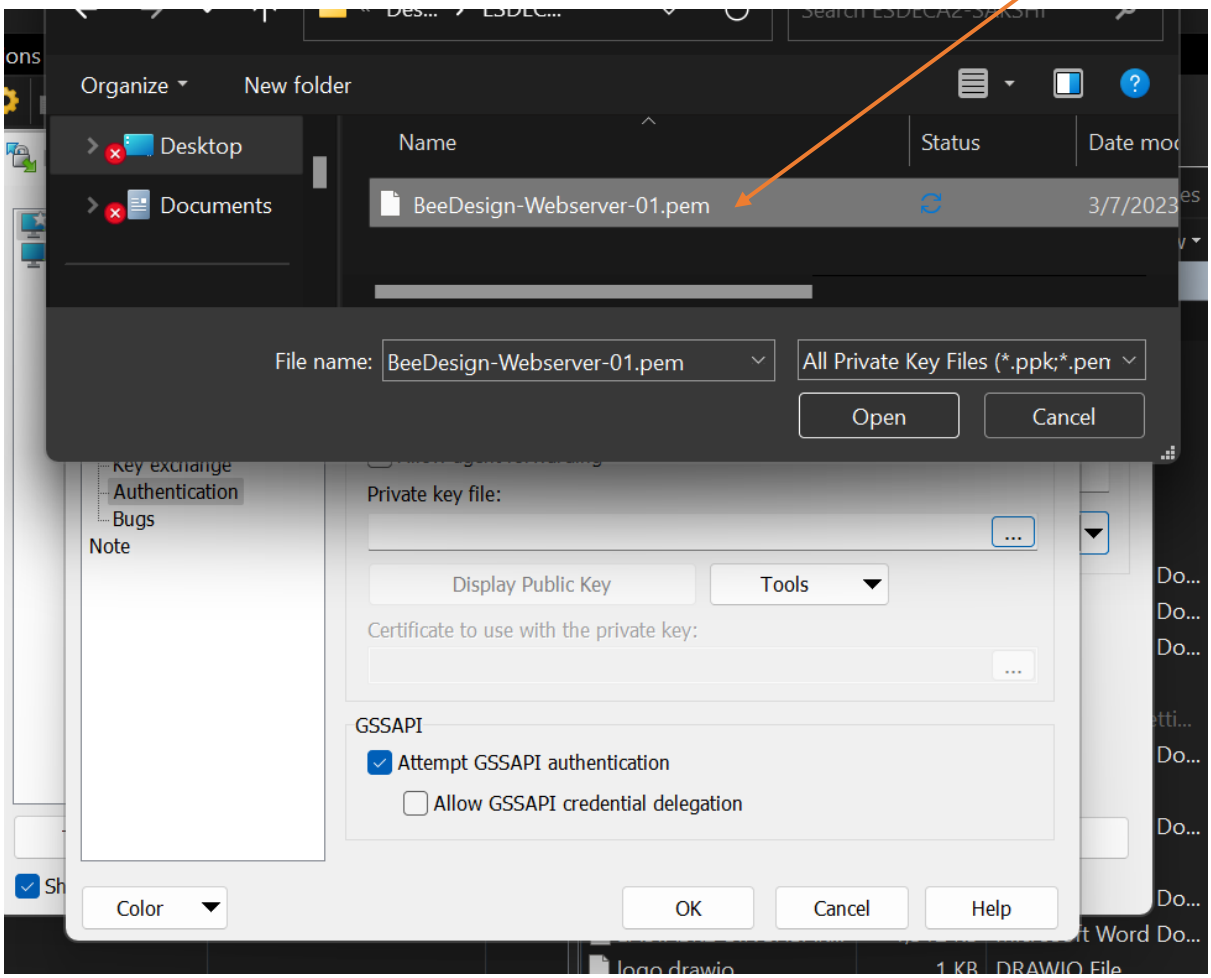


2. Your explorer page will pop up, **navigate to your Pem file**.

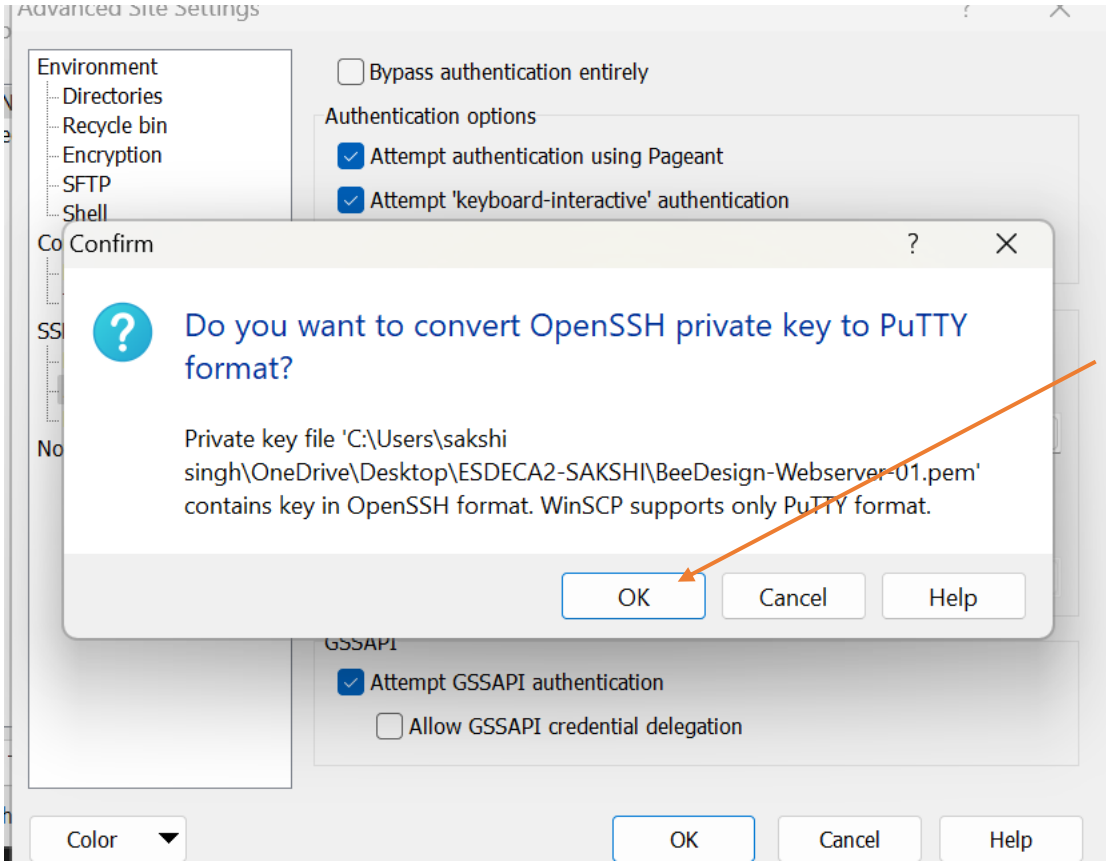


.pem file

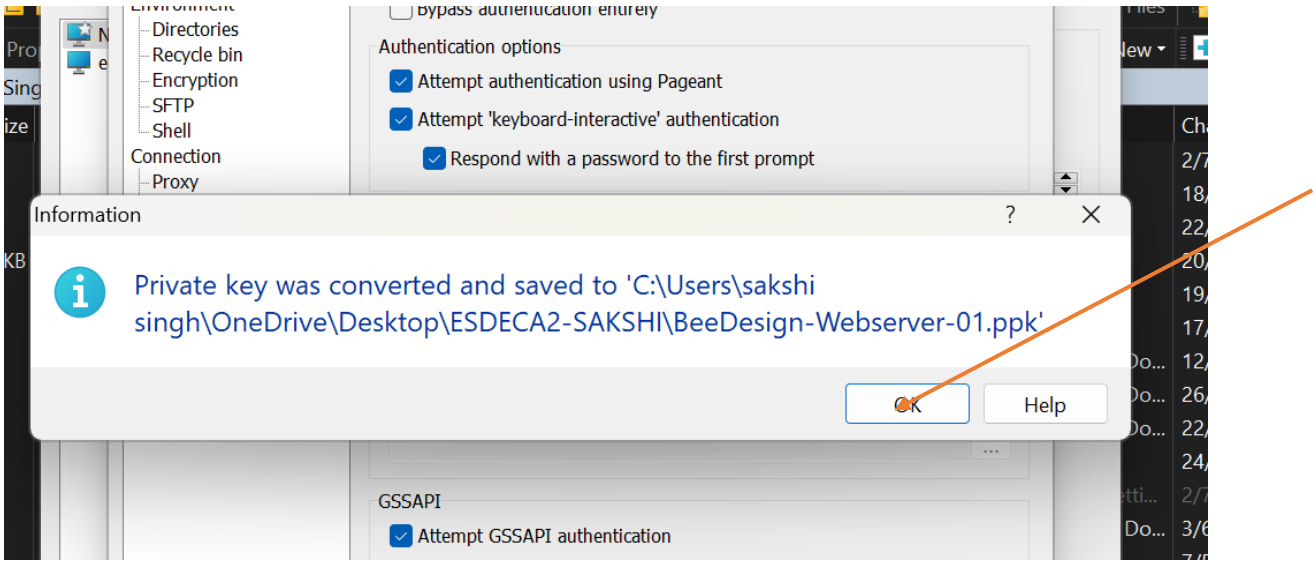
2. Change the type, to **“all private key files”.**



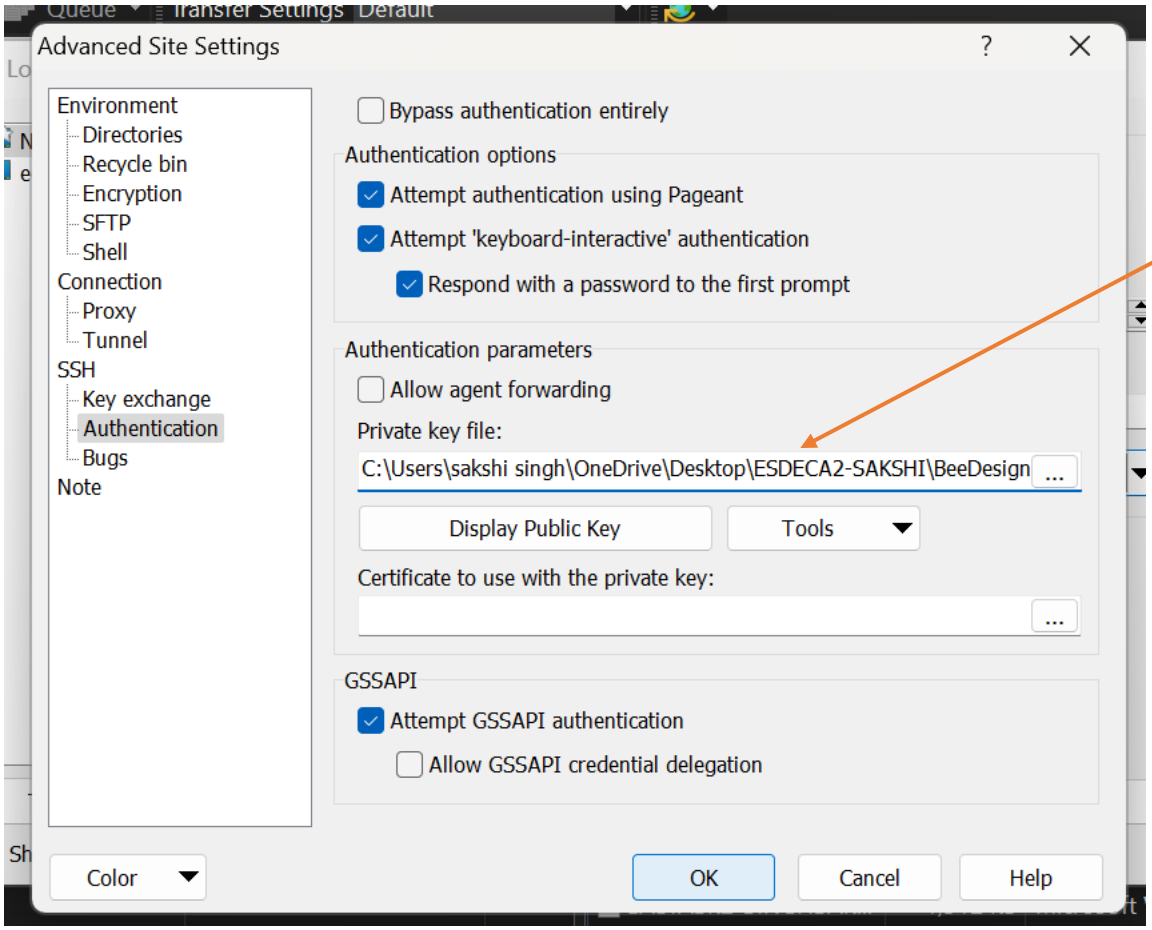
4.Upload the .pem file.



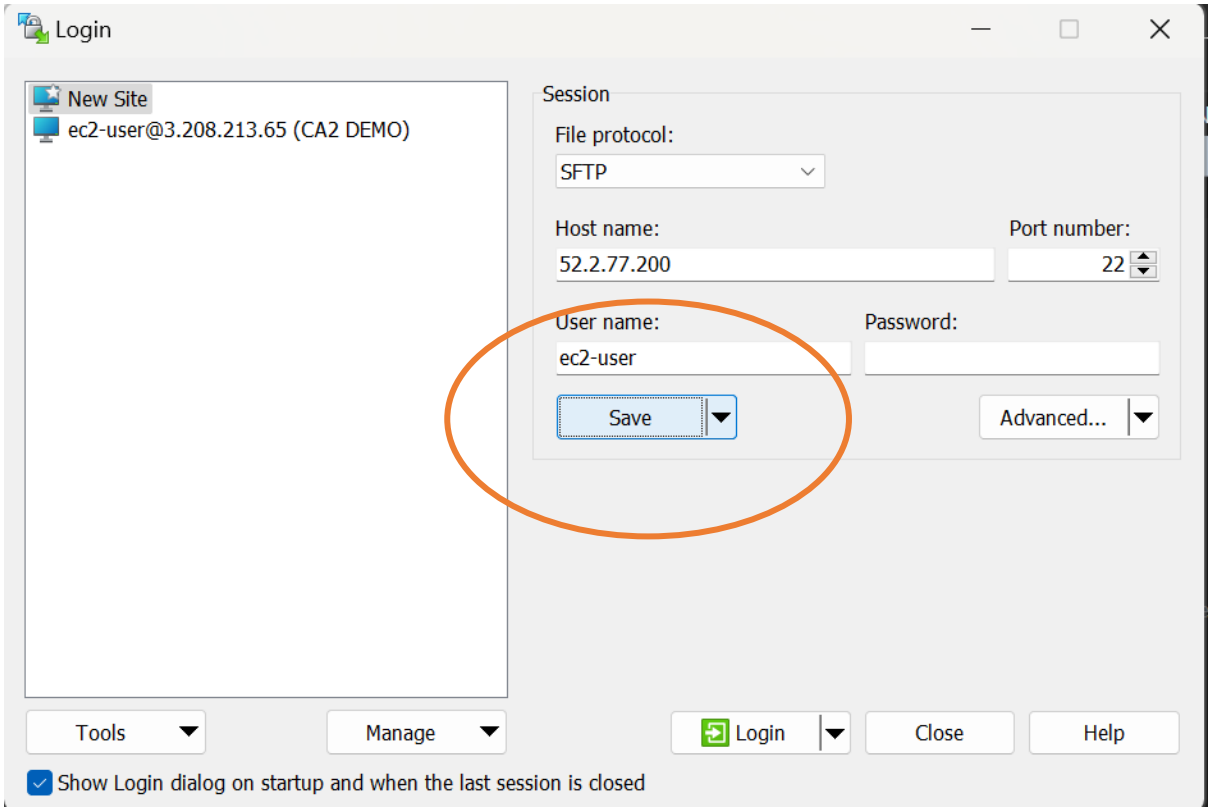
Step 29: Click **“ok”.**



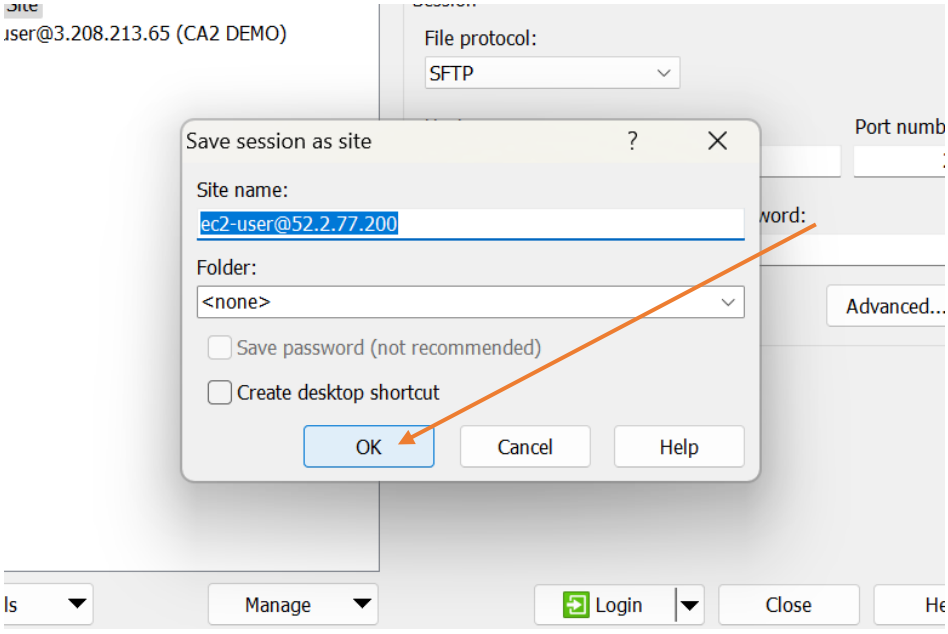
This is how the final settings look like.



Step 29: Click the **“Save” button** to save your configuration settings.

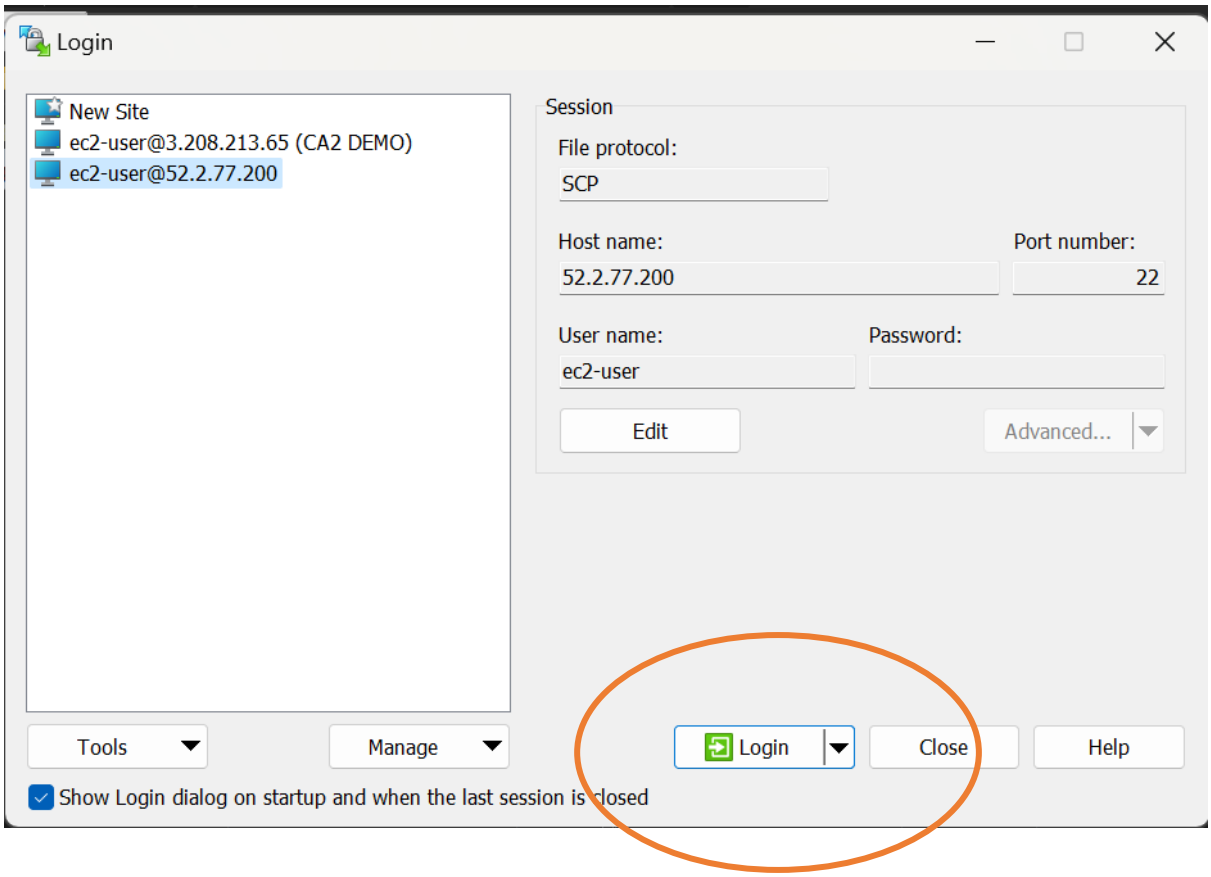


Step 30: Press **‘ok’** to confirmation of the new site name with our EIP address.

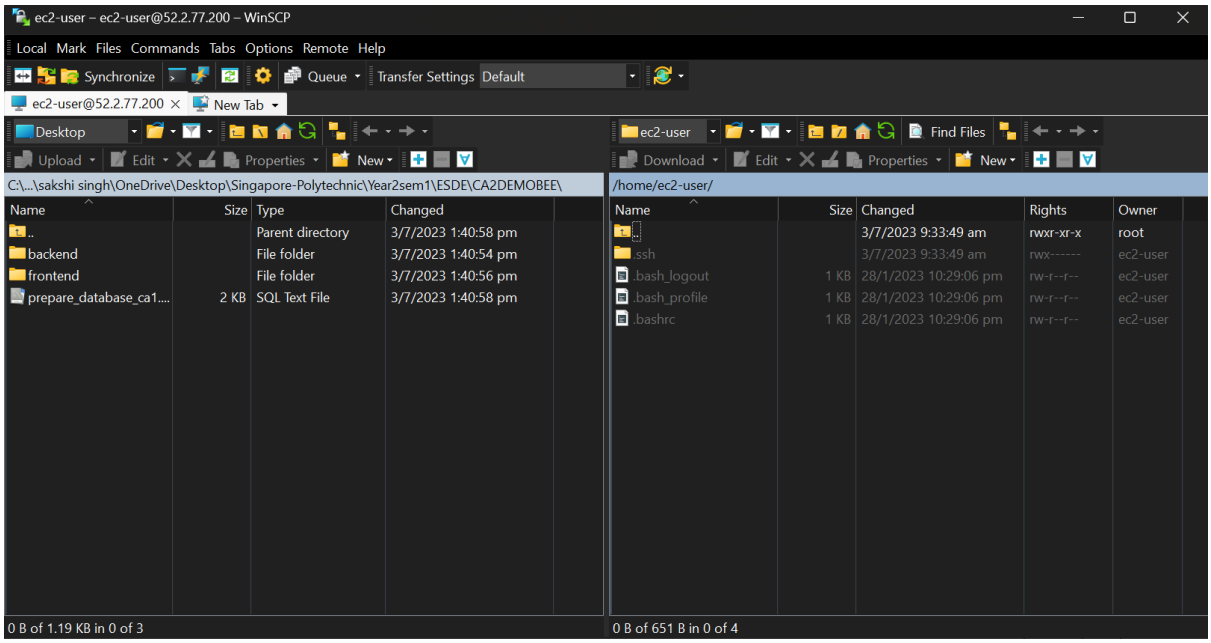


Step 31: Check that the new site name with the username and hostname is created.

Step 32: Press “Login”



Stress 33: If you reach this page, you have successfully logged in.

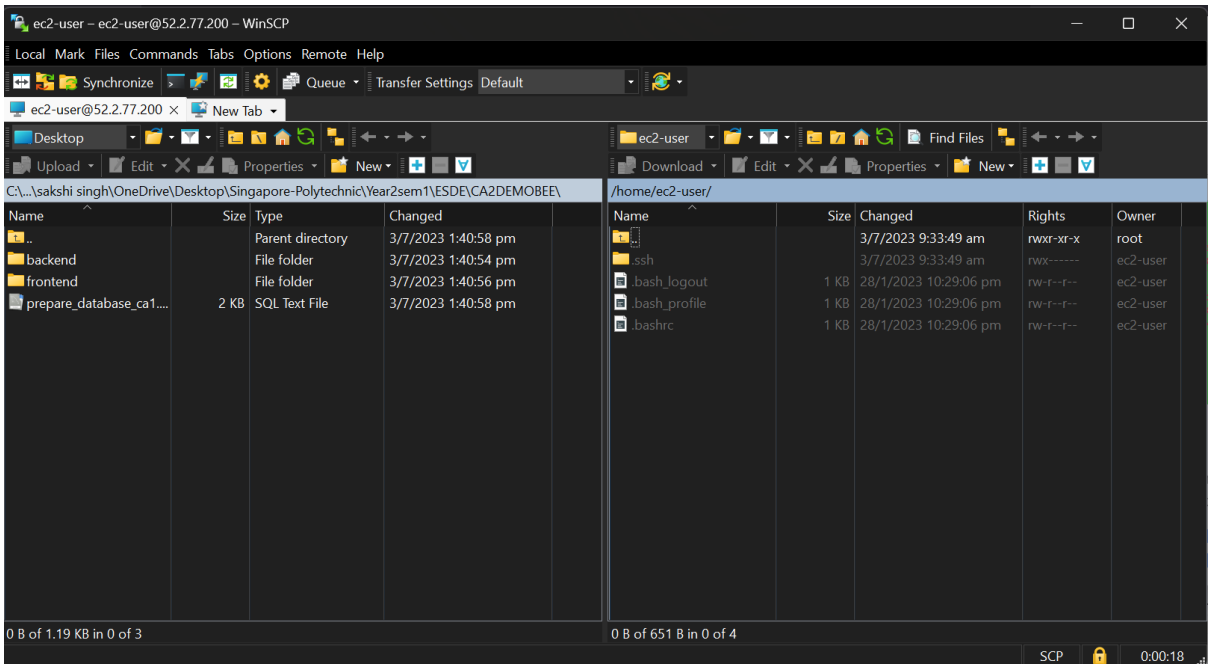


Part 2- Install NodeJS and copy Bee Design app to EC2.

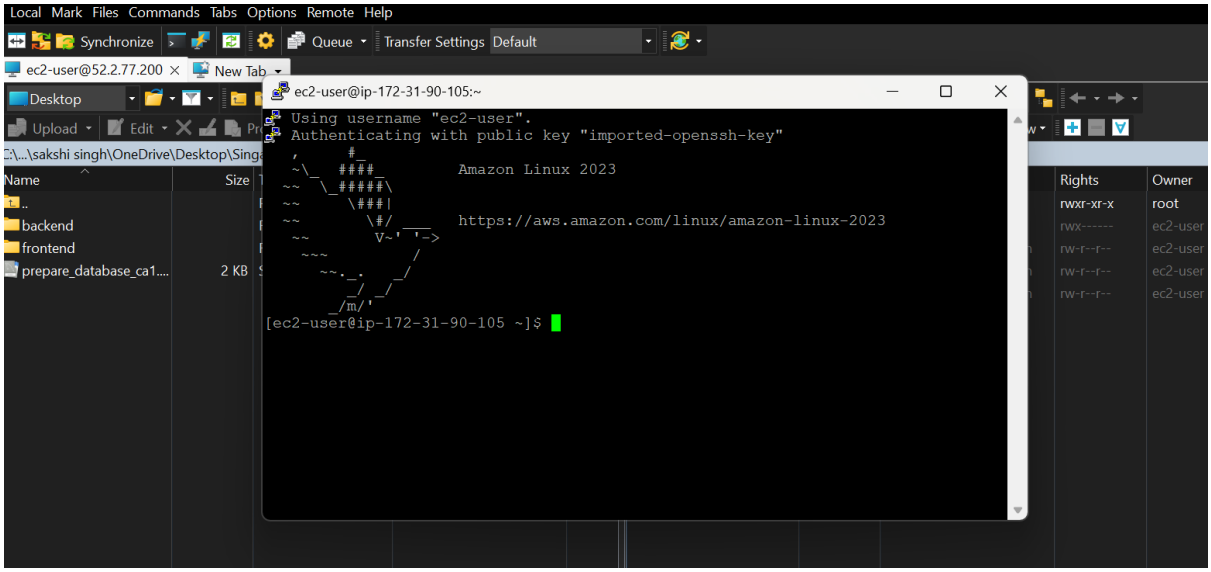
Now that you have set up your ec2 instance, we will be **installing node js** , **deploying the bee design application** and **configuring security settings** to allow the app to be accessed.

Installing Node js

Step 1: Go to WinSCP



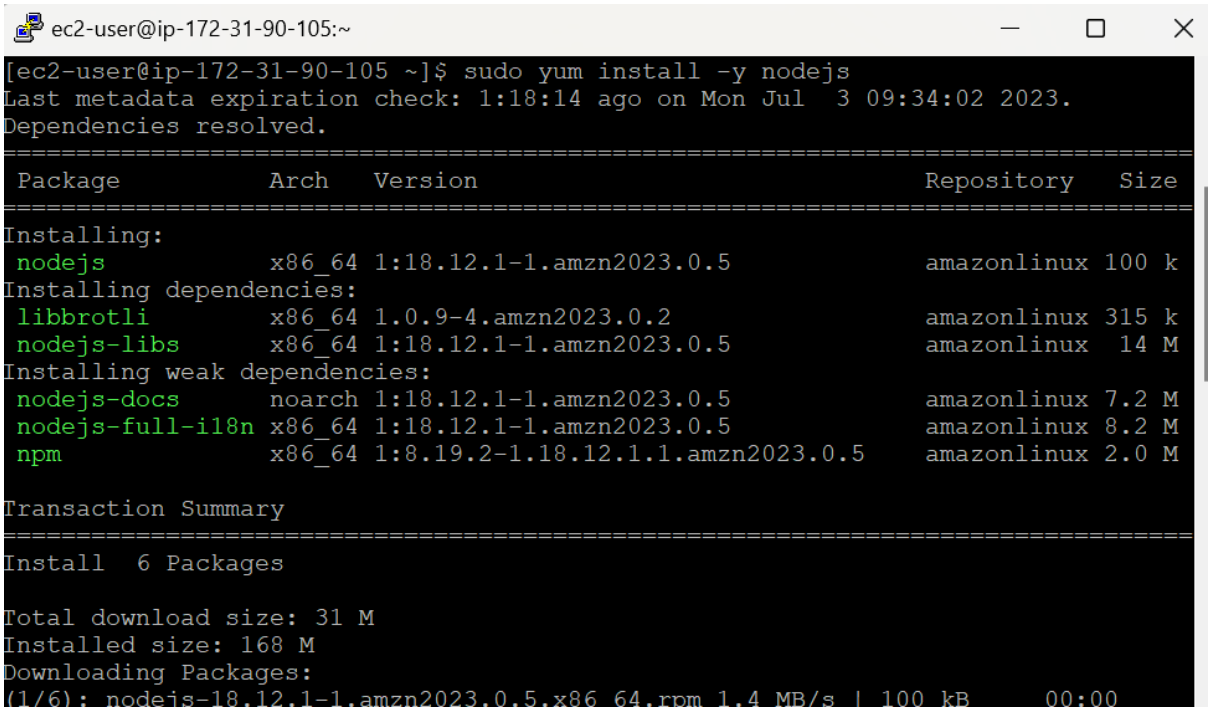
Step 2: Open Putty Terminal



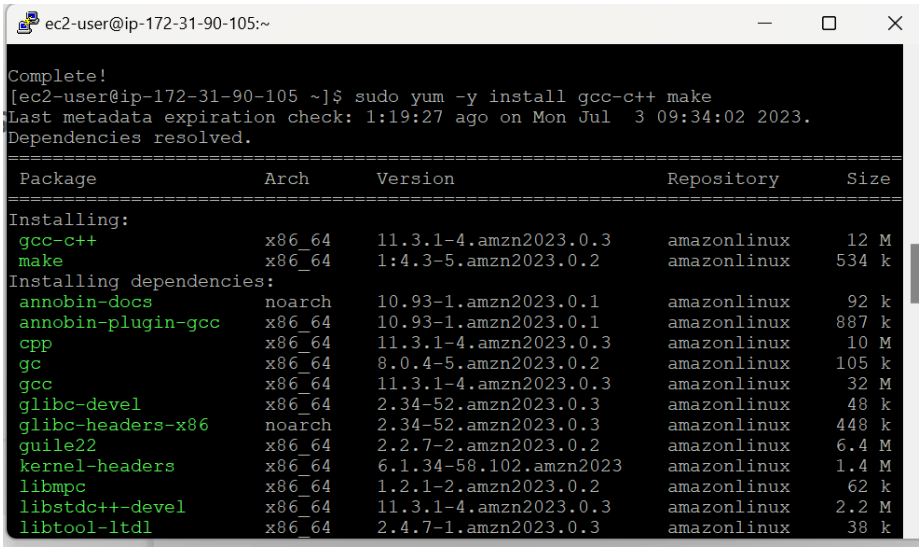
Step 3: Run these following commands

- sudo yum install -y nodejs (Install node js)
- sudo yum -y install gcc-c++ make (install necessary packages)
- node --version (check node version)

Run sudo yum install -y nodejs



Run sudo yum -y install gcc-c++ make



Run node --version

Deploy Bee Design

Step 4: Locate Index.js

```
app.use(serveStatic(__dirname+ /public ));


app.get("/", (req, res) => {
    res.sendFile("/public/home.html", { root: __dirname });
});


app.listen(port,hostname,function(){

    console.log(`Server hosted at http://${hostname}:${port}`);
});
```

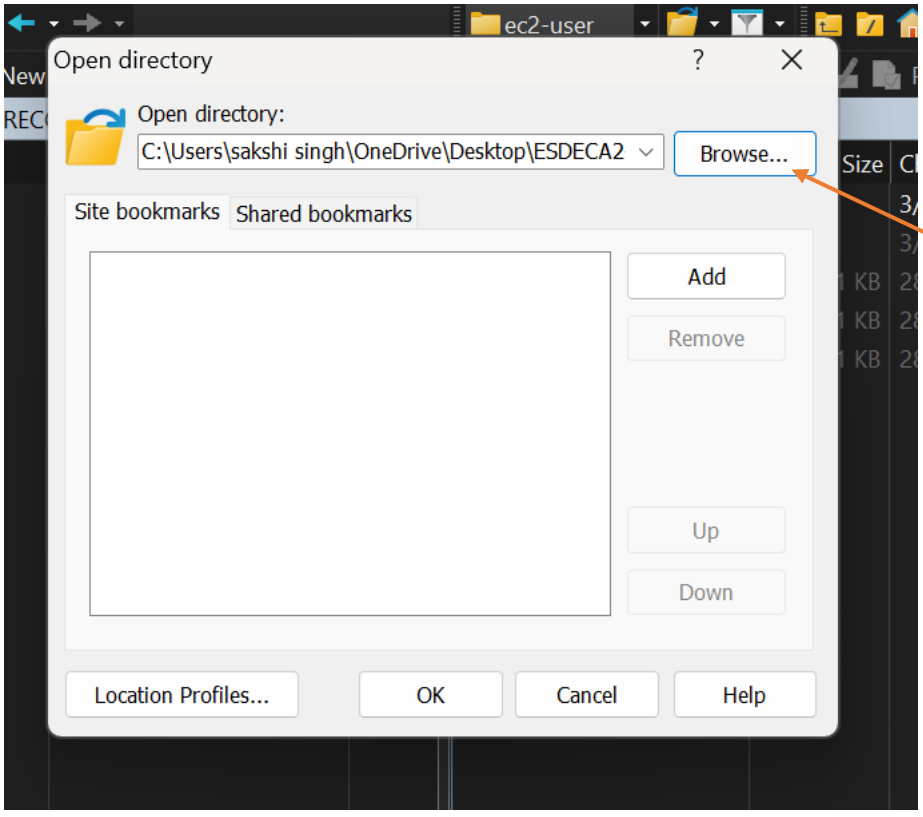
Step 5: Remove hostname

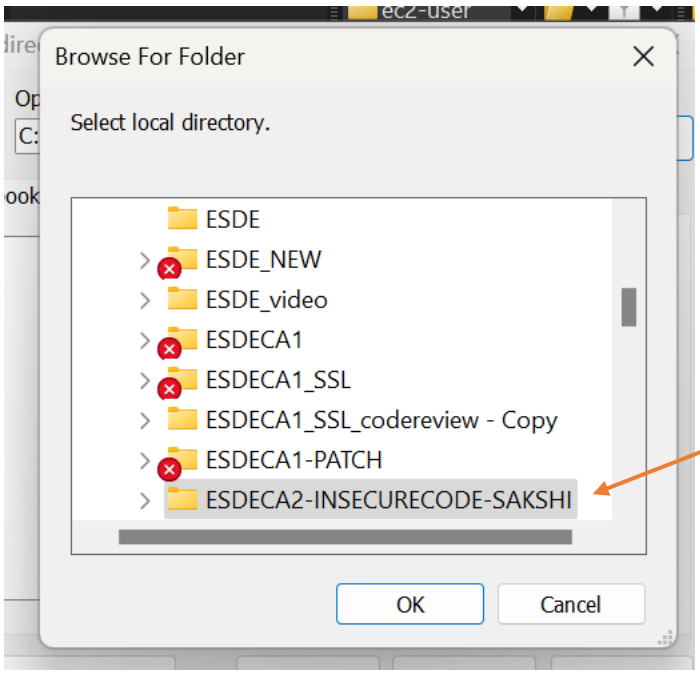
```
app.listen(port,function(){

    console.log(`Server hosted at http://${hostname}:${port}`);
});
```

Step 6 : Locate .env and **add the cloudinary and mailtrap ids** so as to ensure that your Bee Design we website can **successfully request information** from these domains.

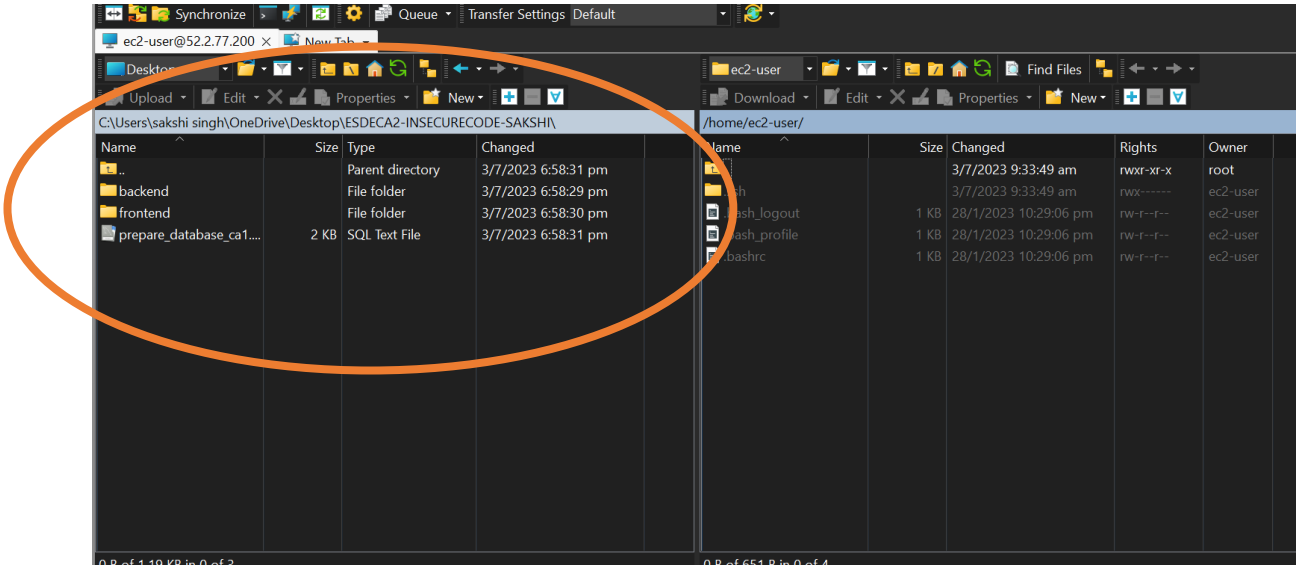
```
backend > .env
1   DB_USERNAME=esde_ca_adminuser
2   DB_PASSWORD=P@ssw0rd
3   DB_DATABASE_NAME=ay2324s1_st0505_esde_ca
4   JWTKEY=QSBKV1QgY29kZSBmb3IgQVkyMjIzczEgRVNERSBzdHVkZW50cw
5
6   CLOUDINARY_CLOUD_NAME=dqmyxllr8
7   CLOUDINARY_API_KEY=678552142329395
8   CLOUDINARY_API_SECRET=XC-xIa54Ma_U4hWf8P49SD17xGk
9
10  MAILTRAP_USERNAME=dd1ce7739fab39
11  MAILTRAP_PASSWORD=90851f84f056ef
```

Step 6 : Open directory and navigate to your Bee Design Website folder

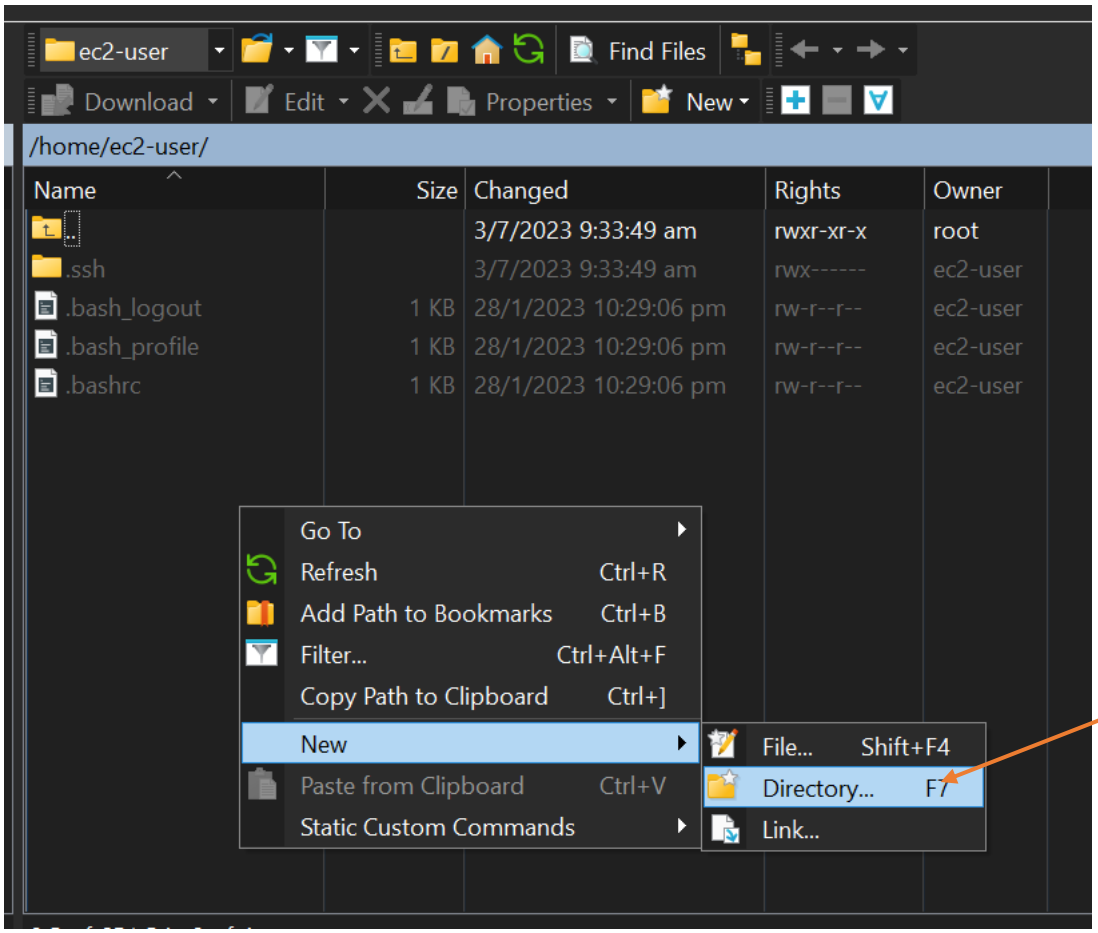


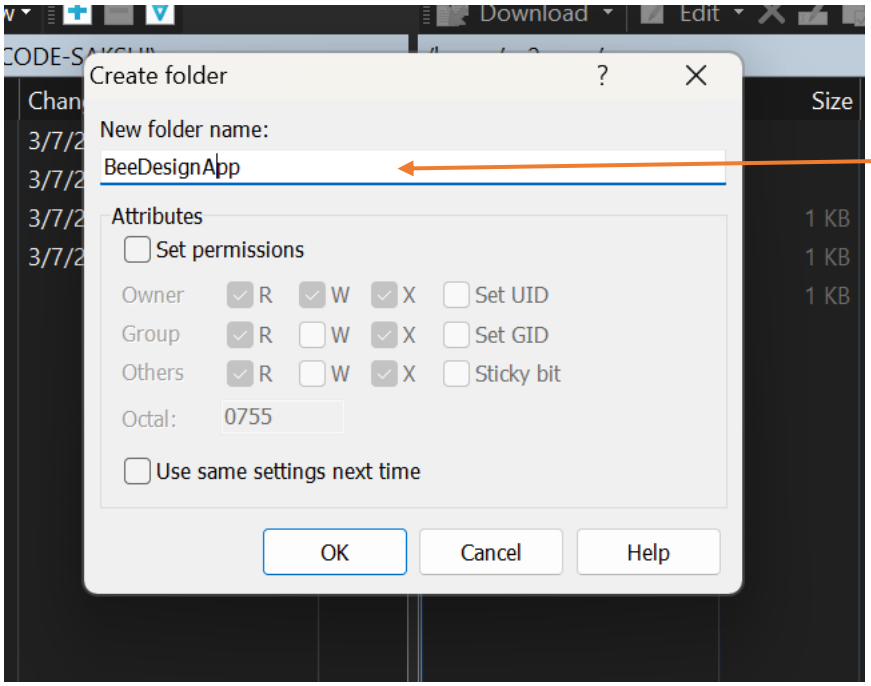


Step 8: You have successfully navigated to the Bee Design folder .

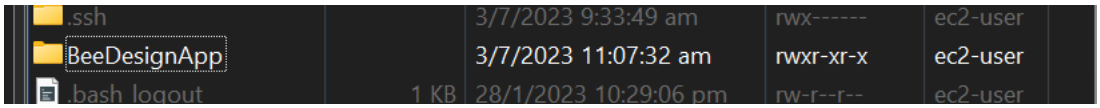


Step 9: Create new directory using WinSCP in remote EC2 called **bee design app.**

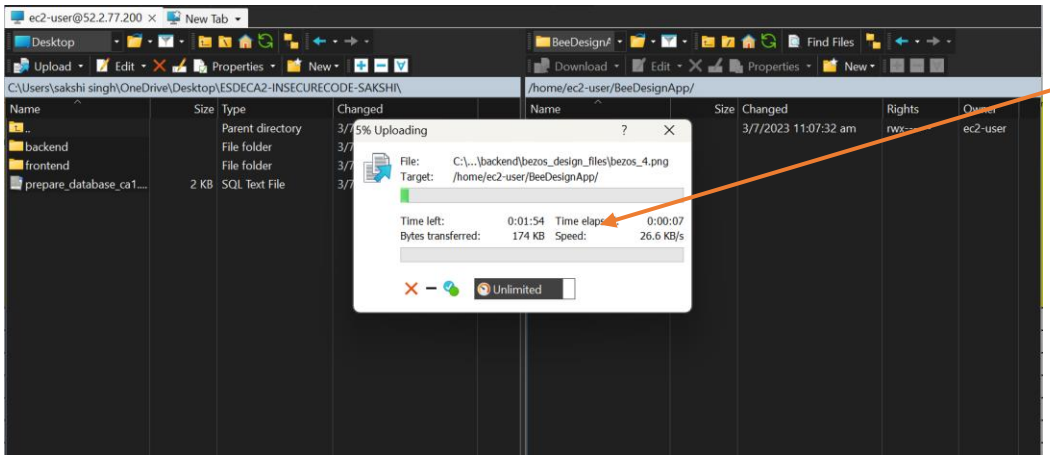




The folder is created.

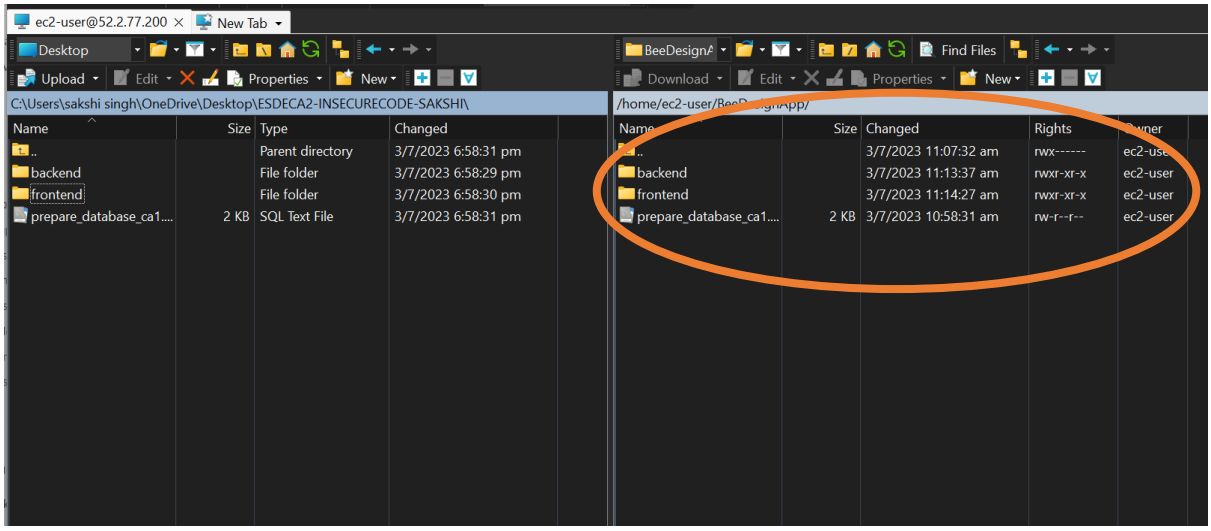


Step 10: Drag the Bee Design files from your laptop into the new directory.



Transfer of files to ec2-user.

Files are successfully transferred to the directory.



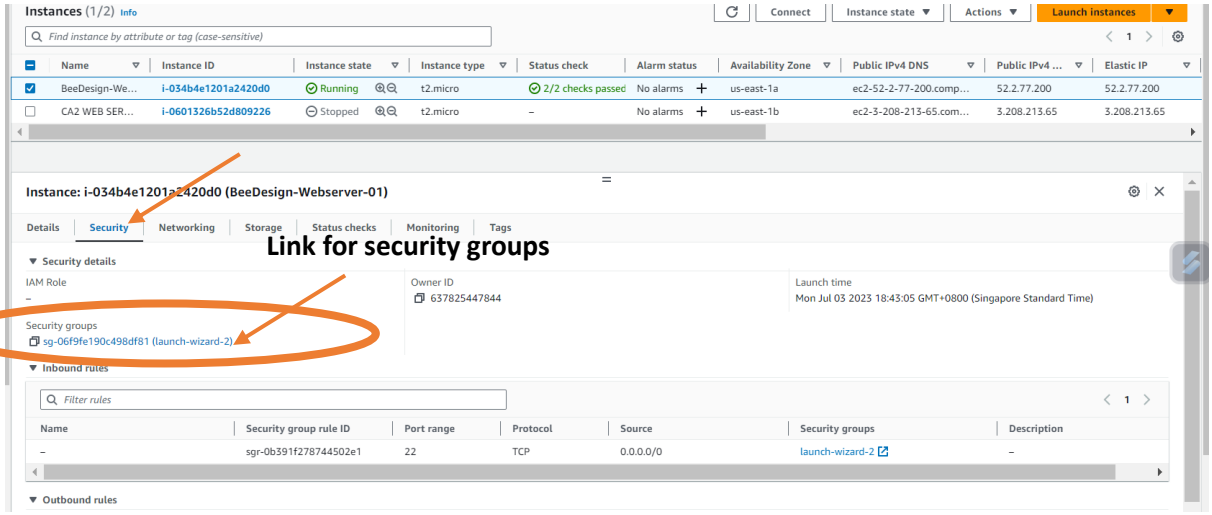
Configuring security settings

Why configure security settings?

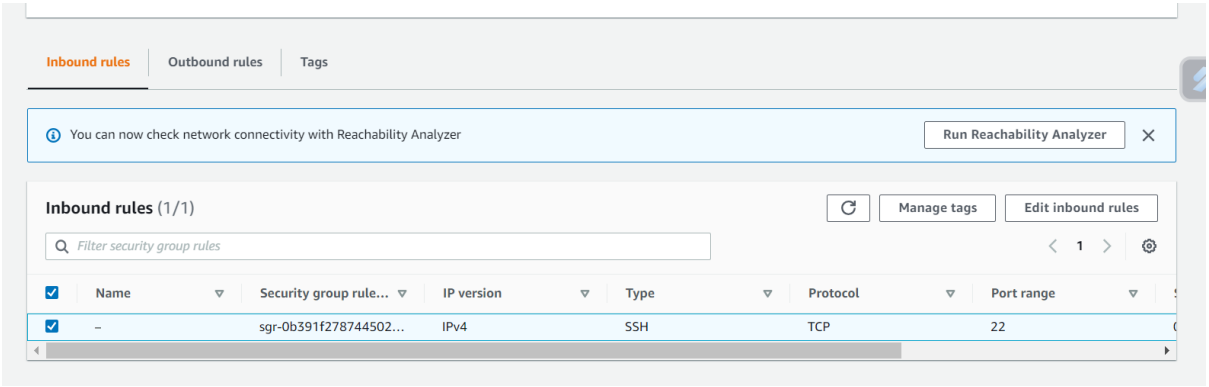
Configuring inbound rules for ports like 3001 and 5000 on an Amazon EC2 instance is important when hosting a website with services that use those ports. **Port numbers are like doors that allow specific types of traffic to reach applications on the server**. By opening these ports in the EC2 instance's security group, you **enable external access to services like Node.js (3001) or Flask (5000**) servers. This is **useful for web development** and allowing visitors to interact with your website.

Step 1: Go to AWS Learner Lab EC2 dashboard and click on **'Security' tab** of your EC2 instance.

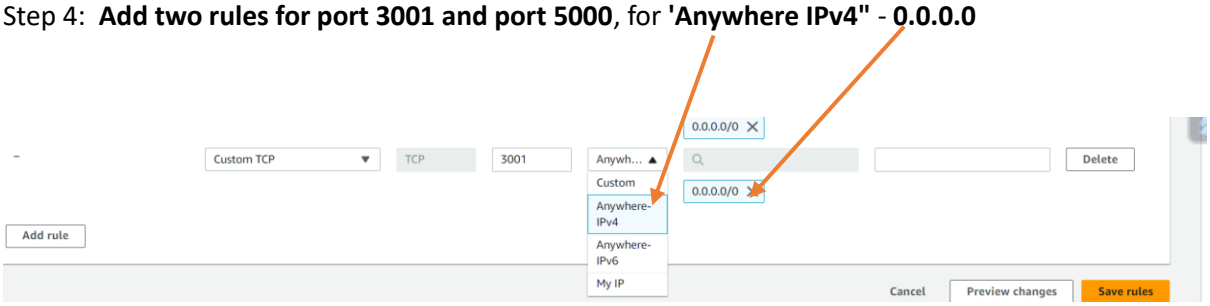
Step 2: Click on the **link** in 'Security Groups to configure the settings for your website.



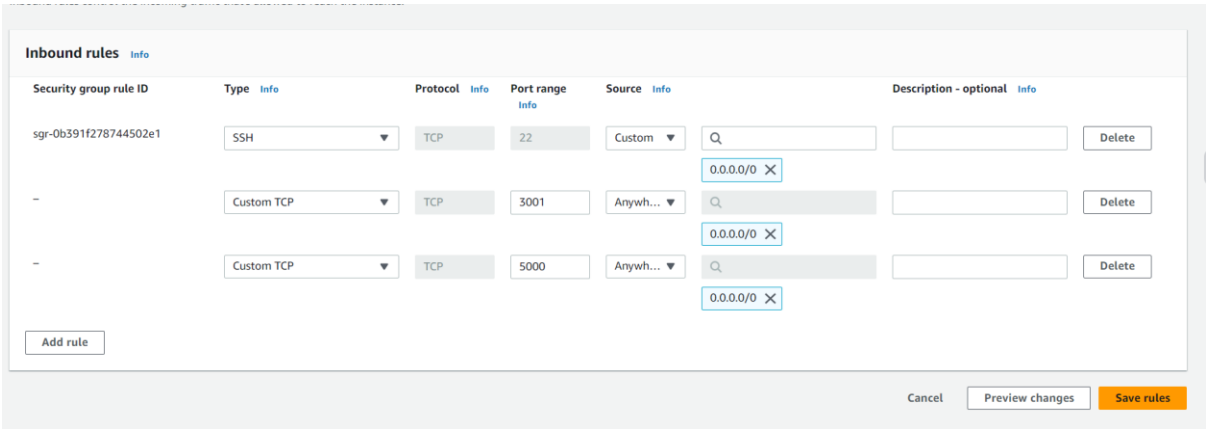
Step 3: Click on Edit Inbound Rules



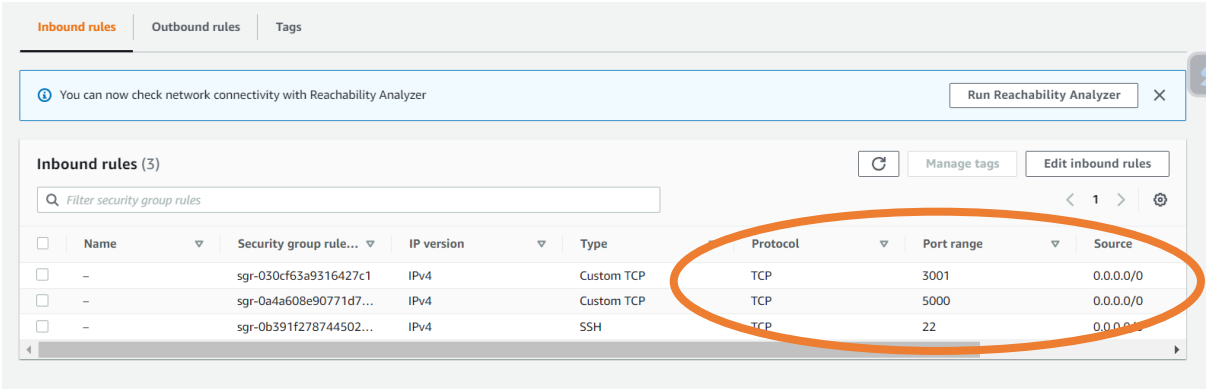
Step 4: **Add two rules for port 3001 and port 5000**, for **'Anywhere IPv4" - 0.0.0.0**



Step 5: **Press "save rules"** after adding both rules for port 3001 and 5000.



Step6: **Check in Inbound rules** that rules for port 3001 and 5000 are added.

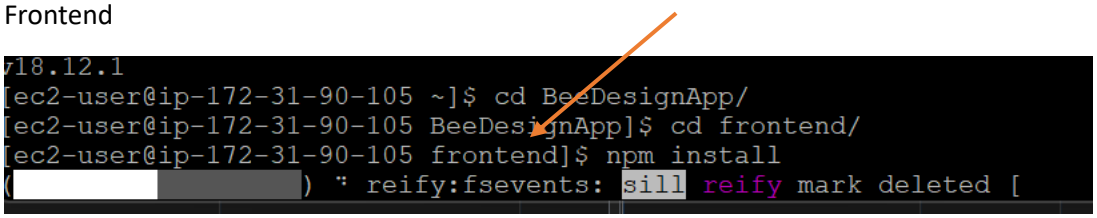


Launch website on EC2

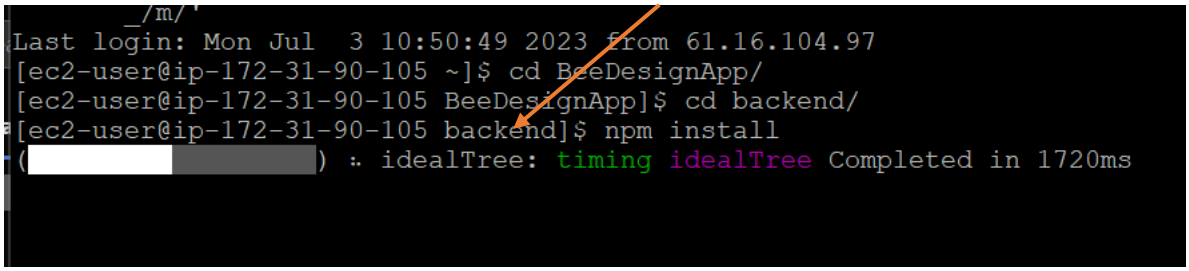
Now we will run both servers through the putty terminals in WinSCP and run our website in the browser.

Step 1: Run npm install in both your front end and back end directory

Frontend

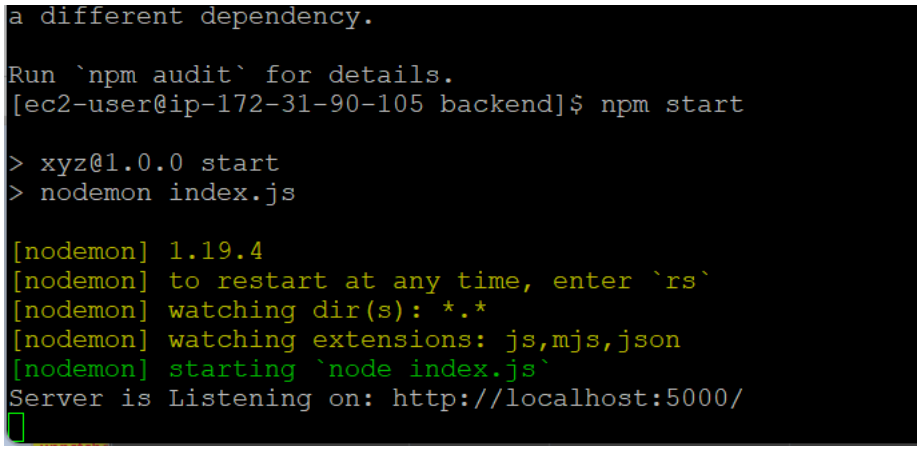


Backend

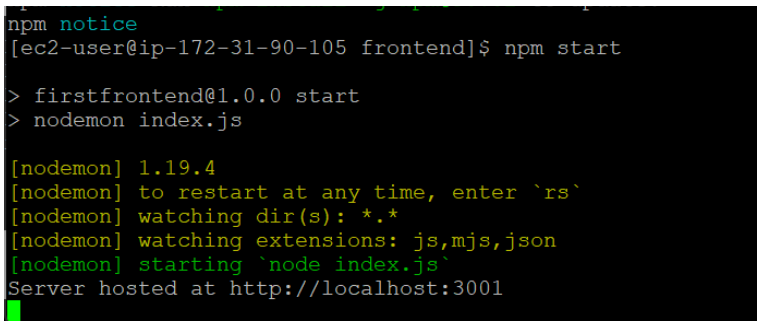


Step 2: Run Npm start to start both frontend and backend servers.

Backend



Frontend

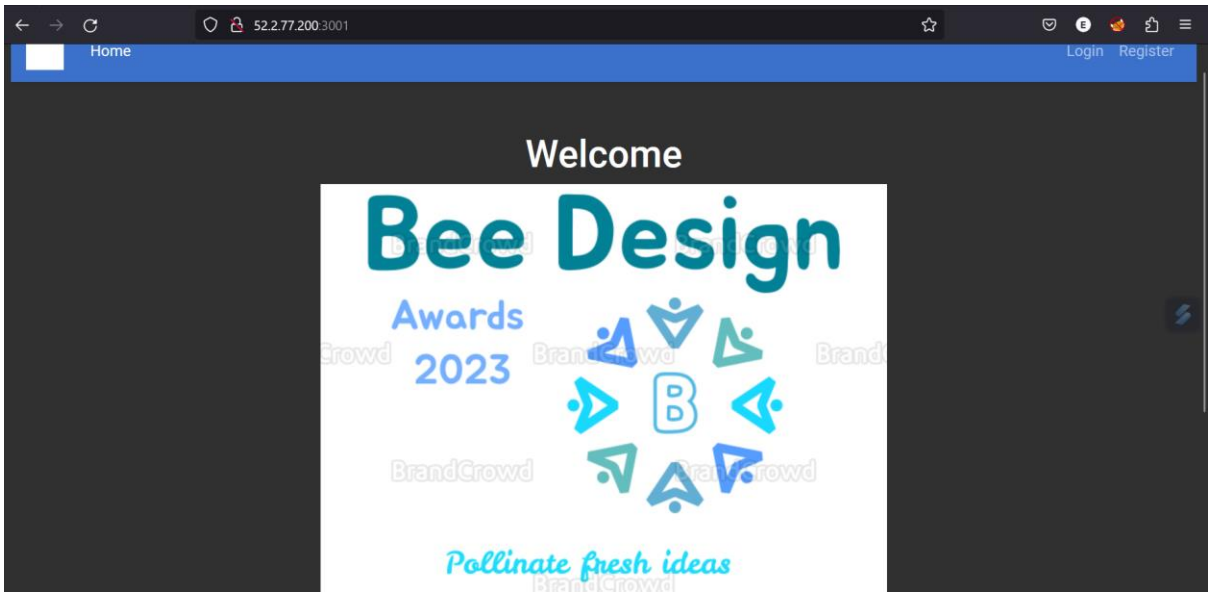


Step 3: Take note that your EIP address is your URL to host your website through your EC2 instance

http://52.2.77.200:3001

Step 4: Run the website with the Eip URL http://52.2.77.200:3001

Congratulation you have successfully hosted your website on Ec2 instance.



Part 3- Setup RDS

RDS

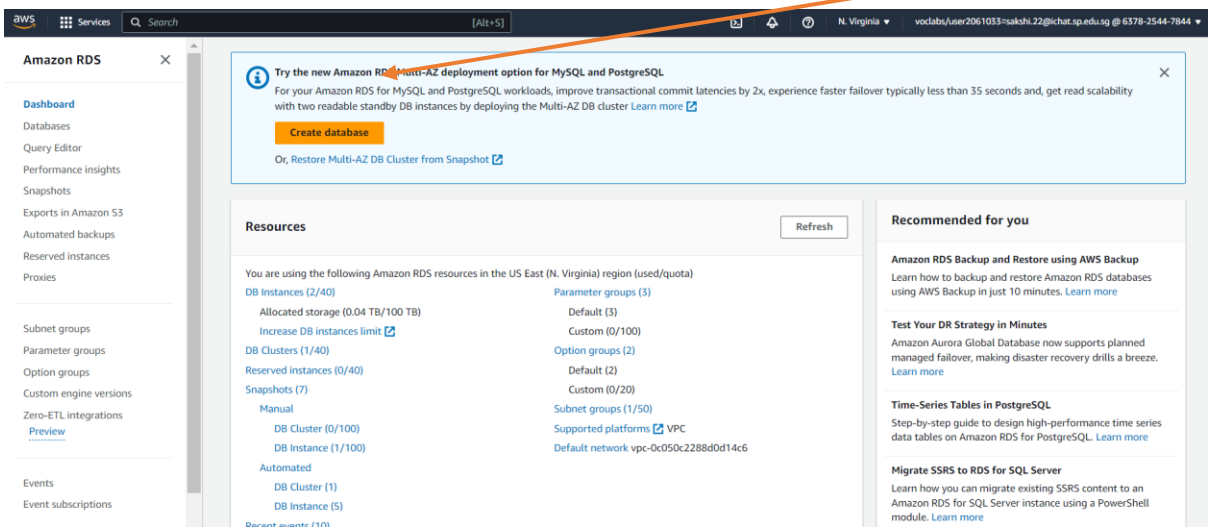
Amazon RDS is a **managed database service on AWS**. It's used for websites to simplify database management, ensure scalability, and enhance reliability.

- It handles backups, scaling, and high availability, freeing developers from infrastructure tasks.
- It supports various databases and offers security features.
- Provides tools for monitoring and optimizing performance.

In essence**, Amazon RDS streamlines database operations, allowing website developers to focus on their applications.**

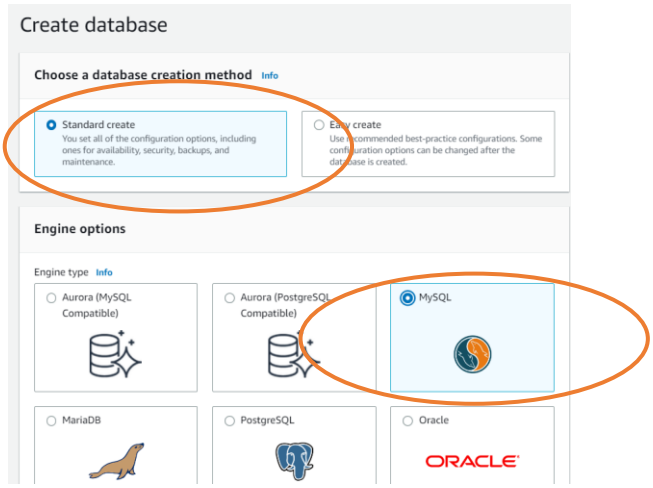
Configure settings to create RDS database.

Step 1: Search for RDS in AWS console and open in new tab **Click 'Create database'.**

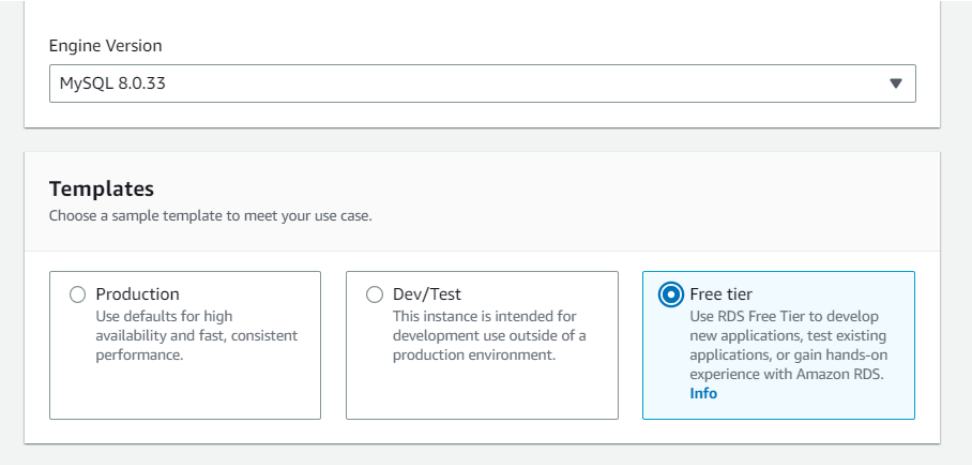


Step 2: For database creation method choose **"Standard Create".**

Step 3: For engine options, **choose MySQL**. Leave the **version as default**.



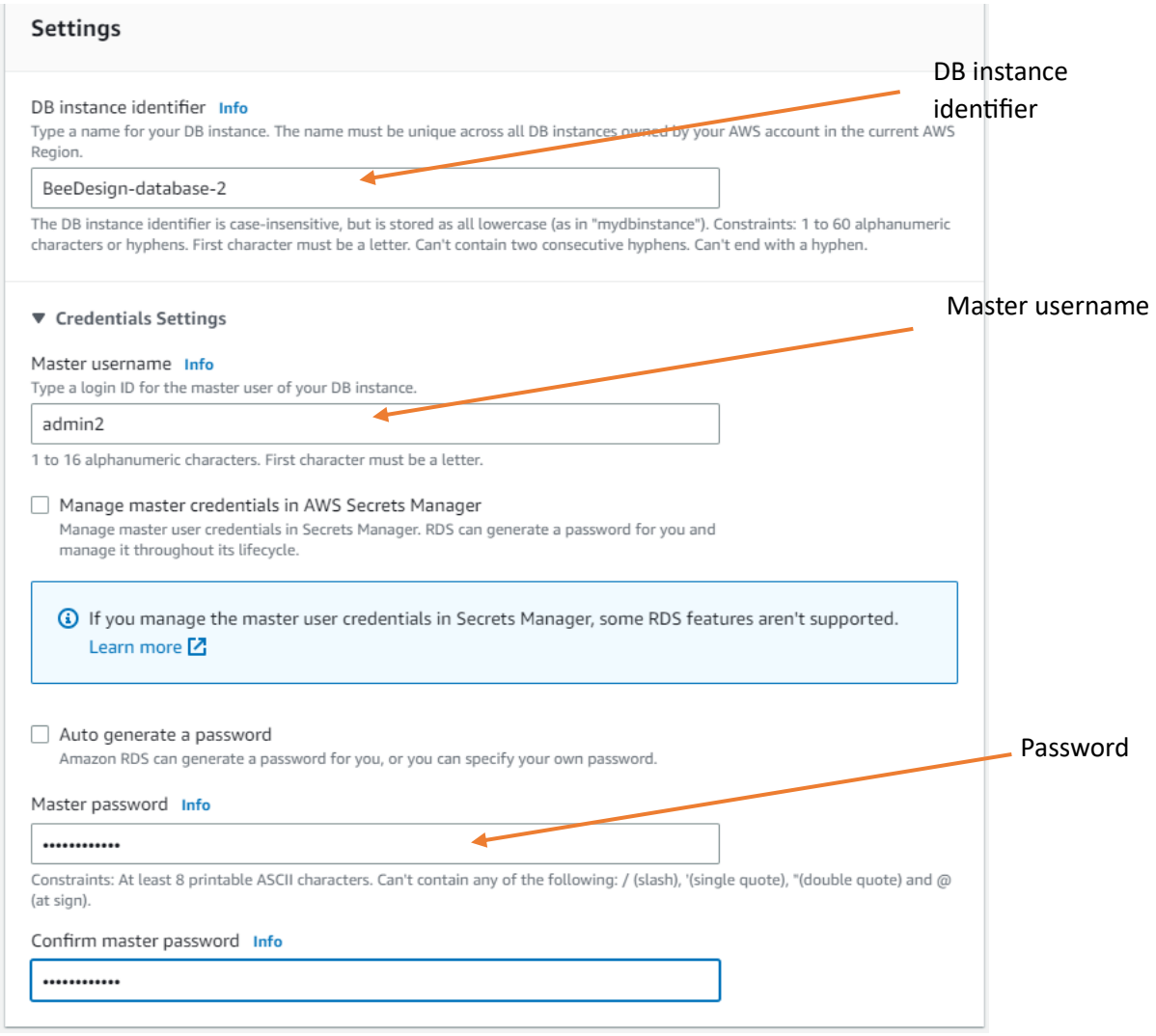
Step 4: For templates , choose free tier



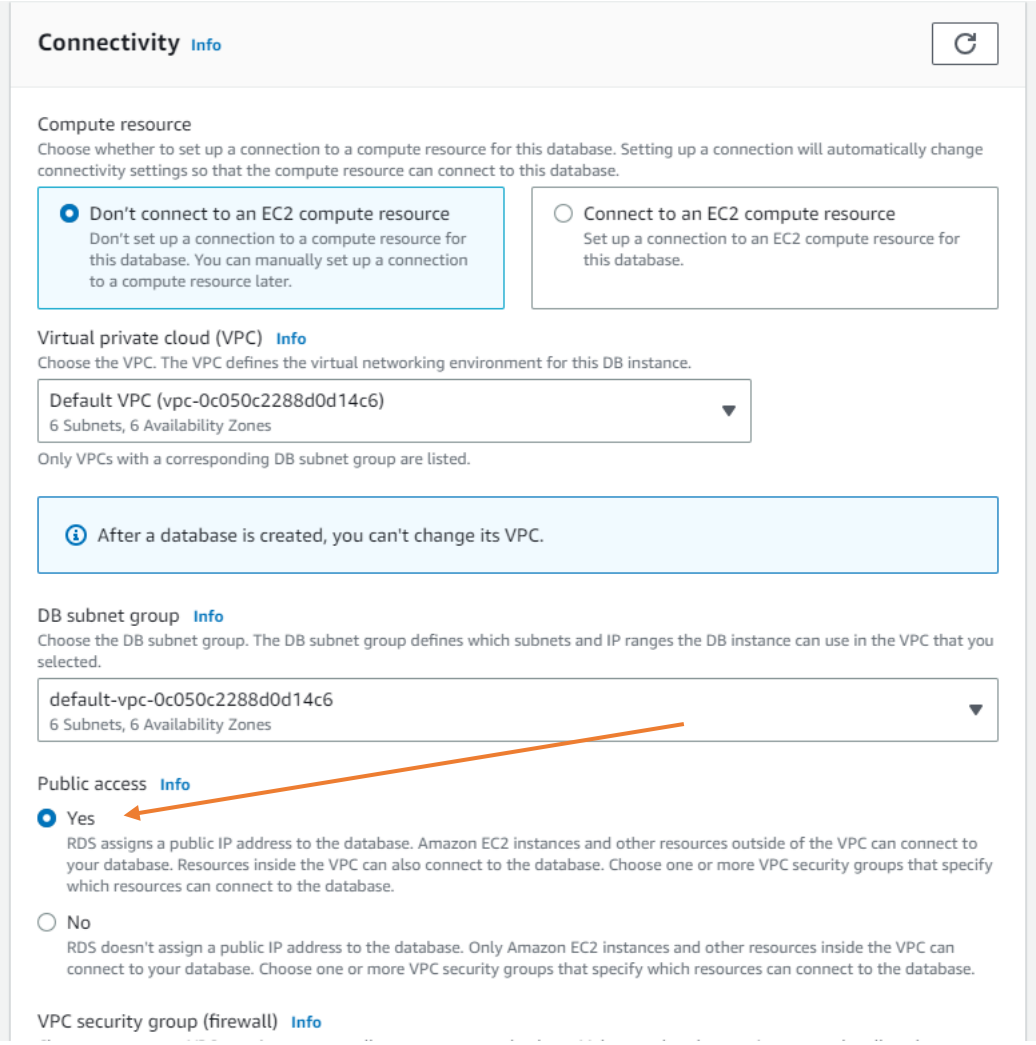
Step 5: For settings, name the DB instance identifier as BeeDesign-database-2

Step 6: In credential settings, set master username as admin2.

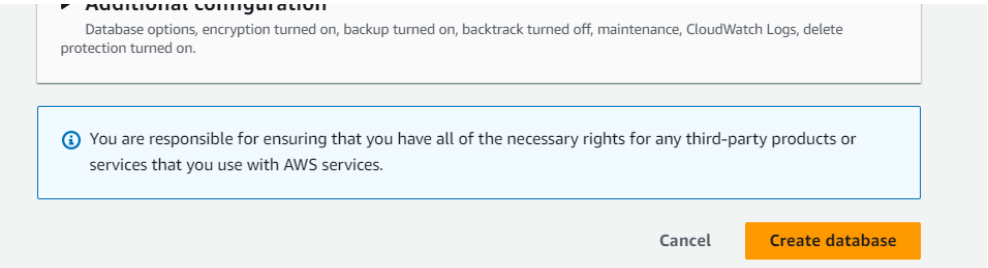
Step 7: In credential settings, set your preferred password.



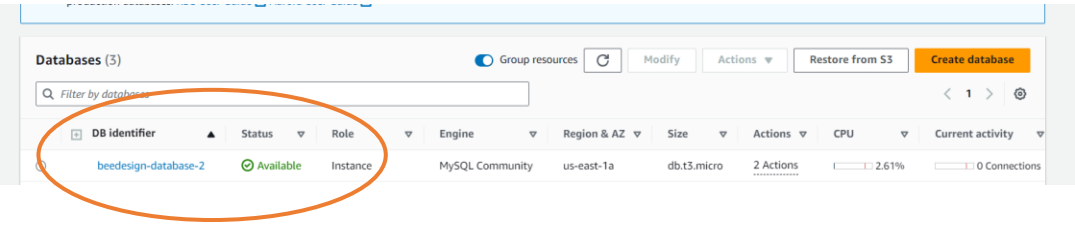
Step 8: Under connectivity, set public access to **"Yes"**



Step 9: Click Create database to create your RDS MYSQL database.



Step 10: Check the database has been created in your RDS console.

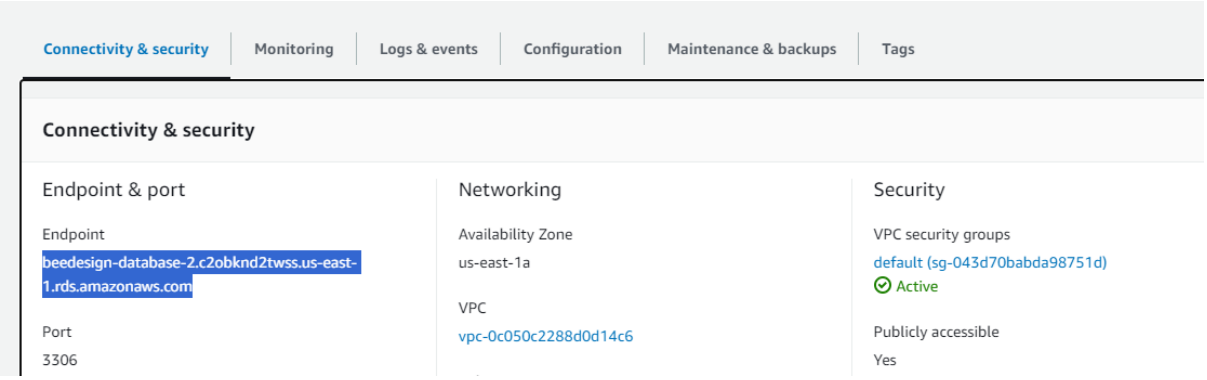


Connect to RDS server by using MySQL workbench.

The purpose of connecting to an Amazon RDS instance using MySQL Workbench is to **efficiently manage, manipulate, and interact with the database.**

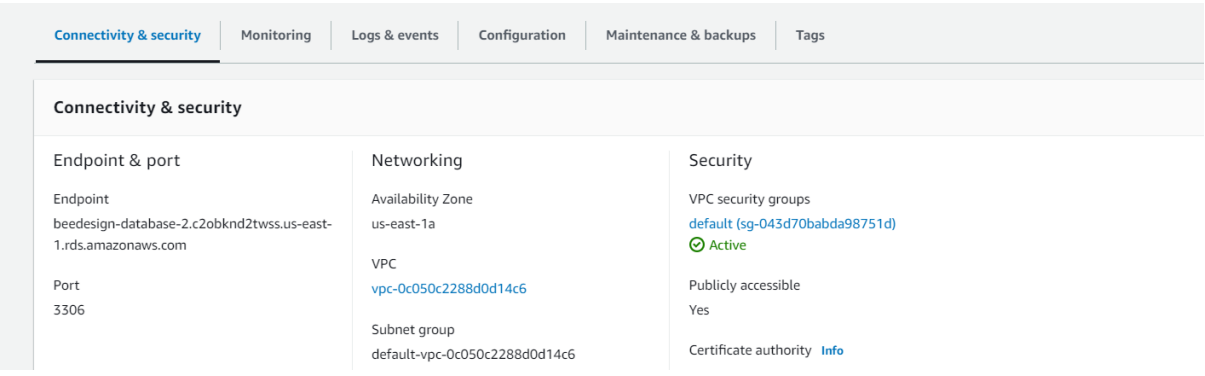
- **Administering the database**, querying, and updating data, optimizing performance, designing schemas, and collaborating with team members.
- It provides a graphical interface for these tasks, enhancing **database management and development processes.**

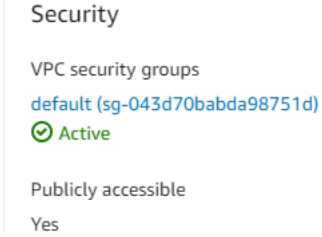
Step 1: Take note of your RDS endpoint and save it in a notepad or elsewhere



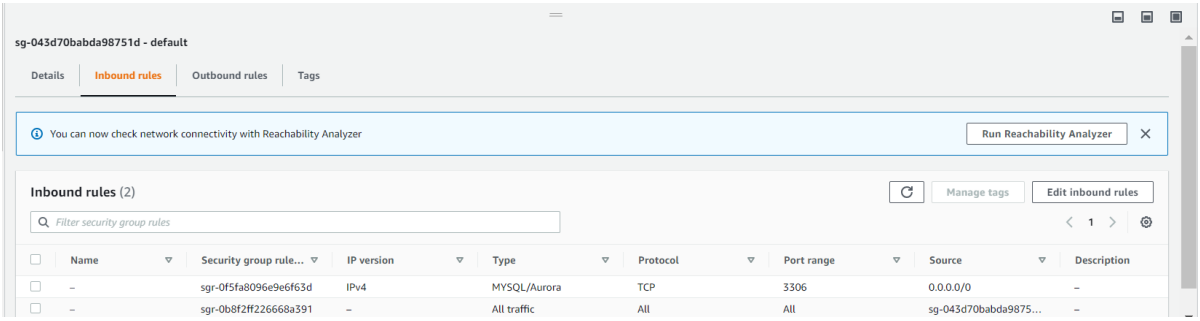
Added security group rule for MySQL.

Step 1: Under 'Connectivity and Security Group' tab, click on VPC Security Group .



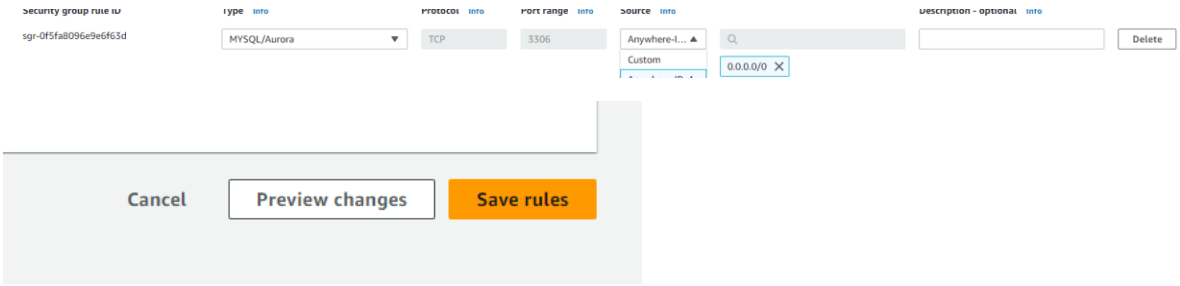


Step 2: In the page that loads, click on the security group at top section, then click on Inbound rules at bottom Click 'Edit inbound rules'



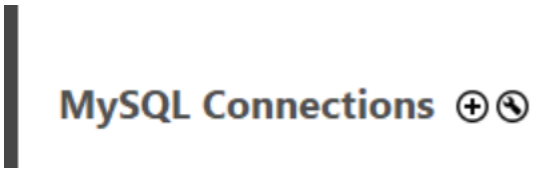
Step 3: Add rule to allow MySQL /Aurora traffic for Anywhere IPv4 as source allowing incoming network traffic from any device with an IPv4 address on the internet to access your MySQL or Aurora database.

Step 4: Press Save.



In MySQL configuration

Step 1: click + to create new connection

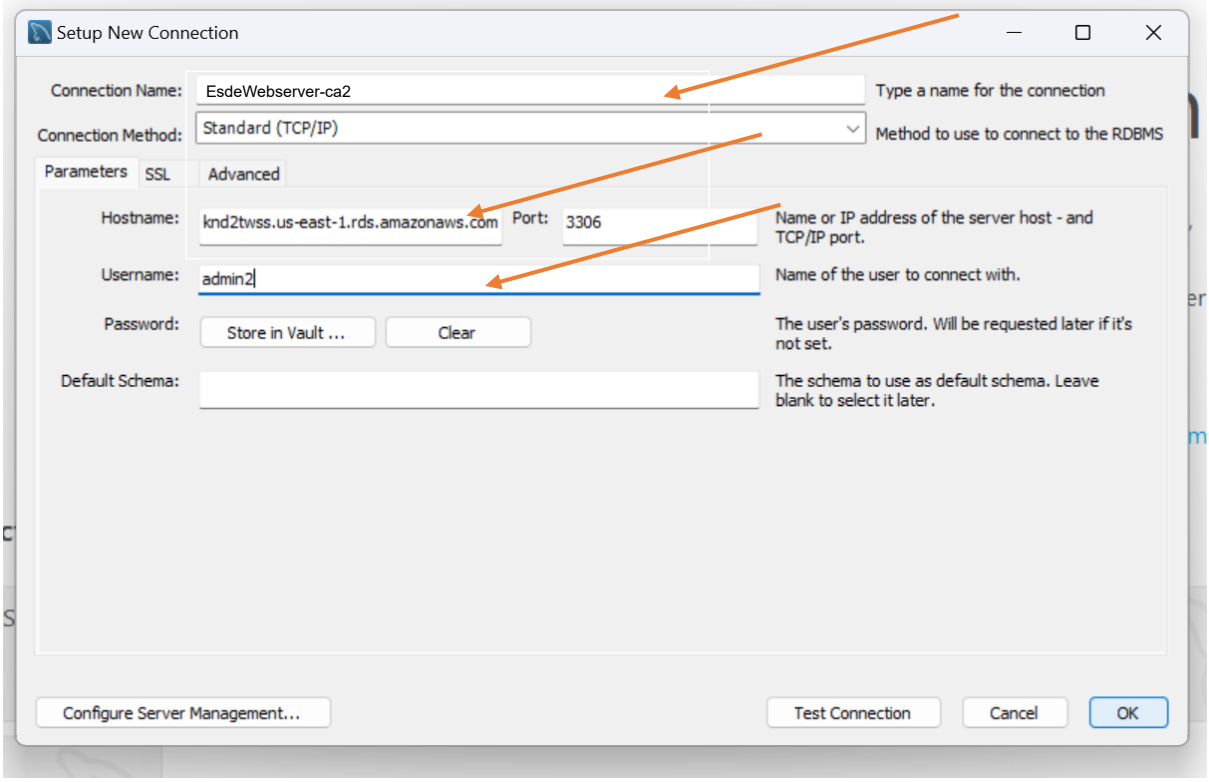


Configure Connection in MySQL Workbench:

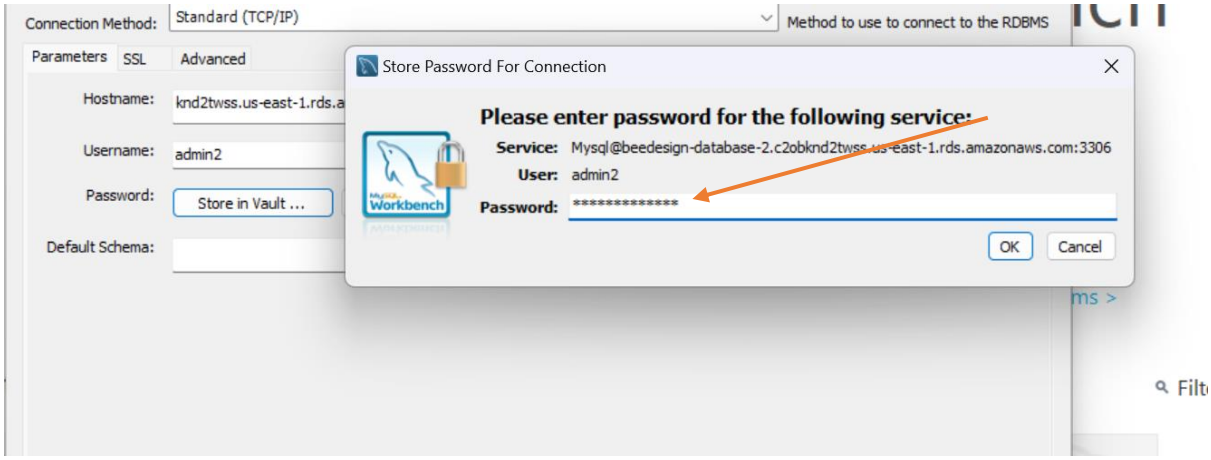
Step 1: Choose a connection name, for instance, 'EsdeWebserver-ca2'.

Step 2: Paste your Amazon RDS endpoint into the "Host" field.

Step 3: Use "admin2" as the username, which you specified during RDS creation.

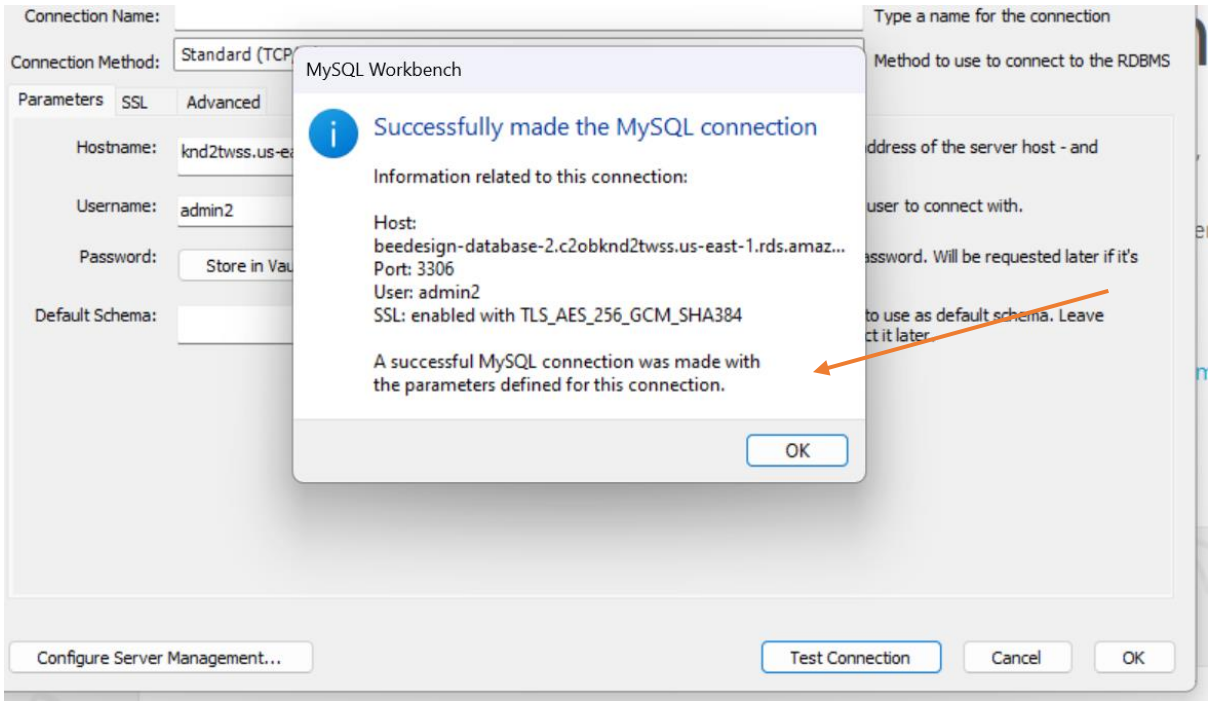


Step 4: Enter the password you selected while setting up the RDS instance.



Step 5: Click the "Test Connection" button to ensure the connection works as expected.

Step 6: If the test is successful, click the "Ok" button to save the configuration settings.



Run Database Script

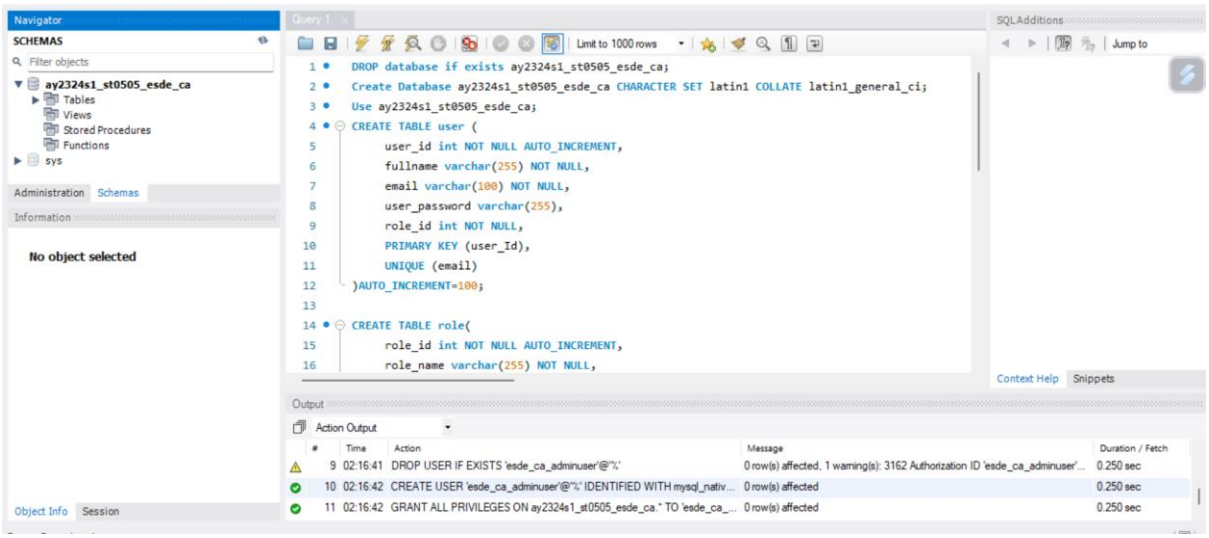
Step 1: Double click your connection in the Workbench to launch it

Step 2: Navigate to folder with **prepare_database_ca1.sql.**

Step 3: Run the script.

Step 4: **Click refresh button** on top left.

Step5: Check that the **database is in your schema**.

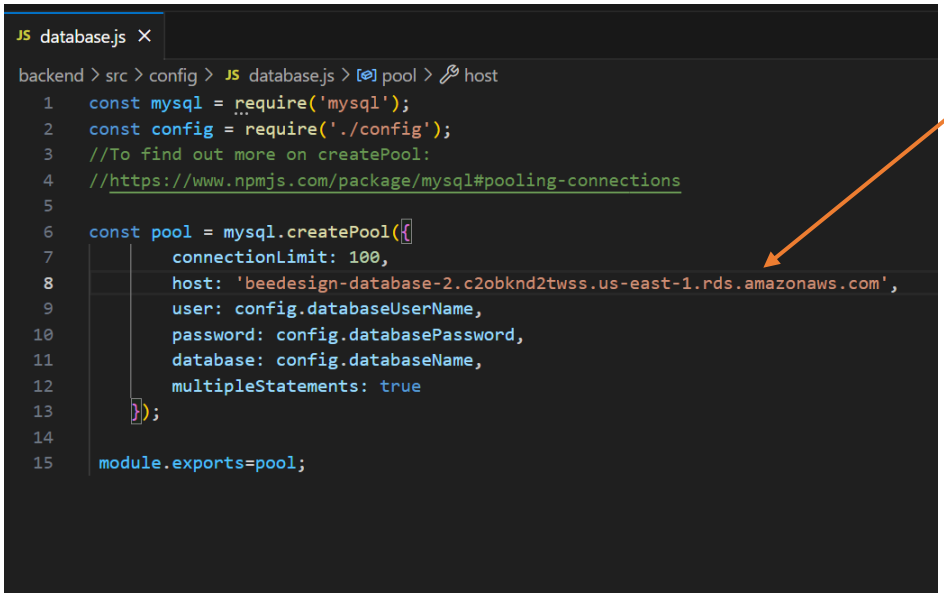


Modify database.js.

Step 1: In Visual Studio code, modify the 'backend/src/config/database.js' source code to use your RDS endpoint.

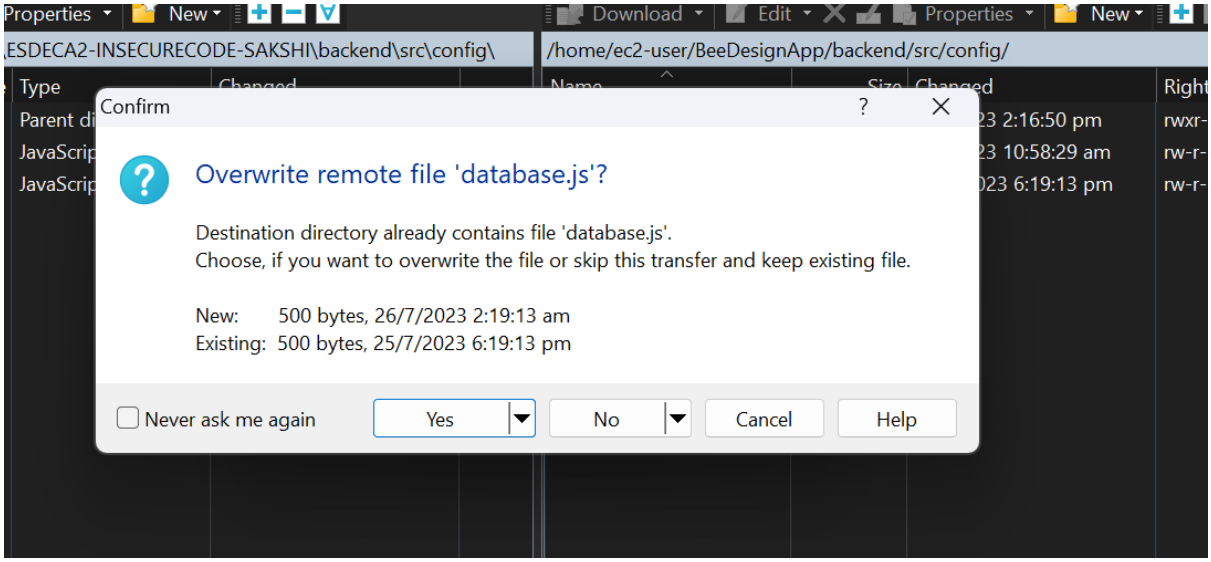
Step 2: Change Line 8 to your RDS endpoint**. host: beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com'** to ensure it connects to the Amazon RDS instance.

Step 3: Save your changes.

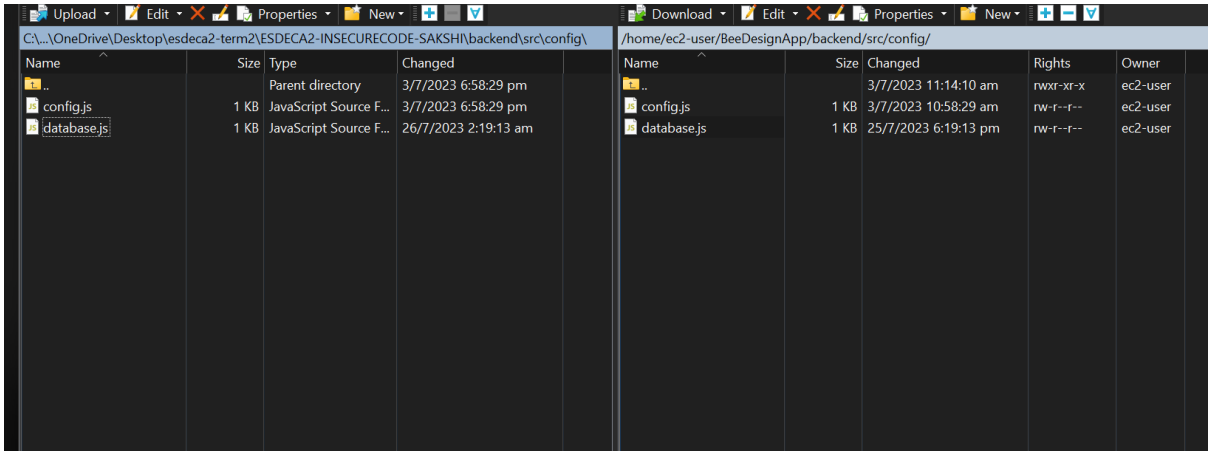


Step 4: Upload your changes into your ec2 user folder as well by right clicking and choosing upload.

Press yes.



Successfully Uploaded to ec2 user directory .



Run SeedData.js

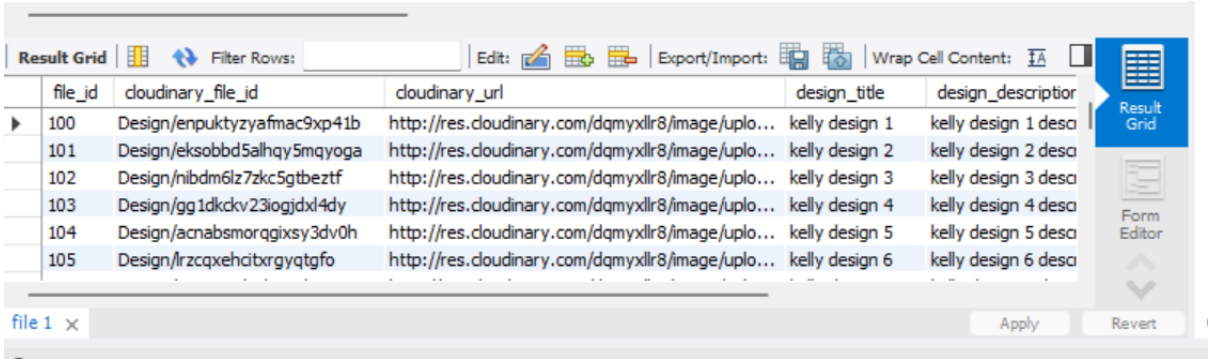
Step 1: Open a new terminal window in WinSCP.

Step 2 Navigate to backend directory.

Step 3: Run seeddata.js node seeddata.js so that we can populate all the files into the files table in MySQL with the cloudinary data.

```
[ec2-user@ip-172-31-90-105 backend]$ node seeddata.js
preparing to create user and data for kelly
Inspecting createUserResult variable
 OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 100,
  serverStatus: 2,
  warningCount: 0,
  message: '',
  protocol41: true,
  changedRows: 0
}
Checking uploadResult before calling createFileData in try block
 {
  imageURL: 'http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design
/enpuktyzyafmac9xp41b.png',
  publicId: 'Design/enpuktyzyafmac9xp41b',
  status: 'success'
}
kelly_design1.png is uploaded. 1 record created in file table.
Checking uploadResult before calling createFileData in try block
 {
  imageURL: 'http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design
/eksobbd5alhqy5mqyoga.png',
  publicId: 'Design/eksobbd5alhqy5mqyoga',
  status: 'success'
}
kelly_design2.png is uploaded. 1 record created in file table.
Checking uploadResult before calling createFileData in try block
 {
  imageURL: 'http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design
/nibdm6lz7zkc5gtbeztf.png',
```

Step 4: Check that the records have been inserted into the table.



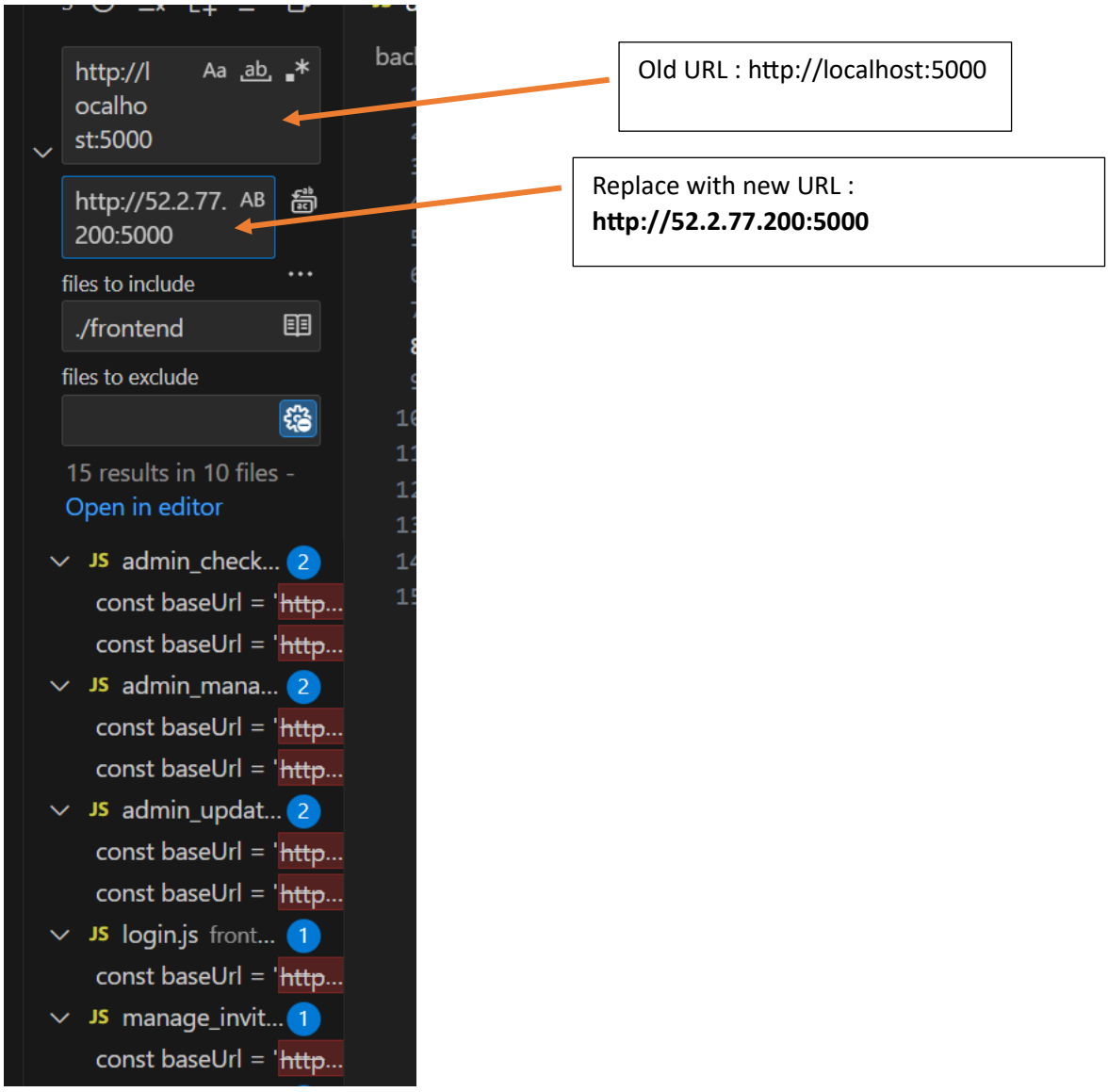
Change URL

Step 1: **Search and Replace URL**s in Visual Studio Code:

- Open Visual Studio Code and select 'Find in Folder' from the menu for the 'frontend/js' folder.

Step 2: **Update URLs** with External IP:

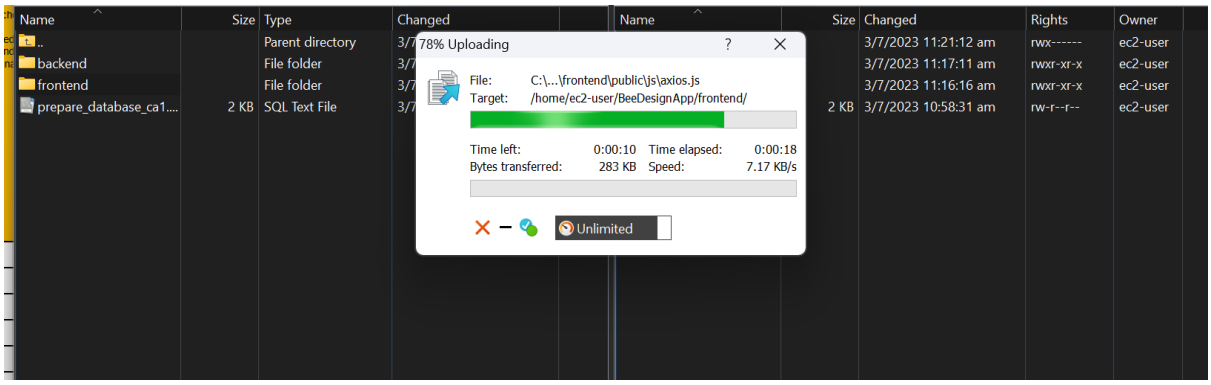
- Search for 'http://localhost:5000' across files.
- Replace 'localhost' with your External IP (EIP), 52.2.77.200
- Click 'Replace all' to update all occurrences.
- Save the changes.



Step 3: Check that your base URL is **http://52.2.77.200:5000.**

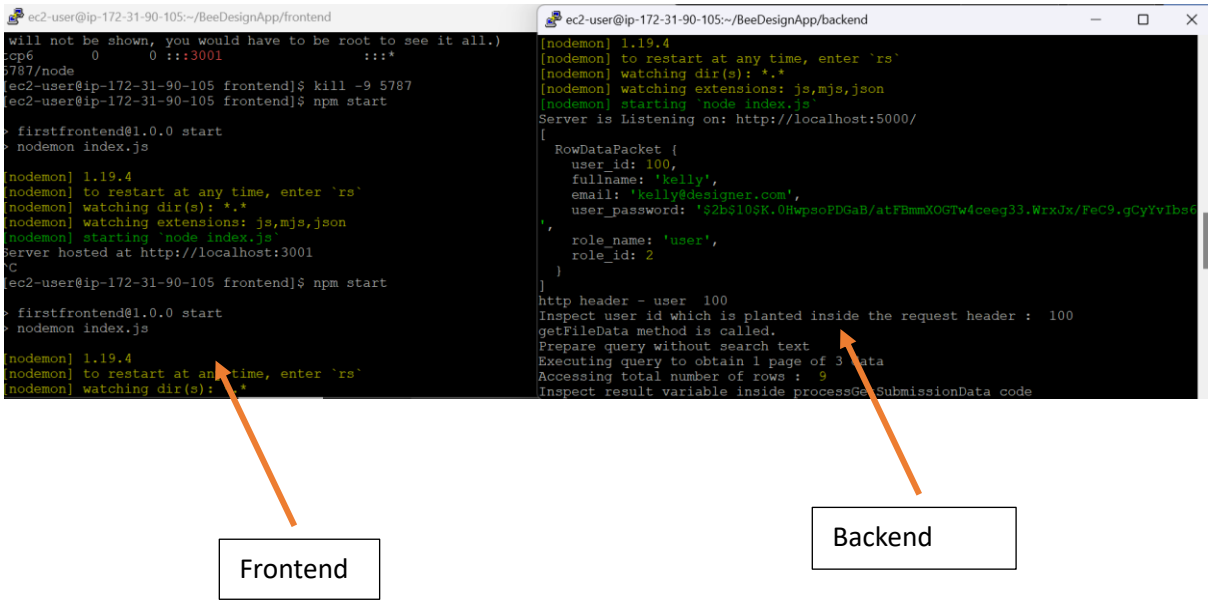
```
//to server-side api when the #submitButton element fires
$('#submitButton').on('click', function(event) {
    event.preventDefault();
    const baseUrl = 'http://52.2.77.200:5000';
    let searchInput = $('#searchInput').val();
    let userId = localStorage.getItem('user_id');
    axios({
            headers: {
```

Step 4: Upload your changes to the ec2-user directory



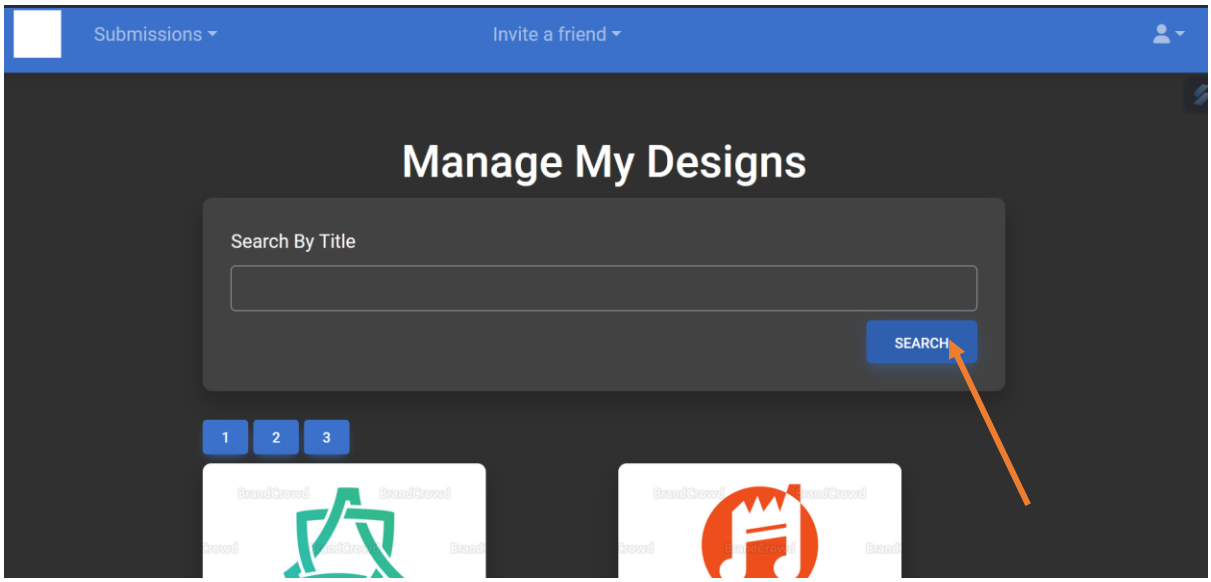
Launch website to check RDS configuration.

Step 1: Run both the backend and frontend server with npm start.



Step 2: Run your website with 52.2.77.200:3001/ and log in as Kelly.

Step 3: Click Search to view all of us Kelly designs. If you are able to do this successfully, congratulations you website is connected to the MySQL RDS database.



Part 4 – DynamoDB

DynamoDB in AWS

Amazon DynamoDB serves to provide a fully managed NoSQL database solution that prioritizes scalability, low-latency access, and ease of use for developers. Its core purposes include:

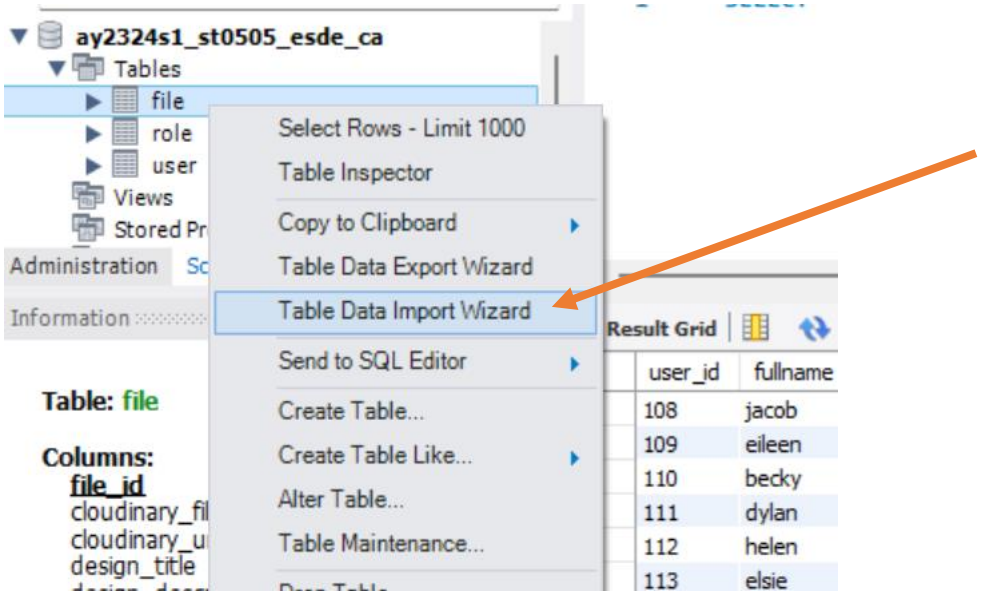
- **Scalability:** DynamoDB scales seamlessly to accommodate evolving workloads.
- **Low Latency:** It ensures rapid data access, making it suitable for real-time applications.
- **Flexibility:** DynamoDB's schema-less design adapts to changing data needs.
- **Managed Service:** Developers are freed from administrative tasks.
- **Security:** Integrates with IAM and offers encryption for data protection.
- **Global Reach**: Global Tables enable low-latency access across regions.
- **Operational Efficiency**: Developers can focus on building, not managing, databases.

In essence, **DynamoDB streamlines database management**, providing a solution optimized for modern application demands.

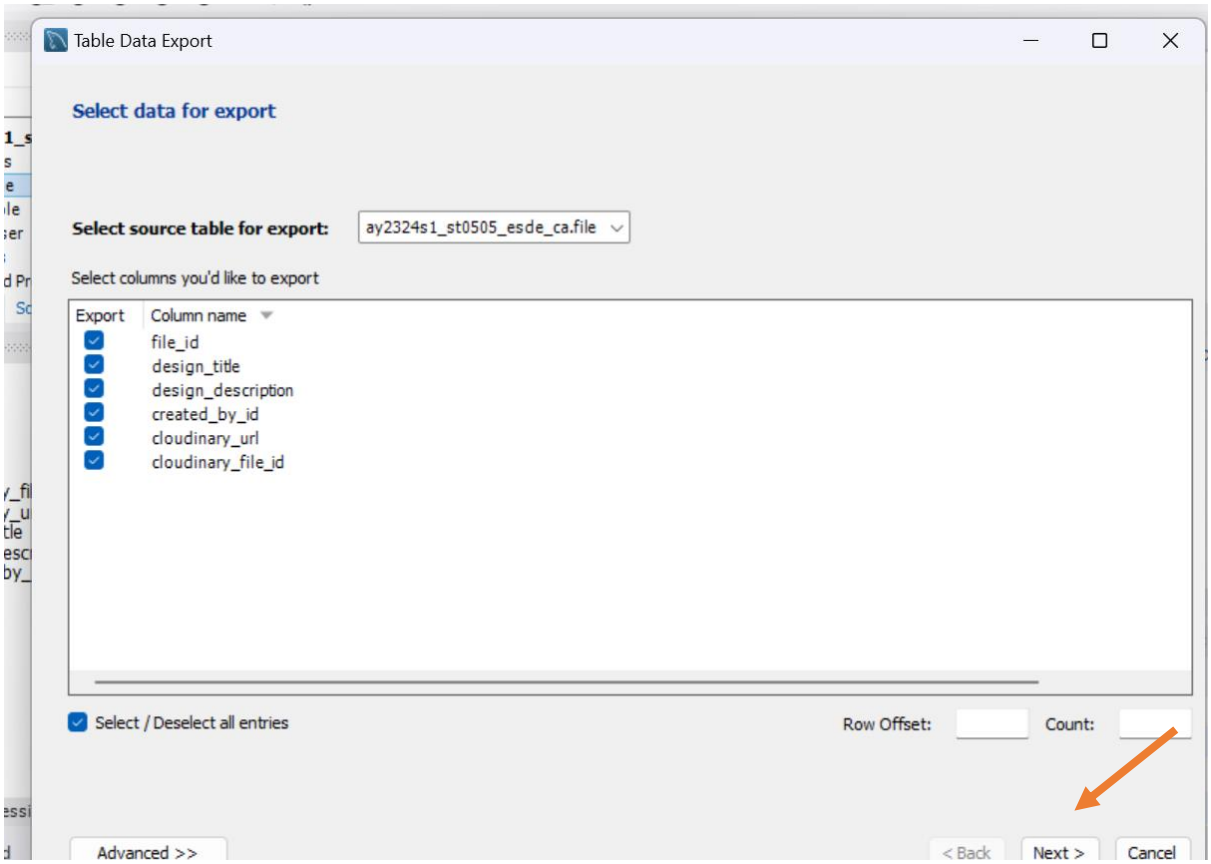
Export records from RDS MySQL to a Json file

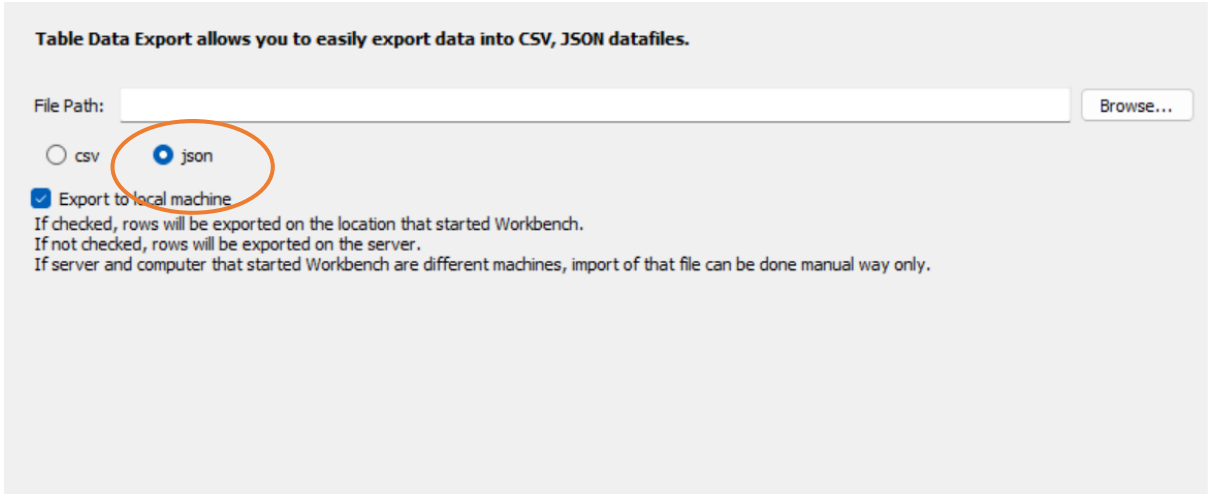
Step 1: Start your RDS MySQL Workbench

Step 2: Perform a right-click on the "Files" table and select the "Table Export Wizard" option.

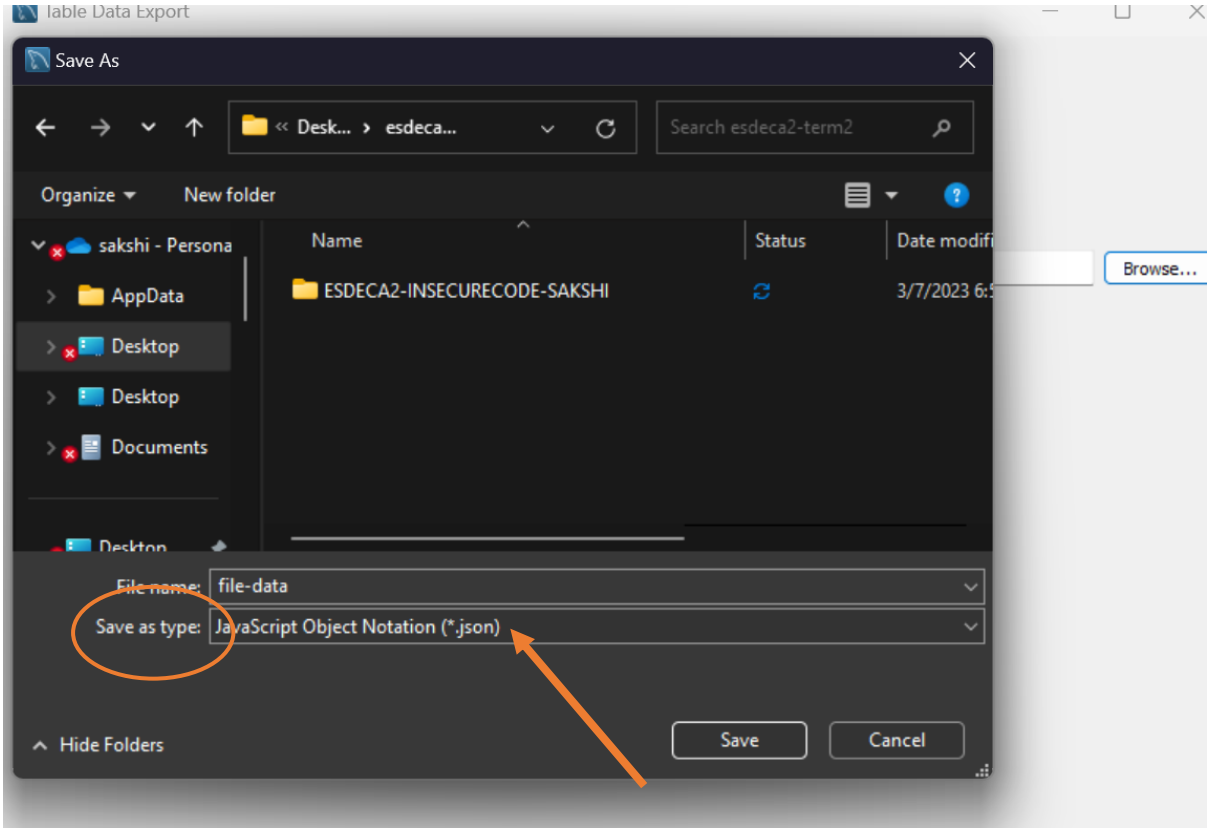


Step 3: Click on the "Next" button, and then select the export format as JSON.

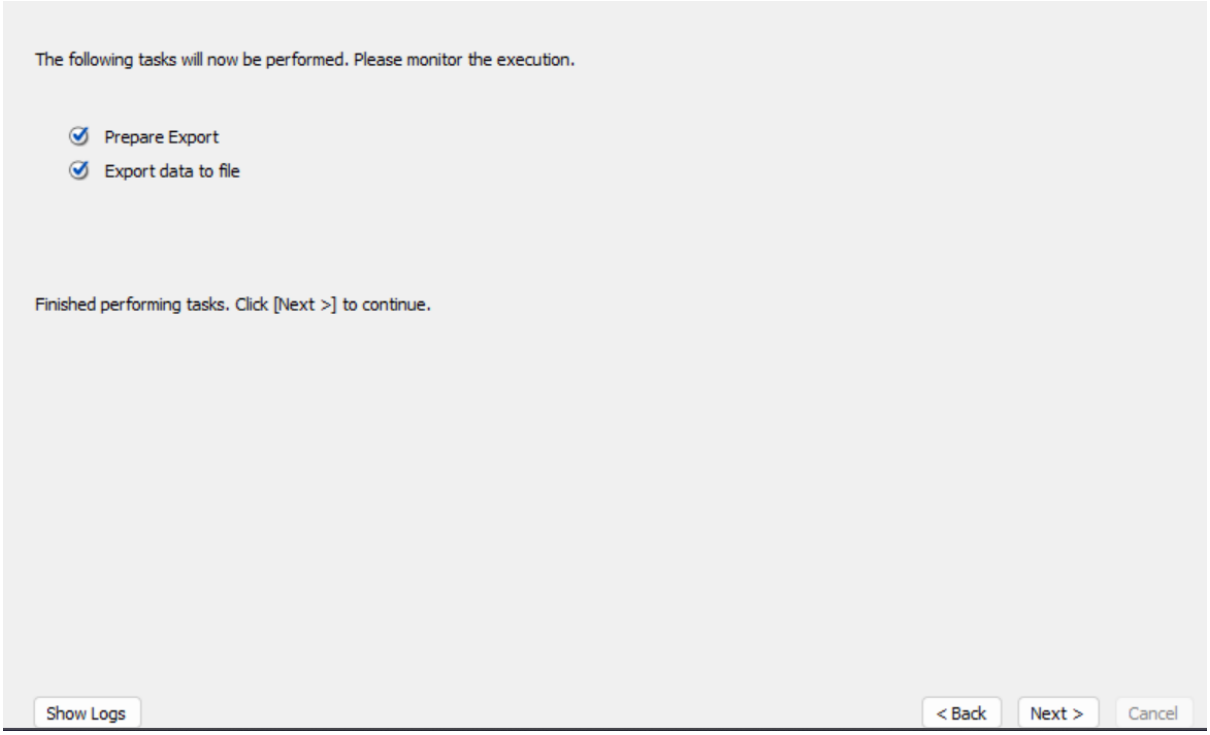




Step 4: Navigate to the folder where you extracted the DynamoDB codes. Save the file as "file-data. Json". Make sure to choose 'JSON' as the 'Save as Type'.

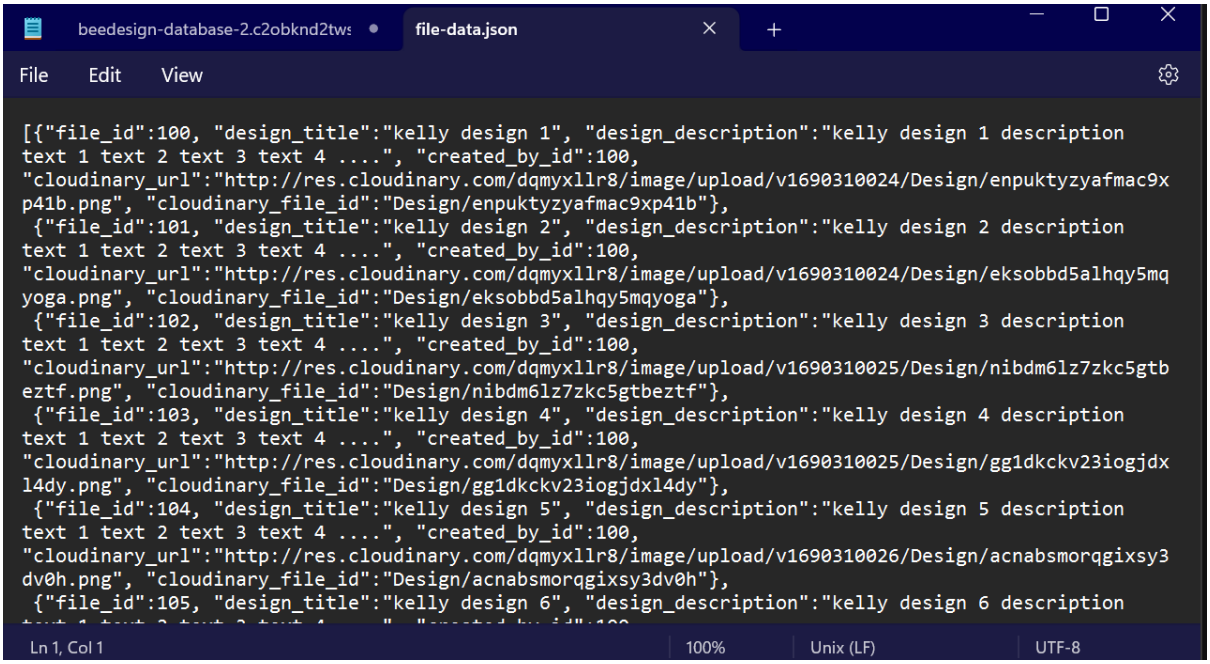


Step 5: Export the data from MySQL to file-data. Json by **clicking next.**



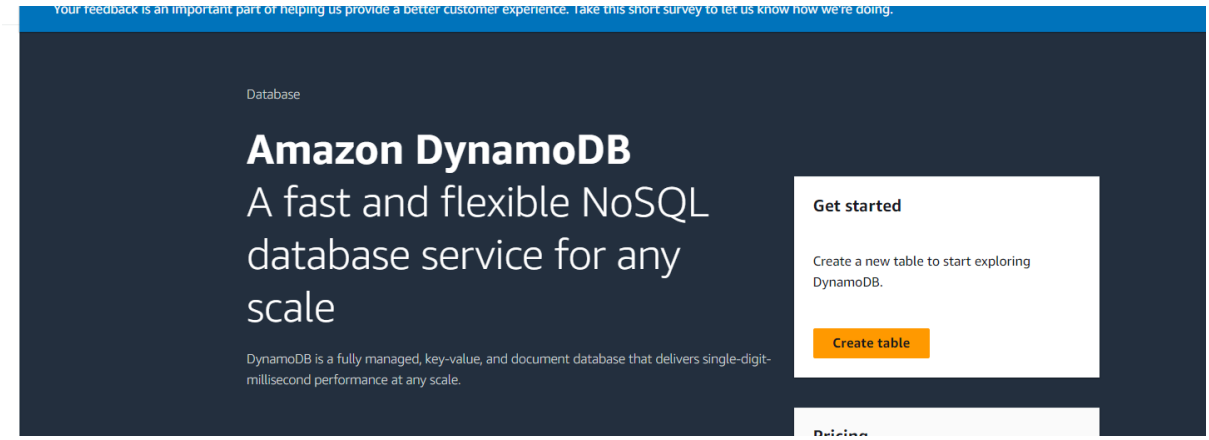
You've successfully exported data from MySQL to JSON, resulting in 36 records being exported.

Step 6 – Check that your file-data. Json with notepad has populated all the data.



Create DynamoDB Table.

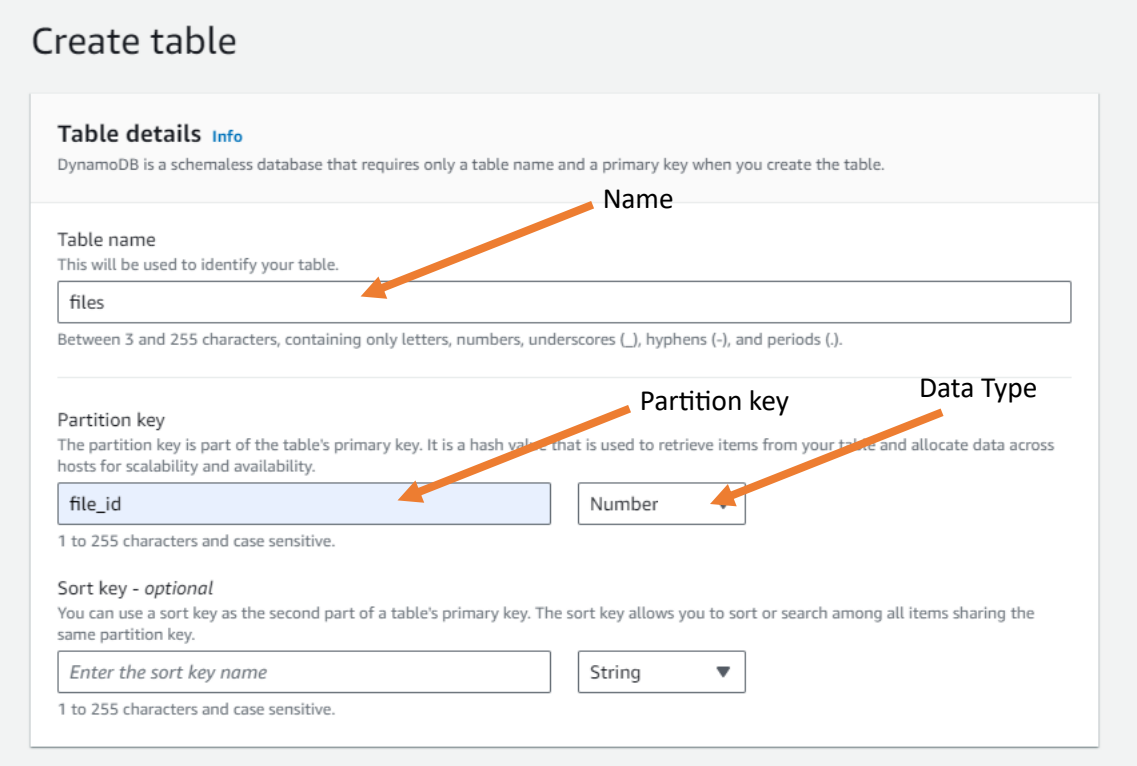
Step 1: Click the orange 'Create Table' button.



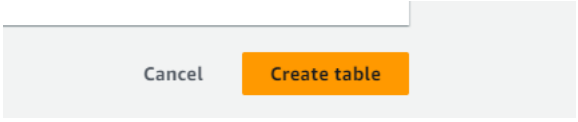
Step 2:

In the settings page, provide the following details:

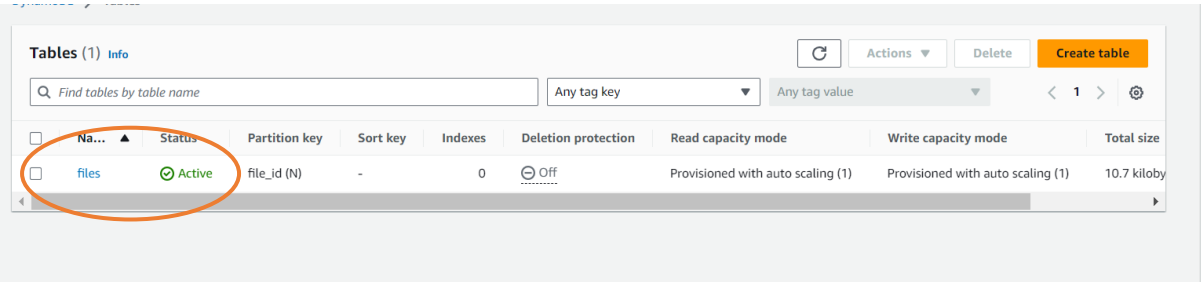
1. Table Name: **files**
2. Partition Key: **file_id**
3. Data Type: **Number**



Then, proceed to click the **'Create Table'** button.



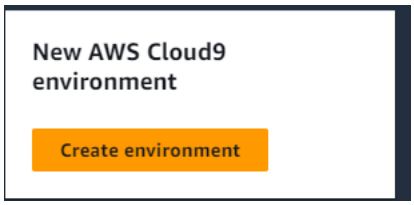
Step 3: Check that your new table “files” is successfully created.



Populate DynamoDB using Node js.

Create Cloud 9 Environment

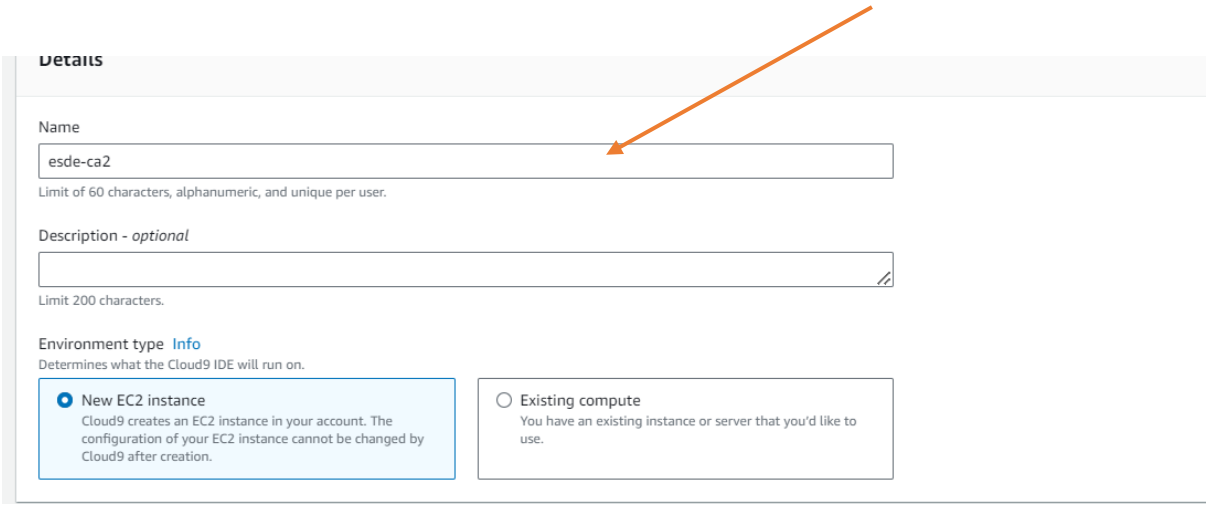
Step 1: Open Cloud9 in a new tab and click the orange 'Create Environment' button to initiate the environment creation process.

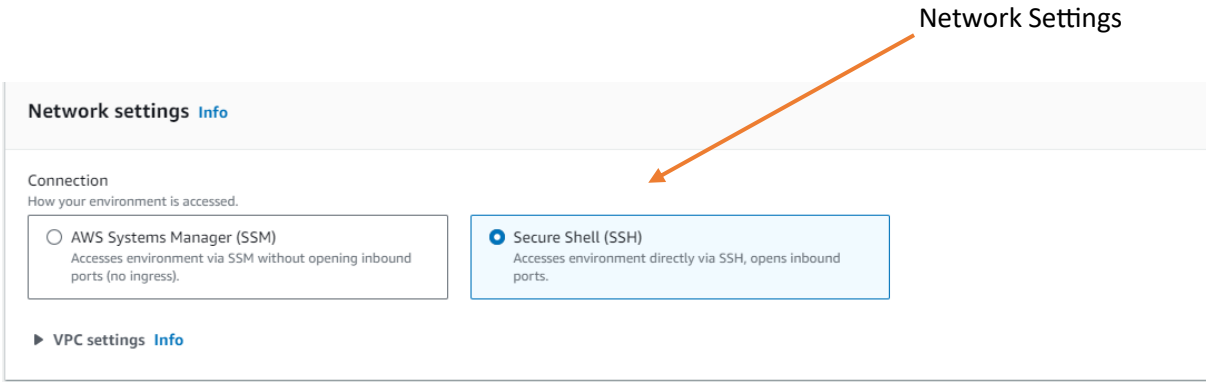


Step 2:

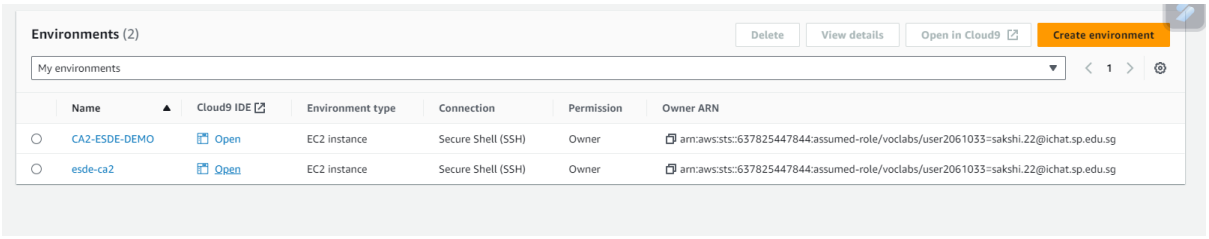
1. Name the **environment** as "esde-ca2".
2. Choose the "SSH" option for the **network settings**.
3. Leave the remaining settings at their **default values**.
4. Press create environment

Environment Name





Step 3: Check the environment esde-ca2 has been created.



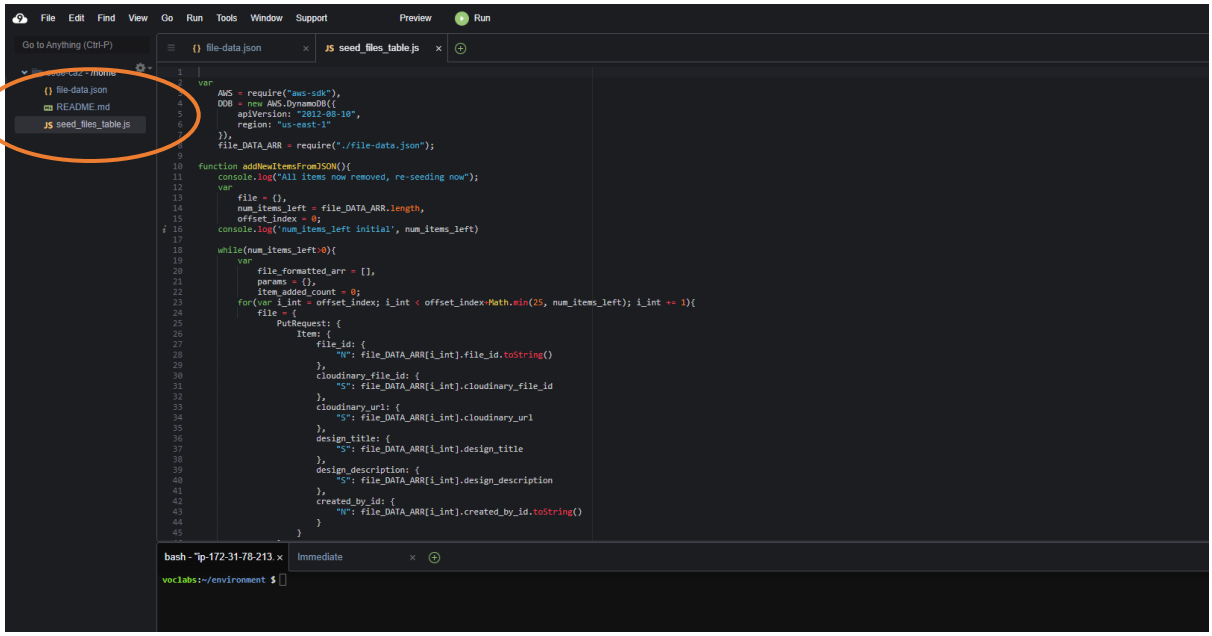
Uploaded file-data.json and seed_files_table.js to Cloud9

Step 1: Open your cloud 9 IDE

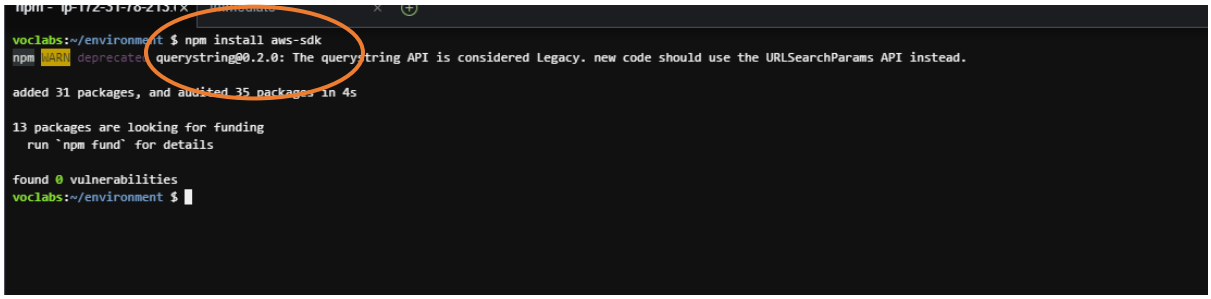
Step 2: From the menu, click on "File" and then select "Upload Local Files".

Step 3: Upload file-data.json and seed_tables.js.

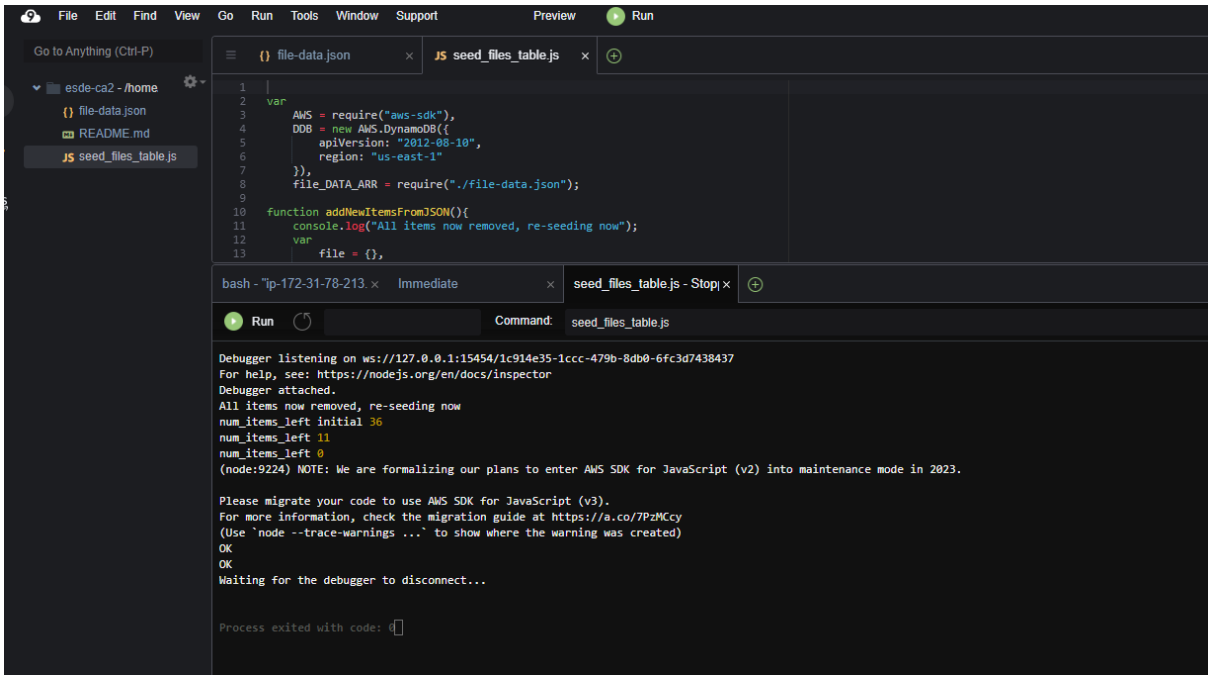
Step 4: Check that they have been uploaded at the left side of the screen



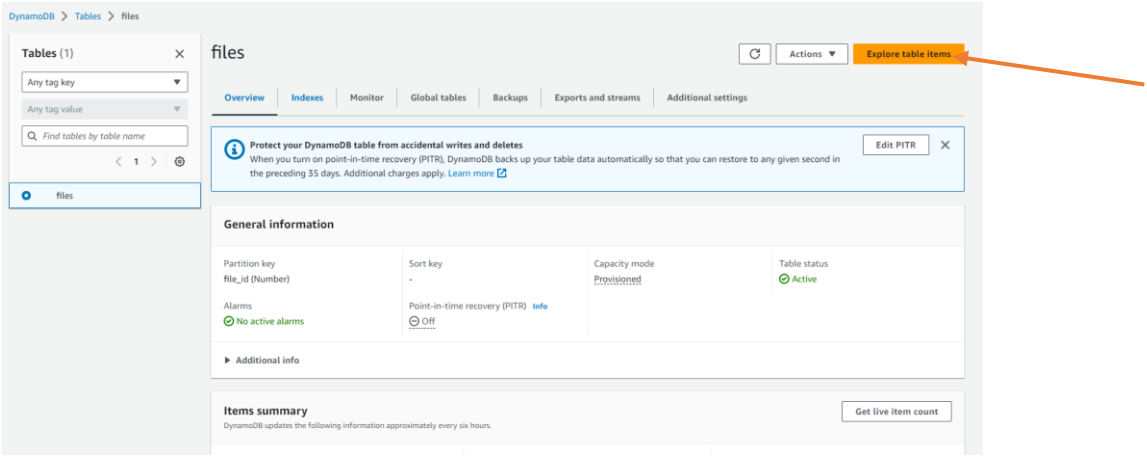
Step 3 : Run npm install aws-sdk in the bash environment.

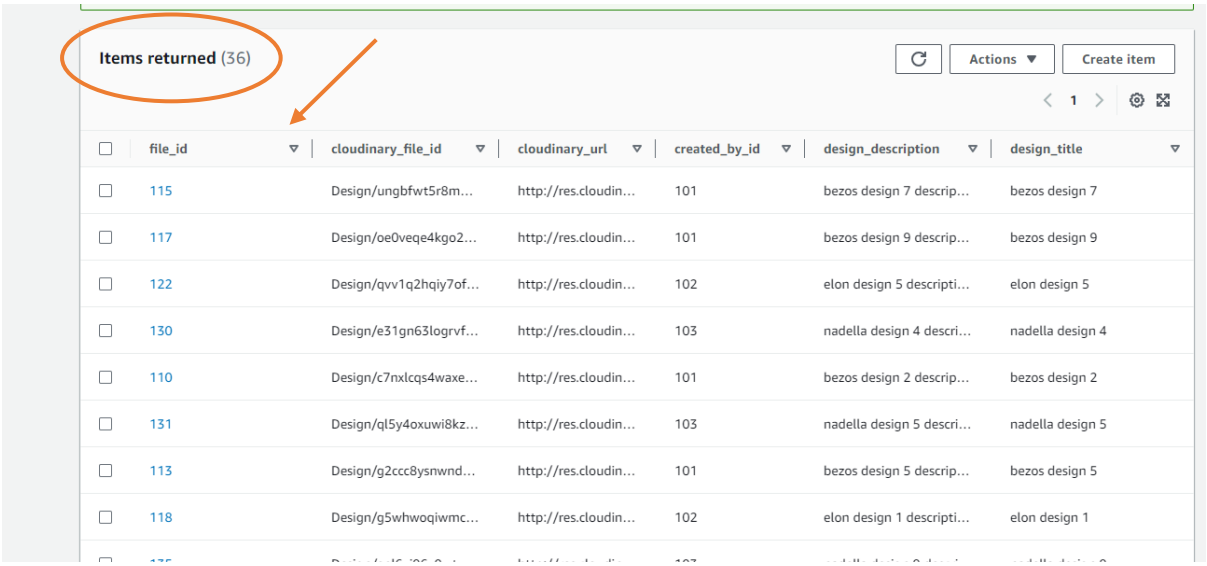


Step 4: Run seed_files_table.js to **upload the records to our DynamoDB files table**.



Step 5: Navigate to DynamoDB console click **“Explore Table Items”,** and check files table that all the record items have been uploaded. There should be **36 items uploaded**.





Part 5- Setup Lambda

Lambda in AWS

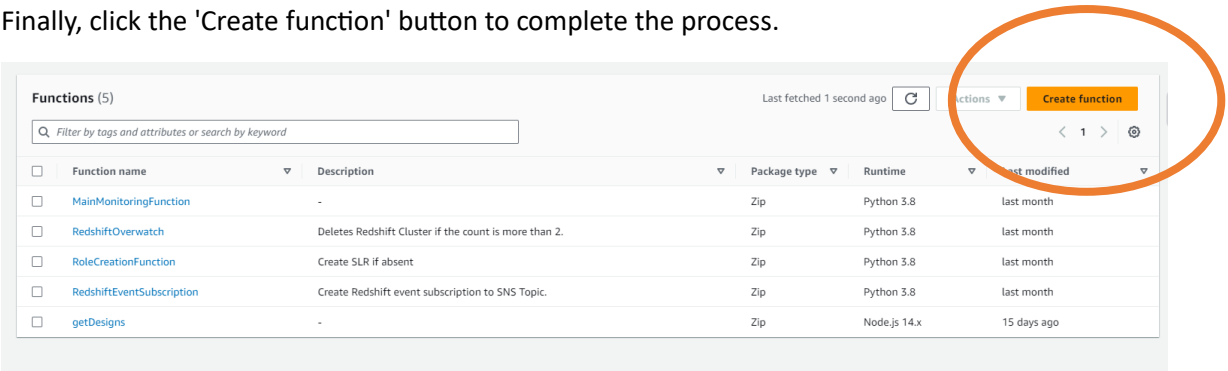
AWS Lambda is a **serverless compute service** that allows you to run code in response to events without managing servers. Its

It's commonly used for data processing, building APIs, IoT, automation, and microservices, **offering quick development and reducing administrative tasks.**

Create Lambda function.

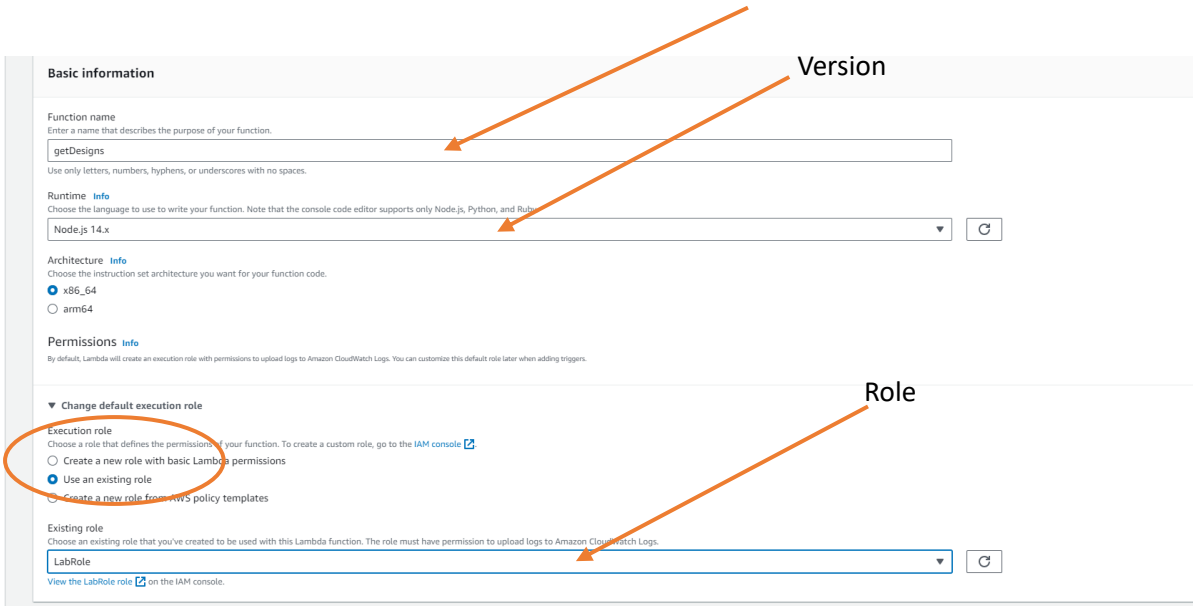
Step 1: Navigate to the Lambda dashboard and click on the "Create Function" button.

Finally, click the 'Create function' button to complete the process.

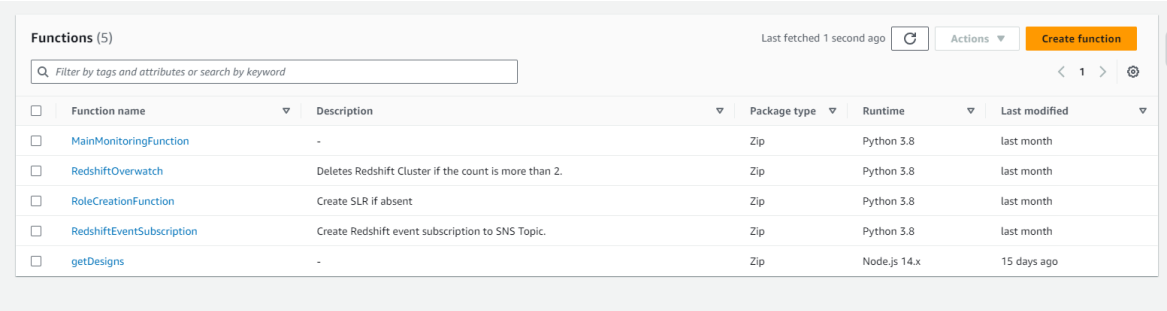


Step 2: Follow these steps

1. Name your function as "get Designs".
2. Make sure to select Node.js v14.x as the runtime.
3. Under the "Permissions" section, click on 'Change default execution role' to choose an existing role.
4. From the dropdown list, select 'LabRole'.
5. Click Create Function

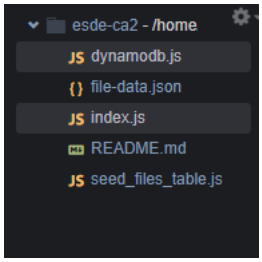


Check that your function has been created

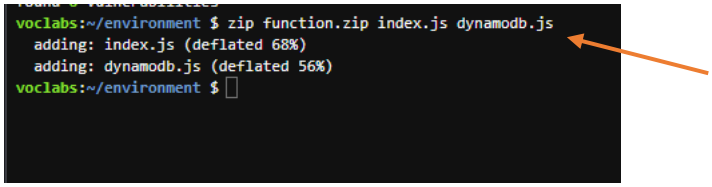


Prepare Lambda codes in cloud 9

Step 1: Upload the **index.js and dynamodb.js** to cloud9



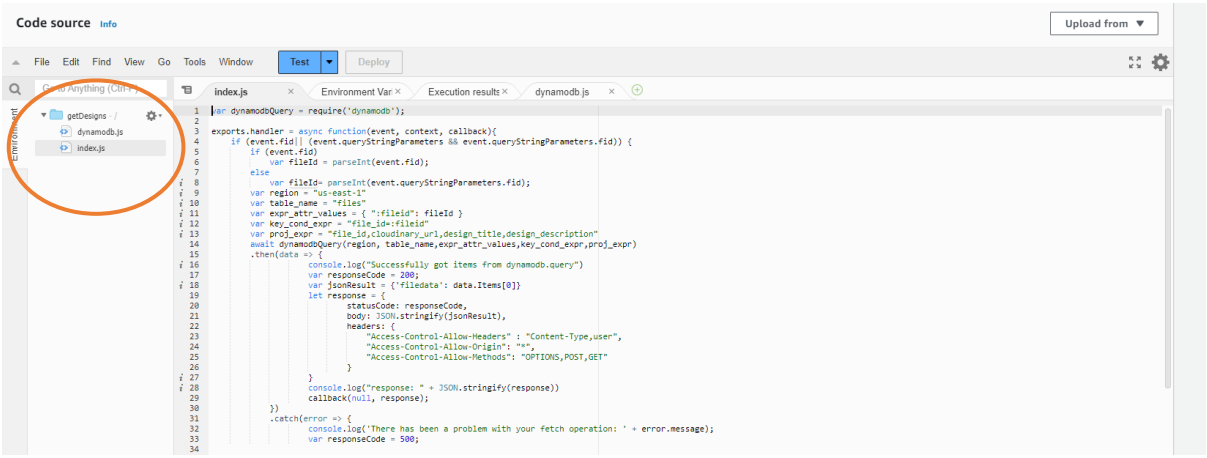
Step 2: Run **zip function.zip index.js dynamodb.js** to put them in a zip file



Upload Lambda zip using AWS CLI

Step 1: Run **aws lambda update-function-code --function-name getDesigns --zip-file fileb://function.zip**, this will upload the zip file to the getDesigns Lamda function.

Step 2: Check that the zip file has been uploaded with index.js and dynamodb.js



Index.js explanation

It retrieves specific data from a DynamoDB table **based on a given file ID**. It handles both direct and query parameter input for the file ID. The **retrieved data is then formatted into a JSON response with necessary headers**, and either a success response with the data or an error response is returned, along with appropriate status 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 = "files"
        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
```

File id query parameter

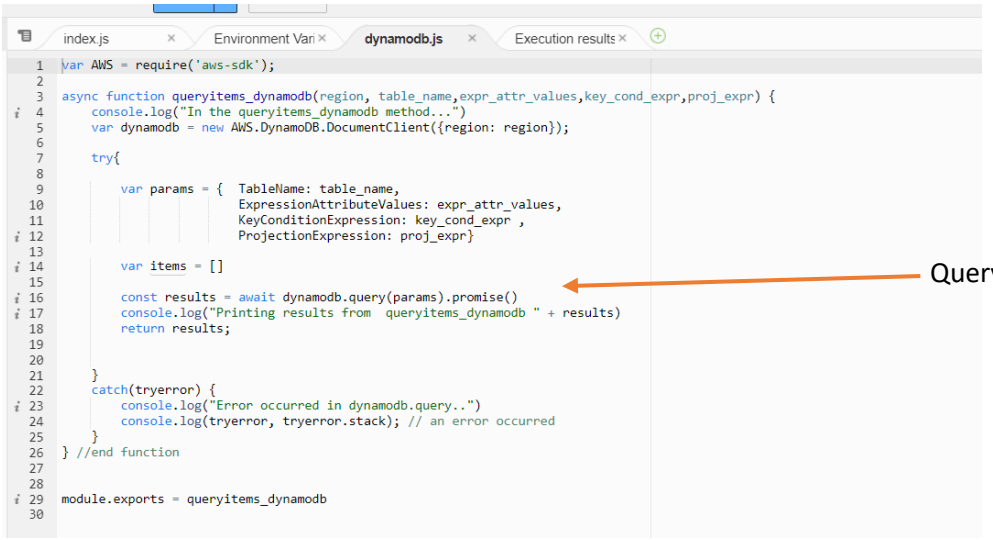
Information retrieved if the file_id

Allow CORS

Error Handling and response

Dynamodbs.js Explanation

This code defines a function called `**queryitems_dynamodb**` that queries an AWS DynamoDB table using provided parameters. It uses the AWS SDK's `DynamoDB.DocumentClient` to **execute the query asynchronously and returns the results on success or logs errors if they occu**r. This function is exported for use in other parts of the code and index.js.



Create test event.

To create a test event in the Lambda code console, follow these steps:

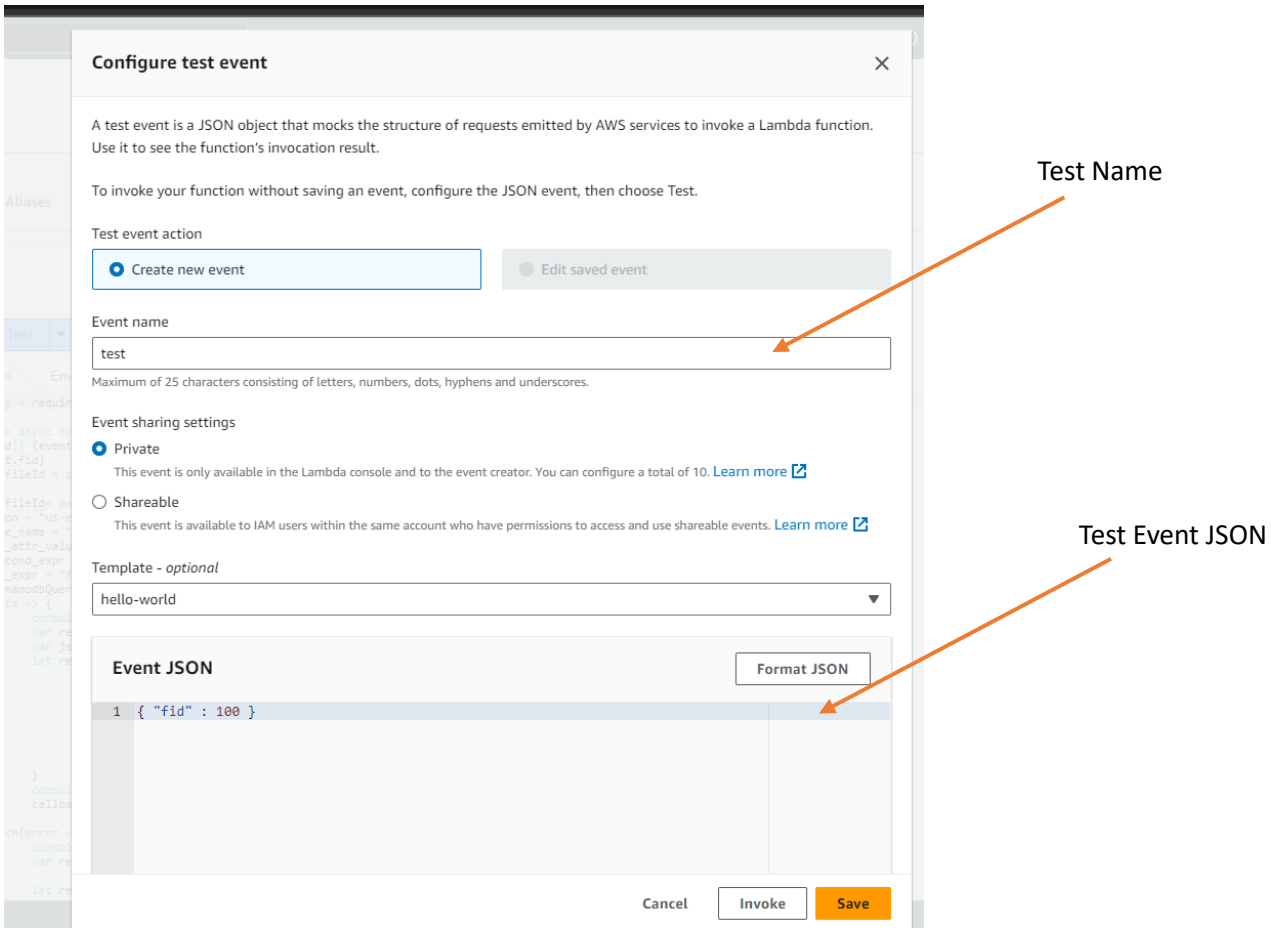
Step 1: Press `Ctrl + Shift + C` in the Lambda code console to open the **"Configure test events"** panel.

Step 2: Name the event as "test".

Step 3: Replace the existing event JSON with the following **{ "fid": "your_file_value" }**

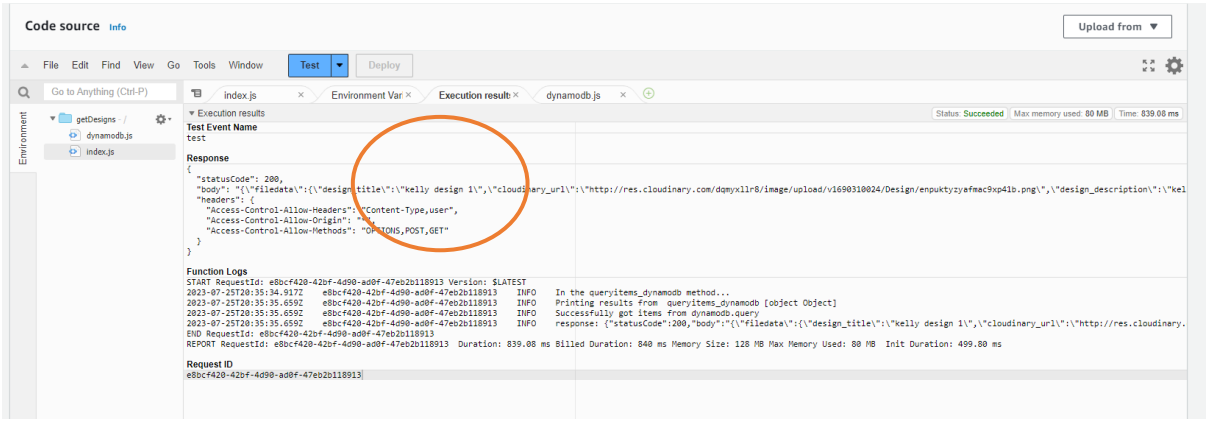
Step 4: Replace `"your_file_value"` with an **actual file value from your AWS files table**.

This test event will **simulate an event with the specified JSON data** when testing your Lambda function as for this case **getting the design information for file_id =100.**

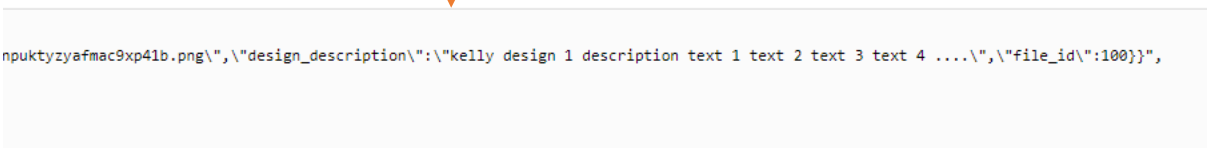


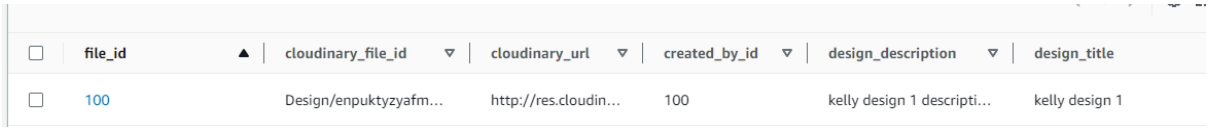
Step 5: Check that the correct file id has been retrieved and the associated information such as **design title , design description , cloudinary url, cloudinary id , created_by_id**

If you receive the values of Kelly design 1. You have successfully set up your getDesigns Lamda function



Information retrieved correctly from DynamoDB





file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
100	Design/enpuktyzyafm...	http://res.cloudin...	100	kelly design 1 descripti...	kelly design 1

Part 6- API Gateway

Api Gateway in Aws

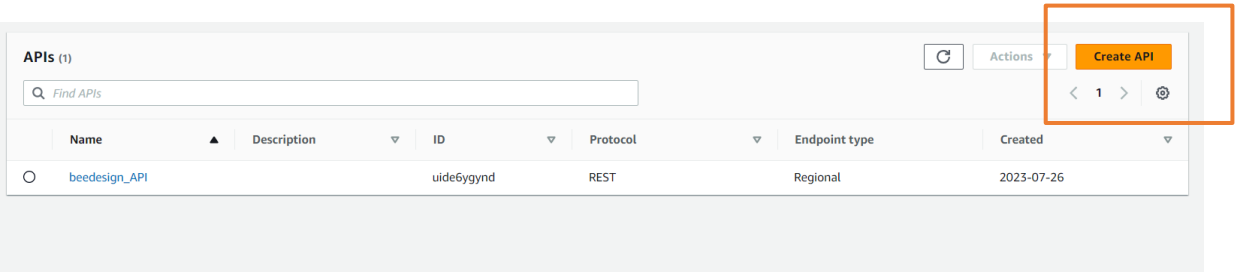
Amazon API Gateway in AWS lets you easily create, manage, and secure APIs. It **acts as a bridge between your applications and backend resources**, offering features like **backend integration**, security controls, rate limiting, caching, monitoring, and documentation. It's ideal for building RESTful APIs, supporting various use cases from serverless to traditional architectures.

Relation to Bee Design

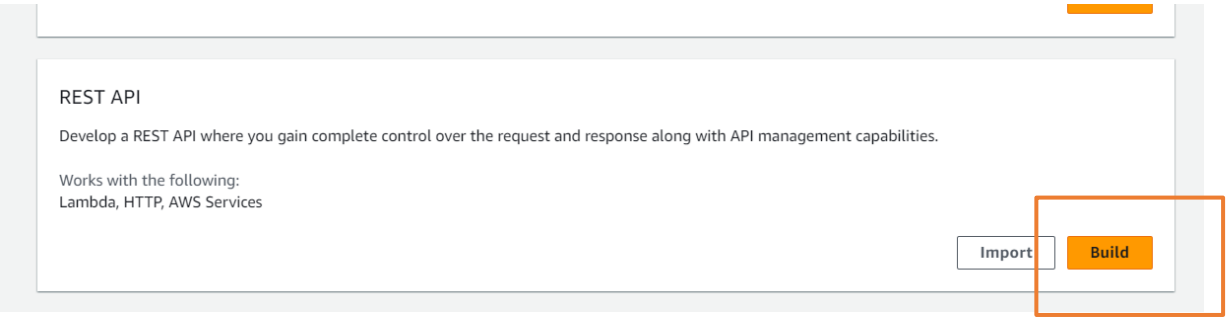
- In this case of Bee design , **Api Gateway receives incoming requests** from clients and forwards them to the appropriate AWS Lambda function based on the **defined API endpoints and methods**.

Create Bee Design Api

Step 1: Access the API Gateway dashboard and click “create API”.



Step 2: Click the "Build" button located under the "REST API" section.

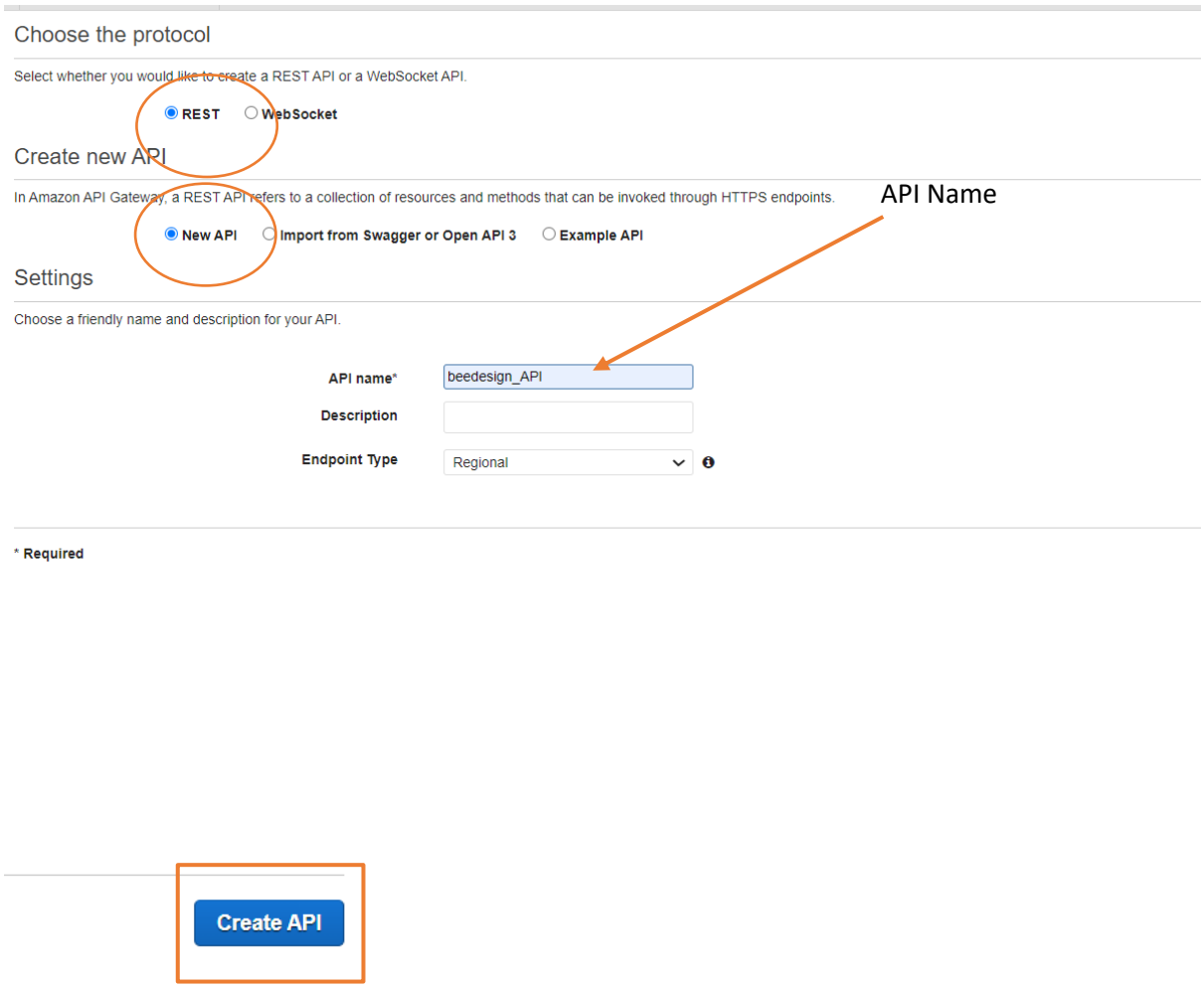


Step 3: On the subsequent screen, maintain the protocol as **REST**, and opt for the **'New API"**

- Selecting a REST API in Amazon API Gateway ensures standardized interactions, stateless communication, caching, and resource-based structuring.

Step 4: Assign the name **"beedesign_API"** to the new API.

Step 5: **Retain the default settings** for other configurations and proceed by clicking the **"Create API"** button.



Create and configure get Designs Method

Step 1: Create a Method for GET Request

1. **Click the '/' button** in the upper left corner of the API Gateway interface.

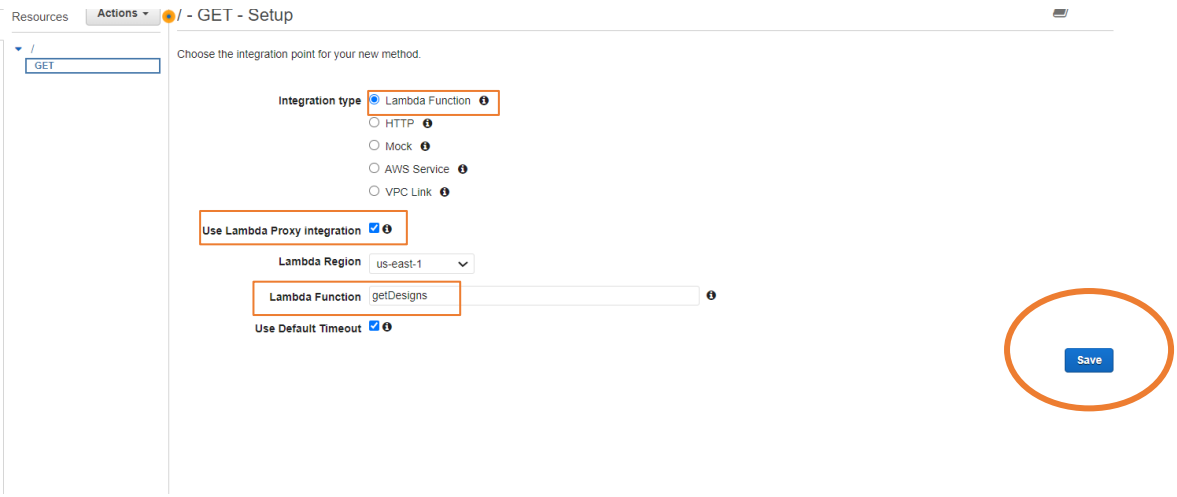
Step 2: Configure the Method

1. From the "Actions" menu, opt for **"Create Method"**.
2. **Select "GET"** from the dropdown list and ensure to save by **clicking the checkmark icon**.

Step 3: Set Up Integration

1. Leave the **Integration Type** as **"Lambda".**
2. **Checkmark** the **"Use Lambda Proxy integration"** option.
3. In the Lambda Function input field, enter **'getDesigns'**.

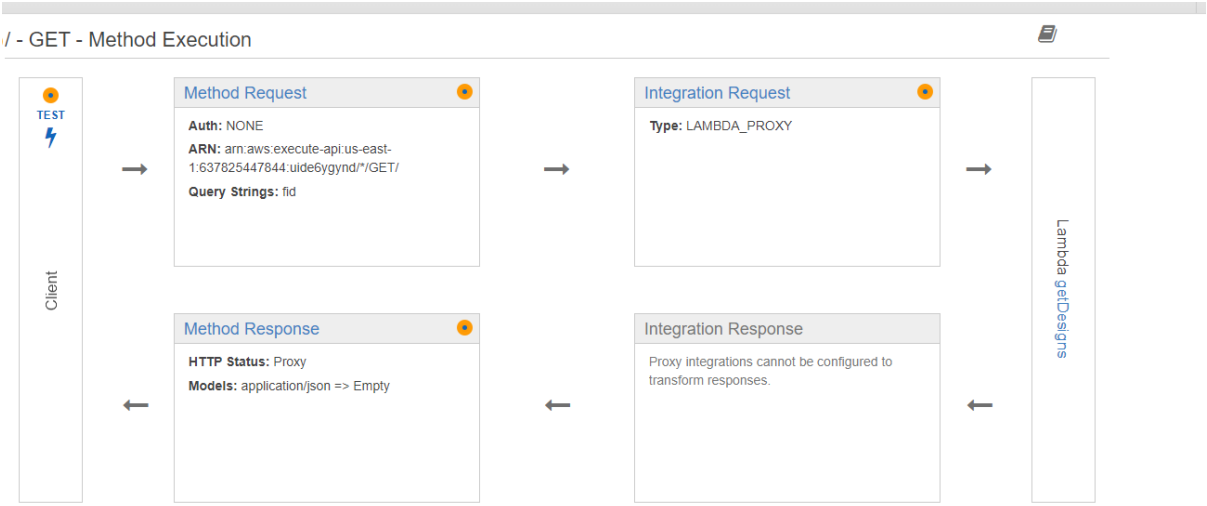
Step 4: Proceed by clicking the 'Save' button.



Method Request

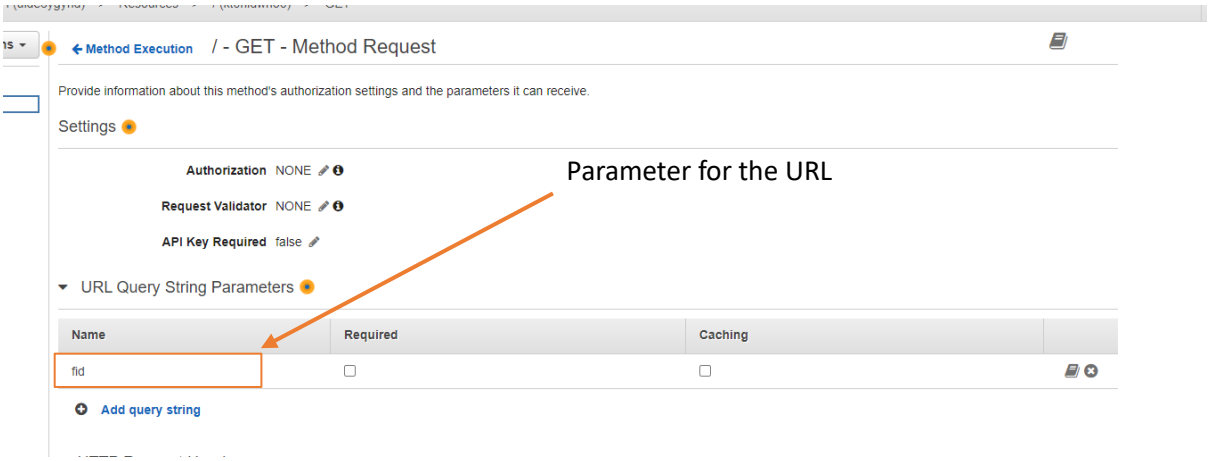
Now we will be **configuring the method request** of our getDesigns Get method.

Step 5: Click on Method Request.



Step 6: Configure Method Request

1. **Expand the URL String Query Parameter** by clicking the triangle icon.
2. Click **'Add query string'.**
3. **Specify the Name as 'fid'**, then save by clicking the checkmark icon.
4. Ensure you remain on this page; do not close it.

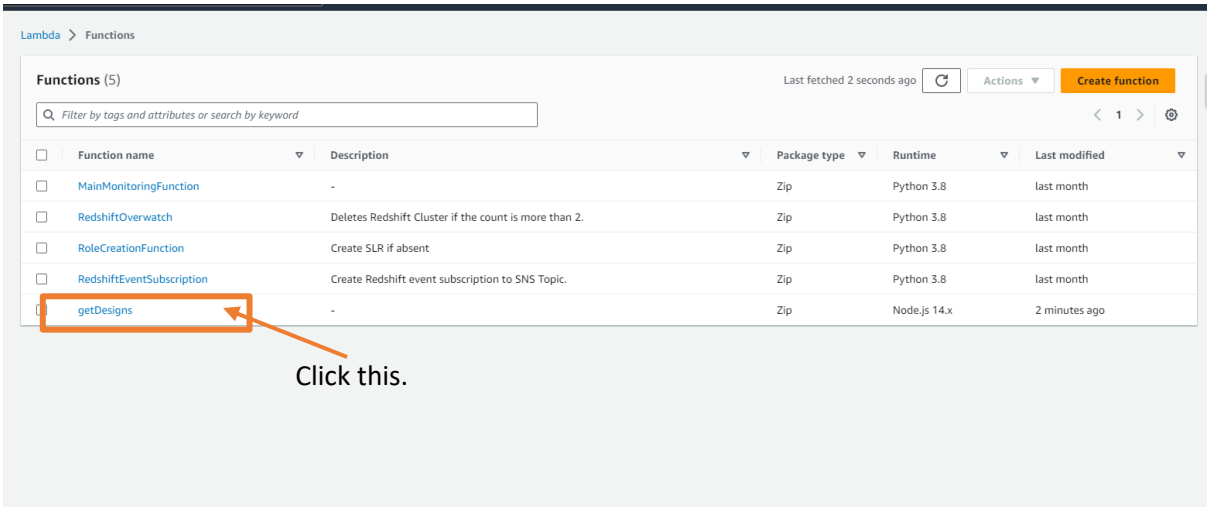


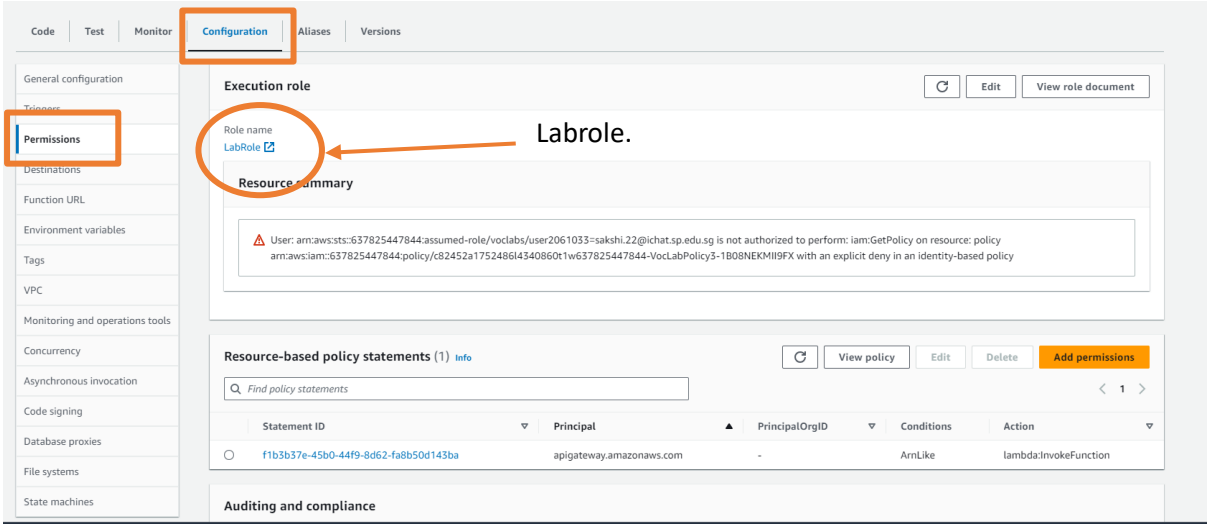
Integration Request

Now we will be **configuring the Integration request** of our getDesigns Get method.

Step 1: Obtain Execution Role for Lambda

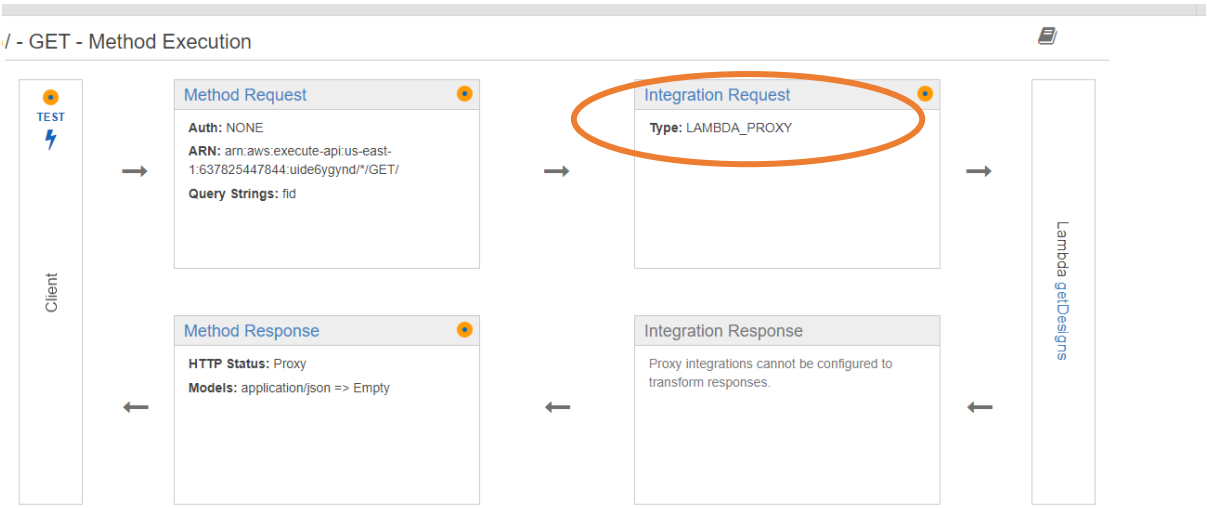
1. Open a new tab for Lambda.
2. Navigate to your **'get Design'** function within Lambda.
3. Click on the **'Configuration'** tab.
4. Under the 'Configuration' tab, choose **'Permissions'**.
5. You'll find the **'LabRole'** listed under Execution Role. Click on 'LabRole' to open it in a new window.
6. Copy the **ARN value** of 'LabRole' and save it in a Notepad for the next step.





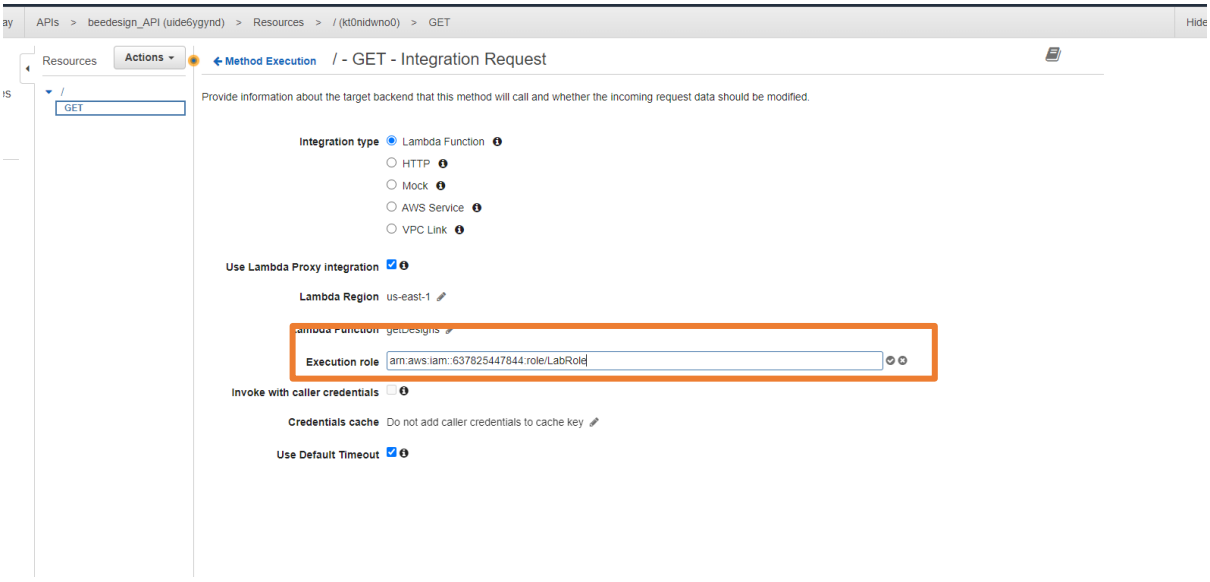
Arn of Labrole : **arn:aws:iam::637825447844:role/LabRole**

Step 2: Go back to Method Execution and Click on **Integration request.**



Step 2: Configure Integration Request

1. Set the **Execution role to 'LabRole'.**
2. Paste the **Labrole Arn Value *arn:aws:iam::637825447844:role/LabRole*** **in Execution role** and click the **checkmark.**



Enable Cors

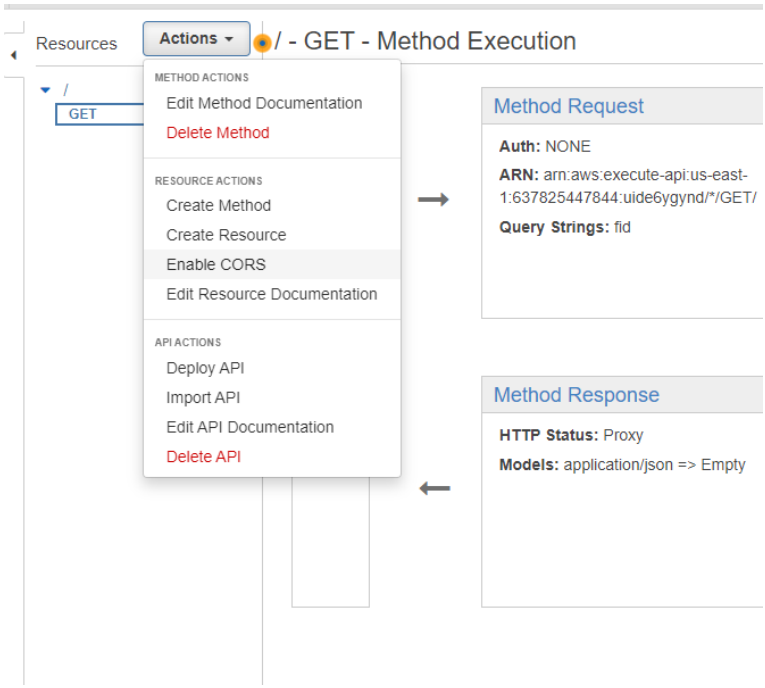
Cors Importance

Enabling CORS in Amazon API Gateway is **vital for secure cross-origin access to API resources**. It supports third-party integration, enhances user experience, and aligns with modern web standards to prevent unauthorized data access.

Relation to Bee Design

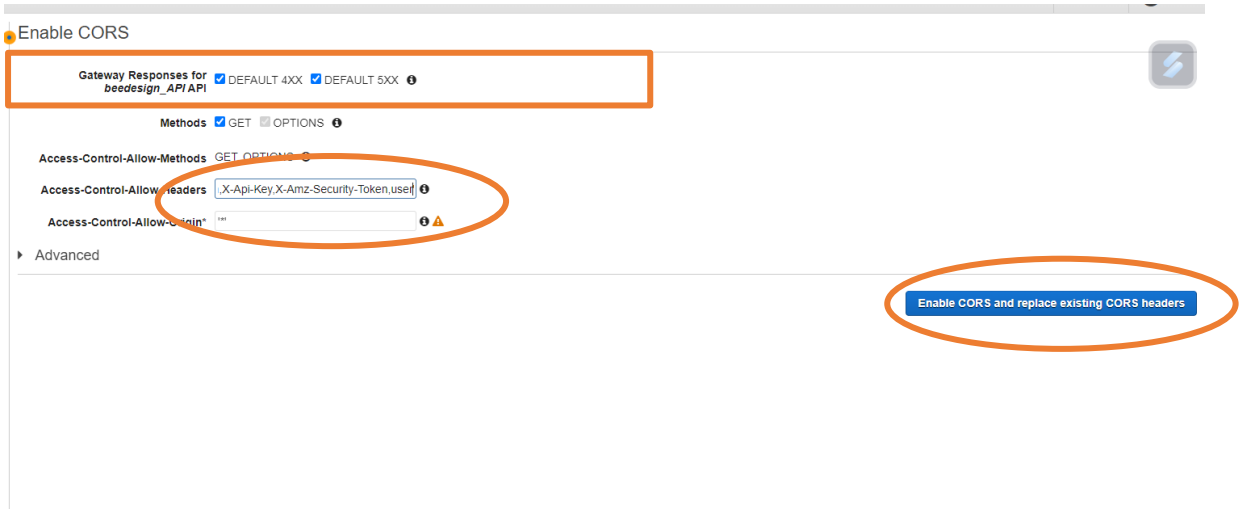
- Enabling Cross-Origin Resource Sharing (CORS) allows other domains to interact with resources such as the **getDesigns function and the website host** .

Step 1: Click on **Actions** and choose **“Enable CORS’.**

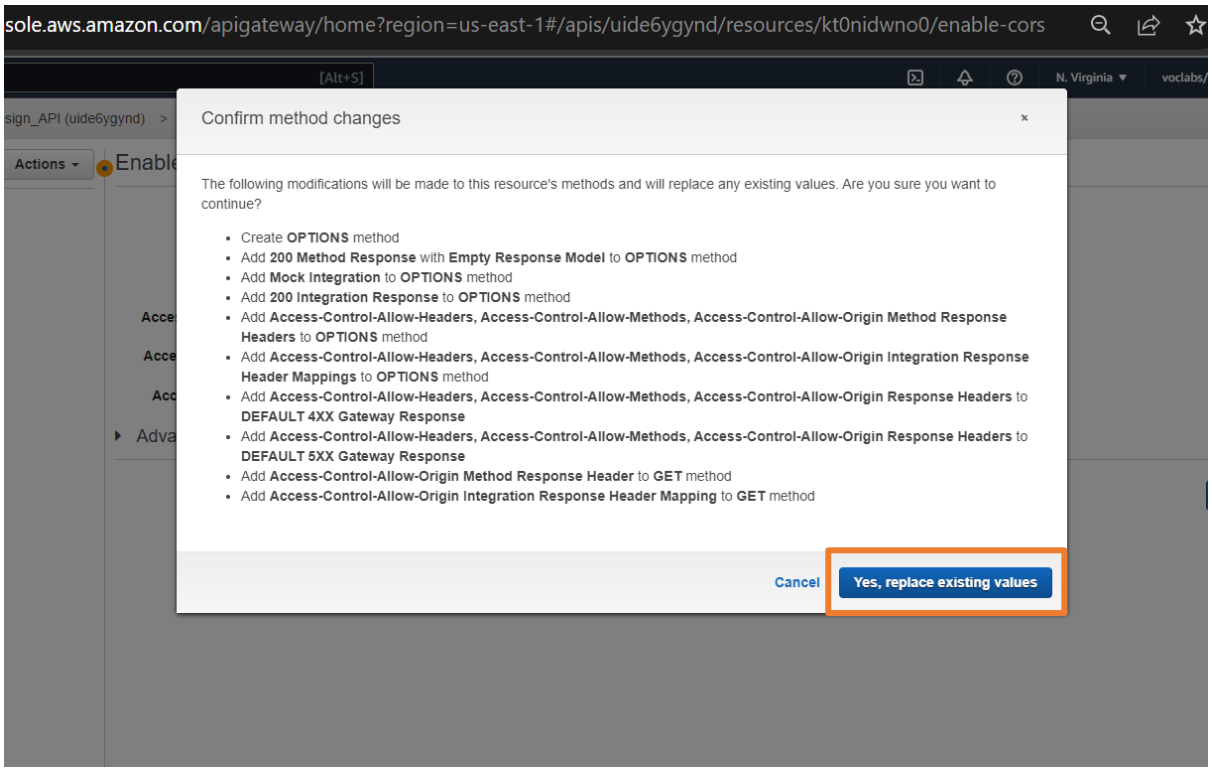


Step 2: Configure Cors Settings

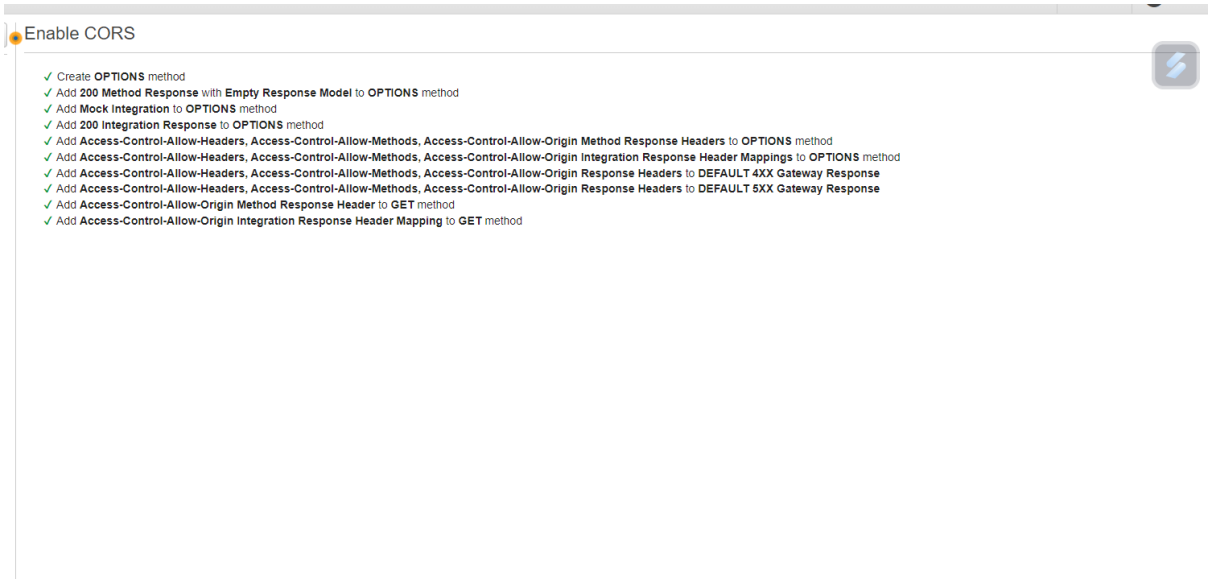
1. Checkmark **DEFAULT 4XX** and **DEFAULT 5XX**
2. Add **', user'** to the end of the existing string for the "Access-Control-Allow-Headers". This modification **expands the allowed headers, including 'user'**, which is **useful for handling CORS requests involving user-related data.**
3. Click **"Enable CORS and replace existing CORS header"** to save your changes and enable cors.



Step 3: Confirm it by pressing **"Yes, replace existing values"**



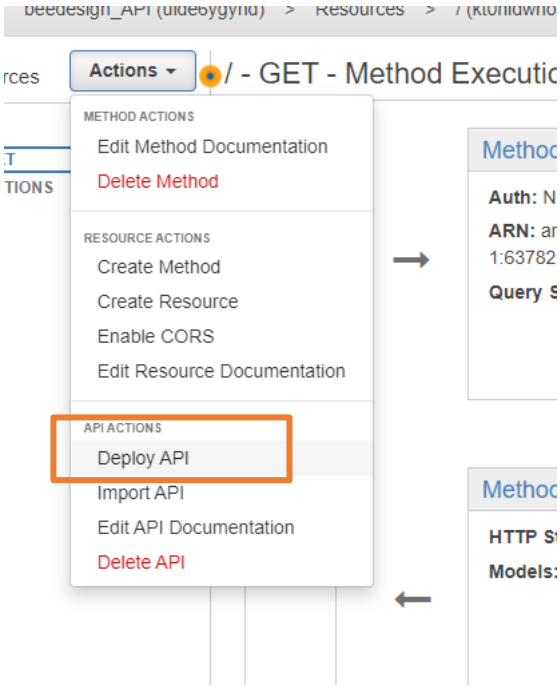
Cors is successfully enabled.



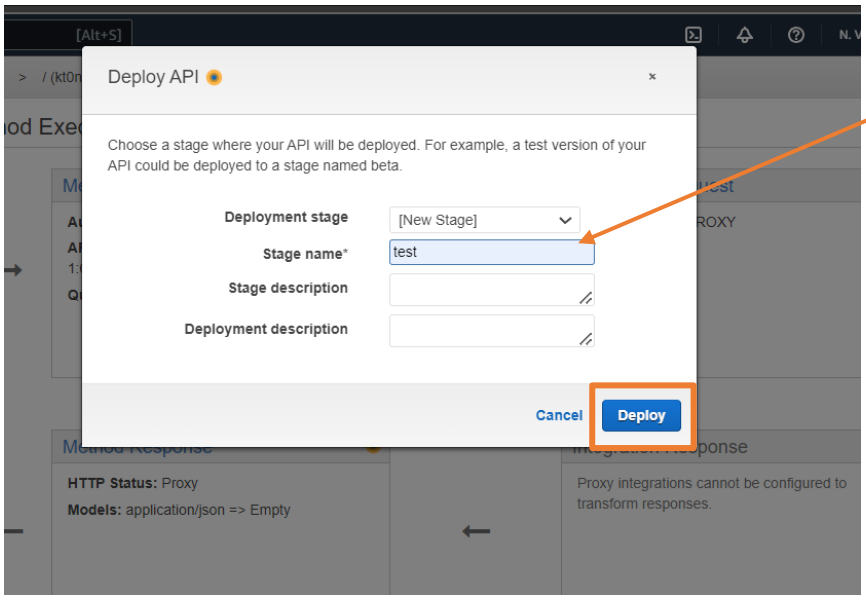
Deploy API

Deploying an API for the method in API Gateway is necessary to make your API accessible to clients. It involves creating a live version of your API configuration, allowing users to interact with your endpoints and access the resources your method provides. Such as our Bee design website interacting with the API, lambda and DynamoDB.

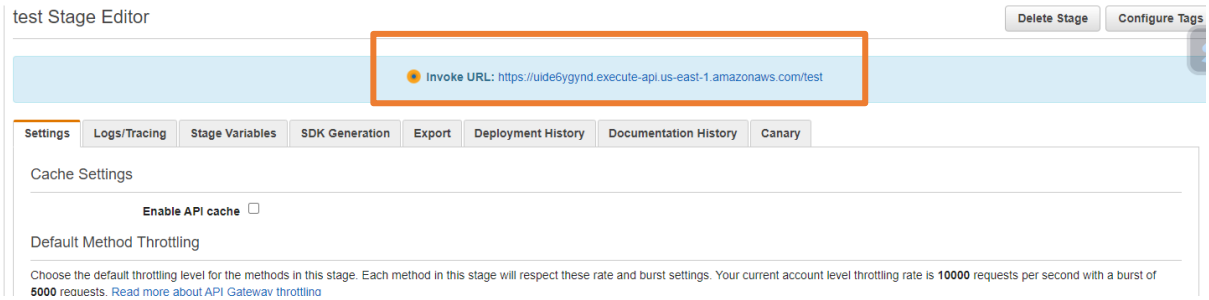
Step 1: Click Actions and click “Deploy Api”.



Step 2: Create new **test** stage and click “Deploy”.



Step 3: Copy the **value of the API end-point** into Notepad



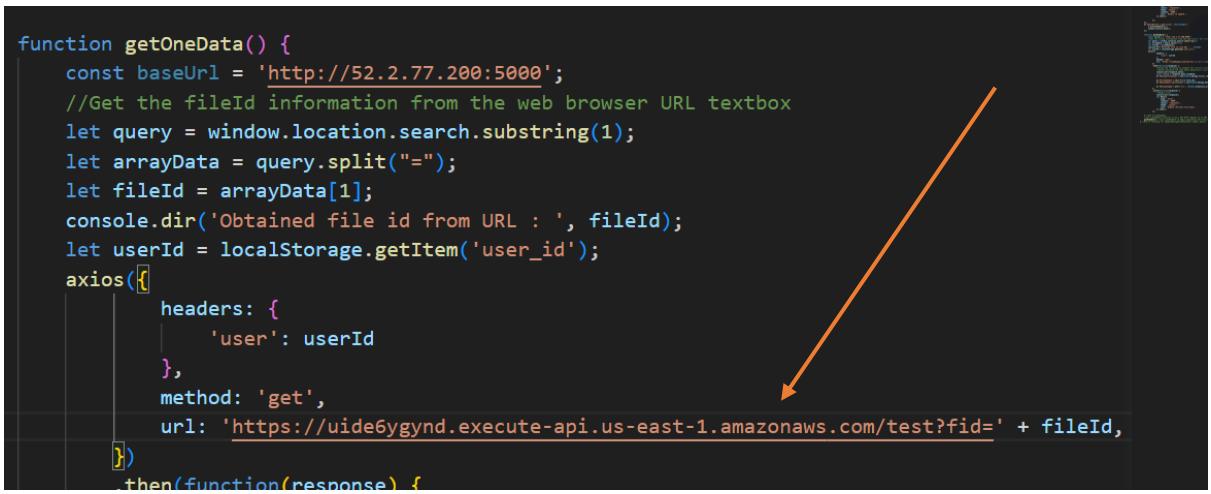
Modify App code to use API endpoint.

Step 1: Edit the **'frontend/public/js/update_design.js' file**:

Step 2: Replace 'localhost' in two instances with your EIP (Elastic IP address) .

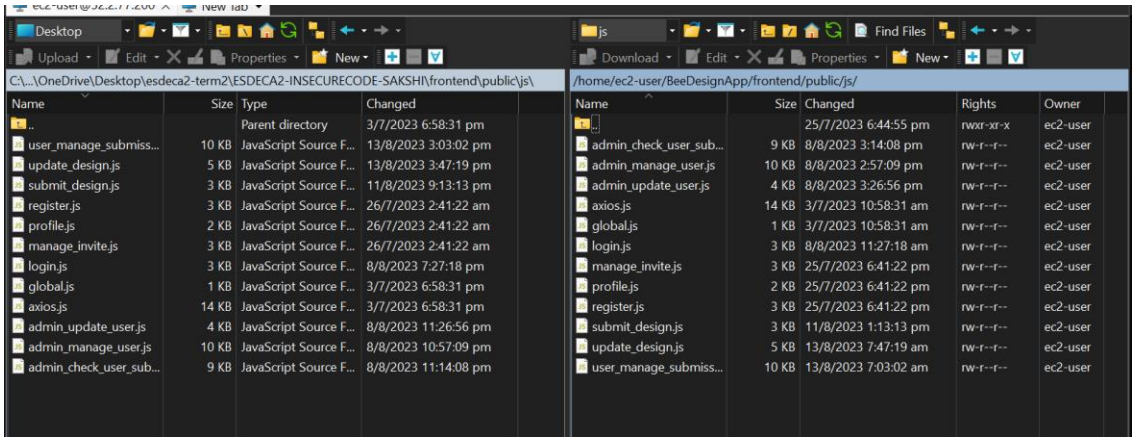
Step 3 : Change the URL to **https://uide6ygynd.execute-api.us-east1.amazonaws.com/test?fid=fileId.**

The API URL specifies the location where your frontend should send requests to interact with the backend services, including the AWS Lambda function.



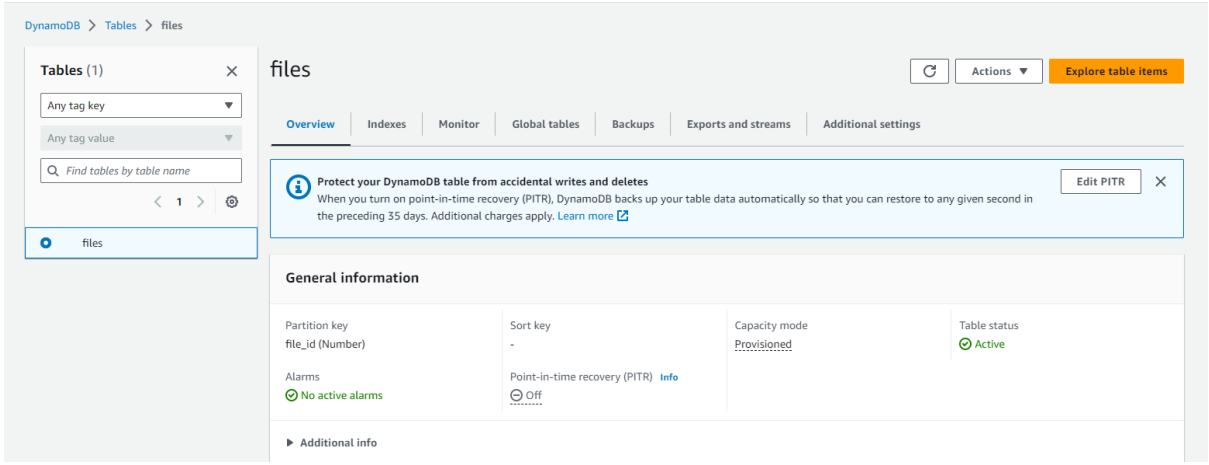
```
function getOneData() {
    const baseUrl = 'http://52.2.77.200:5000';
    //Get the fileId information from the web browser URL textbox
    let query = window.location.search.substring(1);
    let arrayData = query.split("=");
    let fileId = arrayData[1];
    console.dir('Obtained file id from URL : ', fileId);
    let userId = localStorage.getItem('user_id');
    axios({
            headers: {
                'user': userId
            },
            method: 'get',
            url: 'https://uide6ygynd.execute-api.us-east-1.amazonaws.com/test?fid=' + fileId,
        })
        .then(function(response) {
```

Step 4: **Transfer the update_design.js** file to ec2-user.



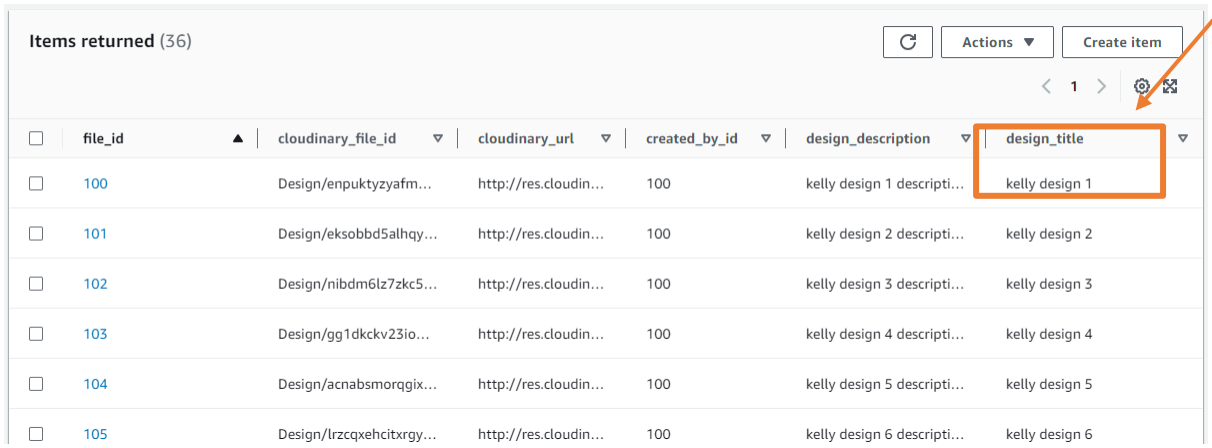
Modify DynamoDB

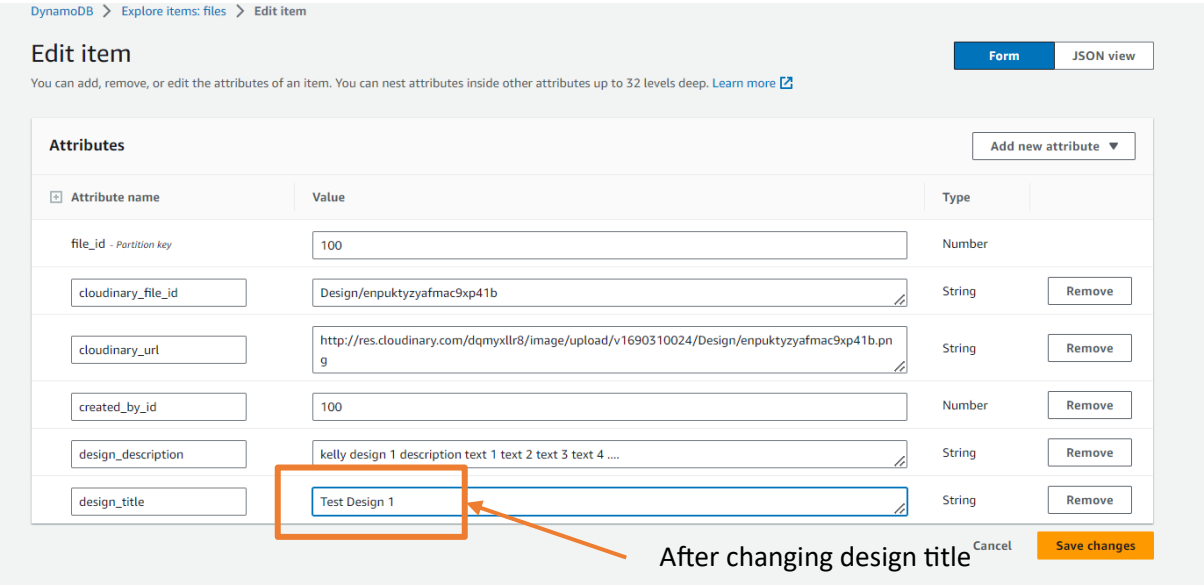
Step 1: Go to the DynamoDB console and press “Explore table items:



Step 2: Modify file_id=100 , design_title to Test design 1,

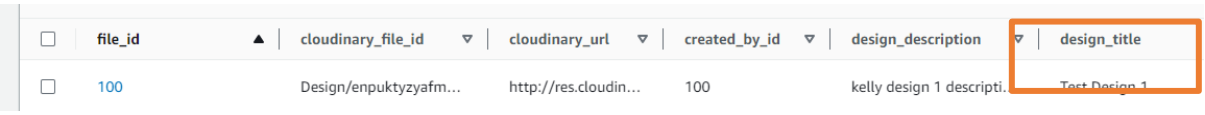
Before





After changing design title in DynamoDB.

This is how file_id=100 should look now , the design title is now Test Design 1 that should reflect when we want to get design file_id=100



Testing the application

We will now test our getDesigns function with our API in our website.

Step 1: Open **two new terminal windows** in WinSCP

Step 2: In one window cd to **frontend** folder. npm start In second window, cd to **backend** folder

Step 3: **Npm start** in both directories.

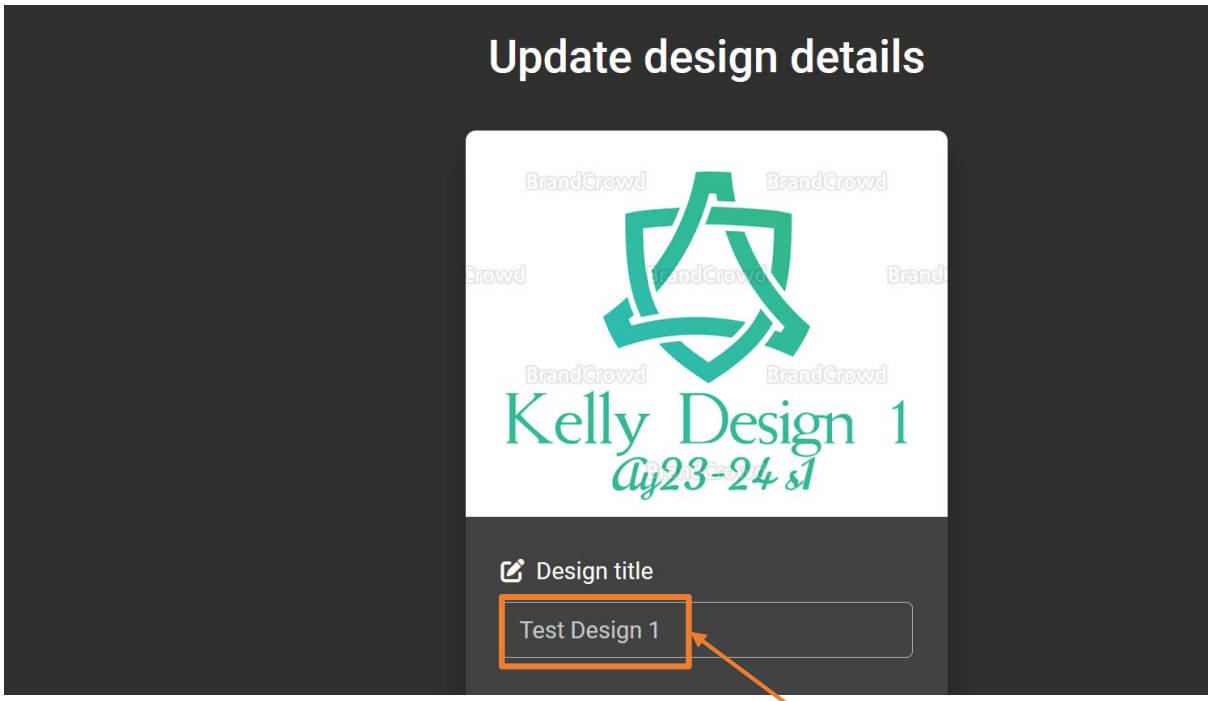
Backend folder

```
^C[ec2-user@ip-172-31-90-105 frontend]$ cd ..
[ec2-user@ip-172-31-90-105 BeeDesignApp]$ cd backend/
[ec2-user@ip-172-31-90-105 backend]$ npm start

> xyz@1.0.0 start
> nodemon index.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server is Listening on: http://localhost:5000/
```

Frontend folder

```
Last login: Tue Jul 25 18:47:06 2023 from 101.127.136.109
[ec2-user@ip-172-31-90-105 ~]$ cd BeeDesignApp/
[ec2-user@ip-172-31-90-105 BeeDesignApp]$ cd frontend/
[ec2-user@ip-172-31-90-105 frontend]$ npm start

> firstfrontend@1.0.0 start
> nodemon index.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server hosted at http://localhost:3001
```

Step 4: If the record you modified displays the value **"Test Design 1"** from the previous step, then you can consider this a successful outcome, indicating that both your endpoint and Lambda function are functioning correctly.



Hence the get designs function, user retrieves the file data from DynamoDB when the lambda function is executed upon receiving the URL with the fid query parameter

Additional Features

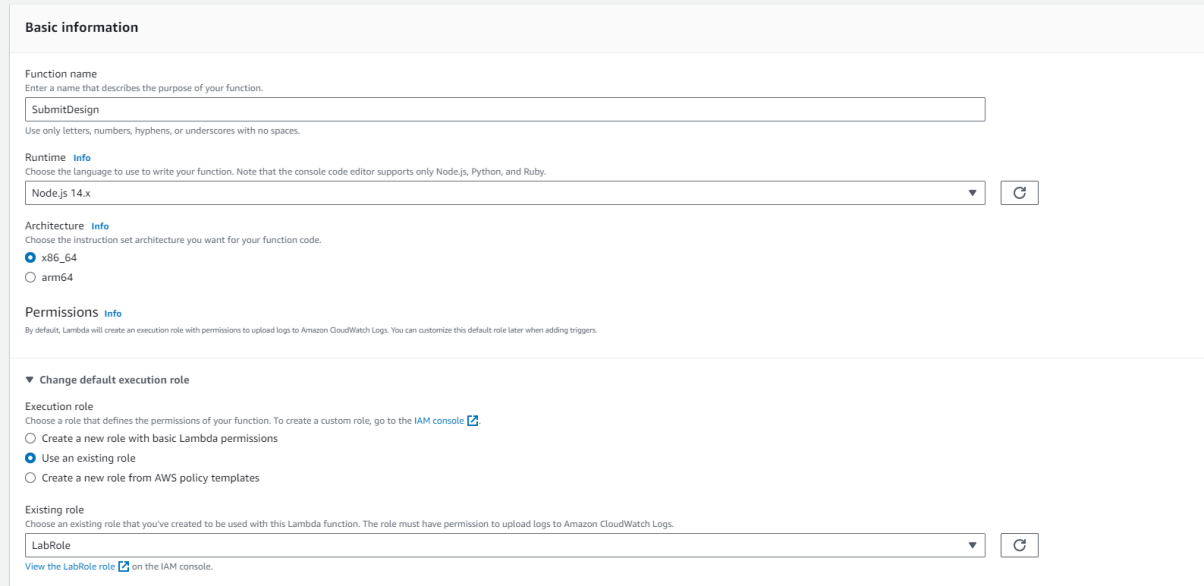
Submit Design (Post)

Submit Design

User can submit new designs by uploading **design title, design description and image.**

Create Lambda Function

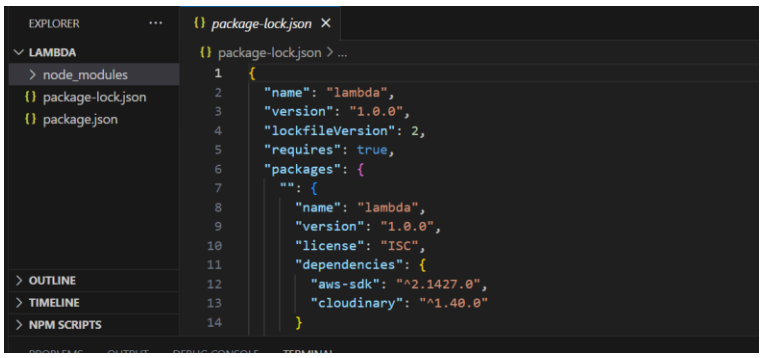
1. Name your function as "Submit Design".
2. Make sure to select Node.js v14.x as the runtime.
3. Under the "Permissions" section, click on 'Change default execution role' to choose an existing role.
4. From the dropdown list, select 'LabRole'.



Create FOLDER in Visual Studio Code

Step 1: Create a folder called “LAMDA” in visual studio code

We will now be installing necessary packages



Step 2: Run Npm init

```
C:\Users\sakshi singh\OneDrive\Desktop\Lambda>npm install

up to date, audited 37 packages in 575ms

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Step 3: Npm install

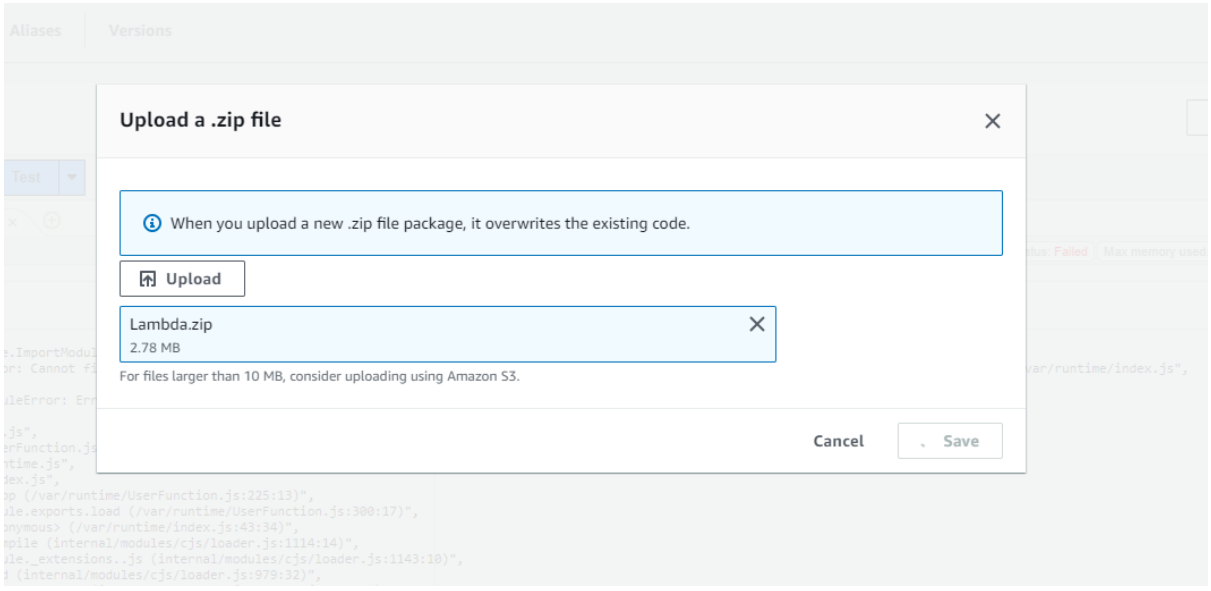
```
C:\Users\sakshi singh\OneDrive\Desktop\Lambda>npm install

up to date, audited 37 packages in 664ms

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Step 4: Npm install cloudinary

```
C:\Users\sakshi singh\OneDrive\Desktop\Lambda> npm install cloudinary

up to date, audited 37 packages in 2s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

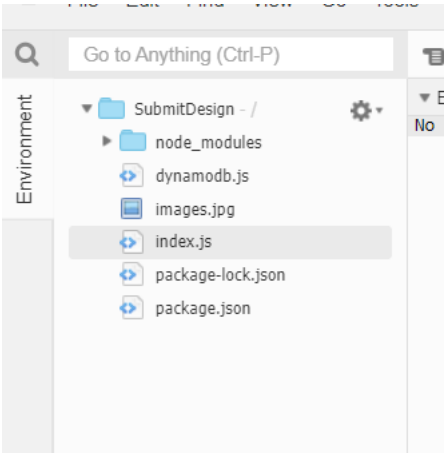
Step 5: Zip the file

Step 6: Upload the “LAMDA” folder into the Submit Design Lambda function.



Step 7: Create Index.js and Dynamodb.js files in the Submit Design function

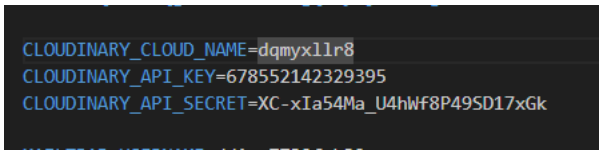
This is how the panel should look like. The node modules and cloudinary is necessary to access cloudinary when we submit design.



Environment Variables

Scripts can directly access environment variables in your code when using AWS Lambda. These environment variables are made available to your Lambda function at runtime, **allowing you to retrieve sensitive or configuration-related information without hardcoding it in your code**. This approach offers **better security** and **flexibility** in managing your application's settings.

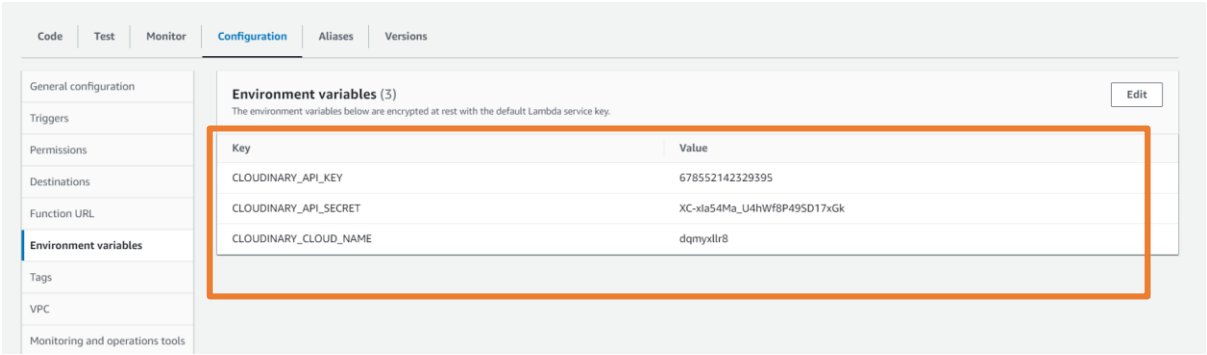
Step 1 : Copy the Cloudinary ids to a notepad .



Step 2: Go to the configuration tab in your Lambda function

Step 3: Click Environment Variables

Step 4: Click Edit and add the 3 key and respective valyes



Dynamodb.js Explanation

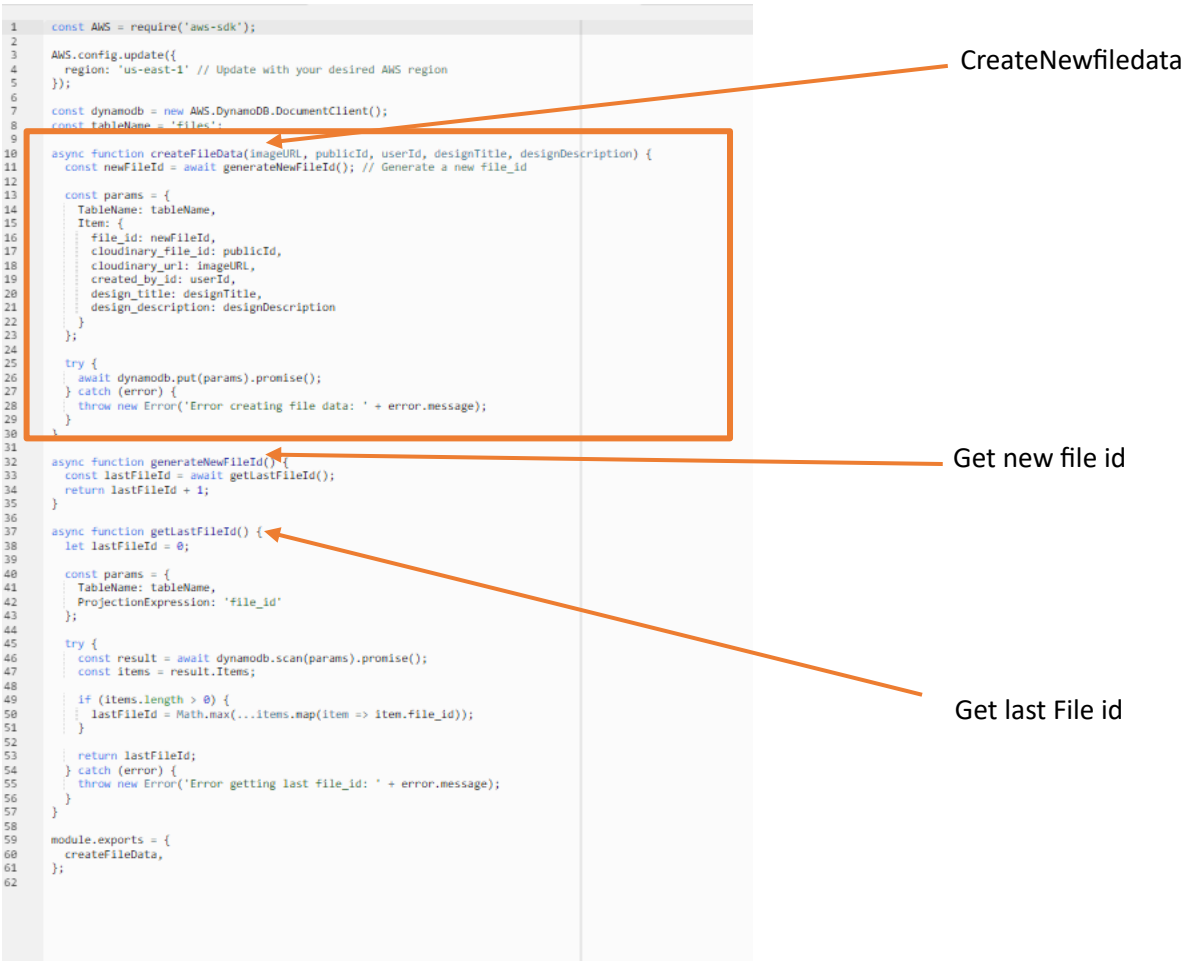
This module interfaces with AWS DynamoDB for file data creation. It includes functions to:

1. **createFileData**: Creates a new entry with given data, including generating a new file ID.

2. **generateNewFileId**: Calculates a new file ID.

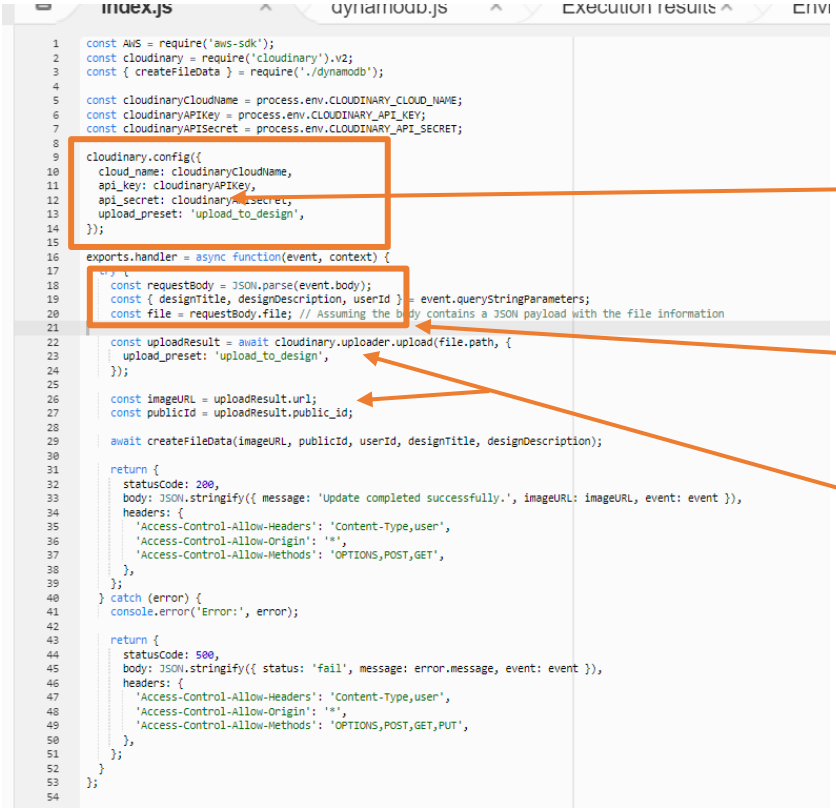
3. **getLastFileId**: Retrieves the last file ID from DynamoDB.

These functions collectively manage the storage of file data in DynamoDB.



Index.js Explanation

This AWS Lambda function in Node.js handles image uploads to Cloudinary. It retrieves Cloudinary credentials from environment variables, processes incoming data, uploads the image to Cloudinary, stores metadata in DynamoDB using the `createFileData` function, and responds with success or error messages including necessary headers for cross-origin requests.




```
const AWS = require('aws-sdk');
const cloudinary = require('cloudinary').v2;
const { createFileData } = require('./dynamodb');

const cloudinaryCloudName = process.env.CLOUDINARY_CLOUD_NAME;
const cloudinaryAPIKey = process.env.CLOUDINARY_API_KEY;
const cloudinaryAPISecret = process.env.CLOUDINARY_API_SECRET;

cloudinary.config({
  cloud_name: cloudinaryCloudName,
  api_key: cloudinaryAPIKey,
  api_secret: cloudinaryAPISecret,
  upload_preset: 'upload_to_design',
});

exports.handler = async function(event, context) {
  try {
    const requestBody = JSON.parse(event.body);
    const { designTitle, designDescription, userId } = event.queryStringParameters;
    const file = requestBody.file; // Assuming the body contains a JSON payload with the file information

    const uploadResult = await cloudinary.uploader.upload(file.path, {
      upload_preset: 'upload_to_design',
    });

    const imageURL = uploadResult.url;
    const publicId = uploadResult.public_id;

    await createFileData(imageURL, publicId, userId, designTitle, designDescription);

    return {
      statusCode: 200,
      body: JSON.stringify({ message: 'Update completed successfully.', imageURL: imageURL, event: event }),
      headers: {
        'Access-Control-Allow-Headers': 'Content-Type,user',
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',
      },
    };
  } catch (error) {
    console.error('Error:', error);

    return {
      statusCode: 500,
      body: JSON.stringify({ status: 'fail', message: error.message, event: event }),
      headers: {
        'Access-Control-Allow-Headers': 'Content-Type,user',
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET,PUT',
      },
    };
  }
};
```

Modify app code

Submit design .js

This code binds a form submit event. When the submit button is clicked, it sends a POST request using Axios to a server API. The payload includes design details and the selected file. Upon success, it displays a notification with the server response message.

The payload includes:

- designTitle: Title of the design.

- designDescription: Description of the design.

- userId: User ID.

- file: Object containing the file path.

This code enhances the user experience by sending and handling design submissions with server interactions and notifications.

This is the modified part of the code.

Api URL with query parameters

Then, `file.name` retrieves the `name` property of the selected file, which represents the original name of the file on the user's local system. This name is included in the JSON payload later in the code.

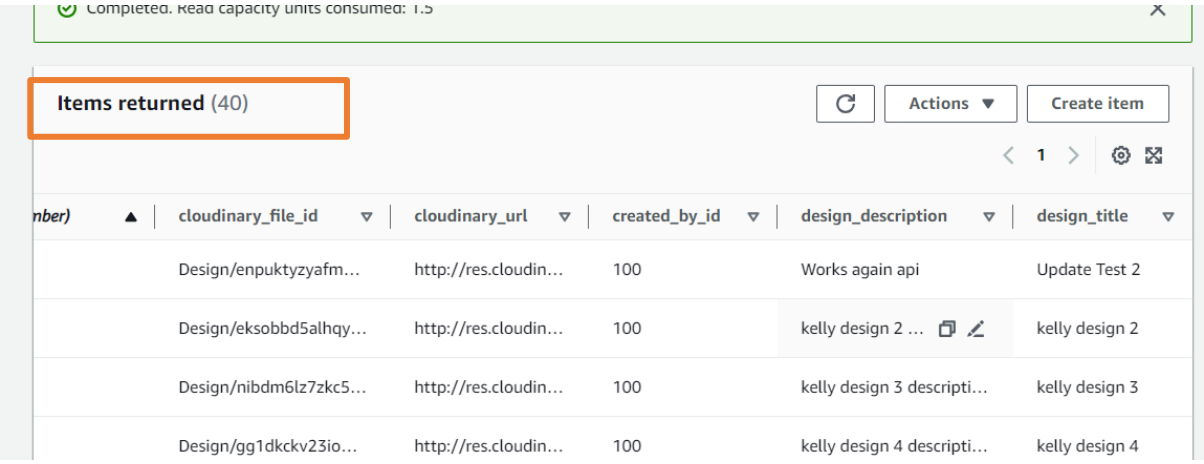
The line `let file = document.getElementById('fileInput').files[0];` retrieves the first selected file from the input element with the ID `fileInput`.

The `files` property of an input element of type `file` holds an array of files selected by the user. By using `[0]`,accessing the first file in that array.

.

Testing the Application

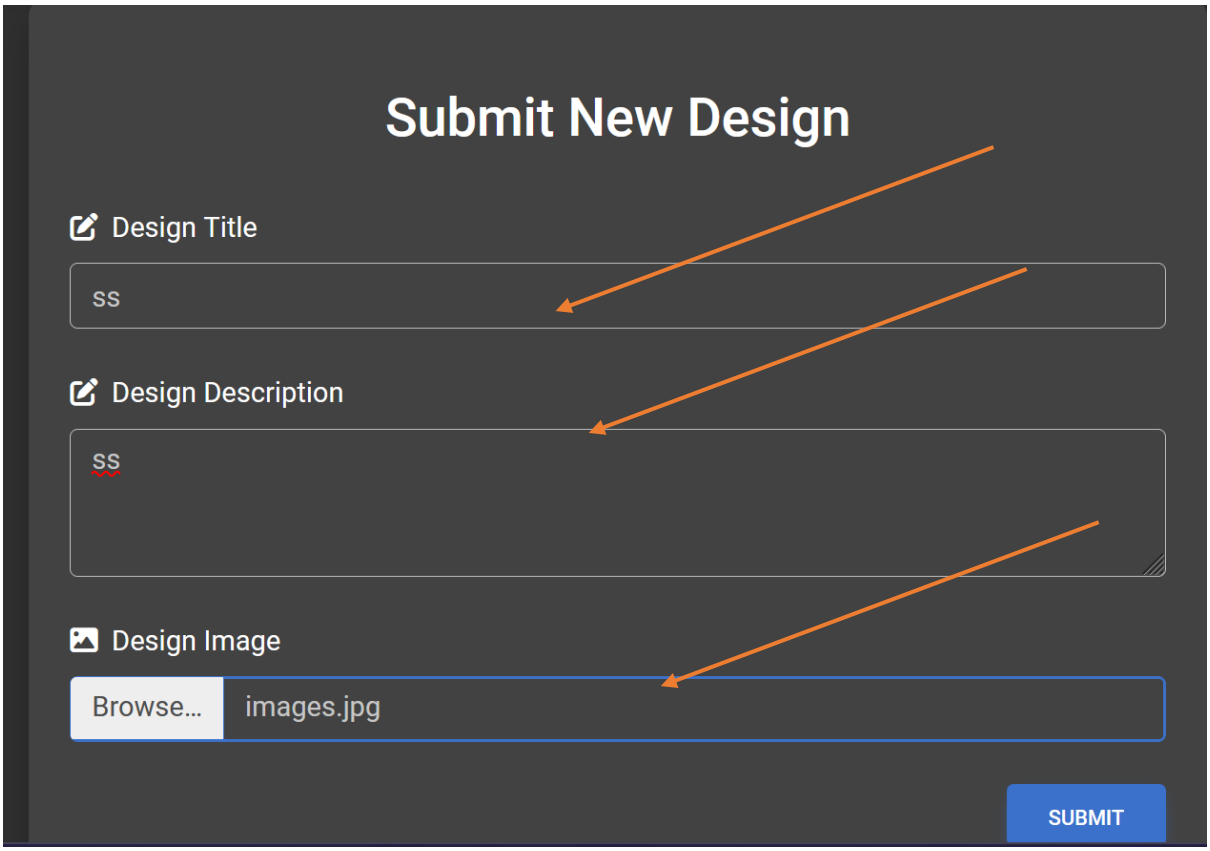
Before we test our application, lets check the item in our DynamoDB files table. There are currently 40 items in the files table.



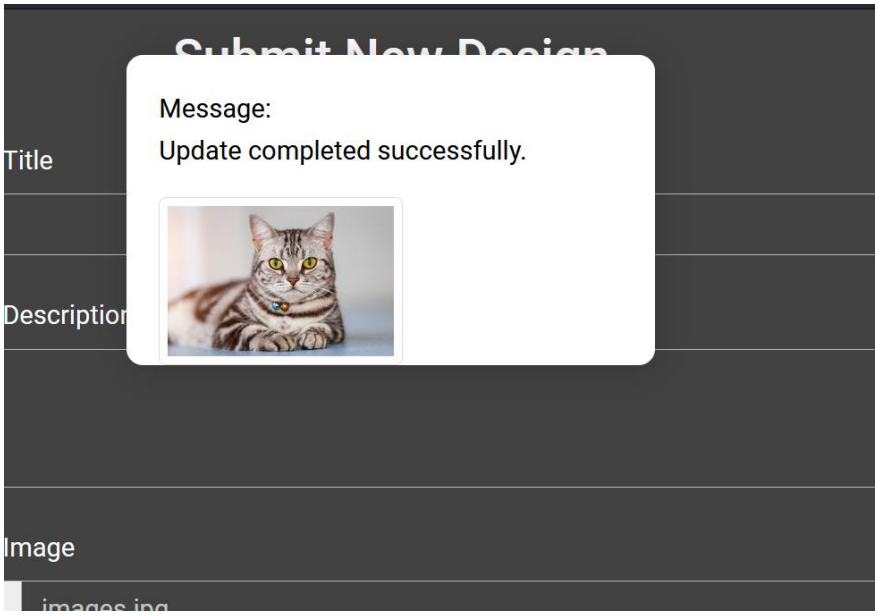
Step 1 : Login as Kelly and navigate to submit new design

Step 2: Enter values for design Title and Design Submission and upload and image

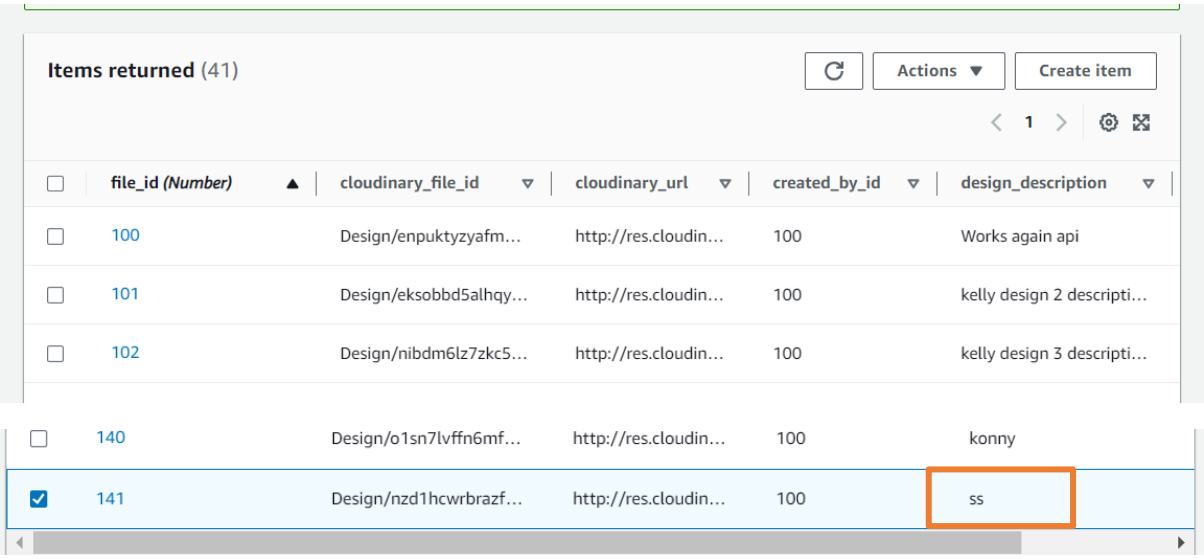
Step 3: Press Submit



Step4: If update message is shown , new post has successfully been updated



Step 5: Check DynamoDb table again , there is now 41 items in the table and as shown the latest file_id has the design description we submitted “ss” showing our Submit Design is effectively working with lambda and DynamoDB.

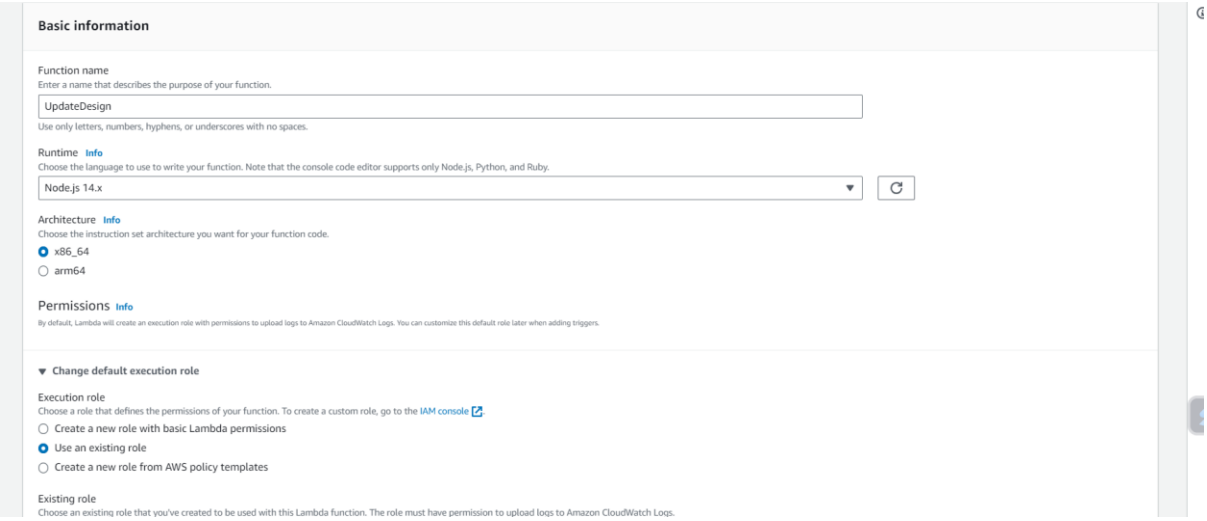


Update Design (Put)

This functions is for users to update their design information.

Create Lambda Fucntion

Step 1: Create Lambda function Update design



Dynamodb.js

It takes parameters like region, table_name, fileId, design_title, and design_description. It uses the AWS SDK to perform an update operation on the specified DynamoDB table. The Update Expression sets the new values for design_description and design_title based on the provided input

index.js | Environment var | dynamodb.js | Execution results

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

async function updateItemDynamoDB(region, table_name, fileId, design_title, design_description) {
  var dynamodb = new AWS.DynamoDB.DocumentClient({ region: region });

  var params = {
    TableName: table_name,
    Key: { "file_id": fileId },
    UpdateExpression: "set design_description = :description, design_title = :title",
    ExpressionAttributeValues: {
      ":description": design_description,
      ":title": design_title
    }
  };

  try {
    await dynamodb.update(params).promise();
    console.log("Update successful.");
  } catch (error) {
    console.log("Error updating item: ", error);
    throw error;
  }
}

module.exports = updateItemDynamoDB;
```

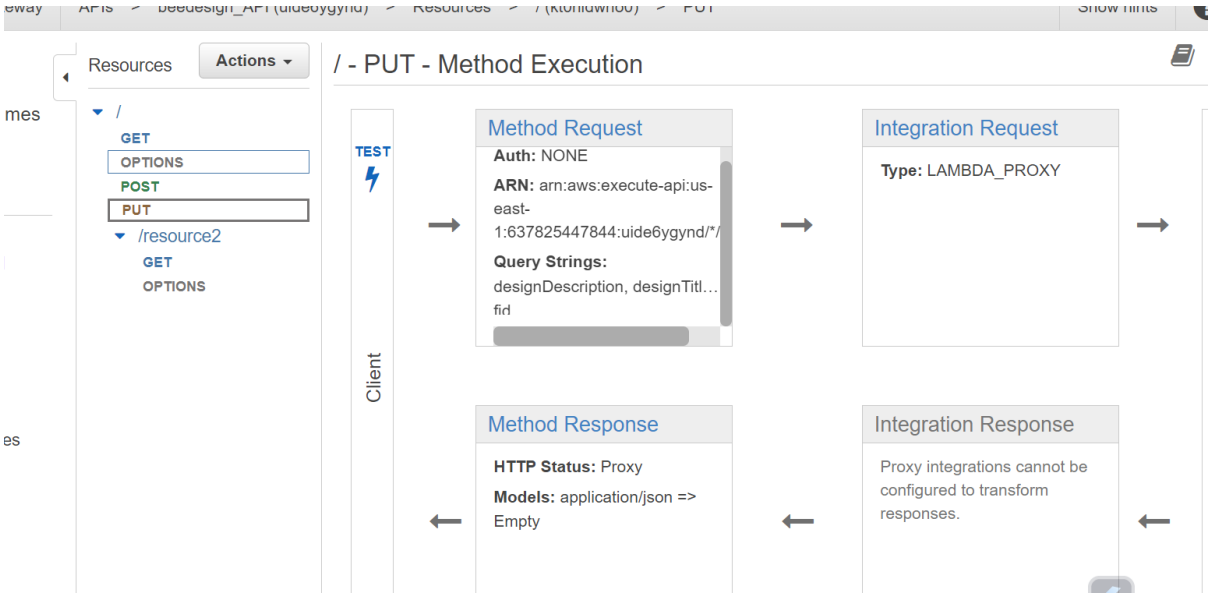
Index.js

The function checks for required parameters (fid, design Title, design Description) either in the event directly or as query string parameters. It extracts these parameters, including parsing fid as an integer. The function then uses the DynamoDB module to perform the update operation.

```
var dynamodb = require('./dynamodb');

exports.handler = function (event, context) {
    if ((event.fid && event.designTitle && event.designDescription) || (event.queryStringParameters && event.queryStringParameters.fid && event.queryStringParameters.designTitle && event.queryStringParameters.designDescription)) {
        var fileId, design_title, design_description;

        if (event.fid && event.designTitle && event.designDescription) {
            fileId = parseInt(event.fid, 10);
            design_title = event.designTitle;
            design_description = event.designDescription;
        } else {
            fileId = parseInt(event.queryStringParameters.fid);
            design_title = event.queryStringParameters.designTitle;
            design_description = event.queryStringParameters.designDescription;
        }

        var region = "us-east-1";
        var table_name = "files";

        return dynamodb(region, table_name, fileId, design_title, design_description)
            .then(() => {
                console.log("Update completed successfully.");

                // Create the response with CORS headers
                let response = {
                    statusCode: 200,
                    body: JSON.stringify({ message: "Update completed successfully." }),
                    headers: {
                        "Access-Control-Allow-Headers": "Content-Type,user",
                        "Access-Control-Allow-Origin": "*", // Allow requests from any origin
                        "Access-Control-Allow-Methods": "OPTIONS,POST,GET,PUT"
                    }
                };

                console.log("response: " + JSON.stringify(response));
                return response;
            })
            .catch((error) => {
                console.log("Update failed.", error);

                // Create the response with CORS headers
                let response = {
                    statusCode: 500,
                    body: JSON.stringify({ message: "Unable to update. Please try again later." }),
                    headers: {
                        "Access-Control-Allow-Headers": "Content-Type,user",
                        "Access-Control-Allow-Origin": "*", // Allow requests from any origin
                        "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
                    }
                };

                console.log("response: " + JSON.stringify(response));
                return response;
            });
    } else {
        // Create the response with CORS headers for the 400 Bad Request case
        let response = {
            statusCode: 400,
            body: JSON.stringify({ message: "Missing required parameters." }),
            headers: {
                "Access-Control-Allow-Headers": "Content-Type,user",
                "Access-Control-Allow-Origin": "*", // Allow requests from any origin
                "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
            }
        };

        console.log("response: " + JSON.stringify(response));
        return response;
    }
};
```

Deploy API

Step 2: This is my API configuration for this function.



Modified Code for App Update_Design.js

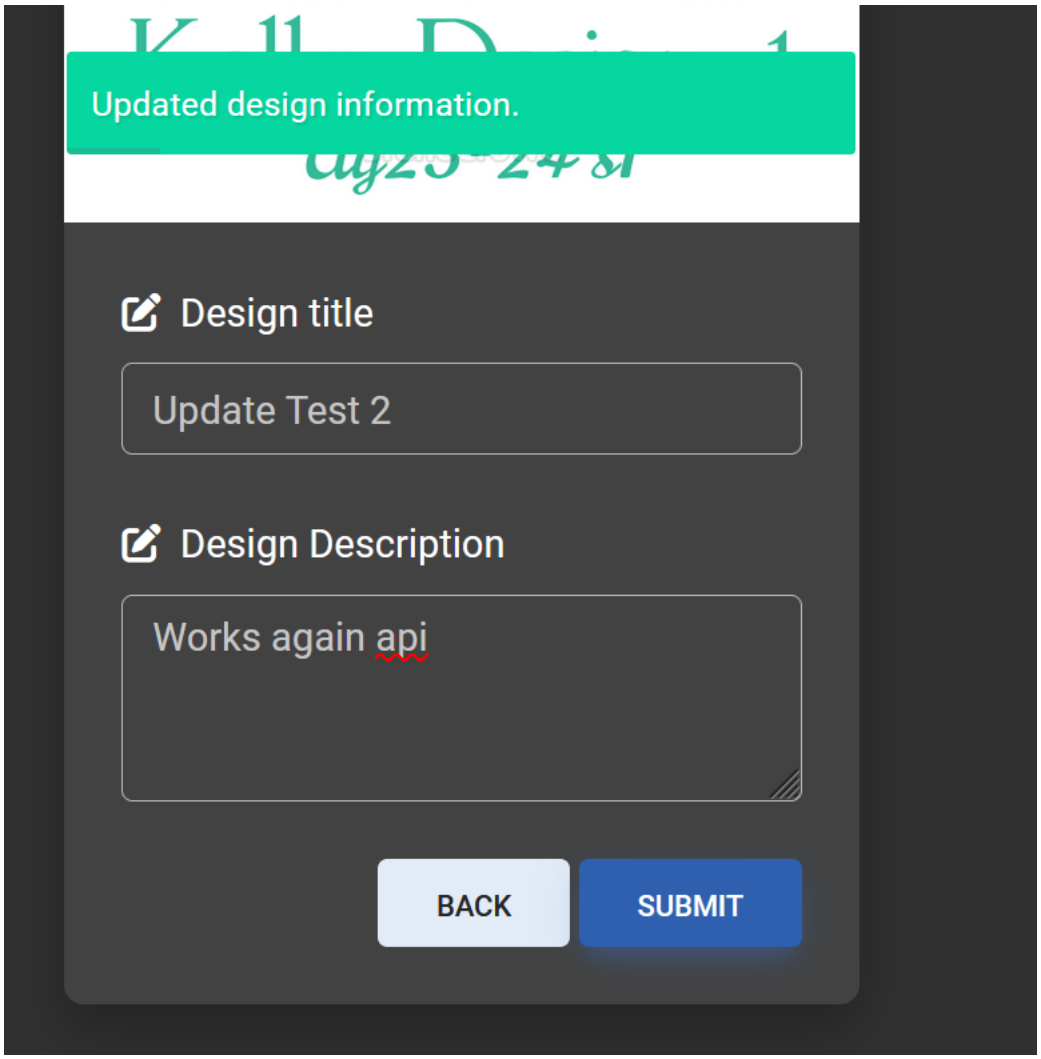
Step 2: Updated api url with query parameters

```
method: 'put',
url: 'https://uide6ygynd.execute-api.us-east-1.amazonaws.com/test?fid=' + fileId + '&designTitle=' + designTitle + '&designDescription=' + designDescription,
```

Testing Application

Step 1: Login as Kelly and enter value new value “Update Test 2” and “Works again API”. Press submit

Step 2: Successful feedback given

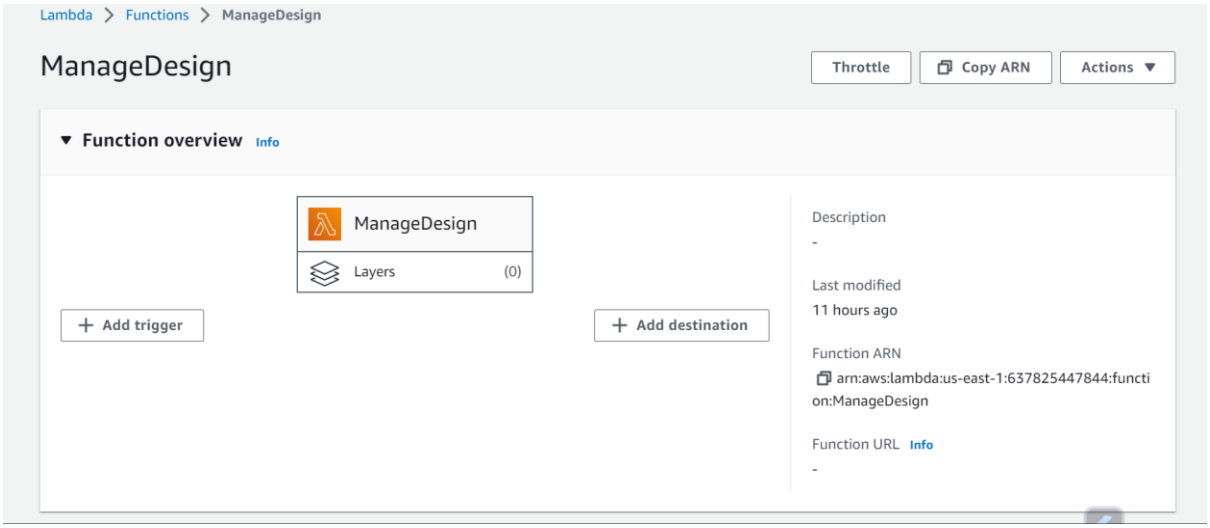


Step 3: Check the file_id you made changes to and as shown below the new updates information is reflected in the DynamoDB table. Hence showing the update function works successfully with the lambda, API and DynamoDB.

Manage User Submission (Get)

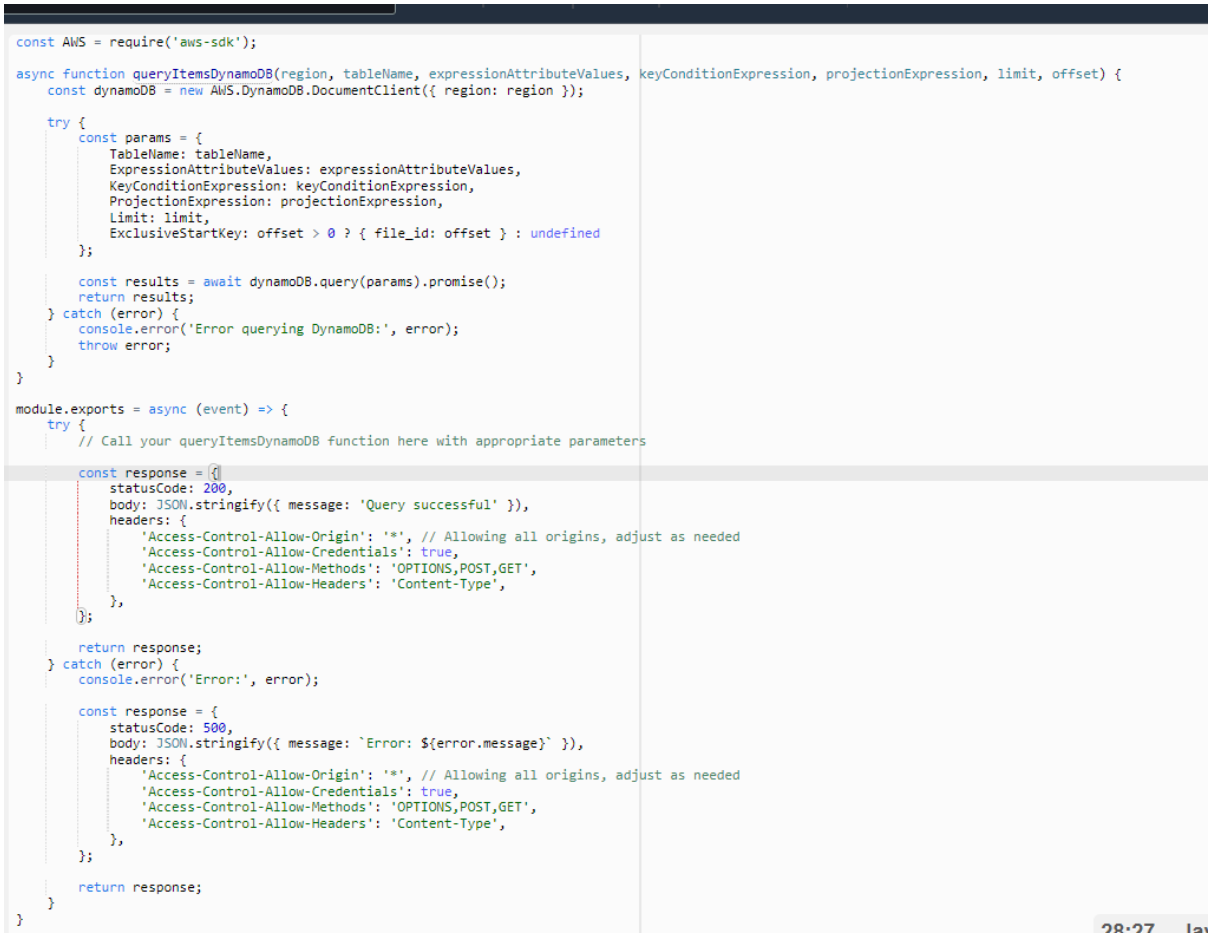
Create Lambda function.

Step 1: create Lambda function Mange Design



Dynamodb.js

This AWS Lambda function queries a DynamoDB table using the provided parameters for filtering, pagination, and projection. It encapsulates the querying process with the `queryItemsDynamoDB` function.. The `queryItemsDynamoDB` function constructs the DynamoDB query based on input parameters, and returns the query results.



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

async function queryItemsDynamoDB(region, tableName, expressionAttributeValues, keyConditionExpression, projectionExpression, limit, offset) {
    const dynamoDB = new AWS.DynamoDB.DocumentClient({ region: region });

    try {
        const params = {
            TableName: tableName,
            ExpressionAttributeValues: expressionAttributeValues,
            KeyConditionExpression: keyConditionExpression,
            ProjectionExpression: projectionExpression,
            Limit: limit,
            ExclusiveStartKey: offset > 0 ? { file_id: offset } : undefined
        };

        const results = await dynamoDB.query(params).promise();
        return results;
    } catch (error) {
        console.error('Error querying DynamoDB:', error);
        throw error;
    }
}

module.exports = async (event) => {
    try {
        // Call your queryItemsDynamoDB function here with appropriate parameters

        const response = {
            statusCode: 200,
            body: JSON.stringify({ message: 'Query successful' }),
            headers: {
                'Access-Control-Allow-Origin': '*', // Allowing all origins, adjust as needed
                'Access-Control-Allow-Credentials': true,
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',
                'Access-Control-Allow-Headers': 'Content-Type',
            },
        };

        return response;
    } catch (error) {
        console.error('Error:', error);

        const response = {
            statusCode: 500,
            body: JSON.stringify({ message: `Error: ${error.message}` }),
            headers: {
                'Access-Control-Allow-Origin': '*', // Allowing all origins, adjust as needed
                'Access-Control-Allow-Credentials': true,
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',
                'Access-Control-Allow-Headers': 'Content-Type',
            },
        };

        return response;
    }
}
```

Index.js

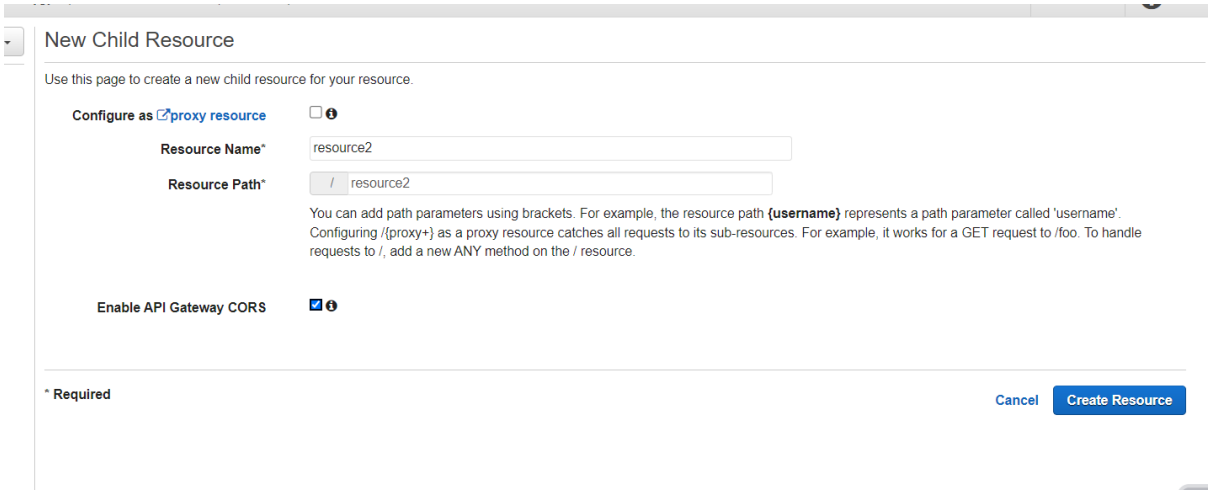
This Manage Design function validates and processes requests to retrieve paginated and filtered file records from a DynamoDB table. It checks query parameters for `pageNumber` and `userId`, then scans the table based on these parameters along with an optional search parameter. It filters, paginates, and sorts the results by `file_id`, and responds with success containing paginated data or a not-found response if no records are found. Errors trigger error responses, all including necessary CORS headers.

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

exports.handler = async (event) => {
    try {
        if (!event.queryStringParameters || !event.queryStringParameters.pageNumber || !event.queryStringParameters.userId) {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: 'Invalid request parameters.' })
            };
        }

        console.log(event.queryStringParameters);
        const pageNumber = parseInt(event.queryStringParameters.pageNumber);
        if (isNaN(pageNumber) || pageNumber <= 0) {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: 'Invalid page number.' })
            };
        }

        const search = event.queryStringParameters.search || ''; // Get the search parameter from the event
        const userId = parseInt(event.queryStringParameters.userId);

        const dynamoDB = new AWS.DynamoDB.DocumentClient({ region: 'us-east-1' }); // Update region as needed

        const scanParams = {
            TableName: 'files', // Your table name
            FilterExpression: 'created_by_id = :userId AND contains(design_title, :search)', // Use the search parameter in the filter expression
            ExpressionAttributeValues: {
                ':userId': userId,
                ':search': search
            }
        };

        const scanResult = await dynamoDB.scan(scanParams).promise();
        console.log(scanResult);

        // Perform client-side pagination
        const limit = 4; // Records per page
        const startIdx = (pageNumber - 1) * limit;
        const endIdx = startIdx + limit;

        const paginatedItems = scanResult.Items.slice(startIdx, endIdx);

        // Sort the paginated items based on file_id in ascending order
        paginatedItems.sort((item1, item2) => {
            return item1.file_id - item2.file_id;
        });
```

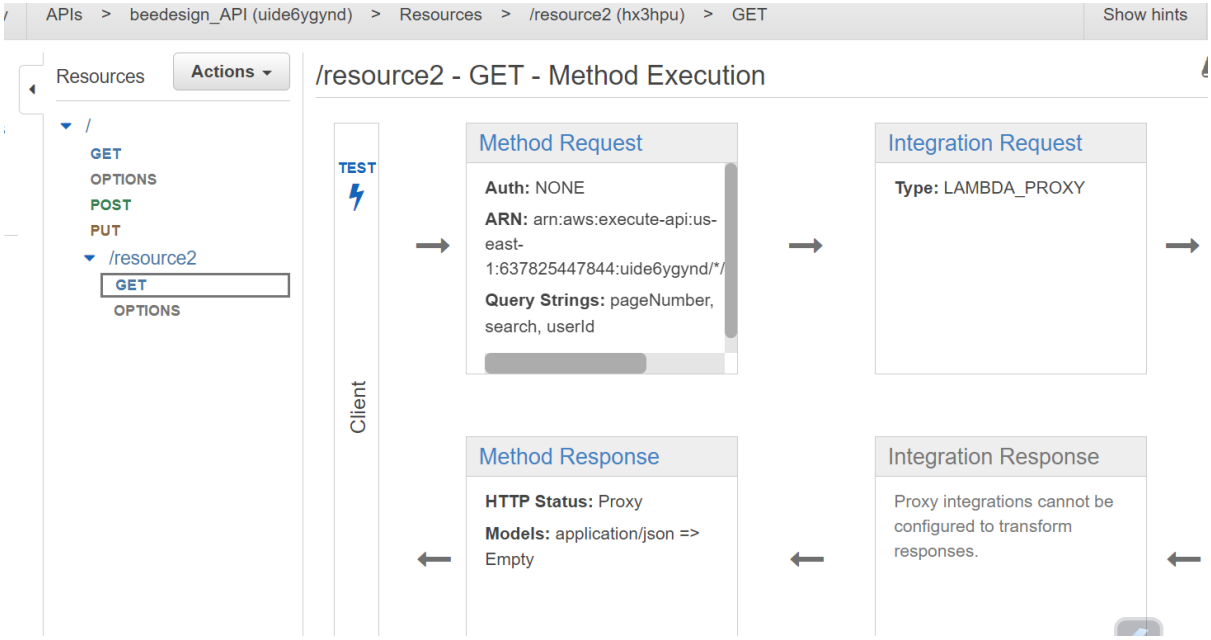
```
        });

        if (paginatedItems.length > 0) {
            const jsonResult = {
                'number_of_records': paginatedItems.length,
                'page_number': pageNumber,
                'filedata': paginatedItems,
                'total_number_of_records': scanResult.Count
            };

            return {
                statusCode: 200,
                body: JSON.stringify(jsonResult),
                headers: {
                    "Access-Control-Allow-Headers": "Content-Type,user",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
                }
            };
        } else {
            return {
                statusCode: 404,
                body: JSON.stringify({ message: 'No records found.' }),
                headers: {
                    "Access-Control-Allow-Headers": "Content-Type,user",
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
                }
            };
        }
    } catch (error) {
        console.error('Error:', error);

        return {
            statusCode: 500,
            body: JSON.stringify({ message: `Error: ${error.message}` }),
            headers: {
                "Access-Control-Allow-Headers": "Content-Type,user",
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
            }
        };
    }
;
```

API

Step 1 : Create New child resource



Step2 : This is the API configuration



Modified code for app

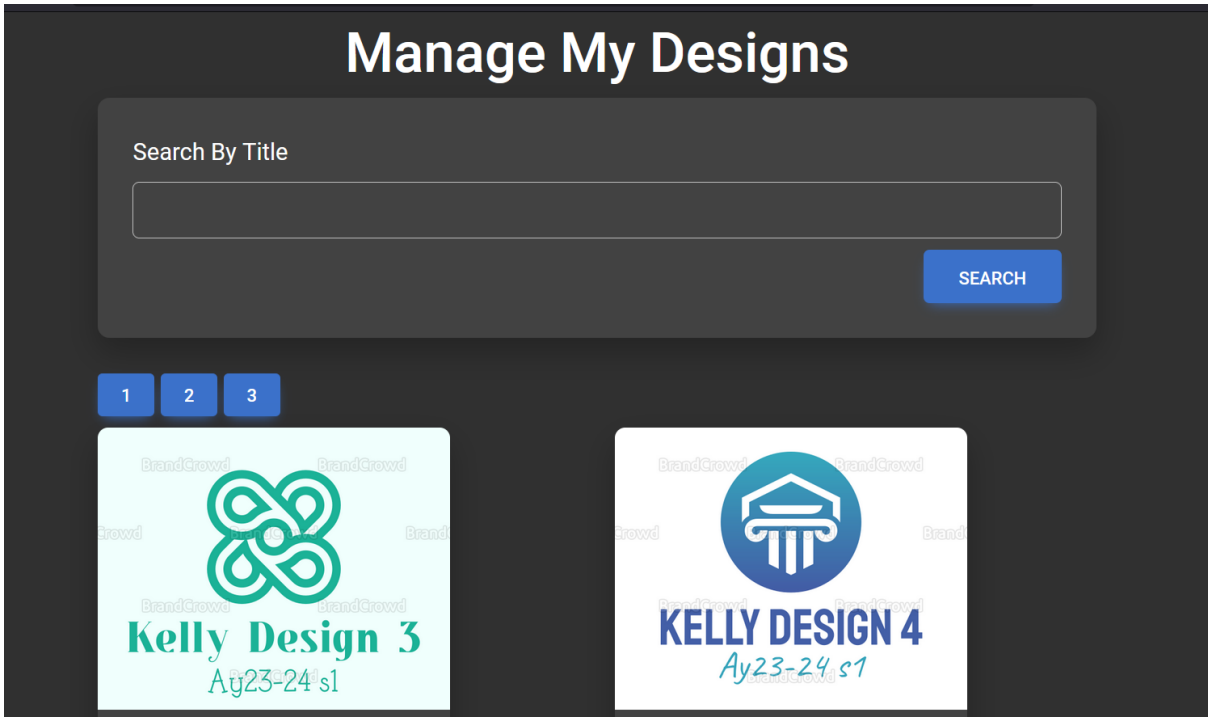
User_Manage_Submission.js

```
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 submit registration details
    //to server-side api when the #submitButton element fires the click event.
    $('#submitButton').on('click', function(event) {
        event.preventDefault();
        let page = 1;
        let searchInput = $('#searchInput').val();
        let userId = localStorage.getItem('user_id');

        let requestData = {
            pageNumber: page,
            userId: parseInt(userId),
            search: searchInput
        }
        axios({
                headers: {
                    'Content-Type': 'application/json',
                    'user': userId
                },
                method: 'get',
                url: 'https://uide6ygynd.execute-api.us-east-1.amazonaws.com/test/resource2'    ,
                params: requestData
            })
```

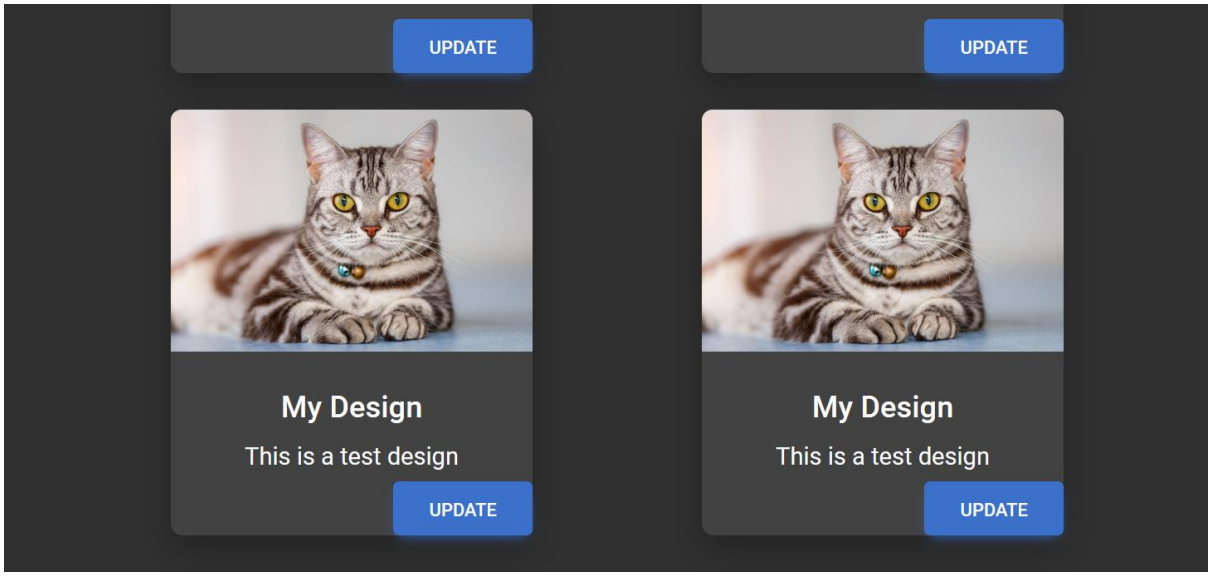
```
    //to server-side api.

function clickHandlerForPageButton(event) {
        event.preventDefault();
        // const baseUrl = 'http://52.2.77.200:5000';
        let userId = localStorage.getItem('user_id');
        let pageNumber = $(event.target).text().trim();
        let searchInput = $('#searchInput').val();
    let requestData = {
            pageNumber: pageNumber,
            userId: parseInt(userId),
            search: searchInput
        }

        console.log(pageNumber);
        axios({
                headers: {
                    'Content-Type': 'application/json',
                    'user': userId
                },
                method: 'get',
                url: 'https://uide6ygynd.execute-api.us-east-1.amazonaws.com/test/resource2',
                params: requestData
            })
            .then(function(response) {
                //Using the following to inspect the response.data data structure
                //before deciding the code which dynamically generates cards.
                //Each card describes a design record.
                //console.dir(response.data);
                const records = response.data.filedata;
                const totalNumOfRecords = response.data.total_number_of_records;
```

Testing Application

Step 1: Log in as Kelly, and press “Search”



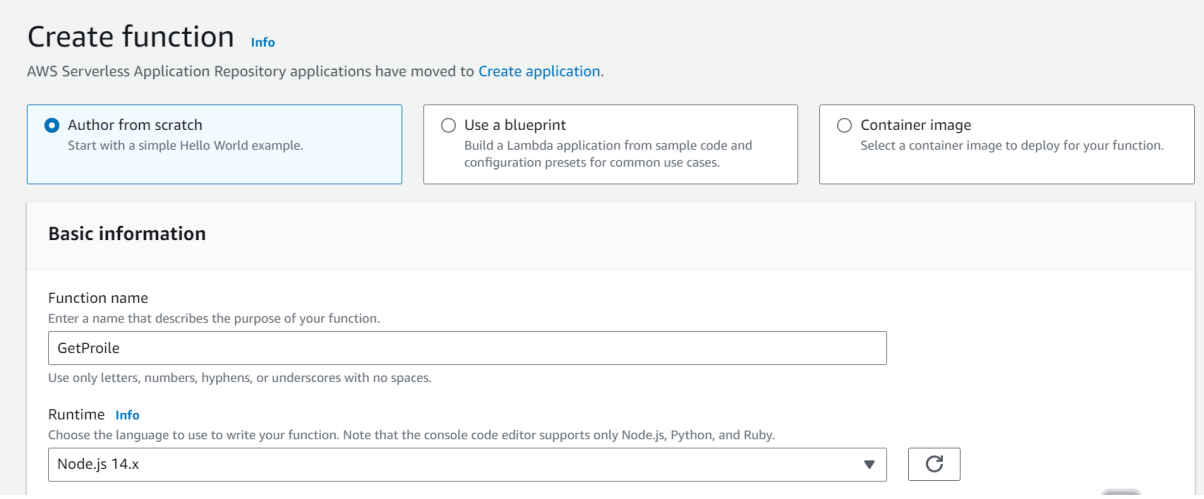
Step 2: You can verify that its retrieving from DynamoDb as it also retives the new submisions from the Submit Design we have done earlier.



Get profile (Get)

Create Lambda function

Step 1: Create Get Profile function.



Dyanamodb.js

This code defines a module for querying items from a DynamoDB table. It uses the AWS SDK to perform a query operation based on provided parameters like region, table name, expression attribute values, key condition expression, and projection expression. The results are returned, and any errors during the query process are logged. The module exports the `queryitems_dynamodb` function for external use.

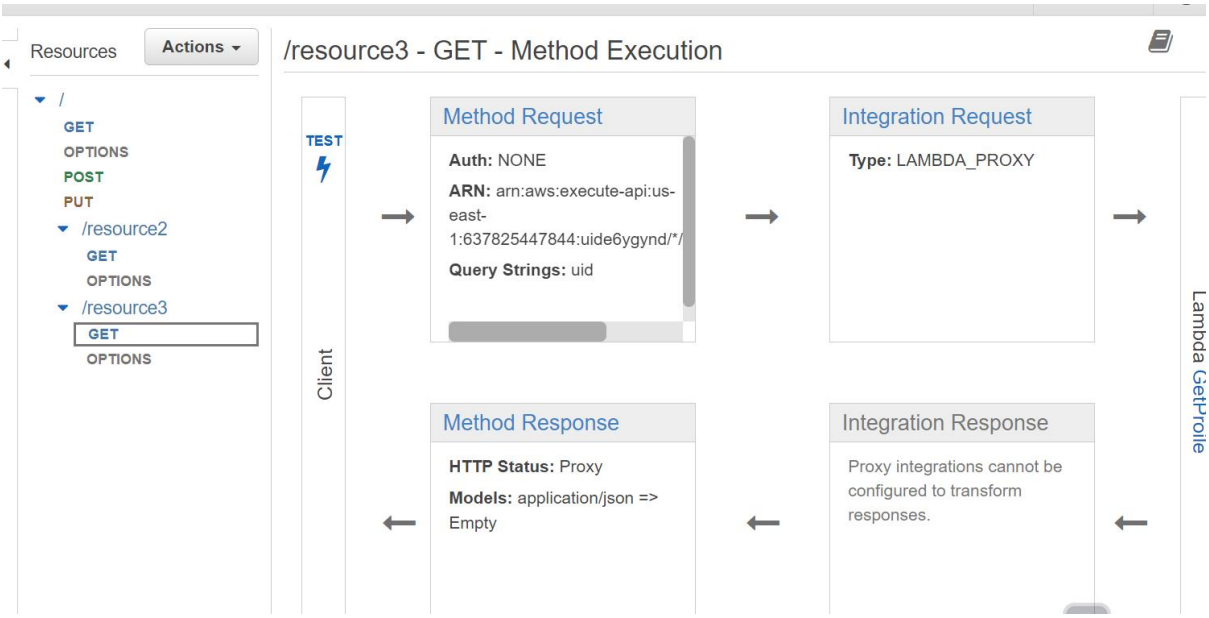
```
var AWS = require('aws-sdk');

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

This Get profile function retrieves user data from a DynamoDB table based on the provided user ID. It checks for the user ID in the event or query parameters, then uses the `dynamodbQuery` function to fetch user information. The function generates responses with user data, handles missing user IDs, and manages errors, all while adhering to CORS policies for cross-origin requests.

API

This is my API configuration.



Modify app code

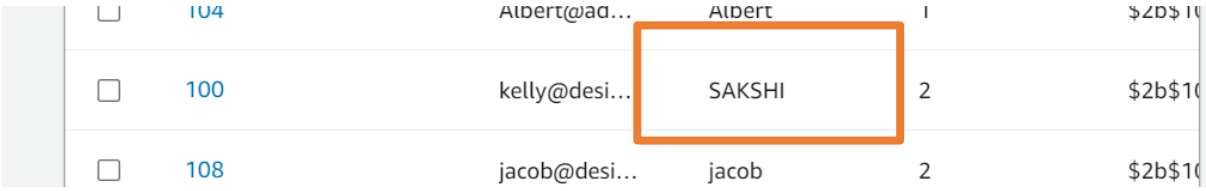
Profile.js

Step 1: Update the URL with the API url , child resource path and query parameters

```
axios({
    headers: {
      'user': userId
    },
    method: 'get',
    url: ' https://uide6ygynd.execute-api.us-east-1.amazonaws.com/testS/resource3?uid=' + userId,
    })
    .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);
        const record = response.data.userdata;
        $('#fullNameOutput').append(record.fullname); //text(record.fullname);
```

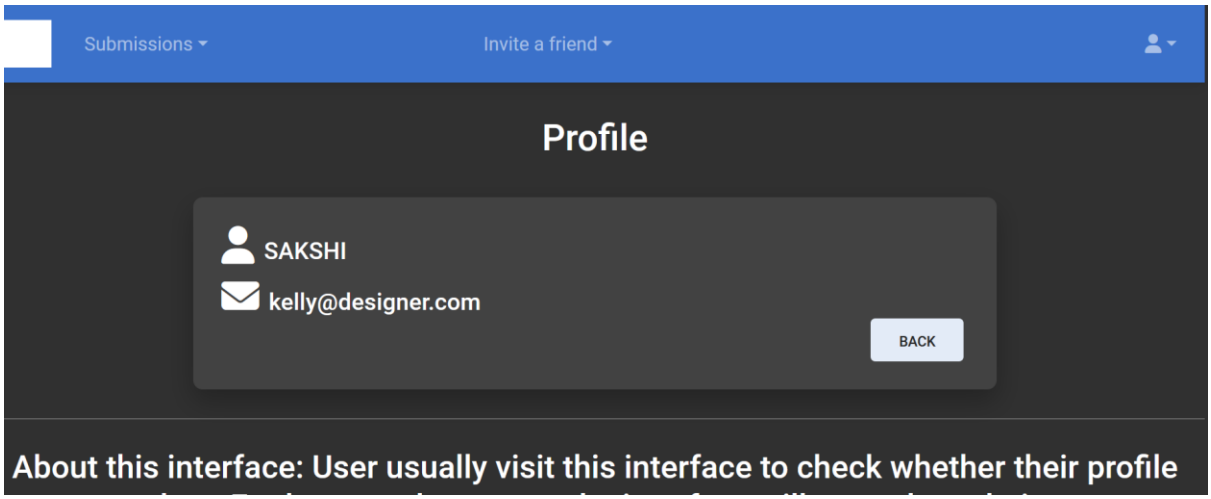
Testing application

Before, testing. I have changed Kelly name to SAKSHI.



Step 1: Log in as Kelly and check your profile.

As shown below , the name shows SAKSHI , showing the get profile function retrieves the profile information from Dynamodb.



MySQL RDS security

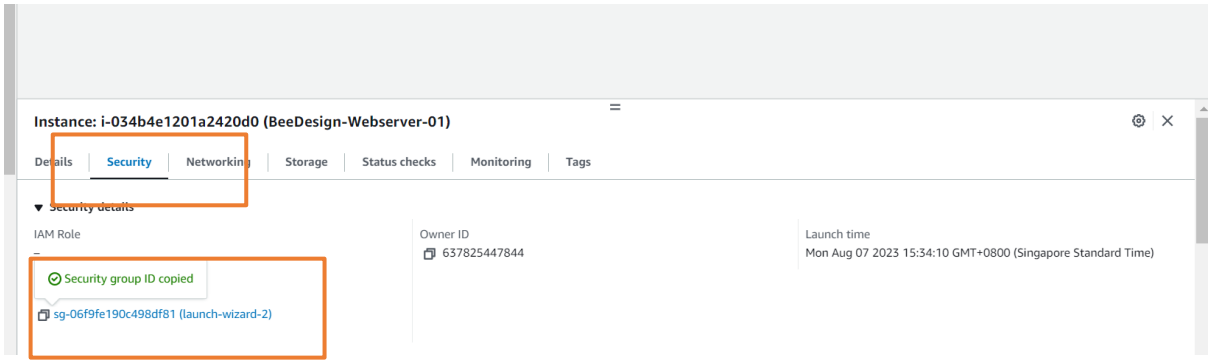
Ensuring only the EC2 instance running NODEJS code can have access to RDS

We will be configuring **security group rules** for the RDS instance ensure that only the website (or any other application) running on the specific EC2 instance with the associated security group can access the RDS instance. This **helps enhance the security of your backend setup** by **restricting database access to only the authorized resources.**

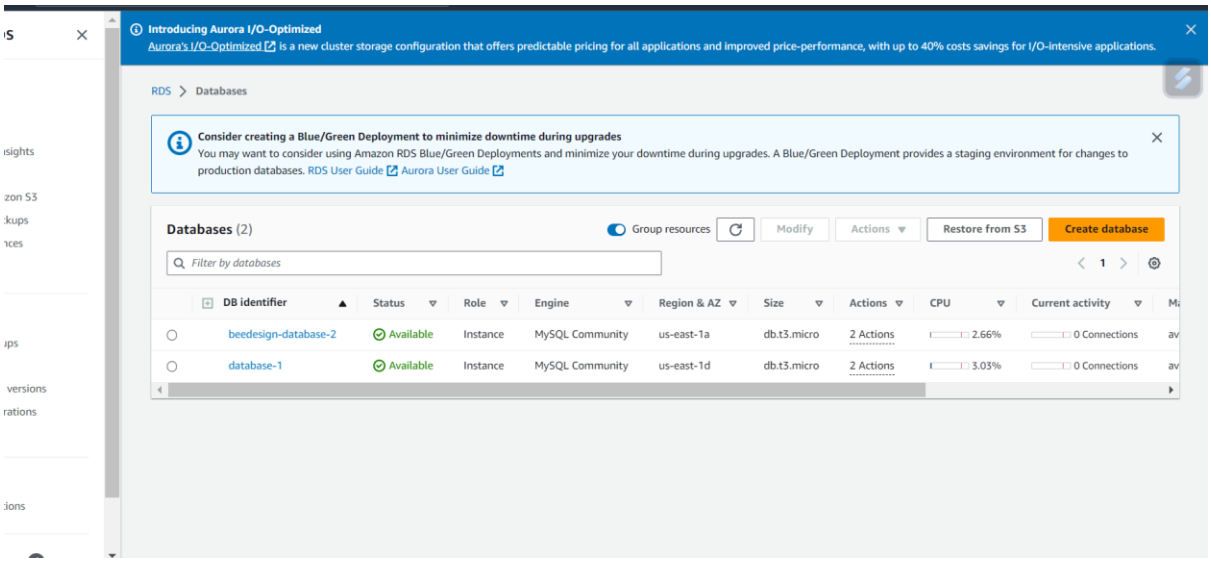
This configuration ensures that the Bee Design Website on the our EC2 instance can communicate with the RDS instance while maintaining a secure network environment.

Step 1: Navigate to Ec2 console , go to your instance BeeDesign-Websever-01,

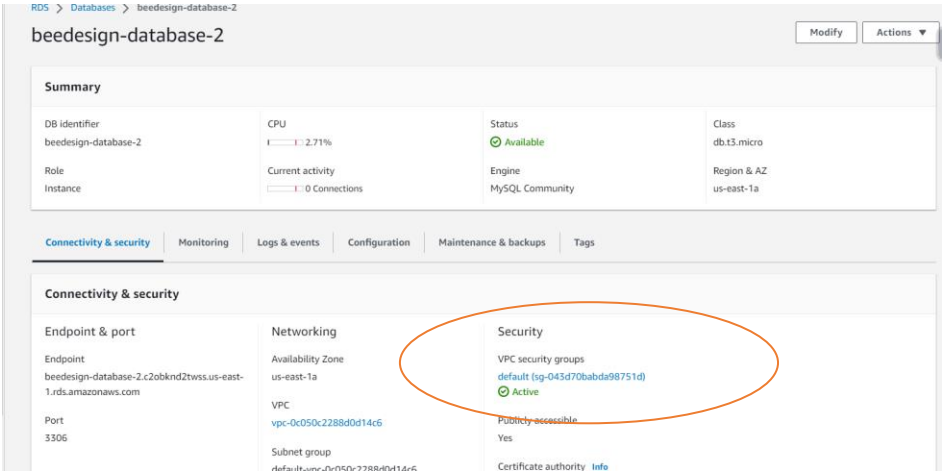
Step 2: Go to security and **copy the security group id**



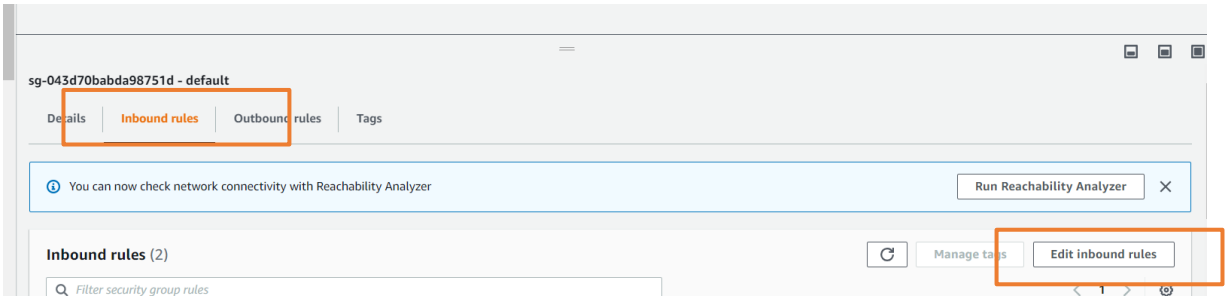
Step 3: Go to RDS console and click on one beedesign-database-2.



Step 4 : In the Connectivity and Security , press on the link of VPC security groups



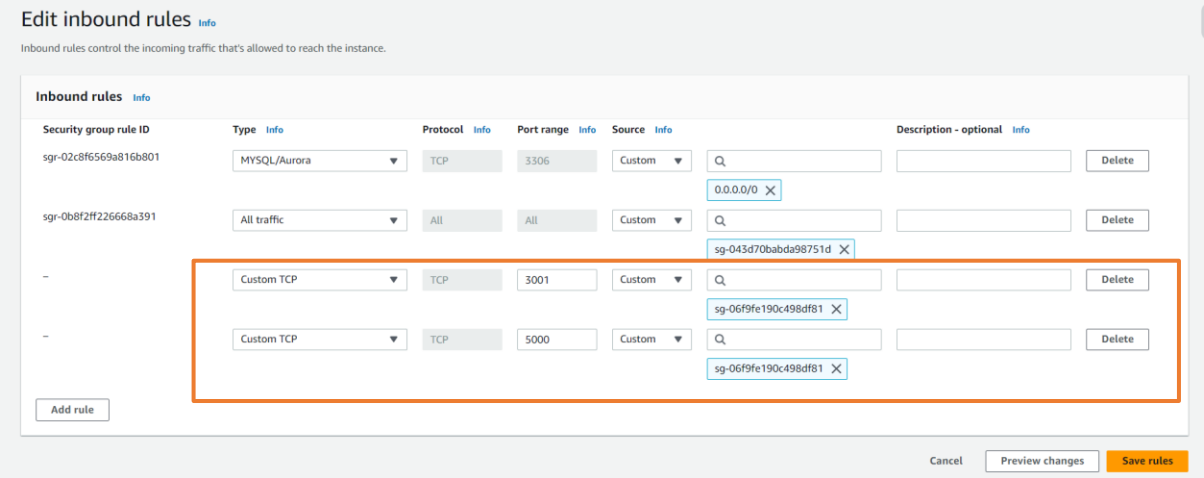
Step 5: In the next page , go to Inbound rules and click “Edit Inbound rules”



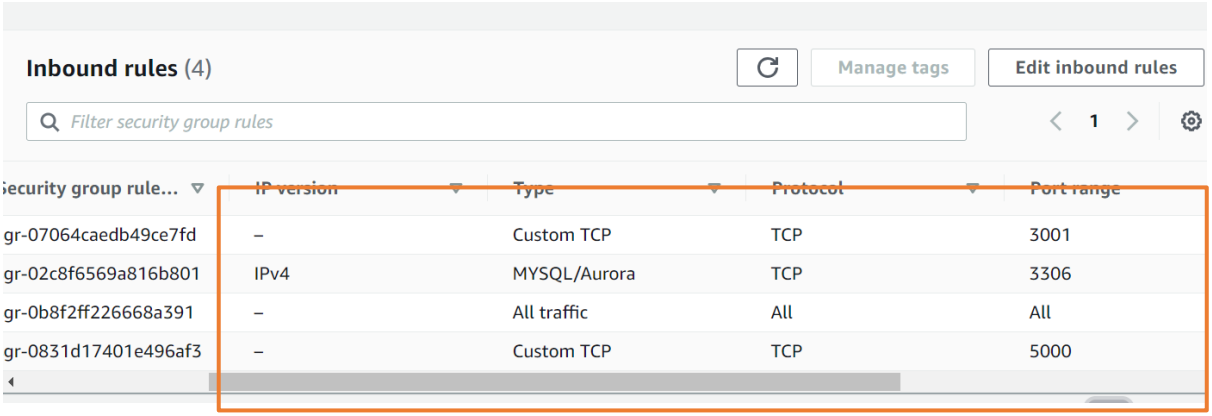
Step 6: Configure **port 3001 and 5000** with the **security group id of the EC2 instance** and protocol to TCP

- By specifying security group as the source, you ensure that only resources within that security group can access the RDS instance.
- This enhances security by restricting access to a specific set of resources and prevents unauthorized access from other sources

Step 7: Press save rules



Step 8: Check that the rules have been successfully configured in the RDS



Increase Security Conclusion

1. The security group rules you added to RDS instance **allow inbound traffic only from the security group associated with EC2 instance (`sg-06f9fe190c498df81`).** This means that any resource (in this case, your EC2 instance) that is a member of that security group is allowed to connect to your RDS instance.

2. Associating the security group with the EC2 instance, it effectively ensures that only resources within that security group (in this case, your EC2 instance) can access the RDS instance on the specified port(s).

3. So**, Bee Design website running on the EC2 instance should be able to access the RDS instance, while access from other unauthorized sources is restricted by the security group rulesof 3001 and 5000 set up**. This helps isolate and secure the communication between your website and the database.

Using the least privilege for database connection credentials

We will be granting privileges to admin.

What does it mean?

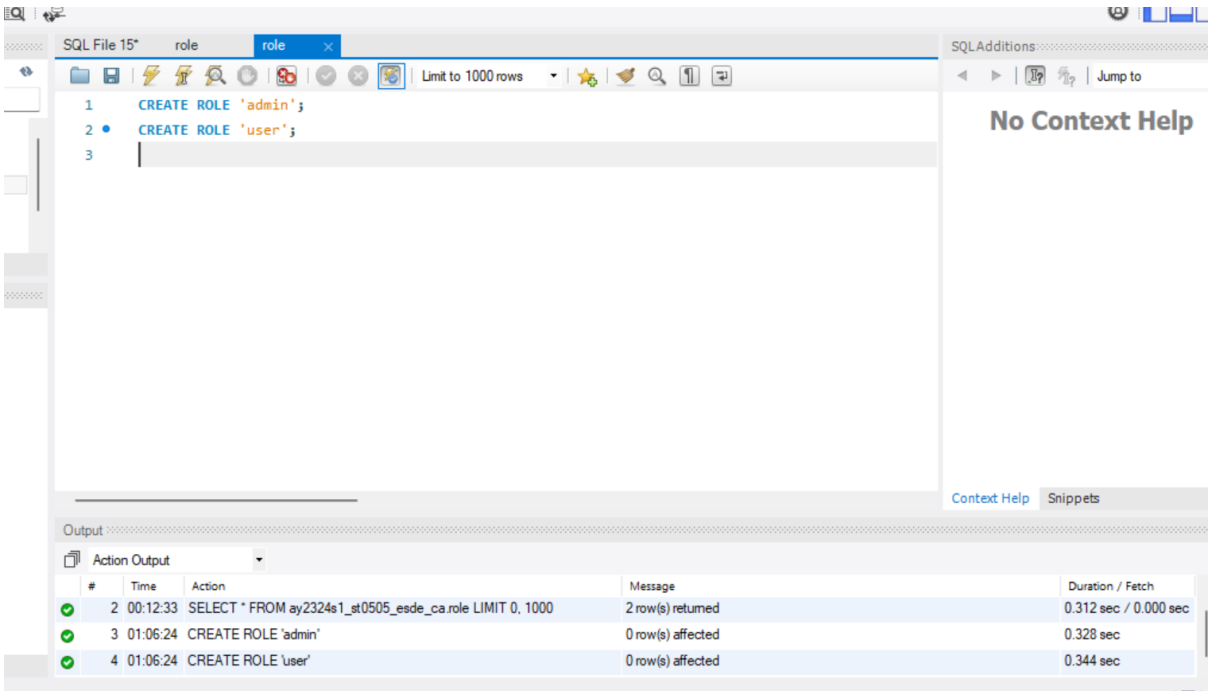
The privileges you granted in MySQL will be enforced at the database level. When you grant privileges to a specific user or role in MySQL, those privileges control what actions that user or role can perform on the specified database objects (e.g., tables, views). These privileges include actions like SELECT, INSERT, UPDATE, DELETE, and more.

Step 1: Using MySQL least privilege to create user and admin

Run these commands

CREATE ROLE 'user_role';

CREATE ROLE 'admin';



Admin can manage user – select privilege to get users, select one user details

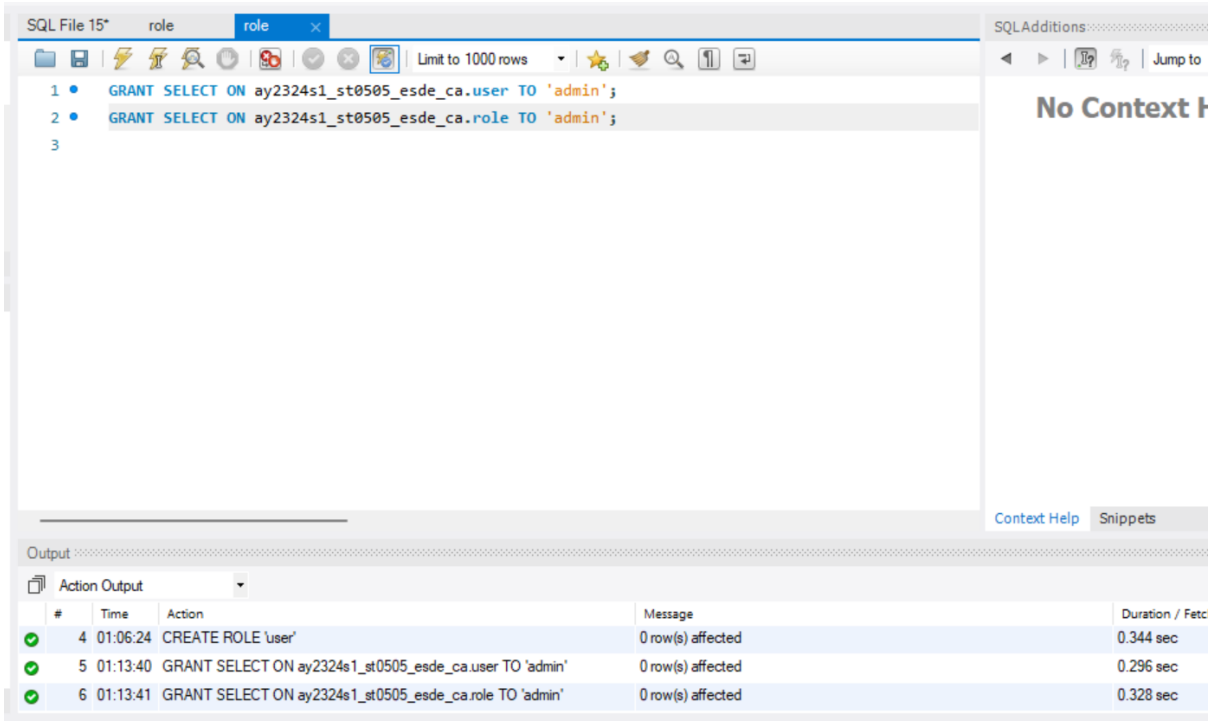
Hence , SELECT: Grant the SELECT privilege on the user and role tables **to allow the admin role to retrieve user data and associated role information.**

Step 2: Run these commands

GRANT SELECT ON your_database.user TO 'admin';

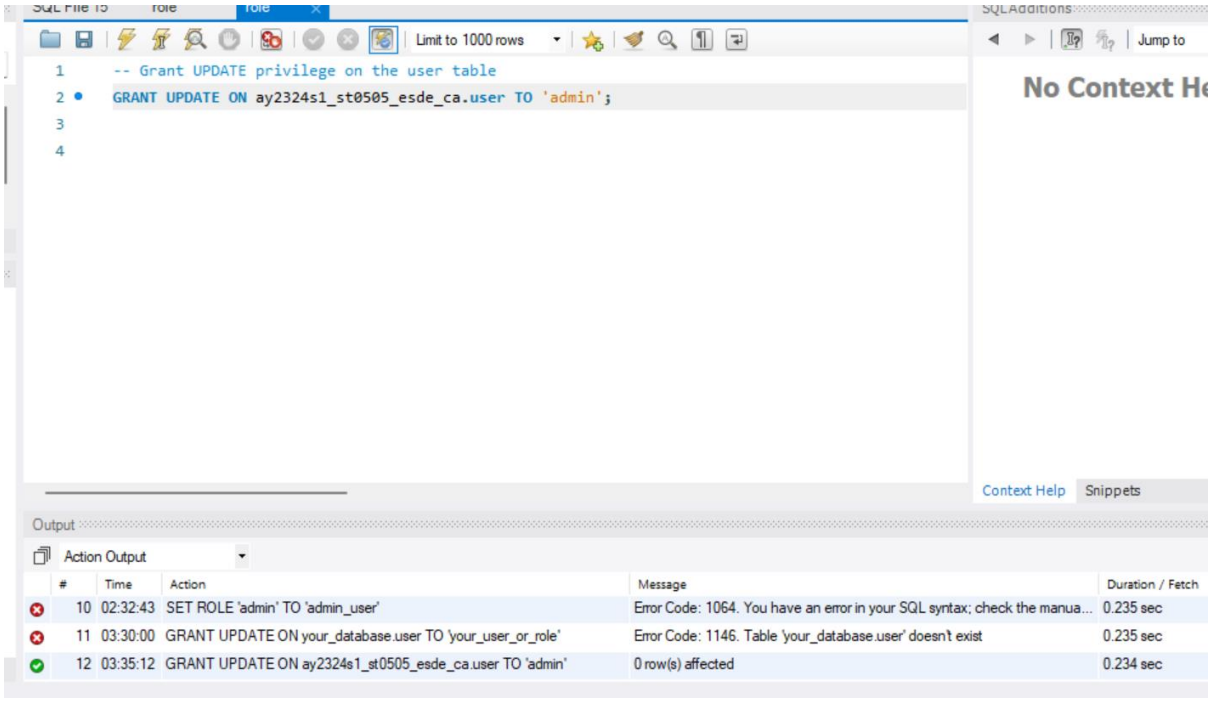
GRANT SELECT ON your_database.roleTO 'admin';

This means that only users with the `admin` role will be able to execute `SELECT` queries on these tables in the database.



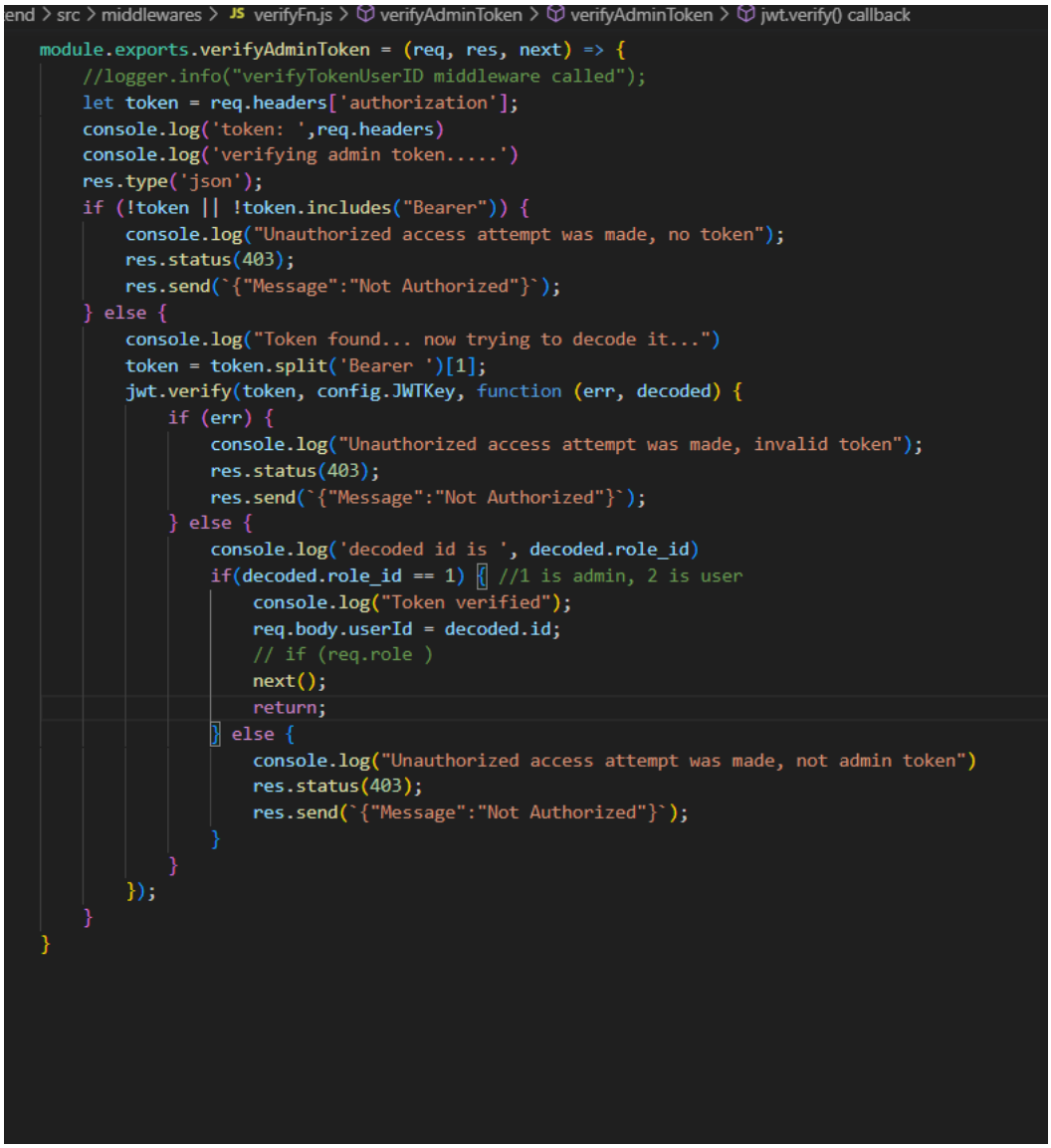
Step 3: Admin – can update user privilege

Run these command to allow admin execute update queries on role of user or admin.



VerifyAdmin function

The middleware and role-based authorization adds an additional layer of security and control. It ensure that even if someone manages to access your endpoints, they won't be able to retrieve data from the database unless their role (admin) matches the required role for that action



In conclusion advantage to security

1. **Granting SELECT privilege and Update** Privileges to the admin role is a step towards improving the security of your Amazon RDS database. By doing so, you are following the principle of least privilege, which means you are giving the admin role only the minimum permissions necessary to perform its required tasks. This approach reduces the potential attack surface and limits the potential impact of any security breaches.

2. By granting SELECT and Update privilege specifically to the admin role, **you are ensuring that only authorized users with admin privileges can access and retrieve data from the database**. This helps **prevent unauthorized access** and data leakage.

3. So, the **combination of MySQL privileges and role-based authorization** in your Node.js application will work together to enforce access controls and ensure that only authorized users with the proper roles can perform certain actions and access specific data.

.