

ESDE CA2



Name: Singh Sakshi

Student id : 2228479

Class: DIT/2a06

14 AUGUST 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
<i>Configure Settings</i>	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.

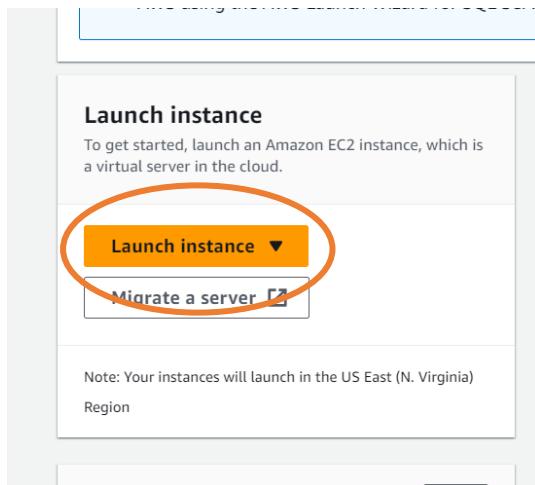
The screenshot shows the AWS EC2 Dashboard. On the left, a sidebar lists navigation options like 'EC2 Dashboard', 'Instances', 'Images', and 'Services'. The main area displays resource statistics in a grid:

Instances (running)	1	Auto Scaling Groups	0	Dedicated Hosts	0
Elastic IPs	1	Instances	1	Key pairs	2
Load balancers	0	Placement groups	0	Security groups	2
Snapshots	0	Volumes	1		

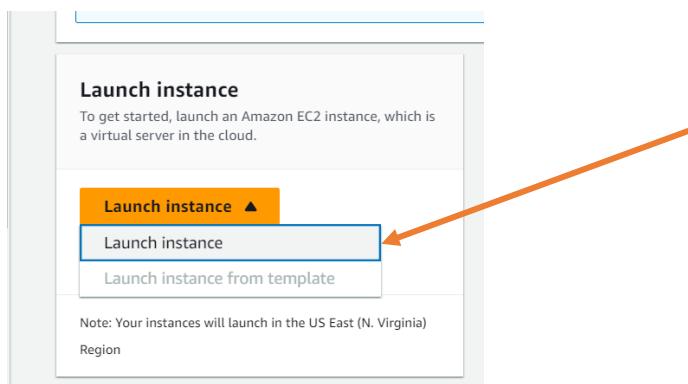
A callout box provides instructions for launching Microsoft SQL Server Always On availability groups. The right side of the screen includes sections for 'Account attributes' (with links to VPC, Default VPC, and Settings), 'Explore AWS' (with links to Best Price-Performance with AWS Graviton2 and Graviton2 powered EC2 instances), and 'Service health'.

Create new instance.

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

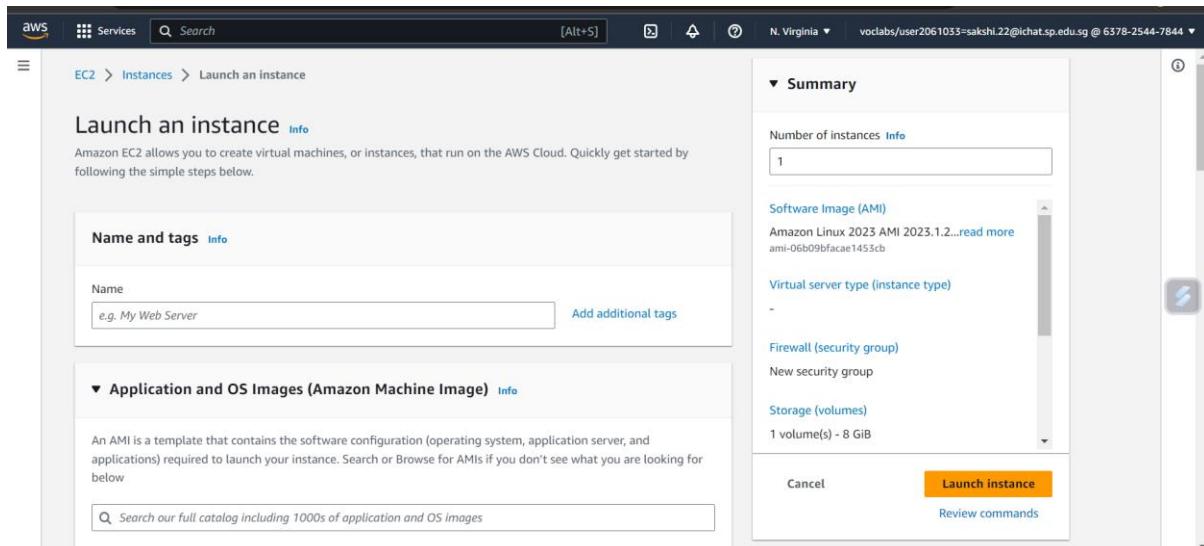


Step 4: Upon clicking, you will see a **dropdown with “Launch Instance”**. Click on it.



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.

The screenshot shows the 'Launch an instance' wizard. In the 'Name and tags' step, the 'Name' field contains 'BeeDesign-Webserver-01'. An orange arrow points to this field.

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 kernel6.1l- as your Amazon Machine Image (AMI).

The screenshot shows the 'Application and OS Images (Amazon Machine Image)' search results. The 'Amazon Linux' thumbnail is highlighted with an orange arrow. Below it, a detailed view of the 'Amazon Machine Image (AMI)' is shown, including its description, architecture, boot mode, and AMI ID.

Architecture	Boot mode	AMI ID
64-bit (x86)	uefi-preferred	ami-06b09bfacae1453cb

Description: Amazon Linux 2023 AMI 2023.1.20230629.0 x86_64 HVM kernel-6.1

Architecture: 64-bit (x86)

Boot mode: uefi-preferred

AMI ID: ami-06b09bfacae1453cb

Status: Free tier eligible

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

The screenshot shows the 'Instance type' section of the AWS EC2 instance creation wizard. A red circle highlights the 't2.micro' option in the dropdown menu. Below the dropdown, there is detailed pricing information for various operating systems. To the right, there is a toggle switch for 'All generations' and a link to 'Compare instance types'.

On-Demand Windows pricing:	0.0162 USD per Hour
On-Demand SUSE pricing:	0.0116 USD per Hour
On-Demand RHEL pricing:	0.0716 USD per Hour
On-Demand Linux pricing:	0.0116 USD per Hour

Instance type

t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows pricing: 0.0162 USD per Hour
On-Demand SUSE pricing: 0.0116 USD per Hour
On-Demand RHEL pricing: 0.0716 USD per Hour
On-Demand Linux pricing: 0.0116 USD per Hour

Free tier eligible

All generations

Compare instance types

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.

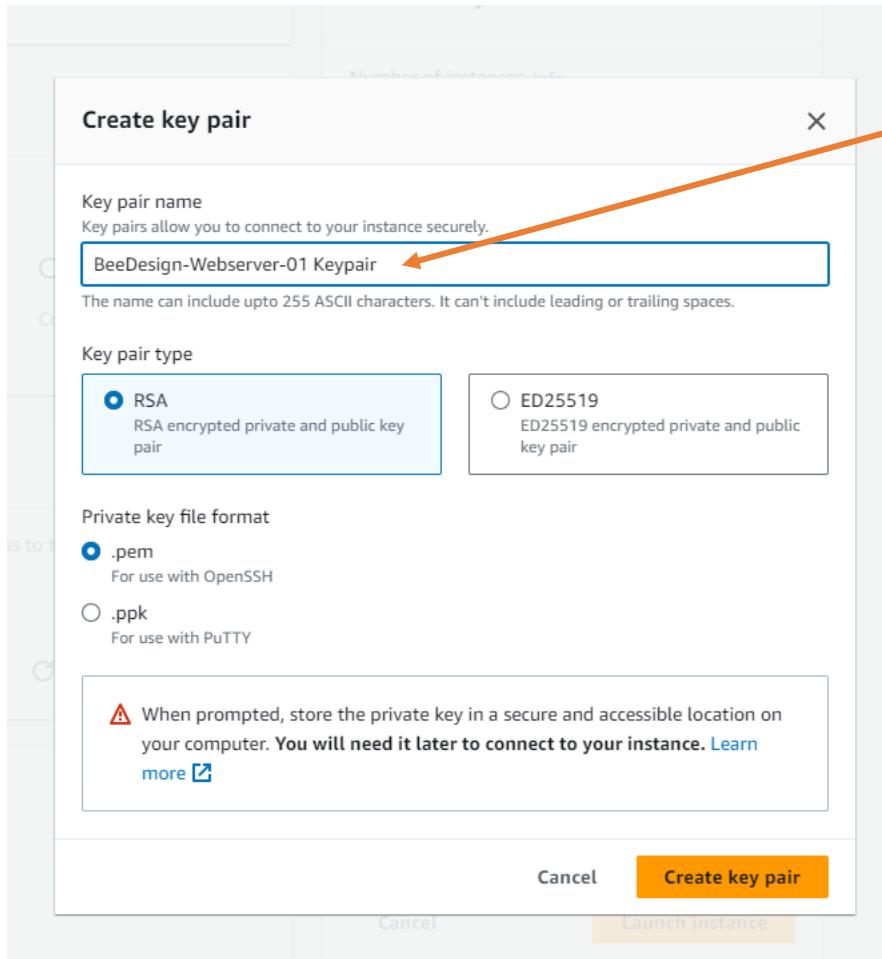
The screenshot shows the 'Key pair (login)' section of the AWS EC2 instance creation wizard. A red circle highlights the 'Create new key pair' button. There is also a dropdown menu for selecting an existing key pair.

Key pair name - *required*

Select Create new key pair

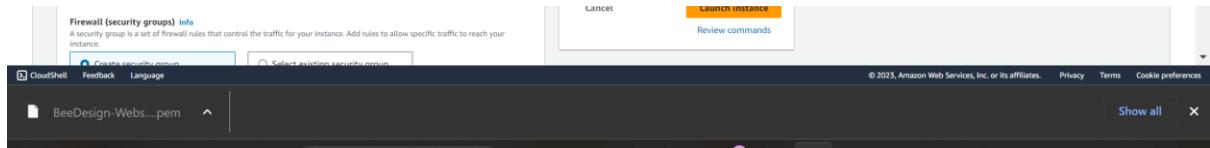
Step 10: Name key pair as **BeeDesign-Webserver-01** and leave the other settings as default.

Step 11: Press “**create key pair**”.

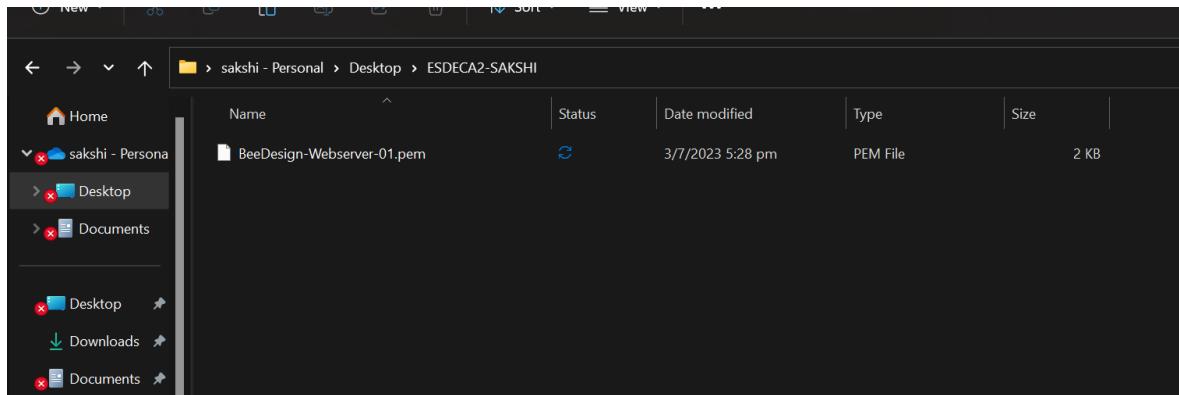


Step 12: **The .pem file** will be automatically downloaded.

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.

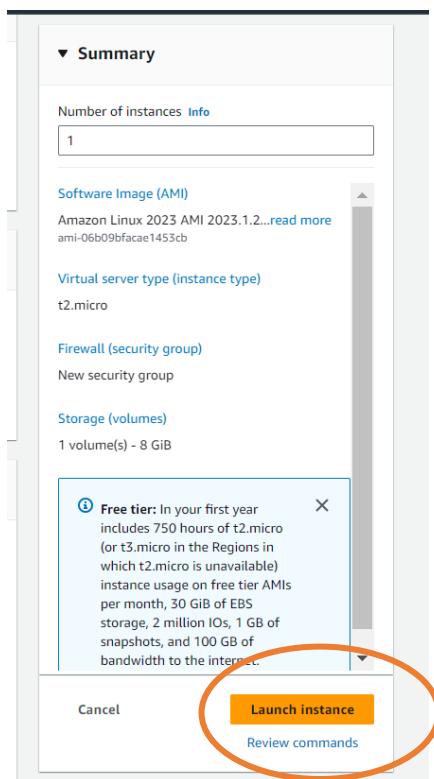


Network Settings

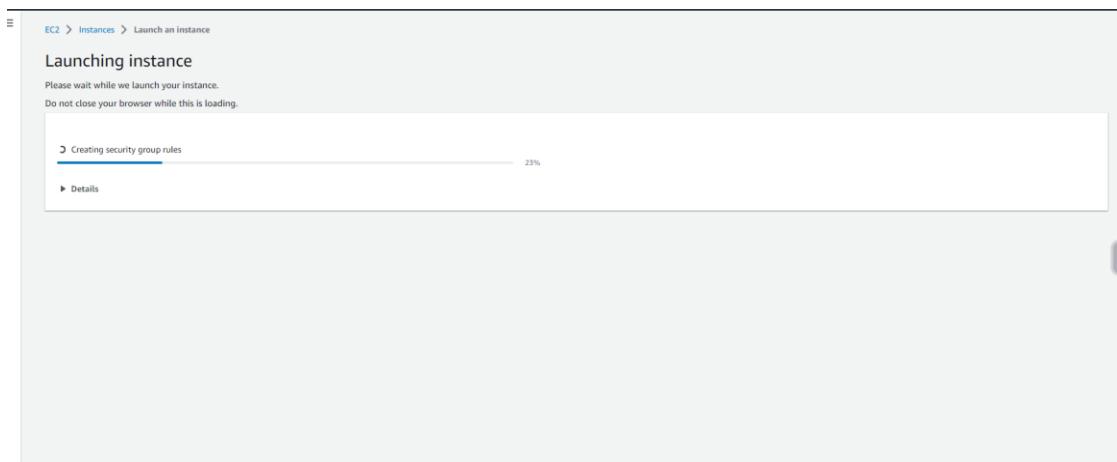
Step 10: Enable Auto-IP to Ensure security group allows SSH.

A screenshot of the AWS Network settings configuration page. It shows basic network information: Network (vpc-0c050c2288d0d14c6), Subnet (No preference), and Auto-assign public IP (Enable). In the Firewall (security groups) section, there is a note about security groups controlling traffic to instances. Below this, there are two radio button options: 'Create security group' (selected) and 'Select existing security group'. A large orange arrow points from the top right towards the 'Create security group' button. Below these buttons, it says 'We'll create a new security group called "launch-wizard-2" with the following rules:' followed by three checkboxes: 'Allow SSH traffic from Anywhere' (checked), 'Allow HTTPS traffic from the internet' (unchecked), and 'Allow HTTP traffic from the internet' (unchecked). A warning message at the bottom states: '⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.' with a close button.

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

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
BeeDesign-We...	i-034b4e1201a2420d0	Running	t2.micro	Initializing	No alarms	us-east-1a	ec2-44-202-128-106.co...	44.202.128.106	-
CA2 WEB SER...	i-0601326b52d809226	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	ec2-3-208-213-65.com...	3.208.213.65	3.208.213.65

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

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
BeeDesign-We...	i-034b4e1201a2420d0	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-44-202-128-106.co...	44.202.128.106	-
CA2 WEB SER...	i-0601326b52d809226	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	ec2-3-208-213-65.com...	3.208.213.65	3.208.213.65

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**

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
BeeDesign-We...	i-034b4e1201a2420d0	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-44-202-128-106.co...	44.202.128.106	-
CA2 WEB SER...	i-0601326b52d809226	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	ec2-3-208-213-65.com...	3.208.213.65	3.208.213.65

Instance: i-034b4e1201a2420d0 (BeeDesign-Webserver-01)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Instance summary

Public IPv4 address
44.202.128.106 | open address

Private IP DNS name (IPv4 only)
ip-172-31-90-105.ec2.internal

Instance type
t2.micro

VPC ID
vpc-0c050c228bd0d14c6

Subnet ID
subnet-006ee127e5b3bdb7b

Auto Scaling Group name

Step 14: Go the left navigation bar.

The screenshot shows the AWS EC2 Instances page. On the left, there is a navigation bar with several categories: Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), Load Balancing (Load Balancers, Target Groups), and Auto Scaling (Auto Scaling Groups). An orange arrow points from the 'Network & Security' category towards the main content area. The main content area displays two instances: 'BeeDesign-Webserver-01' (running, t2.micro) and 'CA2 WEB SER...' (running, t2.micro). Below the instances, there is a detailed view for 'Instance: i-034b4e1201a2420d0 (BeeDesign-Webserver-01)', showing details like Instance ID, IPv6 address, Hostname type, IP name, Answer private resource DNS name, Auto-assigned IP address, IAM Role, IMDSv2, and VPC ID. There are also sections for Public IPv4 address, Instance state, Private IP DNS name, Instance type, Subnet ID, and Tags.

Step 15: Locate Network and Security

The screenshot shows the AWS Network & Security page. On the left, there is a navigation bar with categories: Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces). An orange arrow points from the 'Network & Security' category towards the 'Security Groups' section. The main content area lists the following security groups: Security Groups, Elastic IPs, Placement Groups, Key Pairs, and Network Interfaces.

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

The screenshot shows the AWS Elastic IP addresses page. On the left, there's a sidebar with various services like Savings Plans, Reserved Instances, Dedicated Hosts, etc. The main area shows a table titled "Elastic IP addresses (1/1)". One row is selected, showing the details: Name (EIP-CA2-DEMO), Allocated IPv4 address (3.208.213.65), Type (Public IP), and Allocation ID (eipalloc-091e7e3). At the top right of the table, there's a yellow "Allocate Elastic IP address" button. An orange arrow points to this 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.

This screenshot shows the "Allocate Elastic IP address" configuration page. It has several sections: "Elastic IP address settings" (with a "Network Border Group" dropdown set to "us-east-1"), "Public IPv4 address pool" (radio buttons for "Amazon's pool of IPv4 addresses" (selected), "Public IPv4 address that you bring to your AWS account", and "Customer owned pool of IPv4 addresses"), "Global static IP addresses" (a note about AWS Global Accelerator), and a "Create accelerator" button. Below these is a "Tags - optional" section with a note about tags. A tag named "Name: BeeDesign-Webserver-01-EIP" is listed. A "Name Tag for the EIP" input field is highlighted with a box. At the bottom are "Cancel" and "Allocate" buttons. An orange arrow points to the "X" button of the selected tag.

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

The screenshot shows the AWS Elastic IP Addresses console. At the top, a green banner displays the message "Elastic IP address allocated successfully. Elastic IP address 52.2.77.200 / BeeDesign-Webserver-01-EIP". Below the banner, the main interface shows a table titled "Elastic IP addresses (1/1)". The table has columns: Name, Allocated IPv4 addr..., Type, Allocation ID, Reverse DNS record, and Associated. A single row is selected, showing "BeeDesign-Webserver-01-EIP" under Name, "52.2.77.200" under Allocated IPv4 addr..., "Public IP" under Type, and "eipalloc-06ae714ba4dd45b1" under Allocation ID. An orange arrow points from the text "Step 19" to the "Associated" column of the selected row. The bottom section of the screen shows the details for the IP address 52.2.77.200, with tabs for "Summary" and "Tags".

Associate EIP with instance

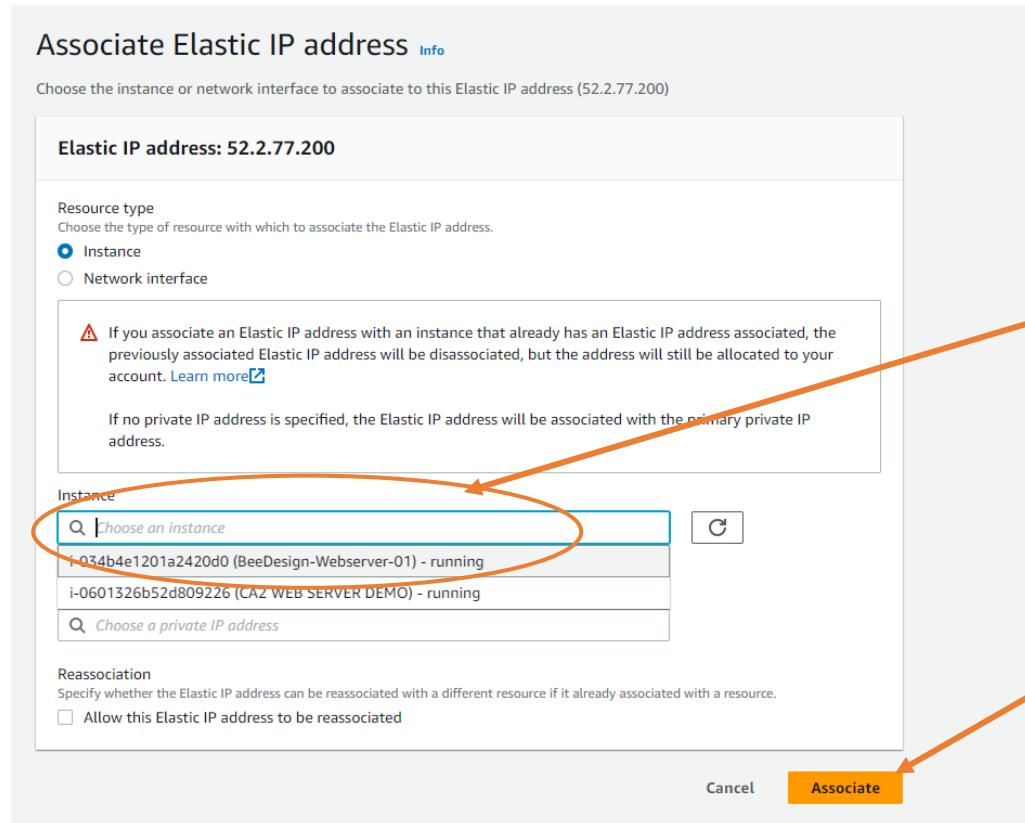
Now we will associate the EIP address to our Ec2 instance.

Step 20: Click “Allocate Elastic IP Address” upon clicking Action button.

The screenshot shows the same AWS Elastic IP Addresses console as the previous one. The green banner at the top still says "Elastic IP address allocated successfully. Elastic IP address 52.2.77.200 / BeeDesign-Webserver-01-EIP". The "Actions" button in the top right corner is highlighted with an orange circle. A secondary orange arrow points from the text "Step 20" to the "Allocate Elastic IP Address" button. The rest of the interface is identical to the previous screenshot, showing the table of elastic IP addresses and the detailed view for 52.2.77.200.

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.

The screenshot shows the AWS Elastic IP addresses page with a success message: 'Elastic IP address associated successfully. Elastic IP address 52.2.77.200 has been associated with instance i-034b4e1201a2420d0'. Below this, a table lists the associated EIP details. To the right, a Windows Notepad window titled 'BeeDesign-EIP' contains the text '52.2.77.200'. The table data is as follows:

Name	Allocated IPv4 address	Type	Allocation ID
BeeDesign-Webserver-01-EIP	52.2.77.200	Public IP	eipalloc-06aee714ba4dd45b1

Step 23: Go back to your **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.

The screenshot shows the AWS CloudWatch Instances interface and a terminal window. In the CloudWatch Instances list, two instances are shown: 'BeeDesign-We...' (i-034b4e1201a2420d0) and 'CA2 WEB SER...' (i-0601326b52d809226). Both are running. The terminal window titled 'BeeDesign-EIP' displays the command 'curl ifconfig.me' with the output '52.2.77.200'. An orange circle highlights the public IP address '52.2.77.200' in the terminal output, and another orange circle highlights the 'Public IPv4 address' field in the CloudWatch Instances details panel.

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.

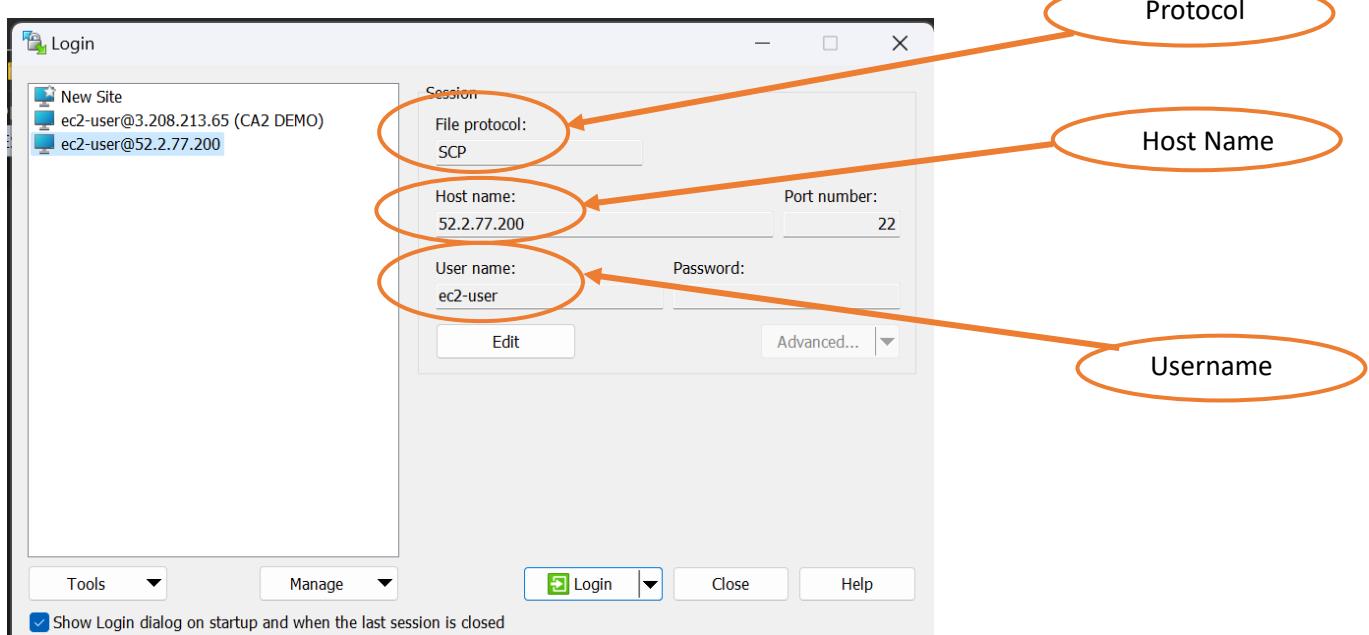
The screenshot shows the WinSCP application interface. The left pane shows a file tree with a folder named 'CA2DEMOBEE - Documents'. The right pane shows a 'Login' dialog for a new site named 'ec2-user@3.208.213.65 (CA2 DEMO)'. The 'Session' tab is selected, showing fields for 'File protocol' (SFTP), 'Host name' (3.208.213.65), 'Port number' (22), 'User name' (ec2-user), and 'Password'. Below the dialog is a file list showing recent changes. At the bottom, there is a status bar with file information: 'LABTASK2-SINGHSAK... 1,312 KB Microsoft Word Document' and 'logo.drawio 1 KB DRAWIO File'.

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.

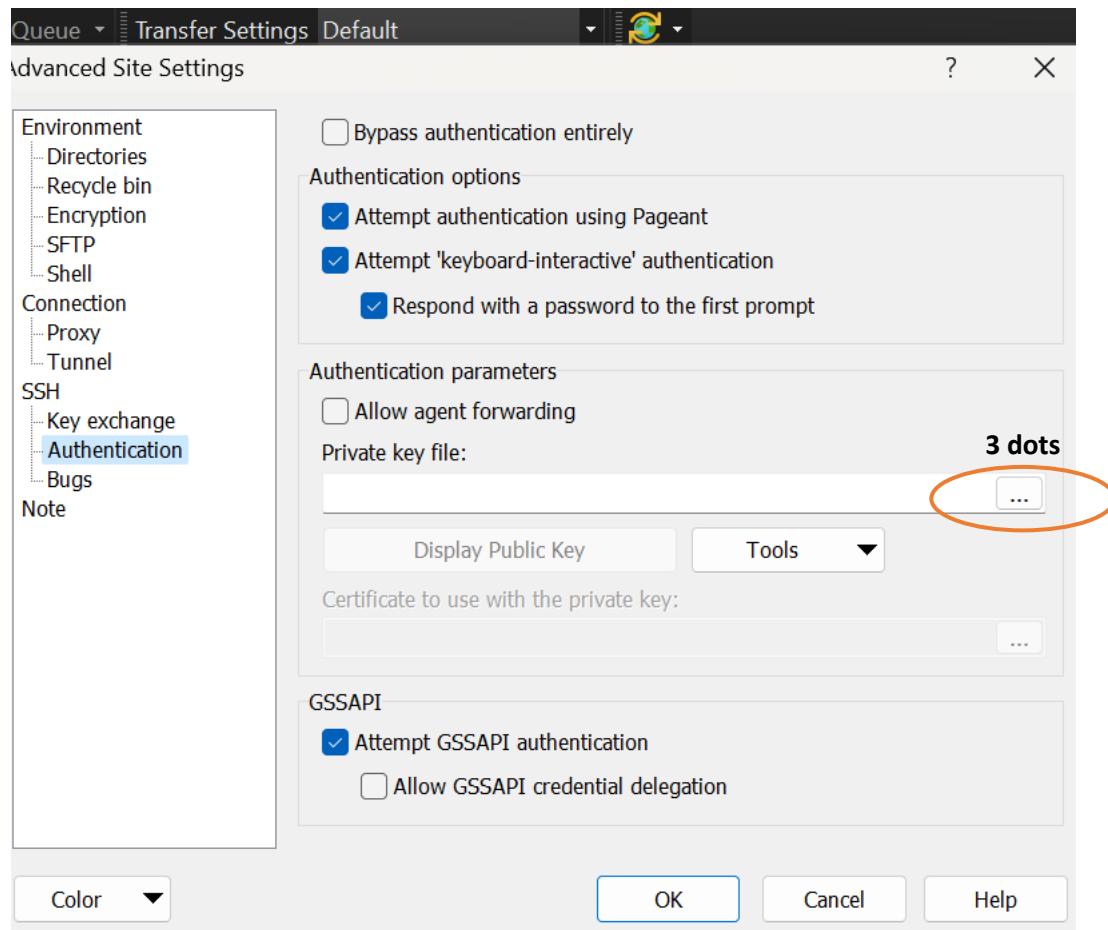
Step 27: Click on the “advanced”.



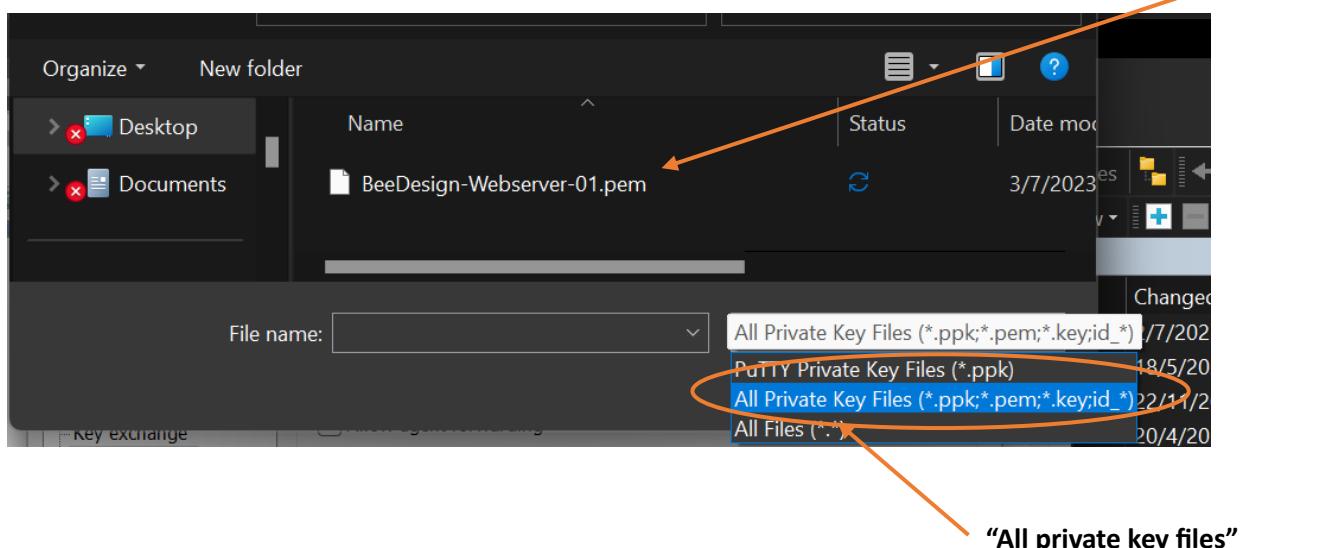
Step 28: Navigate to “Authentication”

1. Press the 3 dots.

1

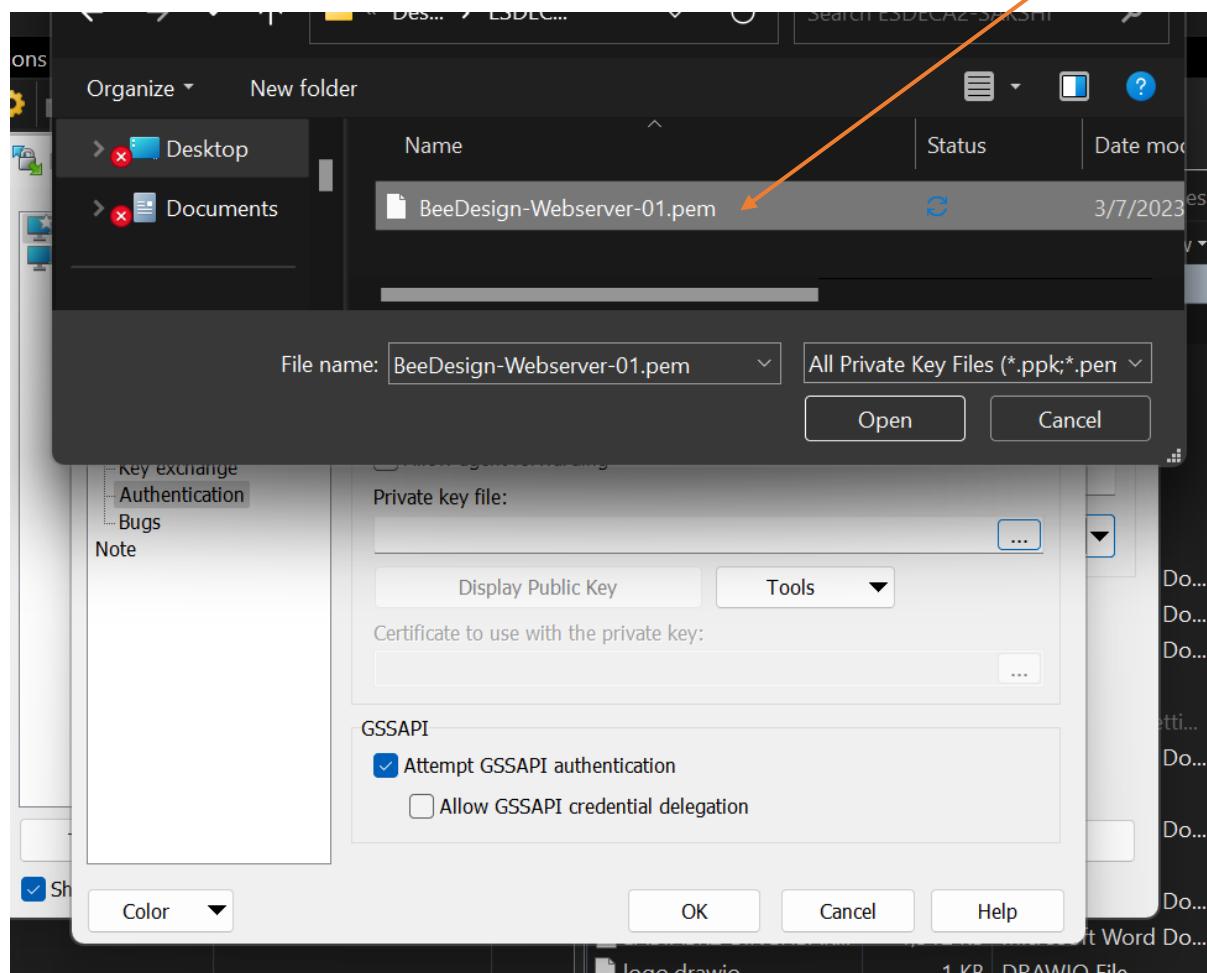


2. Your explorer page will pop up, **navigate to your Pem file**.

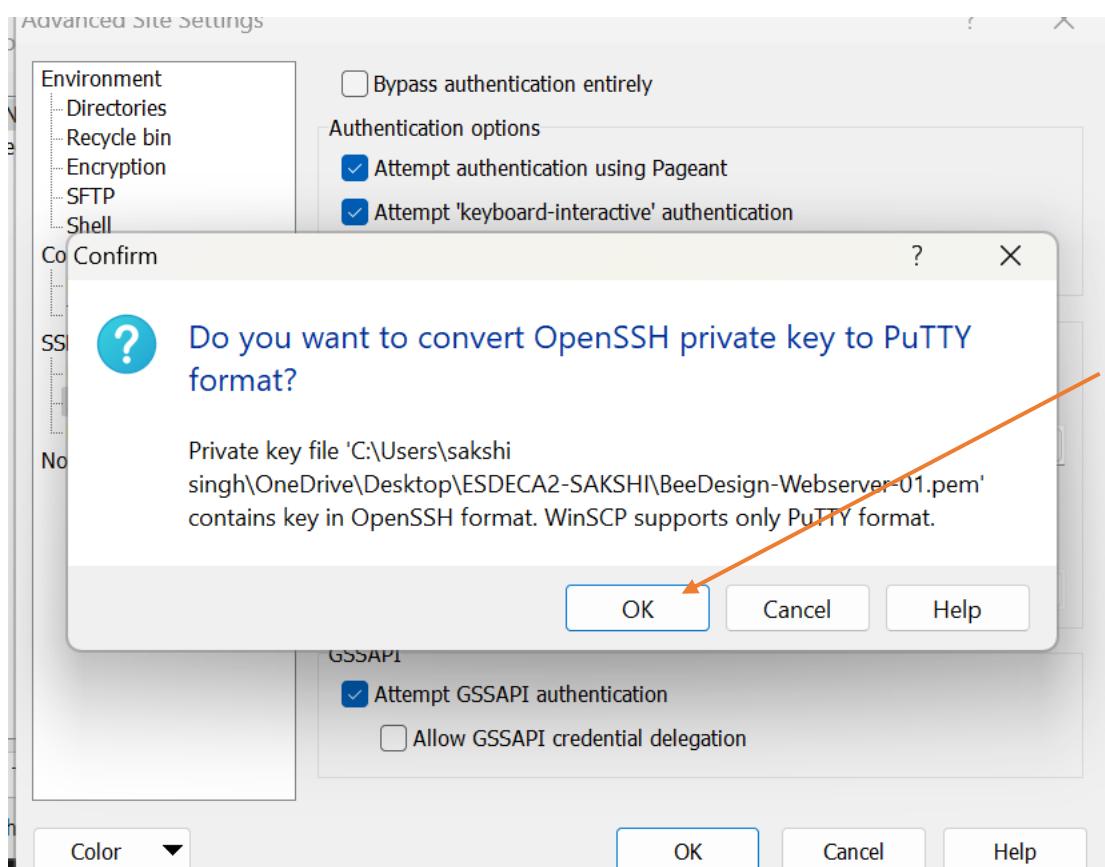


.pem file

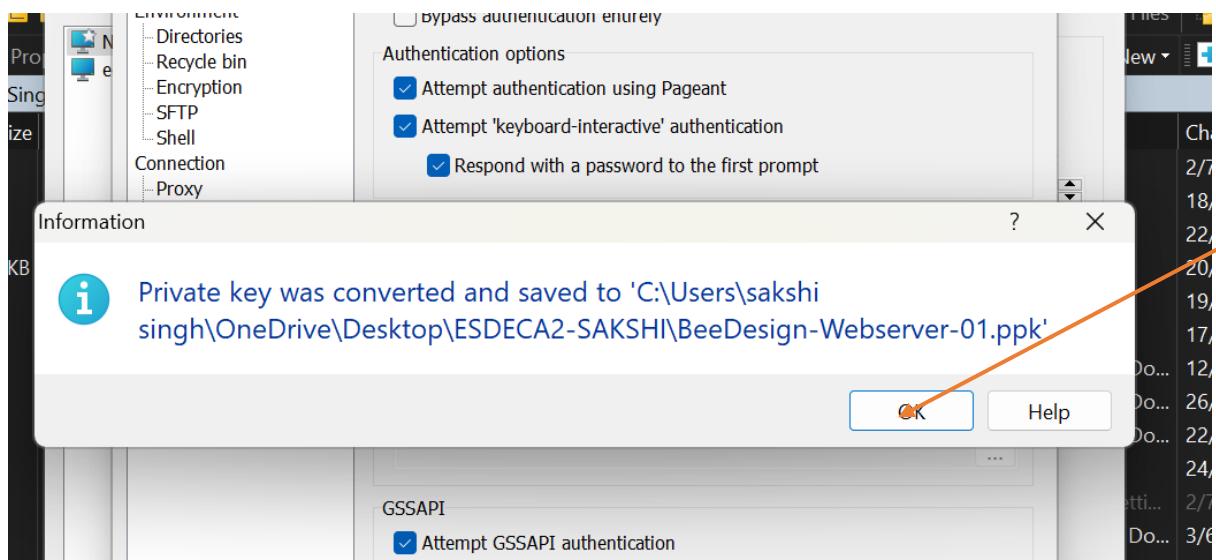
2. Change the type, to “all private key files”.



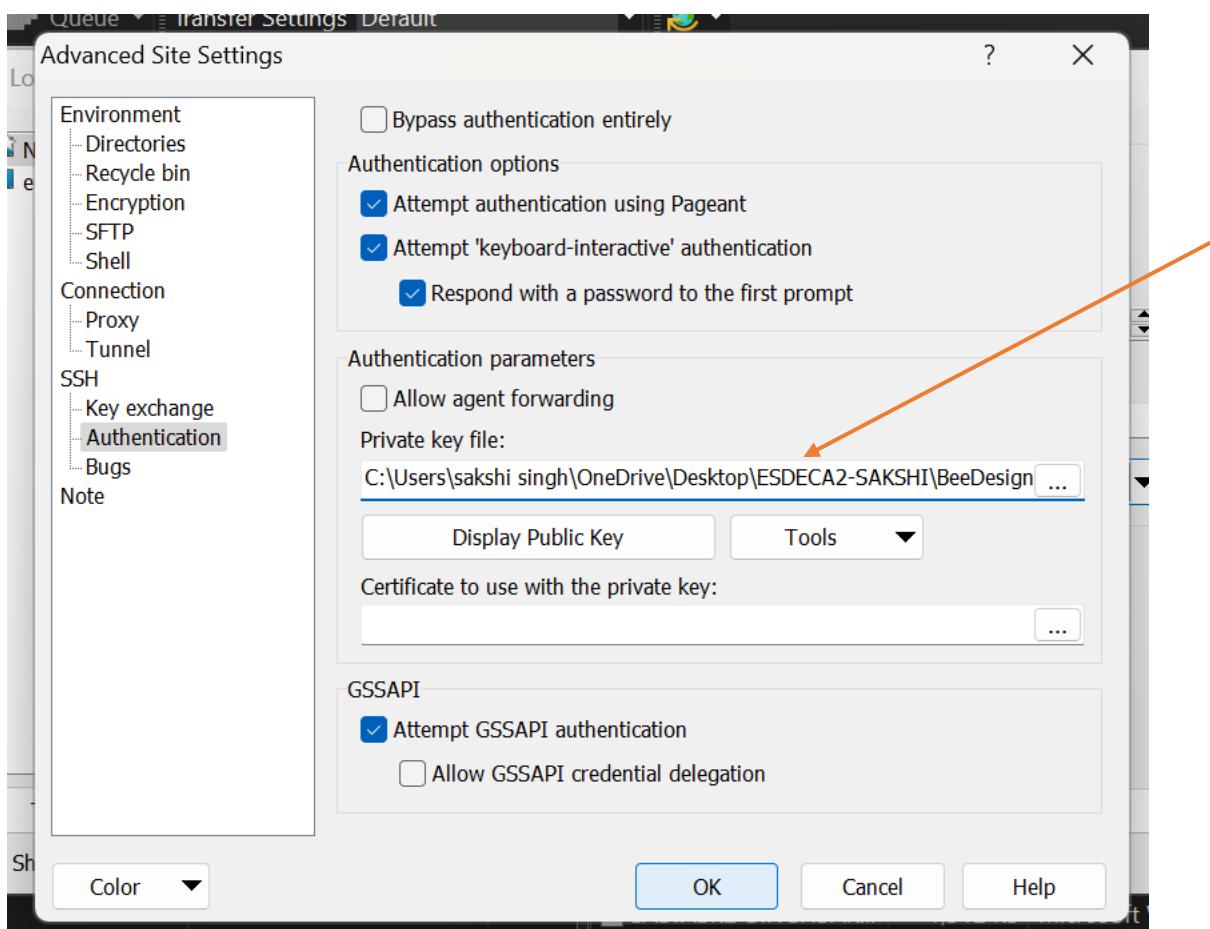
4.Upload the .pem file.



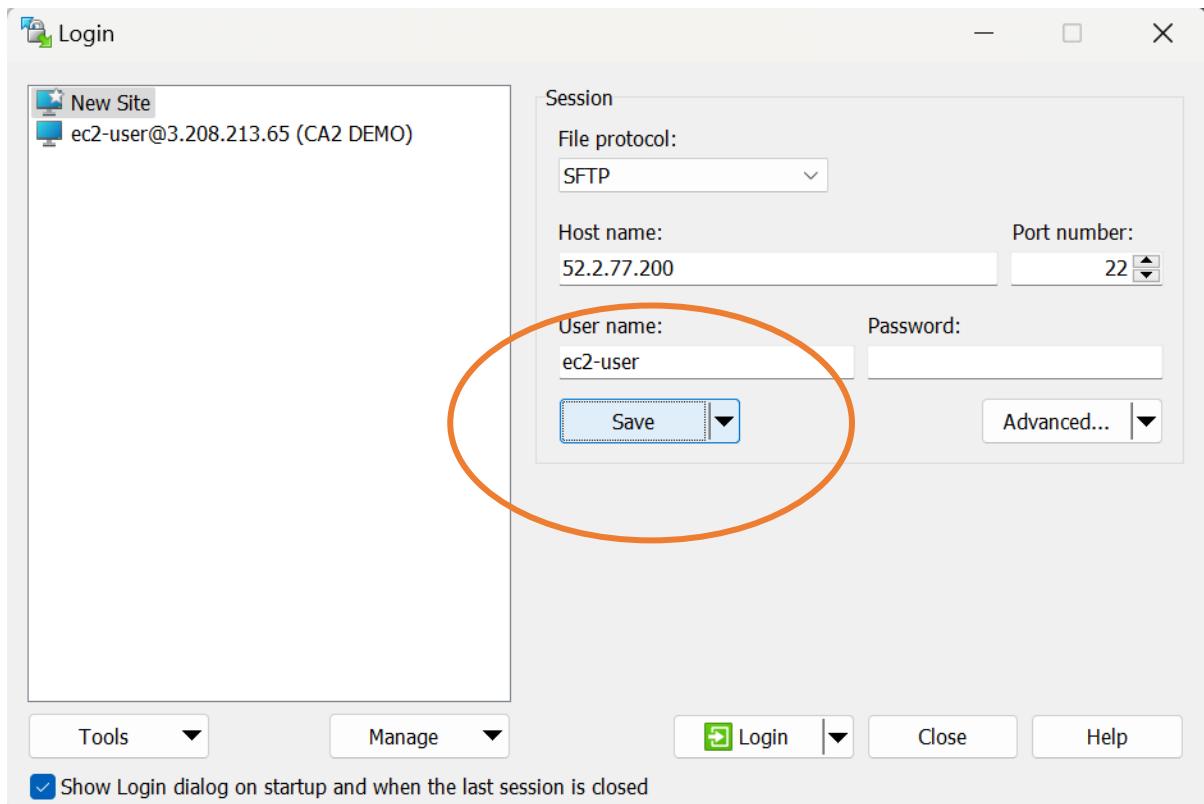
Step 29: Click "ok".



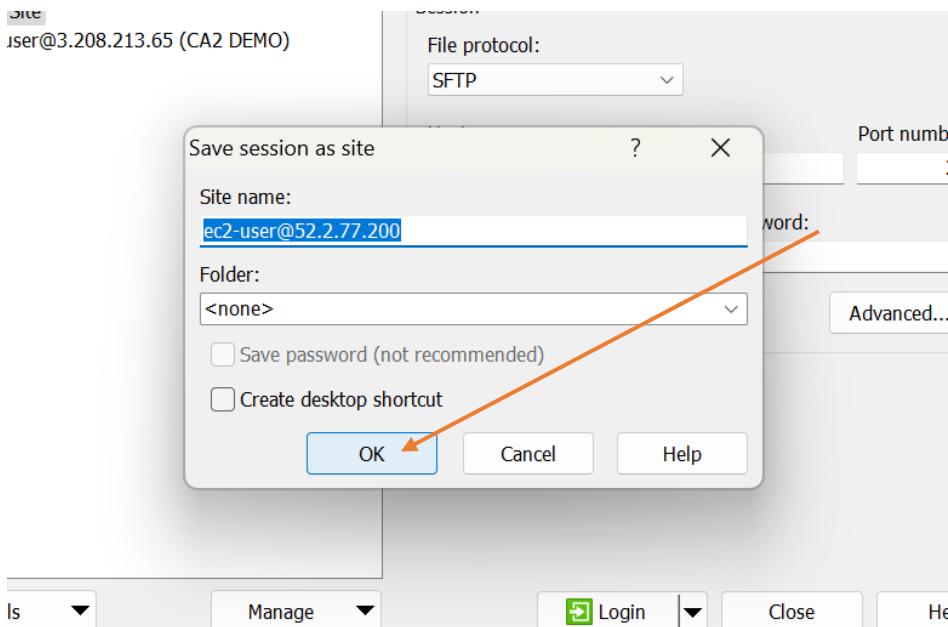
This is how the final settings look like.



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

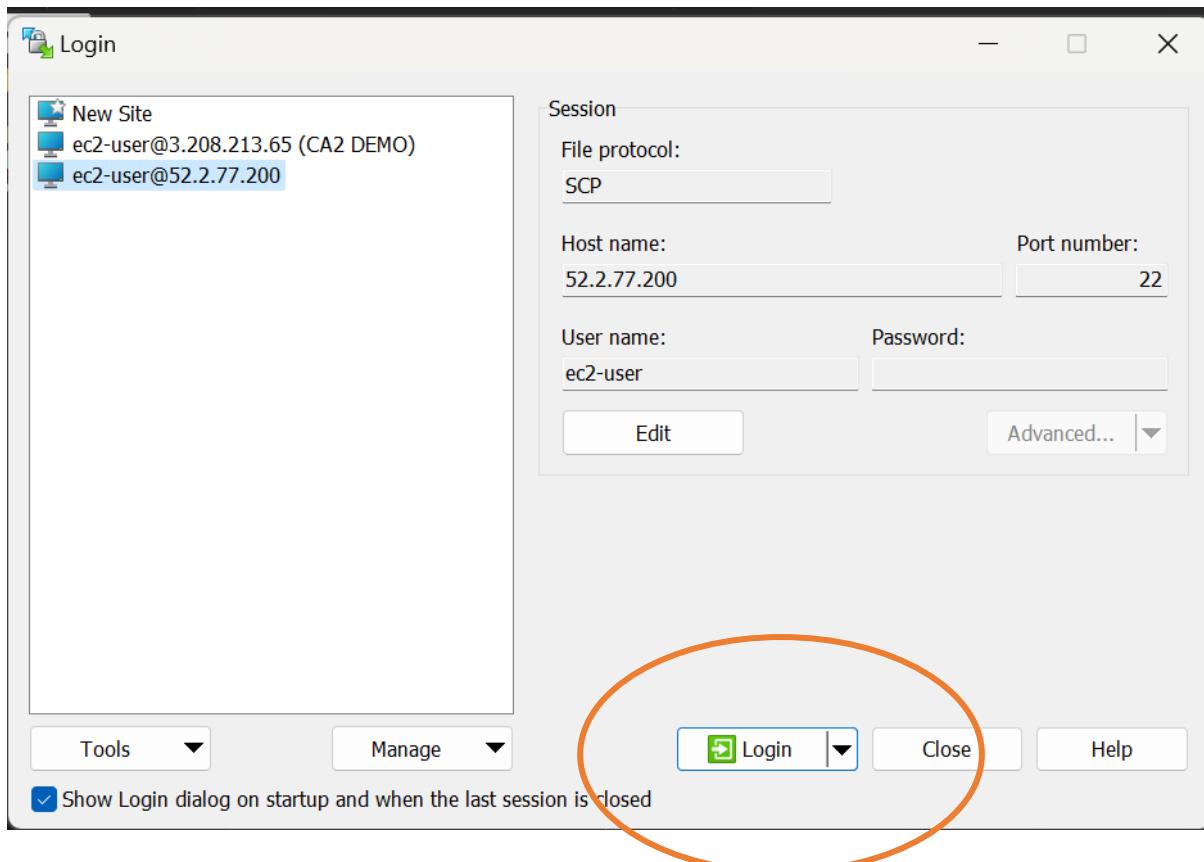


Step 30: Press ‘ok’ to confirmation of the new site name with our EIP address.

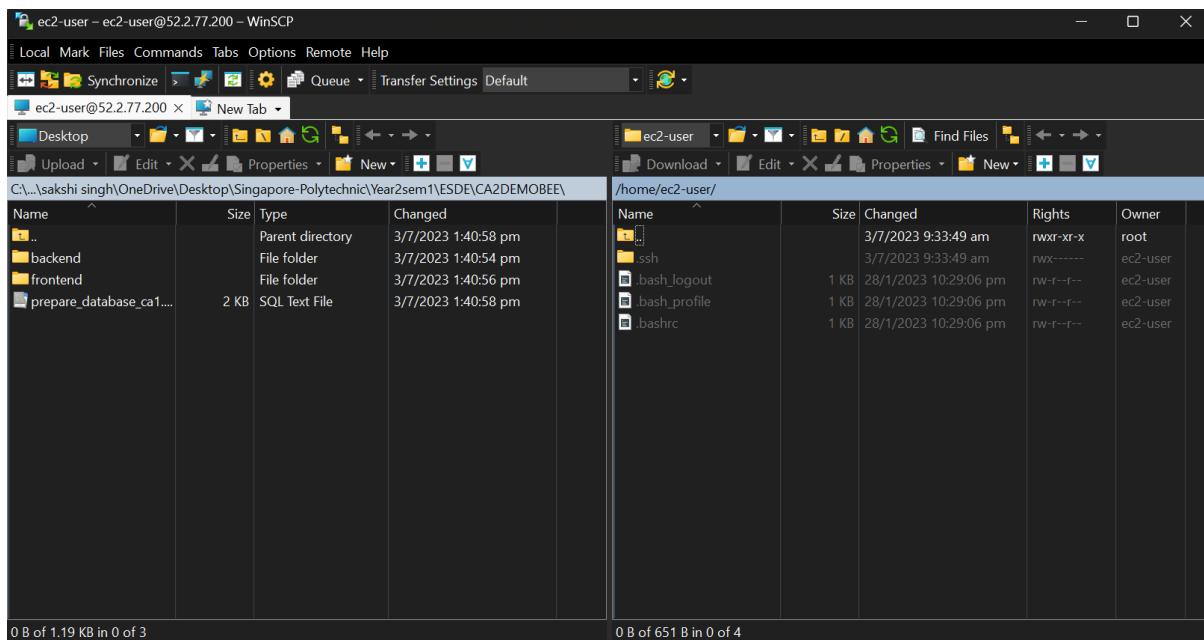


Step 31: Check that the new site name with the username and hostname is created.

Step 32: Press "Login"



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

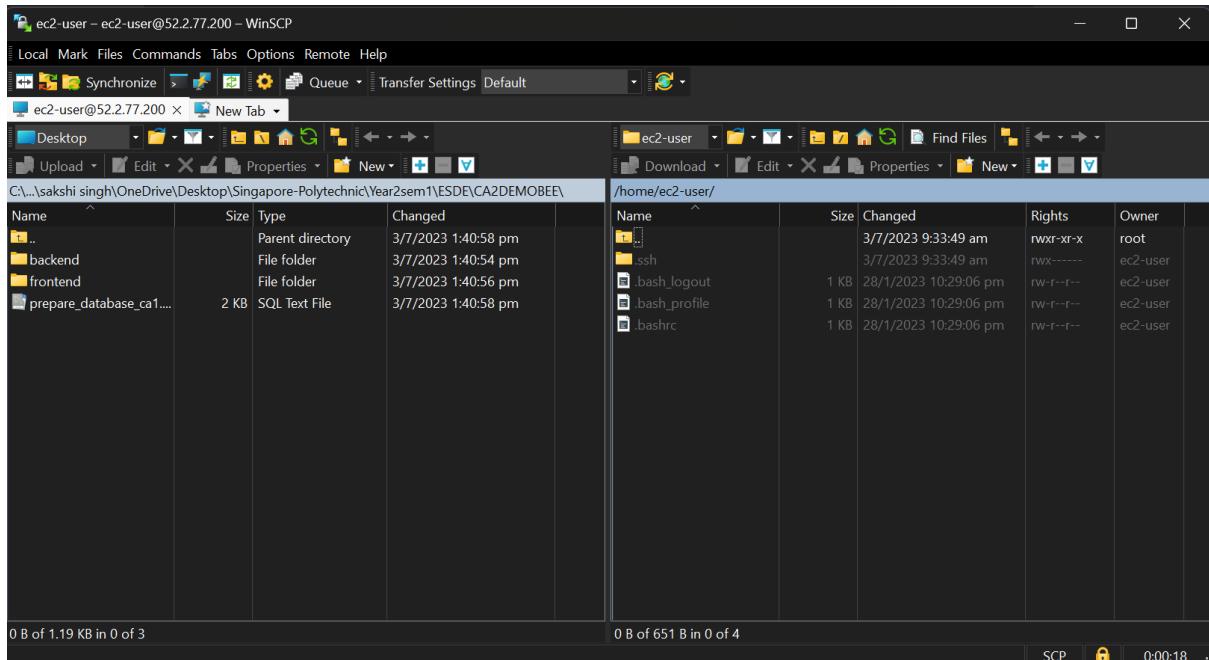


Part 2- Install NodeJS and copy Bee Design app to EC2.

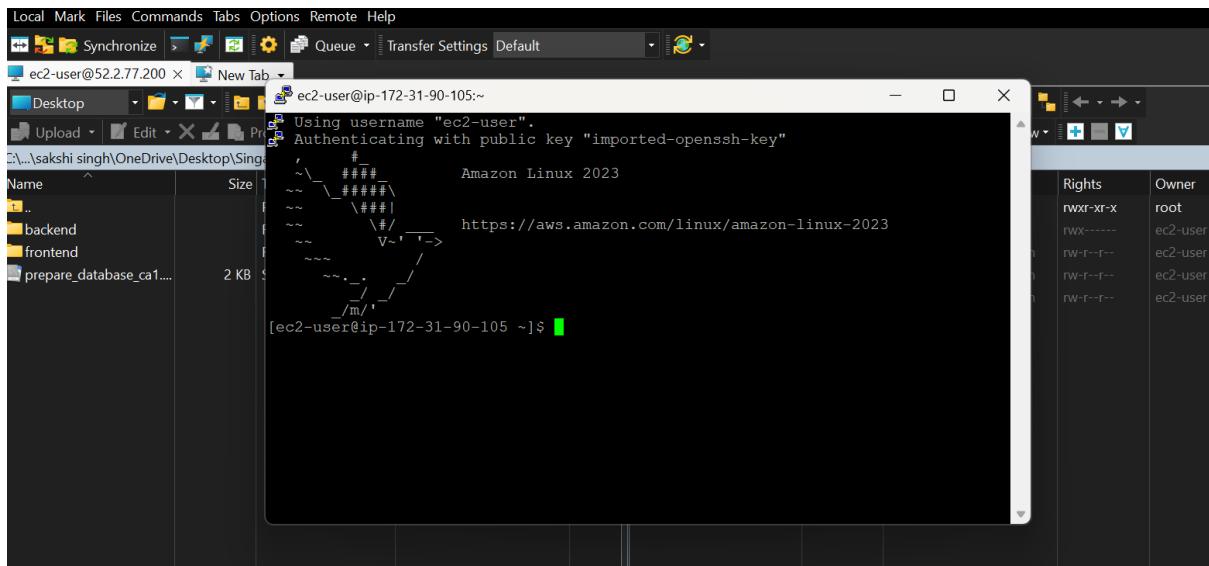
Now that you have set up your ec2 instance, we will be **installing node js , deploying the bee design application** and **configuring security settings** to allow the app to be accessed.

Installing Node js

Step 1: Go to WinSCP



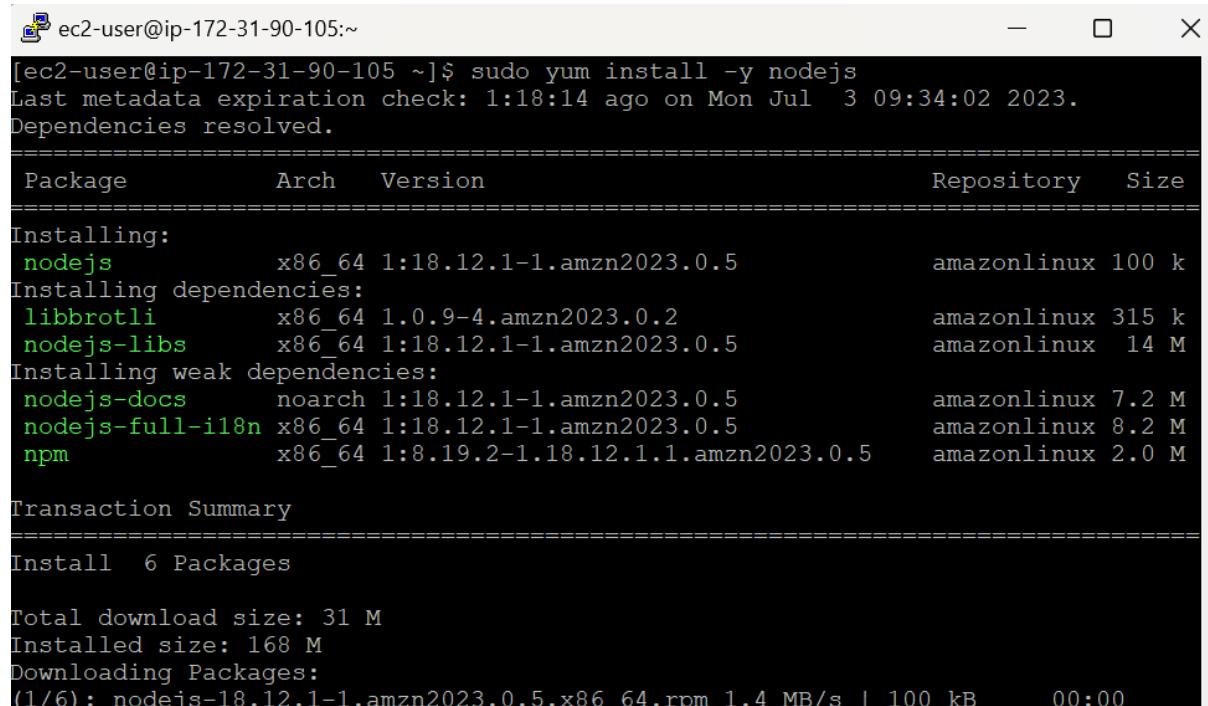
Step 2: Open Putty Terminal



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)

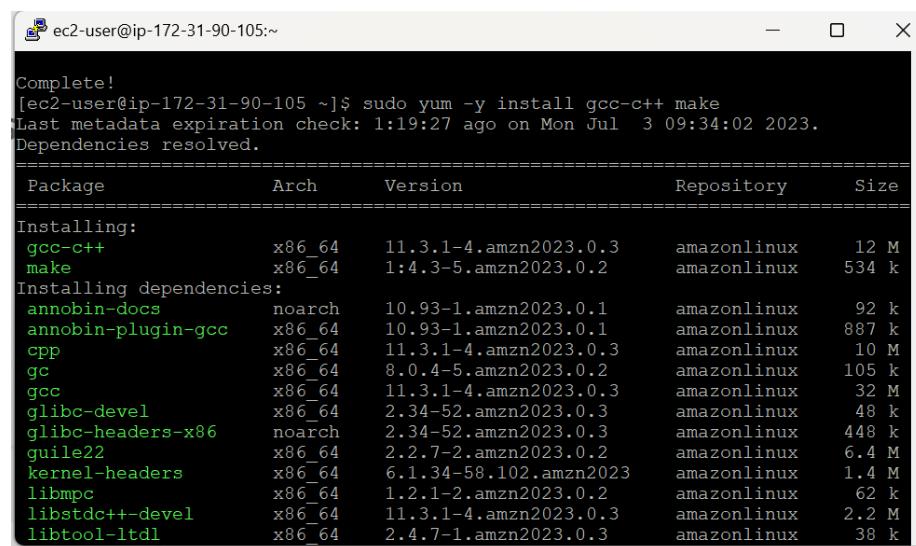
Run sudo yum install -y nodejs



```
[ec2-user@ip-172-31-90-105 ~]$ sudo yum install -y nodejs
Last metadata expiration check: 1:18:14 ago on Mon Jul 3 09:34:02 2023.
Dependencies resolved.
=====
 Package           Arch   Version            Repository  Size
=====
Installing:
 nodejs           x86_64  1:18.12.1-1.amzn2023.0.5      amazonlinux 100 k
Installing dependencies:
 libbrotli        x86_64  1.0.9-4.amzn2023.0.2      amazonlinux 315 k
 nodejs-libs      x86_64  1:18.12.1-1.amzn2023.0.5      amazonlinux 14 M
Installing weak dependencies:
 nodejs-docs      noarch  1:18.12.1-1.amzn2023.0.5      amazonlinux 7.2 M
 nodejs-full-i18n x86_64  1:18.12.1-1.amzn2023.0.5      amazonlinux 8.2 M
 npm              x86_64  1:8.19.2-1.18.12.1.1.amzn2023.0.5  amazonlinux 2.0 M
Transaction Summary
=====
Install 6 Packages

Total download size: 31 M
Installed size: 168 M
Downloading Packages:
(1/6): nodejs-18.12.1-1.amzn2023.0.5.x86_64.rpm 1.4 MB/s | 100 kB     00:00
```

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



```
Complete!
[ec2-user@ip-172-31-90-105 ~]$ sudo yum -y install gcc-c++ make
Last metadata expiration check: 1:19:27 ago on Mon Jul 3 09:34:02 2023.
Dependencies resolved.
=====
 Package           Arch   Version            Repository  Size
=====
Installing:
 gcc-c++          x86_64  11.3.1-4.amzn2023.0.3      amazonlinux 12 M
 make             x86_64  1:4.3-5.amzn2023.0.2      amazonlinux 534 k
Installing dependencies:
 annobin-docs     noarch  10.93-1.amzn2023.0.1      amazonlinux 92 k
 annobin-plugin-gcc x86_64  10.93-1.amzn2023.0.1      amazonlinux 887 k
 cpp              x86_64  11.3.1-4.amzn2023.0.3      amazonlinux 10 M
 gc               x86_64  8.0.4-5.amzn2023.0.2      amazonlinux 105 k
 gcc              x86_64  11.3.1-4.amzn2023.0.3      amazonlinux 32 M
 glibc-devel      x86_64  2.34-52.amzn2023.0.3      amazonlinux 48 k
 glibc-headers-x86 noarch  2.34-52.amzn2023.0.3      amazonlinux 448 k
 guile22         x86_64  2.2.7-2.amzn2023.0.2      amazonlinux 6.4 M
 kernel-headers   x86_64  6.1.34-58.102.amzn2023      amazonlinux 1.4 M
 libmpc           x86_64  1.2.1-2.amzn2023.0.2      amazonlinux 62 k
 libstdc++-devel  x86_64  11.3.1-4.amzn2023.0.3      amazonlinux 2.2 M
 libtool-ltdl    x86_64  2.4.7-1.amzn2023.0.3      amazonlinux 38 k
```

Run node --version

```
Complete!
[ec2-user@ip-172-31-90-105 ~]$ node --version
v18.12.1 ←
[ec2-user@ip-172-31-90-105 ~]$ █
```

Node version
installed in Ec2

Deploy Bee Design

Step 4: Locate Index.js

```
 9  app.use(express.static(__dirname + '/public'));
10
11
12  app.get("/", (req, res) => {
13    |   res.sendFile("/public/home.html", { root: __dirname });
14  });
15
16
17  app.listen(port,hostname,function(){
18
19    |   console.log(`Server hosted at http://${hostname}:${port}`);
20  });
21
```

Step 5: Remove hostname

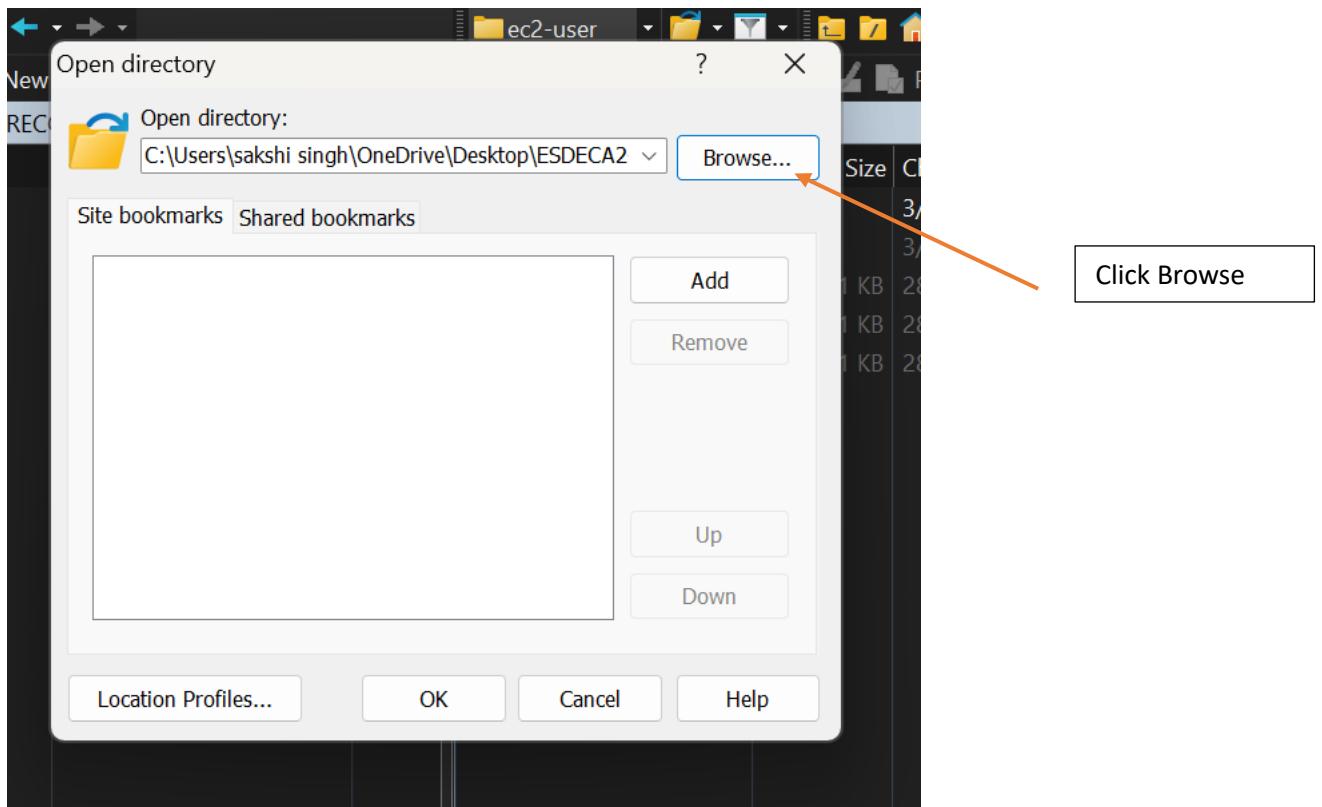
```
app.listen(port,function(){
|
|   console.log(`Server hosted at http://${hostname}:${port}`);
|});
```

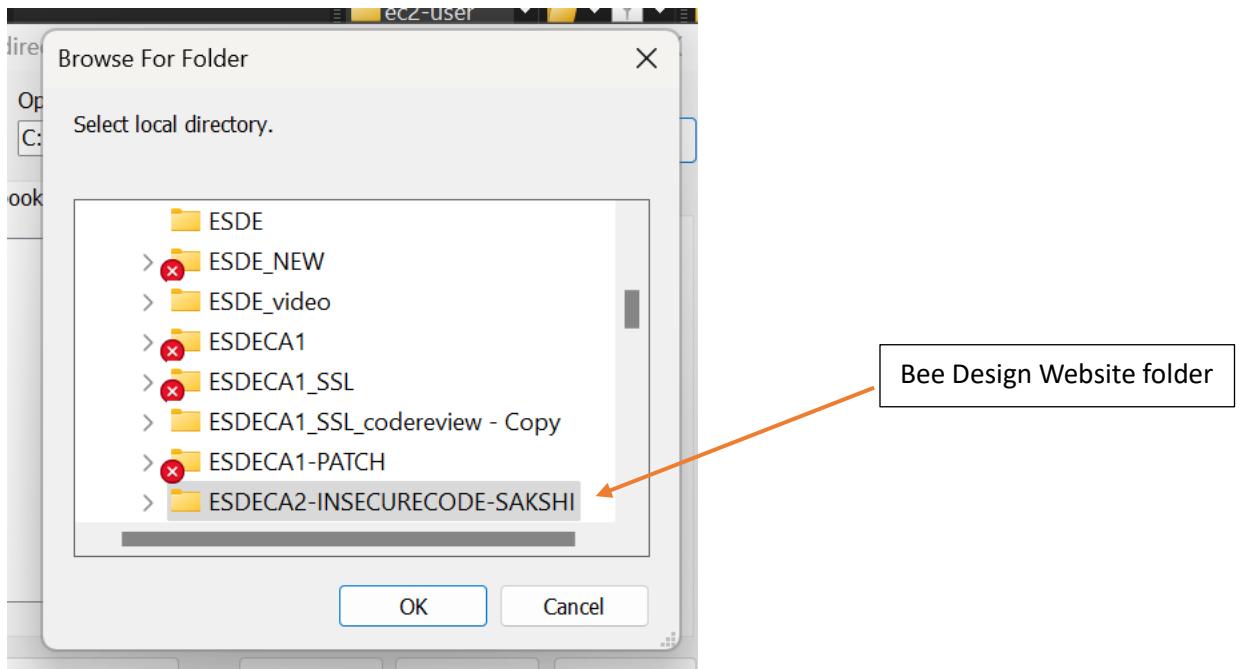
Step 6 : Locate .env and **add the cloudinary and mailtrap ids** so as to ensure that your Bee Design website can **successfully request information** from these domains.

Terminal window showing environment variables:

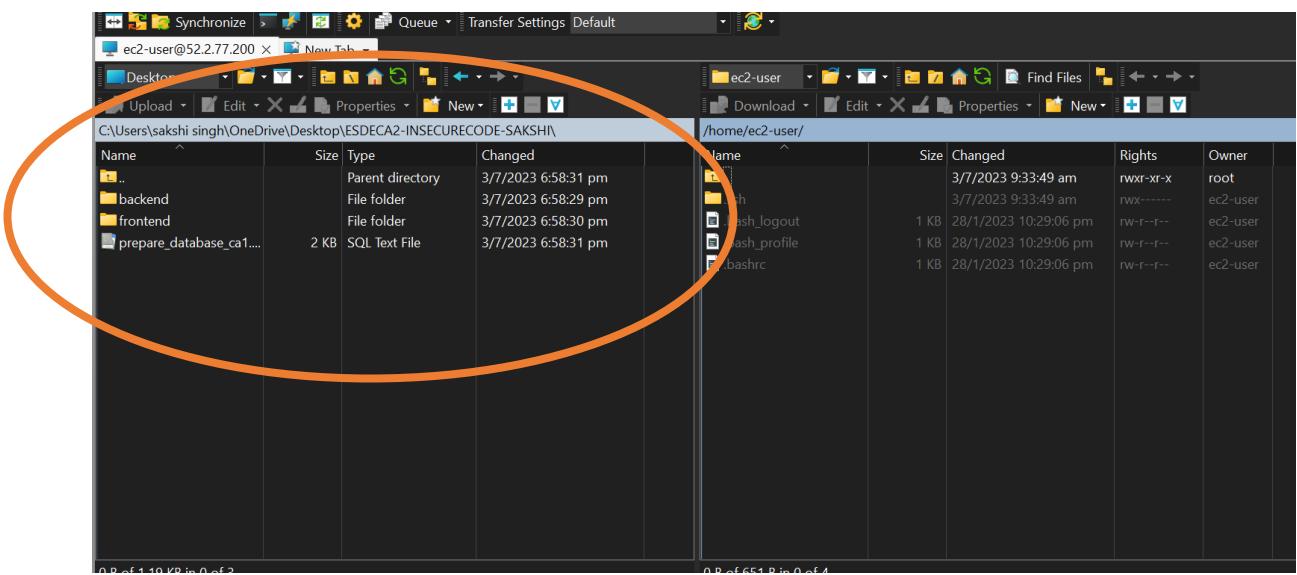
```
js index.js .env
backend > .env
1 DB_USERNAME=esde_ca_adminuser
2 DB_PASSWORD=P@ssw0rd
3 DB_DATABASE_NAME=ay2324s1_st0505_esde_ca
4 JWTKEY=QSBKV1QgY29kZSBmb3IgQVkyMjIzcEgRVNERSBzdHVkZW50cW
5
6 CLOUDINARY_CLOUD_NAME=dqmyxllr8
7 CLOUDINARY_API_KEY=678552142329395
8 CLOUDINARY_API_SECRET=XC-xIa54Ma_U4hWf8P49SD17xGk
9
10 MAILTRAP_USERNAME=dd1ce7739fab39
11 MAILTRAP_PASSWORD=90851f84f056ef
```

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

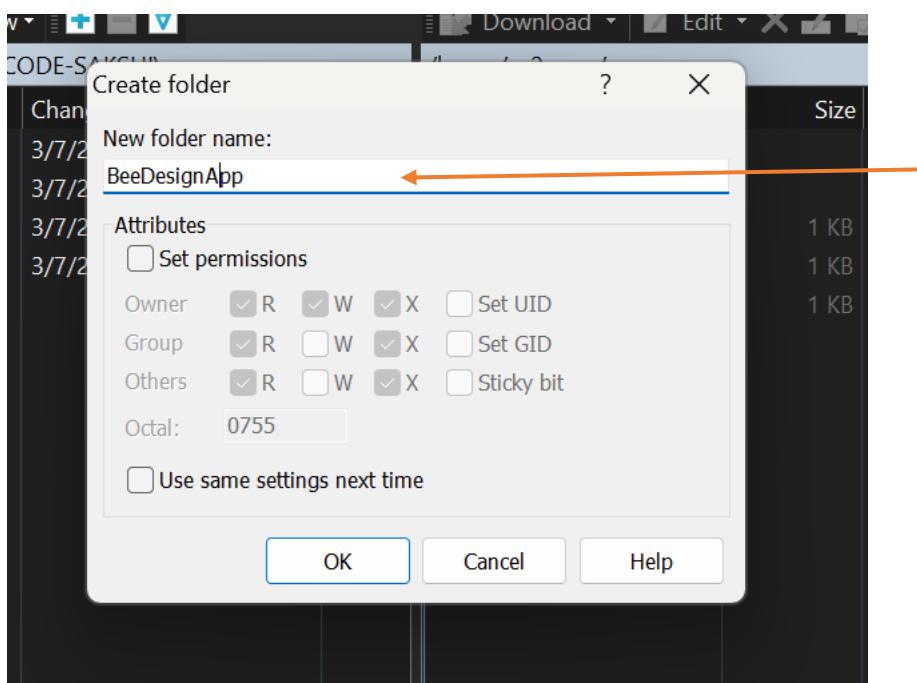
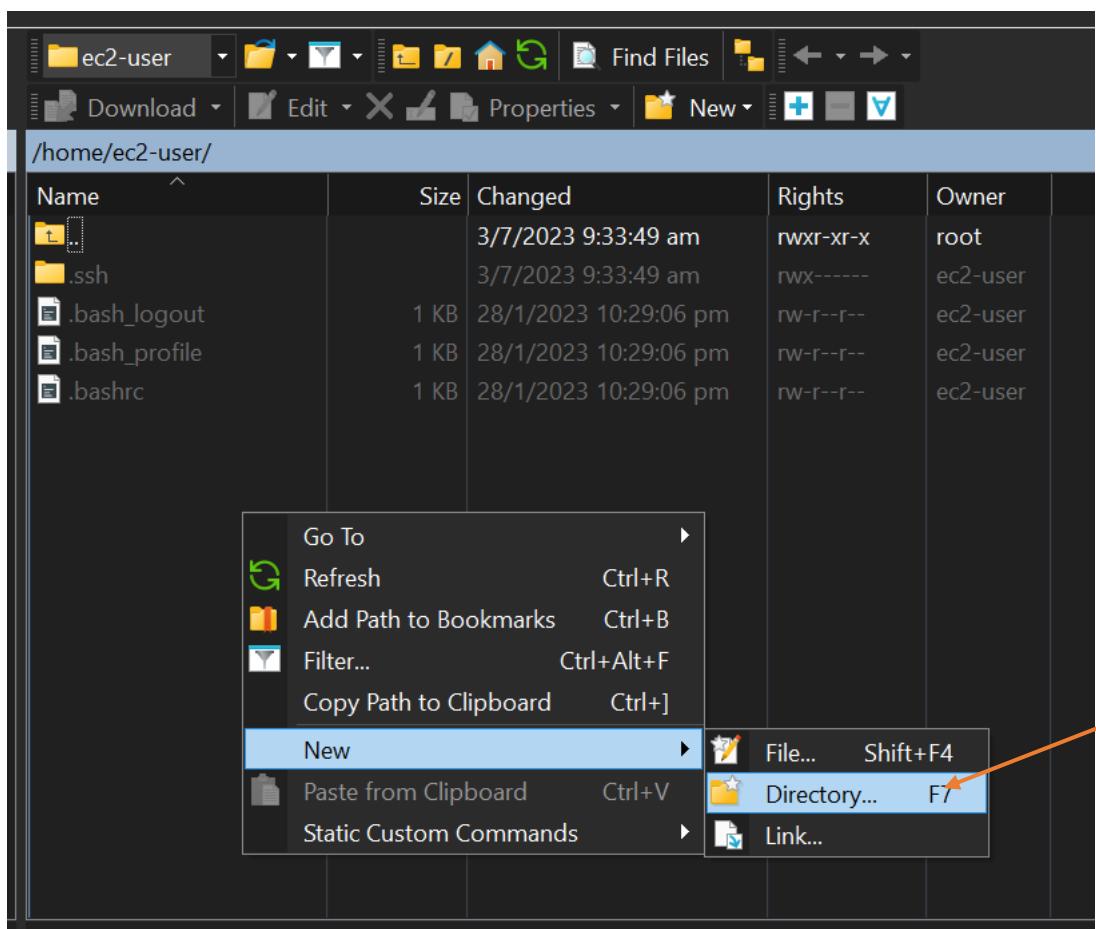




Step 8: You have successfully navigated to the Bee Design folder .



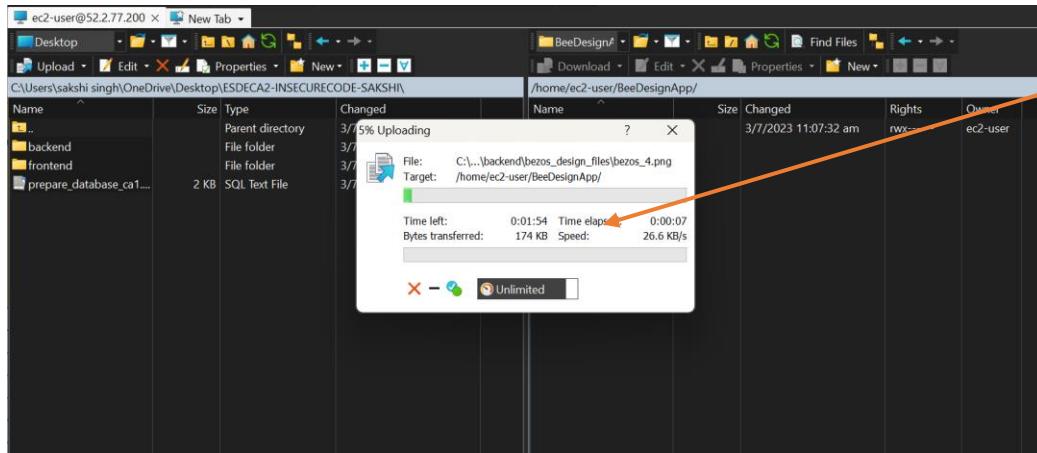
Step 9: Create new directory using WinSCP in remote EC2 called **bee design app**.



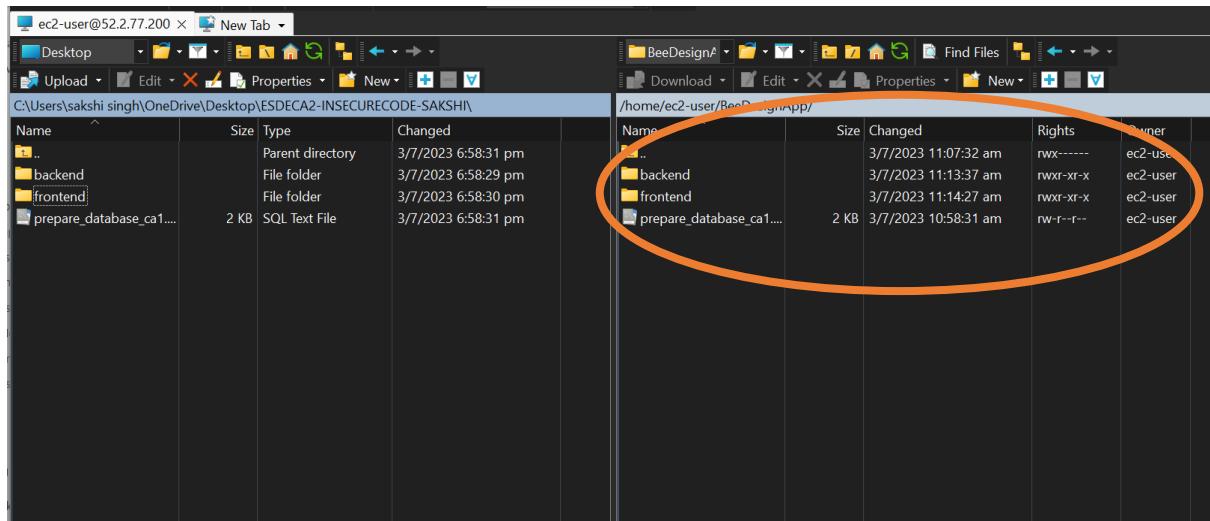
The folder is created.

.ssh	3/7/2023 9:33:49 am	rwx-----	ec2-user
BeeDesignApp	3/7/2023 11:07:32 am	rwxr-xr-x	ec2-user
.bash_logout	1 KB 28/1/2023 10:29:06 pm	rw-r--r--	ec2-user

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



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.

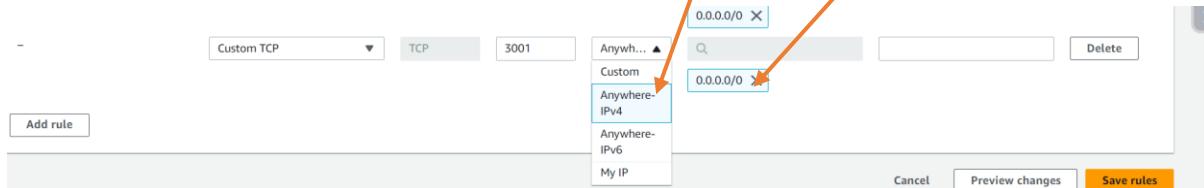
Step 2: Click on the **link** in 'Security Groups' to configure the settings for your website.

The screenshot shows the AWS EC2 Instances dashboard. At the top, there is a table listing two instances: 'BeeDesign-We...' (running, t2.micro, 2/2 checks passed) and 'CA2 WEB SER...' (stopped, t2.micro). Below the table, the details for the selected instance ('BeeDesign-Webserver-01') are shown. The 'Security' tab is highlighted with an orange arrow. A link labeled 'Link for security groups' is visible. An orange circle highlights the 'Security groups' section, which lists 'sg-06f9fe190c498df81 (launch-wizard-2)'. The bottom part of the screen shows the Inbound rules table with one rule: Name: sgr-0b391f278744502e1, Port range: 22, Protocol: TCP, Source: 0.0.0.0/0, Security groups: launch-wizard-2.

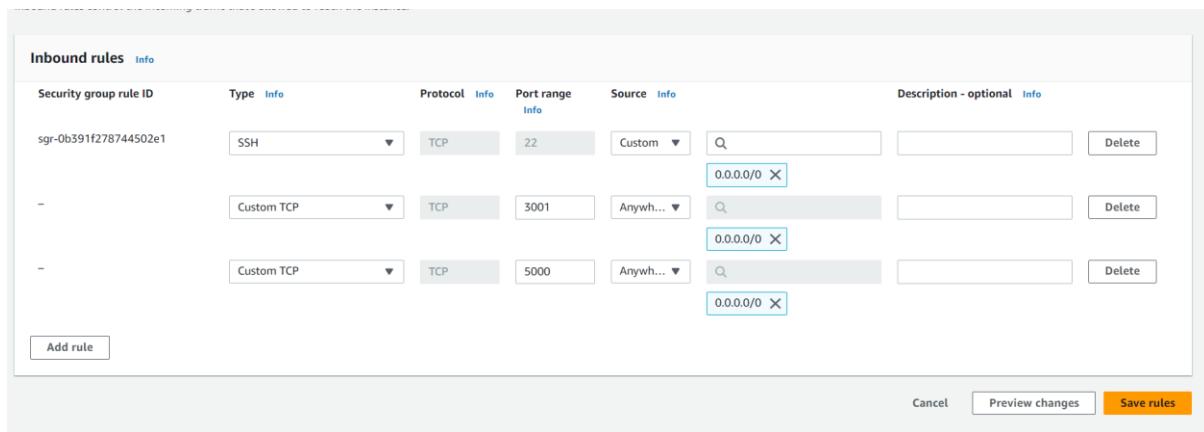
Step 3: Click on Edit Inbound Rules

The screenshot shows the 'Edit inbound rules' page. At the top, there is a message: 'You can now check network connectivity with Reachability Analyzer' and a 'Run Reachability Analyzer' button. Below this, the 'Inbound rules (1/1)' table is displayed. The table has columns: Name, Security group rule ID, IP version, Type, Protocol, Port range, and Description. One rule is listed: Name: sgr-0b391f278744502e1, Security group rule ID: sgr-0b391f278744502e1, IP version: IPv4, Type: SSH, Protocol: TCP, Port range: 22.

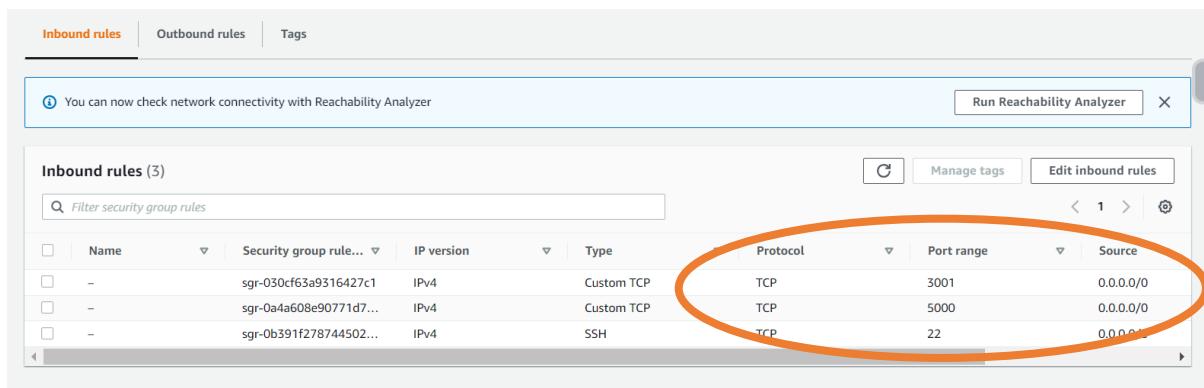
Step 4: Add two rules for port 3001 and port 5000, for 'Anywhere IPv4' - 0.0.0.0



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

```
18.12.1
[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 install
( [ ] ) :: reify:fsevents: sill reify mark deleted [
```

Backend

```
/m/
Last login: Mon Jul 3 10:50:49 2023 from 61.16.104.97
[ec2-user@ip-172-31-90-105 ~]$ cd BeeDesignApp/
[ec2-user@ip-172-31-90-105 BeeDesignApp]$ cd backend/
[ec2-user@ip-172-31-90-105 backend]$ npm install
( [ ] ) :: idealTree: timing idealTree Completed in 1720ms
```

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

Backend

```
a different dependency.

Run `npm audit` for details.
[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

```
npm notice
[ec2-user@ip-172-31-90-105 frontend]$ npm start

> firstfrontend@1.0.0 start
> nodemon index.js

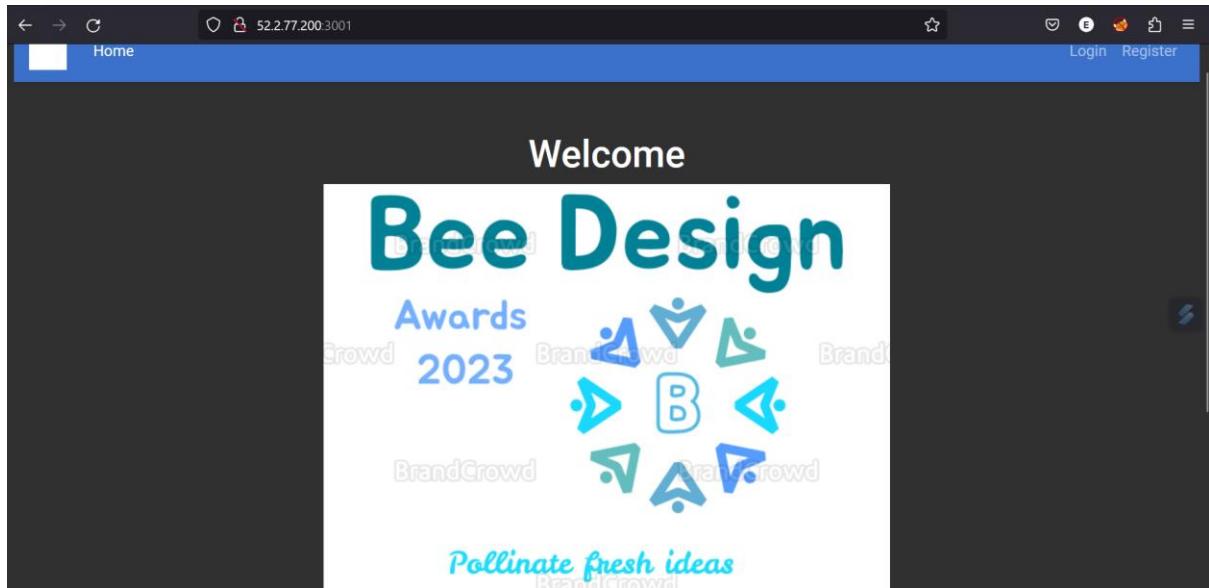
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server hosted at http://localhost:3001
[ ]
```

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

<http://52.277.200:3001>

Step 4: Run the website with the Eip URL <http://52.277.200:3001>

Congratulation you have successfully hosted your website on Ec2 instance.



Part 3- Setup RDS

RDS

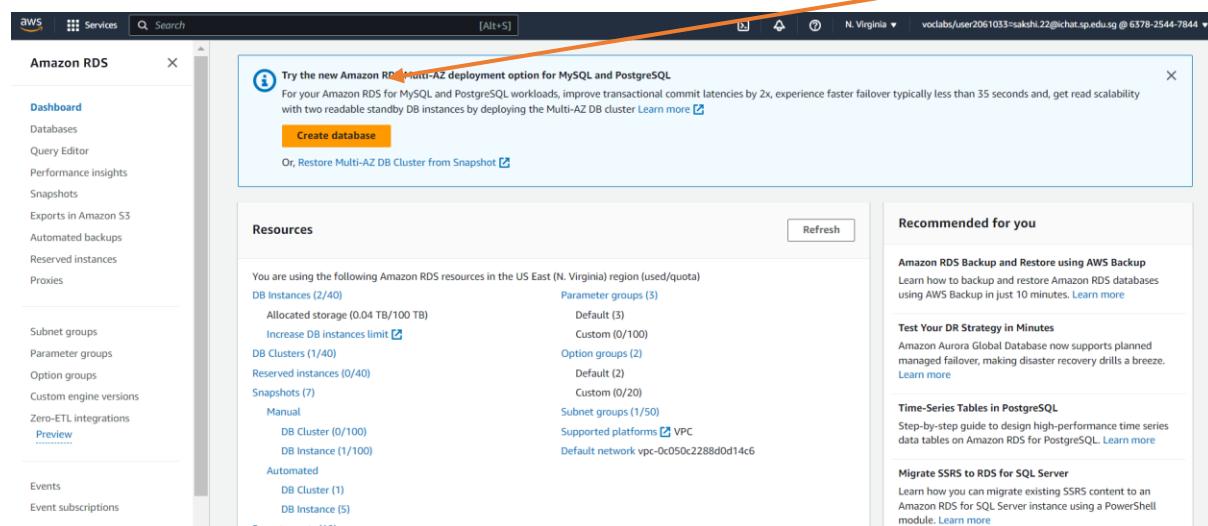
Amazon RDS is a **managed database service on AWS**. It's used for websites to simplify database management, ensure scalability, and enhance reliability.

- 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.

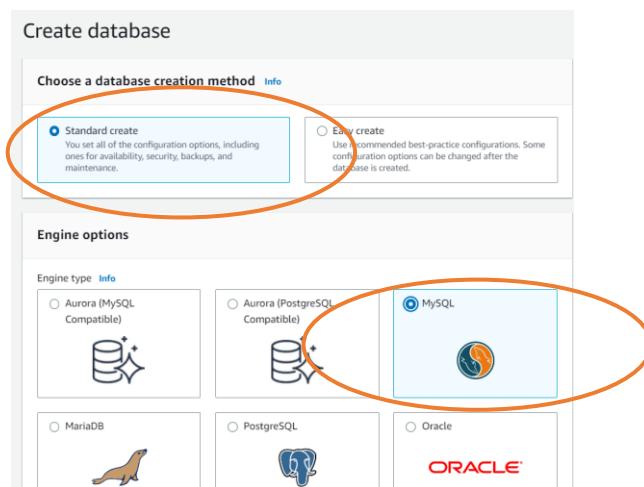
Step 1: Search for RDS in AWS console and open in new tab **Click 'Create database'**.



The screenshot shows the AWS RDS Dashboard. At the top, there is a callout bubble pointing to the 'Create database' button, which is highlighted with an orange arrow. The dashboard displays various RDS resources and metrics. On the left, a sidebar lists navigation options like Dashboard, Databases, Query Editor, etc. The main area shows DB Instances (2/40), Parameter groups (3), and other resource counts. A 'Recommended for you' sidebar on the right suggests topics like Amazon RDS Backup and Restore using AWS Backup, Test Your DR Strategy in Minutes, and Time-Series Tables in PostgreSQL.

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

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



The screenshot shows the 'Create database' wizard. In the first step, 'Choose a database creation method', the 'Standard create' option is selected and highlighted with an orange circle. In the second step, 'Engine options', the 'MySQL' engine type is selected and highlighted with an orange oval. Other engine options shown include Aurora (MySQL Compatible), Aurora (PostgreSQL Compatible), MariaDB, PostgreSQL, and Oracle.

Step 4: For templates , choose free tier

The screenshot shows the 'Templates' section of the AWS RDS setup wizard. At the top, 'Engine Version' is set to 'MySQL 8.0.33'. Below it, there are three options: 'Production' (radio button unselected), 'Dev/Test' (radio button unselected), and 'Free tier' (radio button selected). A callout box points to the 'Free tier' option with the text: 'Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.' An 'Info' link is also present.

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.

The screenshot shows the 'Settings' section of the AWS RDS setup wizard. It includes fields for 'DB instance identifier' (set to 'BeeDesign-database-2'), 'Master username' (set to 'admin2'), and 'Password' (represented by a masked input field). Annotations with orange arrows point from the right side of the image to each of these fields. Callout boxes provide additional information: one for the DB instance identifier stating 'Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.', and another for the master username stating 'Type a login ID for the master user of your DB instance.' A note at the bottom left says 'If you manage the master user credentials in Secrets Manager, some RDS features aren't supported.' with a 'Learn more' link.

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

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

Don't connect to an EC2 compute resource
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

Connect to an EC2 compute resource
Set up a connection to an EC2 compute resource for this database.

Virtual private cloud (VPC) [Info](#)
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

Default VPC (vpc-0c050c2288d0d14c6)
6 Subnets, 6 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

DB subnet group [Info](#)
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

default-vpc-0c050c2288d0d14c6
6 Subnets, 6 Availability Zones

Public access [Info](#)

Yes
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

No
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

VPC security group (firewall) [Info](#)

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

Additional configuration
Database options, encryption turned on, backup turned on, backtrack turned off, maintenance, CloudWatch Logs, delete protection turned on.

Note: You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Create database

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

Databases (3)	Group resources	C	Actions	Restore from S3	Create database
<input type="text"/> Filter by database...					
<input checked="" type="checkbox"/> beedesign-database-2	Available	Instance	MySQL Community	us-east-1a	db.t3.micro
					2 Actions
					2.61% 0 Connections

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

This screenshot shows the 'Connectivity & security' tab of an AWS RDS instance configuration. The endpoint is listed as 'beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com'. The port is set to 3306. The networking section shows the availability zone as 'us-east-1a' and the VPC as 'vpc-0c050c2288d0d14c6'. The security section indicates that the default VPC security group is active and the instance is publicly accessible.

Endpoint & port	Networking	Security
Endpoint beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com	Availability Zone us-east-1a VPC vpc-0c050c2288d0d14c6	VPC security groups default (sg-043d70babda98751d) Active Publicly accessible Yes
Port 3306		

Added security group rule for MySQL.

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

This screenshot shows the 'Connectivity & security' tab of an AWS RDS instance configuration. The endpoint is listed as 'beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com'. The port is set to 3306. The networking section shows the availability zone as 'us-east-1a' and the VPC as 'vpc-0c050c2288d0d14c6'. The security section indicates that the default VPC security group is active and the instance is publicly accessible. Below this, a detailed view of the security group settings is shown, including the VPC security group and its status.

Endpoint & port	Networking	Security
Endpoint beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com	Availability Zone us-east-1a VPC vpc-0c050c2288d0d14c6 Subnet group default-vpc-0c050c2288d0d14c6	VPC security groups default (sg-043d70babda98751d) Active Publicly accessible Yes Certificate authority Info
Port 3306		

Security

VPC security groups
default (sg-043d70babda98751d)
Active

Publicly accessible
Yes

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'

The screenshot shows the AWS Security Groups Inbound Rules page for a security group named 'sg-043d70babda98751d - default'. The 'Inbound rules' tab is selected. There are two rules listed:

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0f5fa8096e9e6f63d	IPv4	MySQL/Aurora	TCP	3306	0.0.0.0/0	-
-	sgr-0b8f2ff226668a391	-	All traffic	All	All	sg-043d70babda98751d - default	-

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.

The screenshot shows the 'Edit inbound rule' dialog box. The rule configuration is as follows:

- Type:** MySQL/Aurora
- Protocol:** TCP
- Port range:** 3306
- Source:** Anywhere-IPv4 (Custom) 0.0.0.0/0
- Description (optional):** (empty)

At the bottom, there are 'Cancel', 'Preview changes', and 'Save rules' buttons. The 'Save rules' button is highlighted in orange.

In MySQL configuration

Step 1: click + to create new connection

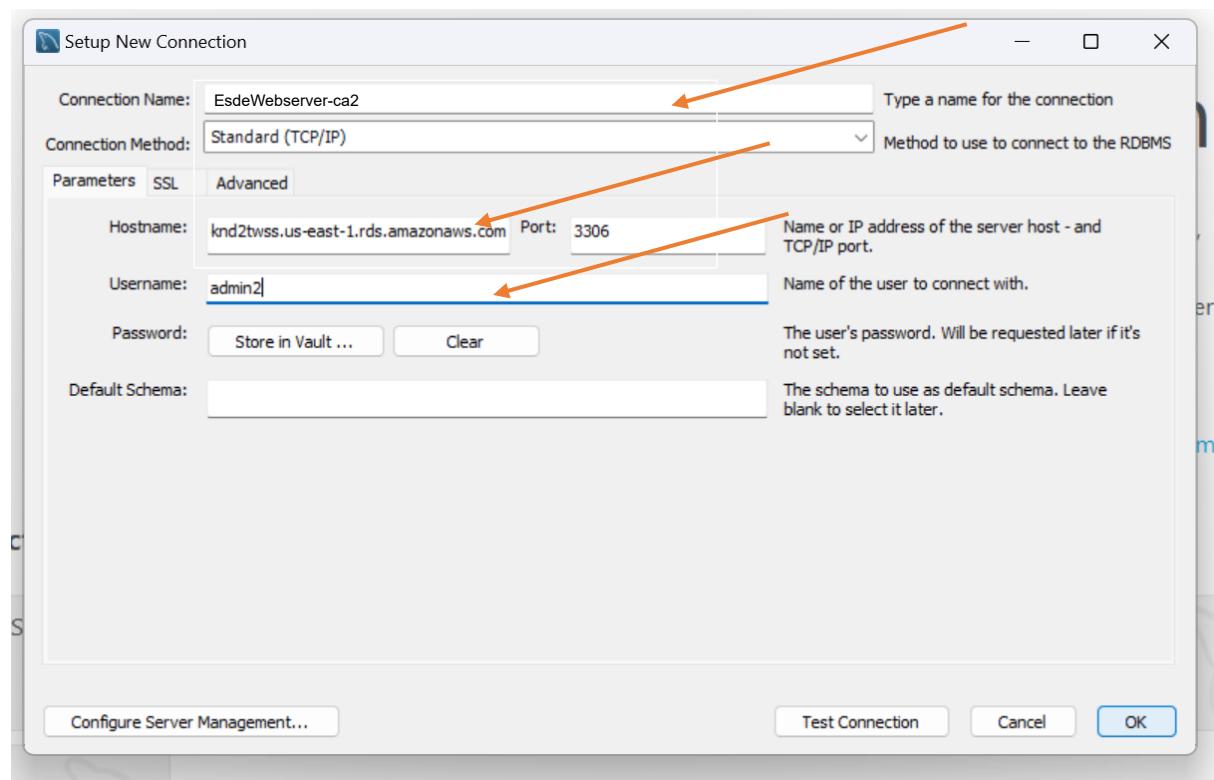


Configure Connection in MySQL Workbench:

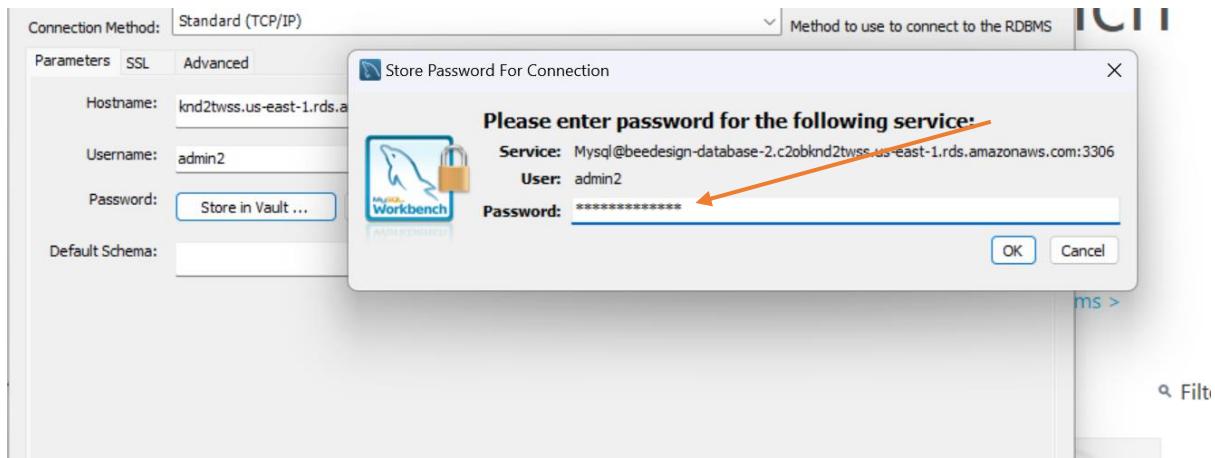
Step 1: Choose a connection name, for instance, 'EsdeWebserver-ca2'.

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

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

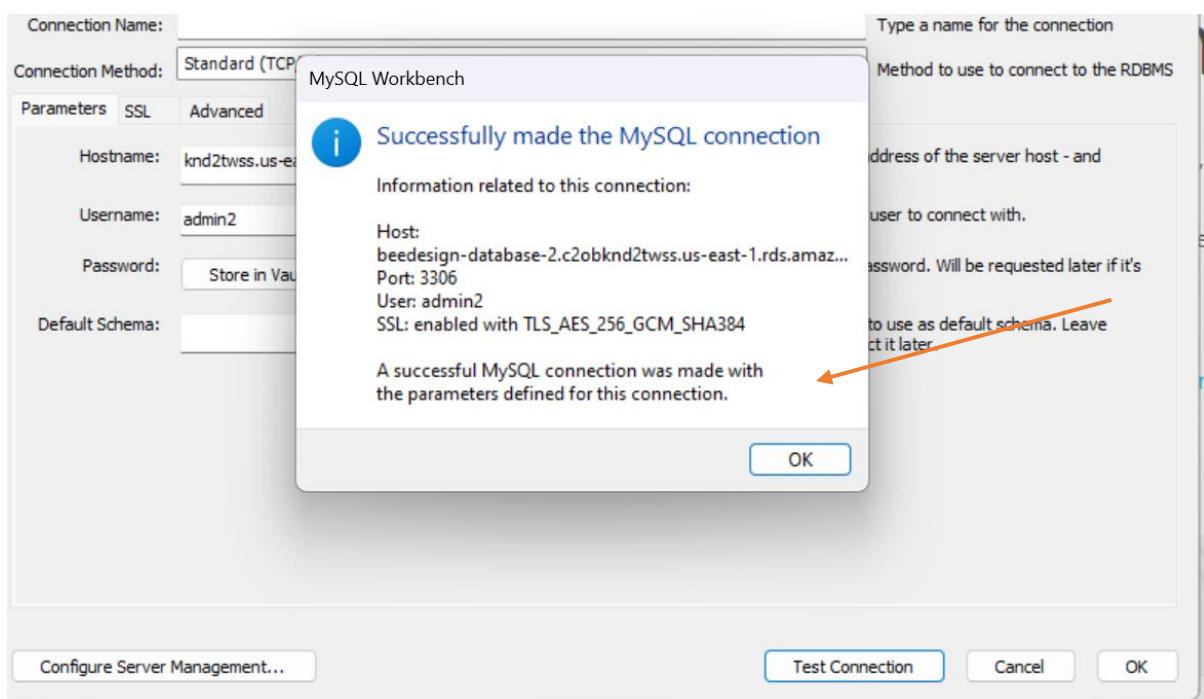


Step 4: Enter the password you selected while setting up the RDS instance.



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

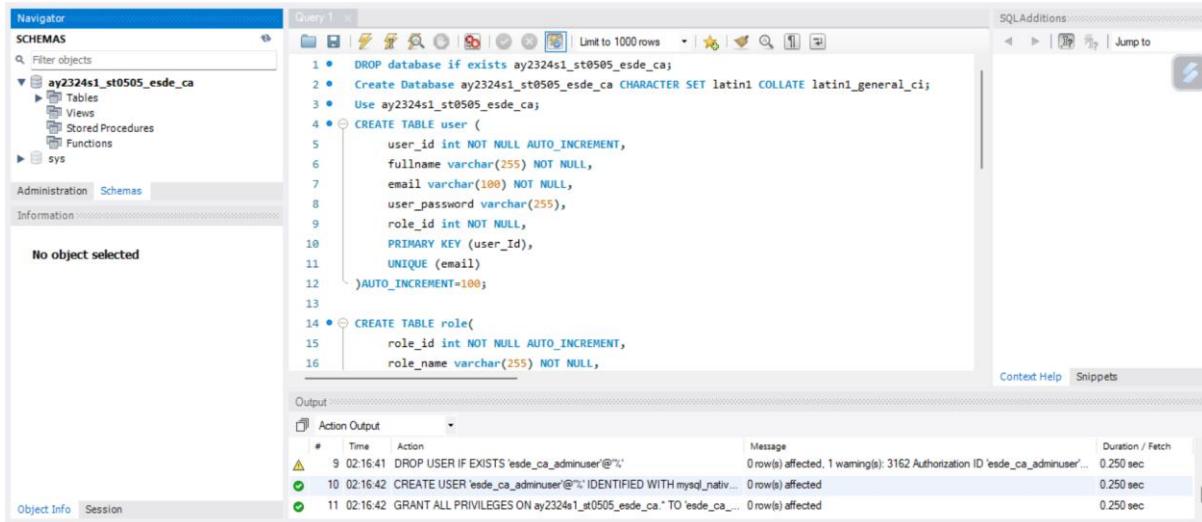
Step 1: Double click your connection in the Workbench to launch it

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**.



```
Query 1
1 • DROP database if exists ay2324s1_st0505_esde_ca;
2 • Create Database ay2324s1_st0505_esde_ca CHARACTER SET latin1 COLLATE latin1_general_ci;
3 • Use ay2324s1_st0505_esde_ca;
4 • CREATE TABLE user (
5     user_id int NOT NULL AUTO_INCREMENT,
6     fullname varchar(255) NOT NULL,
7     email varchar(100) NOT NULL,
8     user_password varchar(255),
9     role_id int NOT NULL,
10    PRIMARY KEY (user_Id),
11    UNIQUE (email)
12 )AUTO_INCREMENT=100;
13
14 • CREATE TABLE role(
15     role_id int NOT NULL AUTO_INCREMENT,
16     role_name varchar(255) NOT NULL,
```

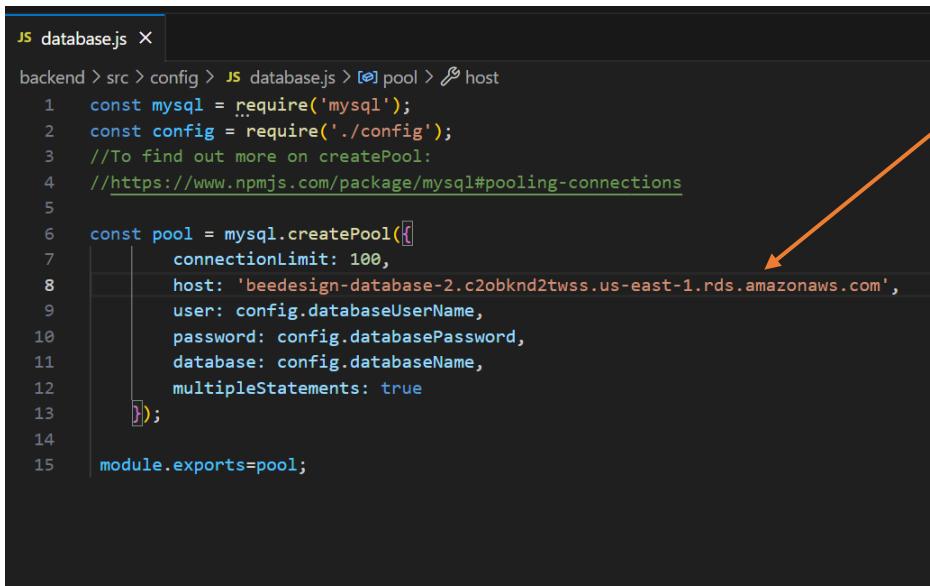
Action	Time	Message	Duration / Fetch
DROP USER IF EXISTS 'esde_ca_adminuser'@'%'	9 02:16:41	0 row(s) affected, 1 warning(s): 3162 Authorization ID 'esde_ca_adminuser'...	0.250 sec
CREATE USER 'esde_ca_adminuser'@'%' IDENTIFIED WITH mysql_native...	10 02:16:42	0 row(s) affected	0.250 sec
GRANT ALL PRIVILEGES ON ay2324s1_st0505_esde_ca.* TO 'esde_ca_...	11 02:16:42	0 row(s) affected	0.250 sec

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.

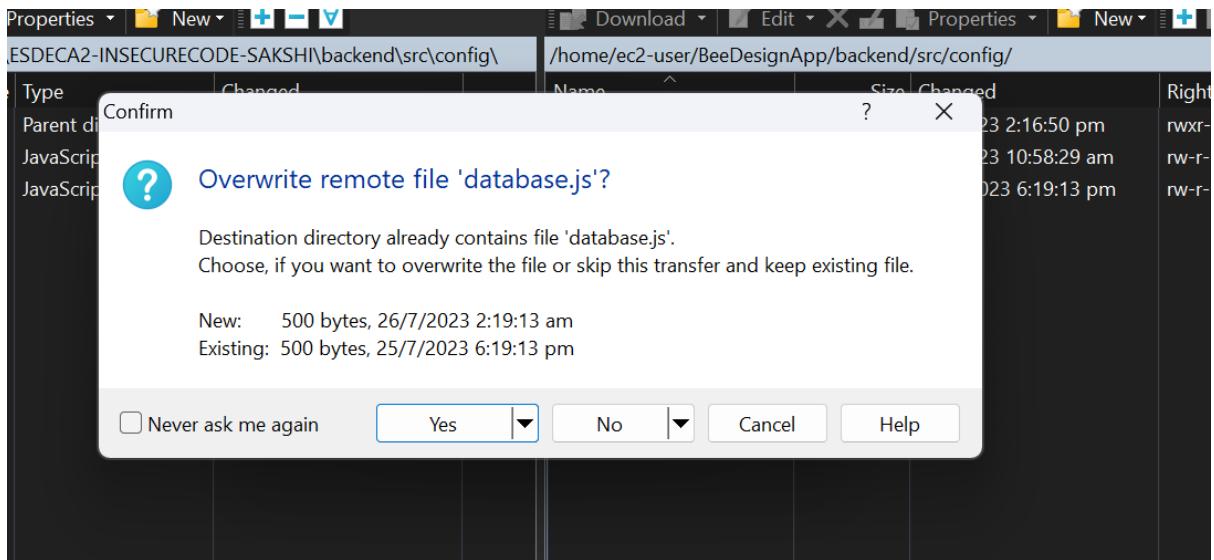
Step 3: Save your changes.



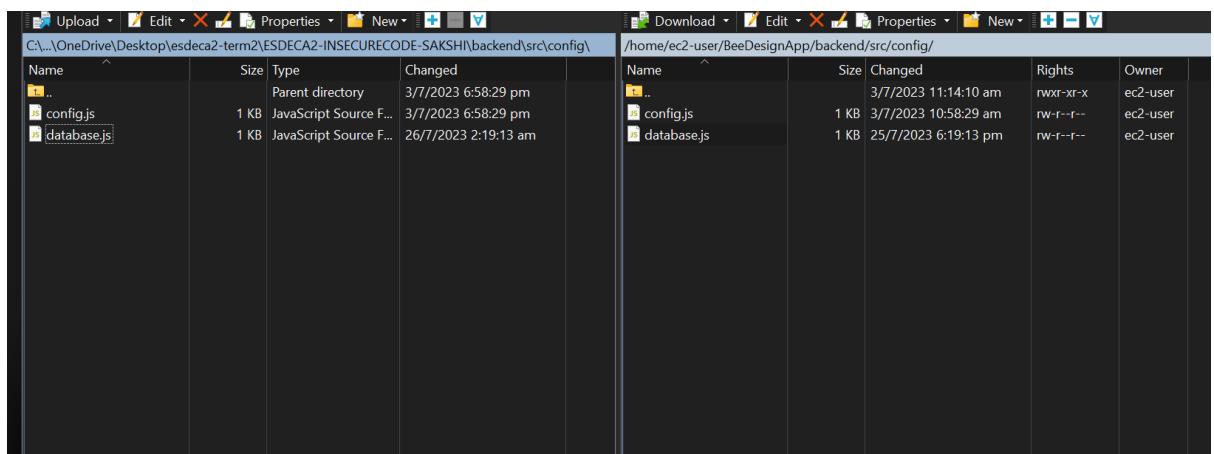
```
js database.js X
backend > src > config > js database.js > pool > host
1  const mysql = require('mysql');
2  const config = require('./config');
3  //To find out more on createPool:
4  //https://www.npmjs.com/package/mysql#pooling-connections
5
6  const pool = mysql.createPool([
7      connectionLimit: 100,
8      host: 'beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com',
9      user: config.databaseUserName,
10     password: config.databasePassword,
11     database: config.databaseName,
12     multipleStatements: true
13   ]);
14
15 module.exports=pool;
```

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

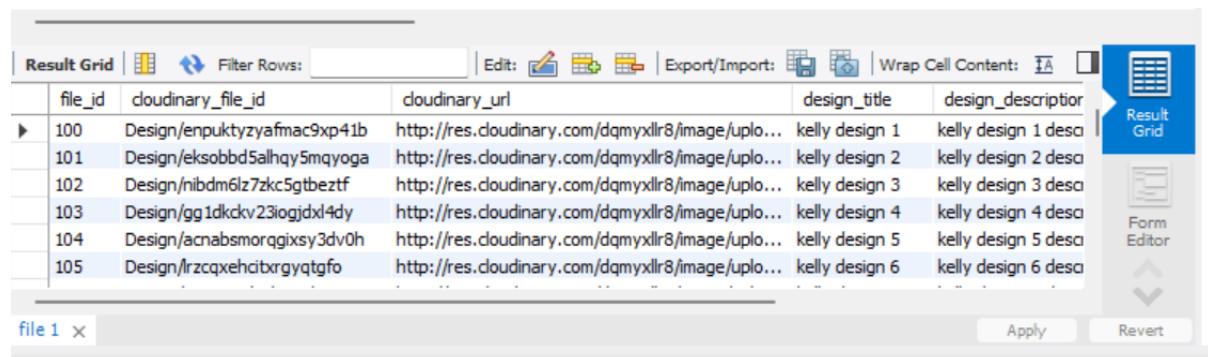
Step 1: Open a new terminal window in WinSCP.

Step 2 Navigate to backend directory.

Step 3: Run seederdata.js node seederdata.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 seederdata.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',
  publicId: 'Design/nibdm6lz7zkc5gtbeztf',
  status: 'success'
}
```

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



The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. A table named 'files' is displayed with the following data:

file_id	cloudinary_file_id	cloudinary_url	design_title	design_description
100	Design/enpuktyzyafmac9xp41b	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design/enpuktyzyafmac9xp41b.png	kelly design 1	kelly design 1 desc
101	Design/eksobbd5alhqy5mqyoga	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design/eksobbd5alhqy5mqyoga.png	kelly design 2	kelly design 2 desc
102	Design/nibdm6lz7zkc5gtbeztf	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/nibdm6lz7zkc5gtbeztf.png	kelly design 3	kelly design 3 desc
103	Design/gg1dkckv23logjdxl4dy	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/gg1dkckv23logjdxl4dy.png	kelly design 4	kelly design 4 desc
104	Design/acnabsmorqixsy3dv0h	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/acnabsmorqixsy3dv0h.png	kelly design 5	kelly design 5 desc
105	Design/lrcqxehtxrgyqtgfo	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/lrcqxehtxrgyqtgfo.png	kelly design 6	kelly design 6 desc

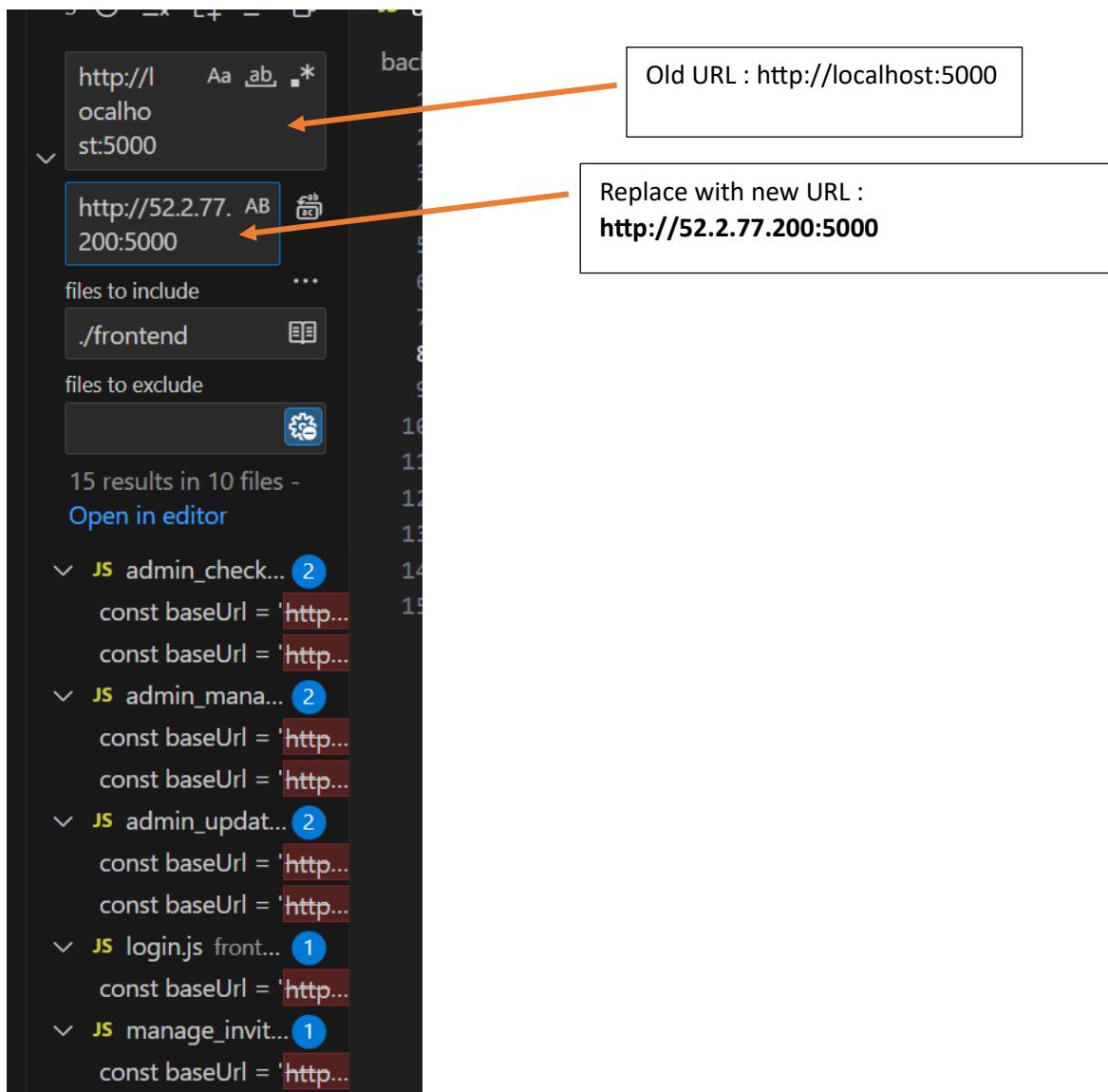
Change URL

Step 1: Search and Replace URLs 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:

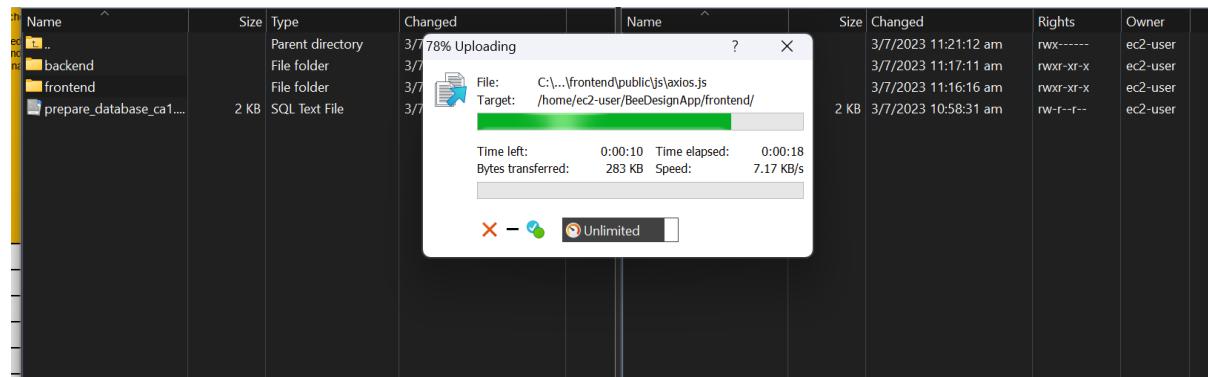
- Search for 'http://localhost:5000' across files.
- Replace 'localhost' with your External IP (EIP), 52.2.77.200
- Click 'Replace all' to update all occurrences.
- Save the changes.



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 = $('#searchInput').val();
    let userId = localStorage.getItem('user_id');
    axios({
        headers: {
            'Content-Type': 'application/json'
        },
        method: 'post',
        url: `${baseUrl}/api/search`,
        data: {
            searchInput: searchInput,
            userId: userId
        }
    }).then(response => {
        console.log(response);
    }).catch(error => {
        console.error(error);
    });
});
```

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.

The image shows two terminal windows. The left window, titled 'Frontend', shows the command 'npm start' being run and the output of the 'nodemon' process. The right window, titled 'Backend', shows the command 'npm start' being run and the output of the 'nodemon' process, which includes a database query log. Two orange arrows point from the text 'Inspect user id which is planted inside the request header : 100' in the Backend window to the 'user_id' field in the JSON object shown in the Backend window.

```
ec2-user@ip-172-31-90-105:~/BeeDesignApp/frontend
will not be shown, you would have to be root to see it all.)
:cp6      0      0 ::::3001          ::::*
:7877/node
[ec2-user@ip-172-31-90-105 frontend]$ kill -9 5787
[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
'C
[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): *.*
```

```
ec2-user@ip-172-31-90-105:~/BeeDesignApp/backend
[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
[
  RowDataPacket {
    user_id: 100,
    fullname: 'kelly',
    email: 'kelly@designer.com',
    user_password: '$2b$10$K.0HwpsPDGaB/atFBmmXOGTw4ceeg33.WrxJx/FeC9.qCyYvIbs'
  ,
    role_name: 'user',
    role_id: 2
  }
]
http header - user 100
Inspect user id which is planted inside the request header : 100
getFileData method is called.
Prepare query without search text
Executing query to obtain 1 page of 3 data
Accessing total number of rows : 9
Inspect result variable inside processGetSubmissionData code
```

Frontend

Backend

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 your website is connected to the MySQL RDS database.

The screenshot shows the 'Manage My Designs' section of the BrandCrowd platform. At the top, there are navigation links for 'Submissions' and 'Invite a friend'. On the right, there is a user profile icon. The main title 'Manage My Designs' is centered above a search bar labeled 'Search By Title'. Below the search bar is a blue 'SEARCH' button with an orange arrow pointing to it. Underneath the search area, there are three small thumbnail images of designs, each with the 'BrandCrowd' logo. To the left of these thumbnails is a page navigation bar with buttons for '1', '2', and '3'.

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:

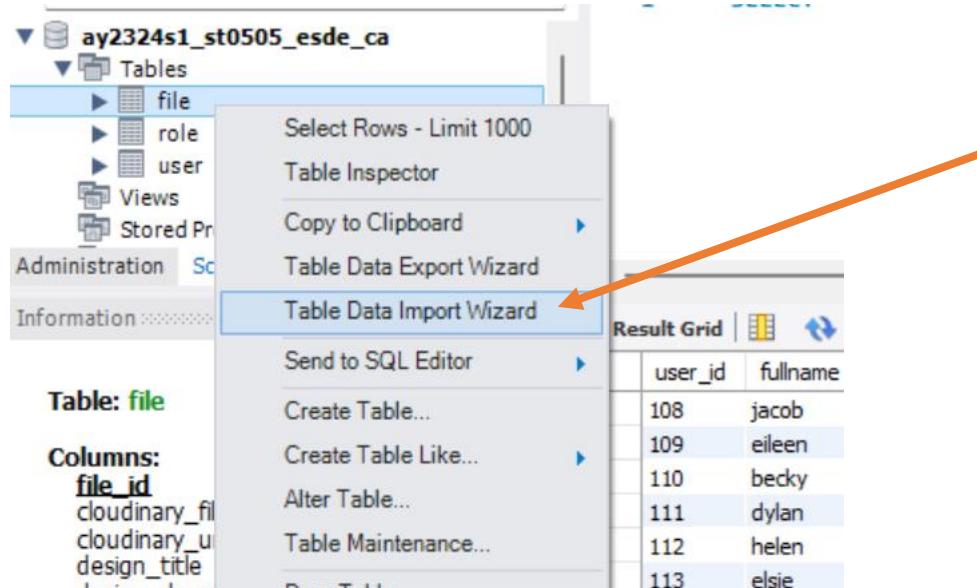
- **Scalability:** DynamoDB scales seamlessly to accommodate evolving workloads.
- **Low Latency:** It ensures rapid data access, making it suitable for real-time applications.
- **Flexibility:** DynamoDB's schema-less design adapts to changing data needs.
- **Managed Service:** Developers are freed 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.

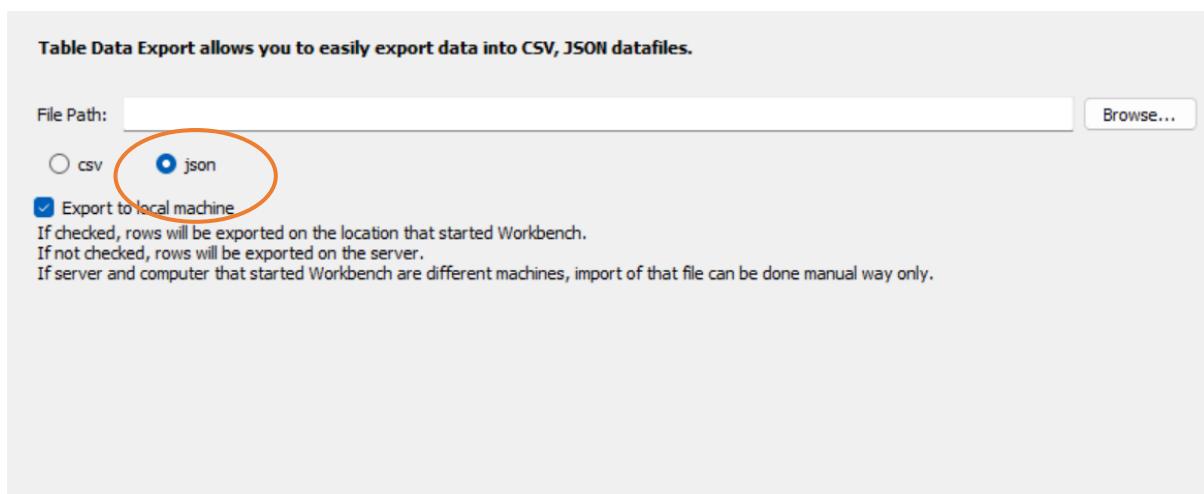
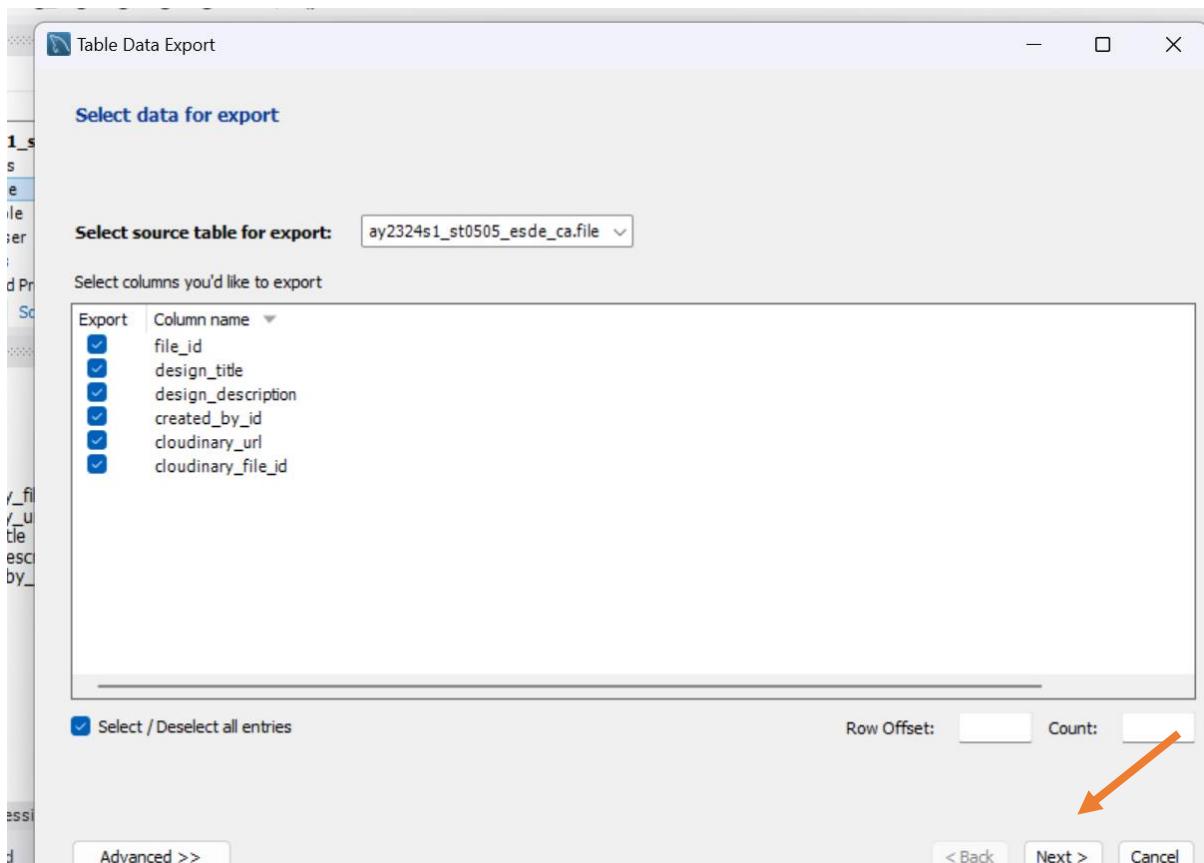
Export records from RDS MySQL to a Json file

Step 1: Start your RDS MySQL Workbench

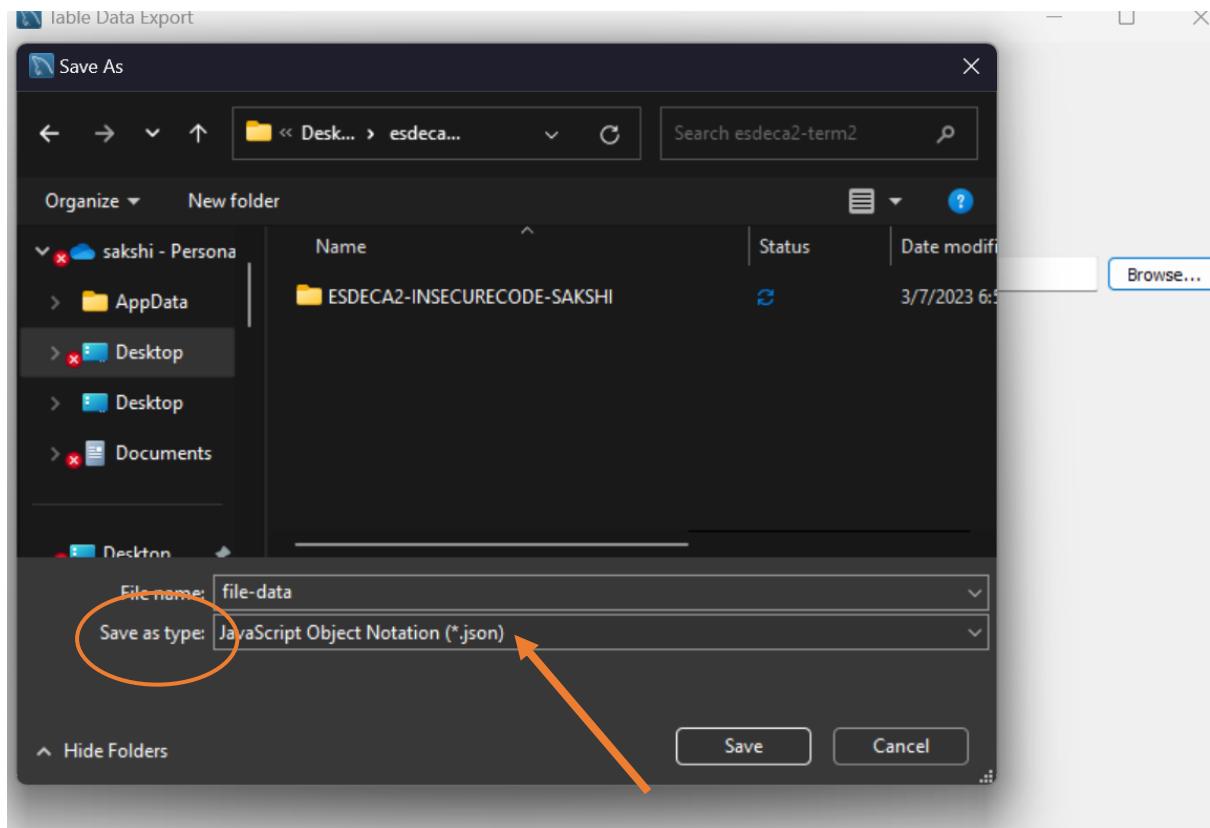
Step 2: Perform a right-click on the "Files" table and select the "Table Export Wizard" option.



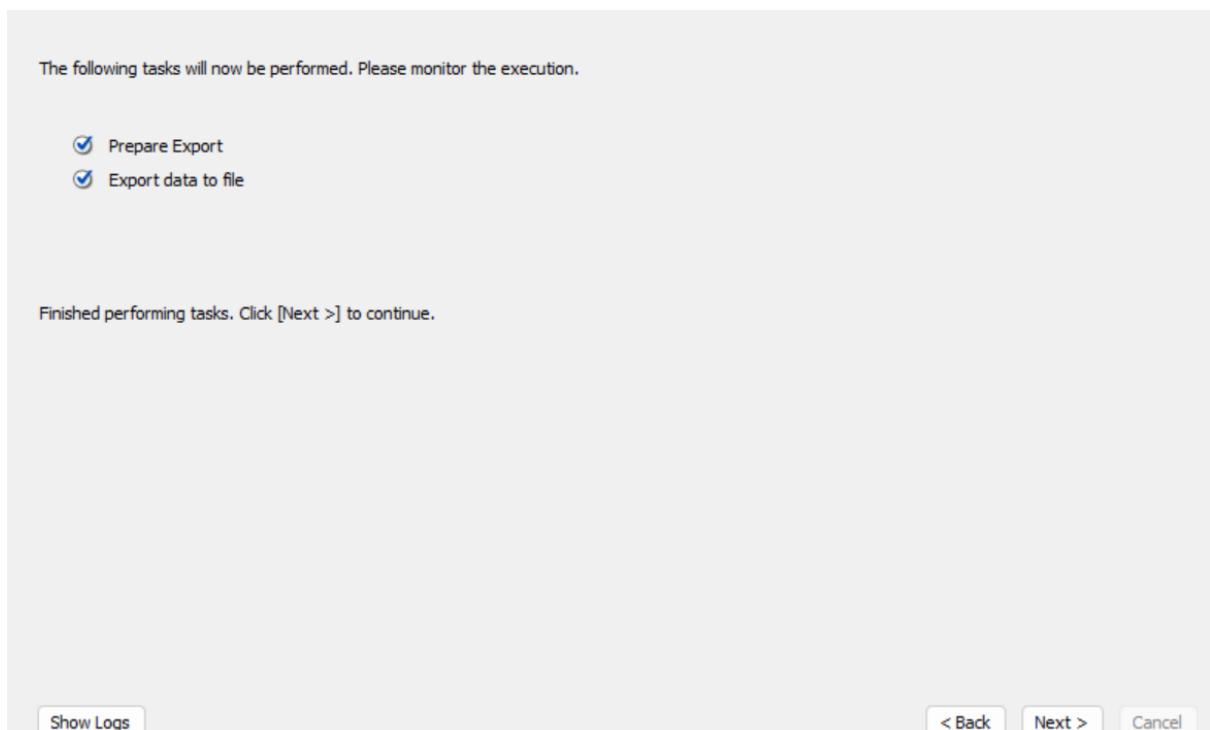
Step 3: Click on the "Next" button, and then select the export format as JSON.



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.

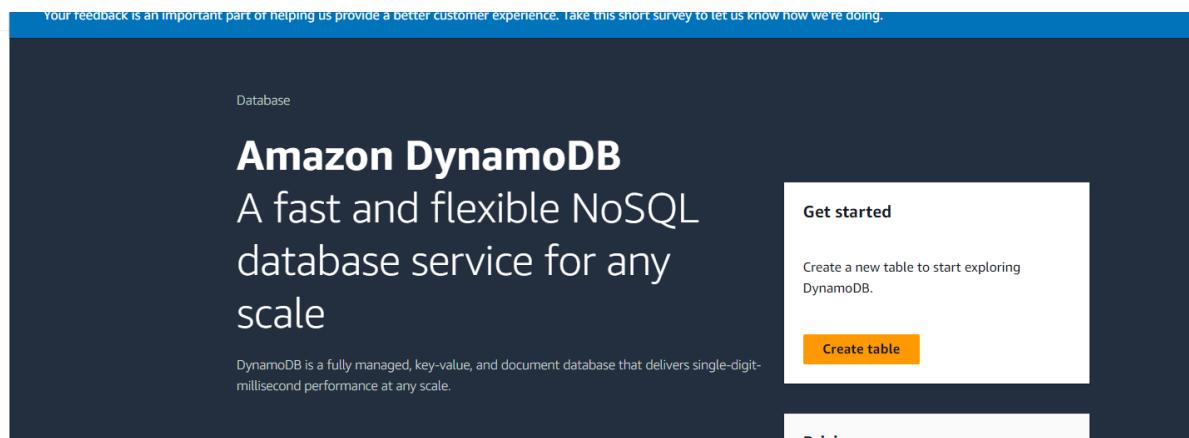
The screenshot shows a Notepad window with the title bar "beedesign-database-2.c2obknd2tw5" and the file name "file-data.json". The content of the file is a JSON array of 36 objects, each representing a design with fields like file_id, design_title, design_description, and cloudinary_url. The JSON is as follows:

```
[{"file_id":100, "design_title":"kelly design 1", "design_description":"kelly design 1 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design/enpuktzyafmac9xp41b.png", "cloudinary_file_id":"Design/enpuktzyafmac9xp41b"}, {"file_id":101, "design_title":"kelly design 2", "design_description":"kelly design 2 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design/eksobbd5alhqy5mqyoga.png", "cloudinary_file_id":"Design/eksobbd5alhqy5mqyoga"}, {"file_id":102, "design_title":"kelly design 3", "design_description":"kelly design 3 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/nibdm6lz7zkc5gtbeztf.png", "cloudinary_file_id":"Design/nibdm6lz7zkc5gtbeztf"}, {"file_id":103, "design_title":"kelly design 4", "design_description":"kelly design 4 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310025/Design/gg1dkckv23iogjdx14dy.png", "cloudinary_file_id":"Design/gg1dkckv23iogjdx14dy"}, {"file_id":104, "design_title":"kelly design 5", "design_description":"kelly design 5 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310026/Design/acnabsmorqgixsy3dv0h.png", "cloudinary_file_id":"Design/acnabsmorqgixsy3dv0h"}, {"file_id":105, "design_title":"kelly design 6", "design_description":"kelly design 6 description text 1 text 2 text 3 text 4 ....", "created_by_id":100, "cloudinary_url":"http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310026/Design/acnabsmorqgixsy3dv0h"}]
```

At the bottom of the Notepad window, it says "Ln 1, Col 1" and "100% Unix (LF) UTF-8".

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**

The screenshot shows the "Create table" settings page. It starts with a "Table details" section where the "Name" field is set to "files". Below this, the "Partition key" section is shown with "file_id" as the key name and "Number" as the data type. There's also a "Sort key - optional" section with an empty input field and a dropdown menu set to "String". Orange arrows point from the text labels "Name", "Partition key", and "Data Type" to their respective input fields and dropdown menu.

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Name

Table name
This will be used to identify your table.

files

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key

Data Type

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

file_id

Number

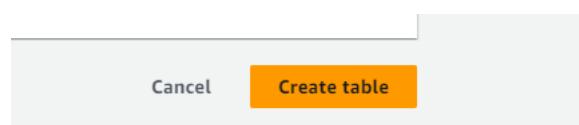
Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name

String

1 to 255 characters and case sensitive.

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



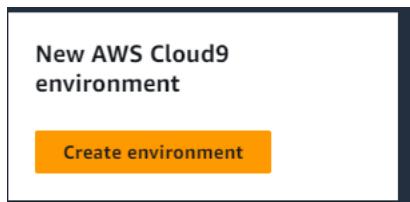
Step 3: Check that your new table “files” is successfully created.

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
files	Active	file_id (N)	-	0	Off	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	10.7 kilobytes

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

Details

Name
esde-ca2

Description - optional

Environment type Info
Determines what the Cloud9 IDE will run on.

New EC2 instance
Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute
You have an existing instance or server that you'd like to use.

Network settings [Info](#)

Connection
How your environment is accessed.

AWS Systems Manager (SSM)
Accesses environment via SSM without opening inbound ports (no ingress).

Secure Shell (SSH)
Accesses environment directly via SSH, opens inbound ports.



▶ VPC settings [Info](#)

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

Environments (2)						
						Delete
						View details
My environments						
Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN	
CA2-ESDE-DEMO	Open	EC2 instance	Secure Shell (SSH)	Owner	arn:aws:sts::637825447844:assumed-role/voclabs/user2061033-sakshi.22@ichat.sp.edu.sg	
esde-ca2	Open	EC2 instance	Secure Shell (SSH)	Owner	arn:aws:sts::637825447844:assumed-role/voclabs/user2061033-sakshi.22@ichat.sp.edu.sg	

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

The screenshot shows a code editor with two tabs: 'file-data.json' and 'seed_files_table.js'. The 'seed_files_table.js' tab contains the following code:

```
1  var AWS = require("aws-sdk");
2  DB = new AWS.DynamoDB({
3      apiVersion: "2012-08-10",
4      region: "us-east-1"
5  });
6  file_DATA_ARR = require("./file-data.json");
7
8  function addNewItemFromJSON(){
9      console.log("All items now removed, re-seeding now");
10     var
11         file = [],
12         num_items_left = file_DATA_ARR.length,
13         offset_index = 0;
14     console.log('num_items_left initial', num_items_left)
15
16     while(num_items_left > 0){
17         var
18             file_formatted_arr = [],
19             params = {},
20             item_added_count = 0;
21         for(var i_int = offset_index; i_int < offset_index+Math.min(25, num_items_left); i_int += 1){
22             file
23                 .PutRequest(
24                     Item: {
25                         file_id: {
26                             "N": file_DATA_ARR[i_int].file_id.toString()
27                         },
28                         cloudinary_file_id: {
29                             "S": file_DATA_ARR[i_int].cloudinary_file_id
30                         },
31                         cloudinary_url: {
32                             "S": file_DATA_ARR[i_int].cloudinary_url
33                         },
34                         design_title: {
35                             "S": file_DATA_ARR[i_int].design_title
36                         },
37                         design_description: {
38                             "S": file_DATA_ARR[i_int].design_description
39                         },
40                         created_by_id: {
41                             "N": file_DATA_ARR[i_int].created_by_id.toString()
42                         }
43                     }
44                 )
45         }
46     }
47 }
```

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

```
vocabs:~/environment $ npm install aws-sdk
npm WARN deprecate querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API instead.

added 31 packages, and audited 35 packages in 4s

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

found 0 vulnerabilities
vocabs:~/environment $
```

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

```
File Edit Find View Go Run Tools Window Support Preview Run

Go to Anything (Ctrl-P)
esde-ca2 - /home
  () file-data.json
  () README.md
JS seed_files_table.js

1  | var
2  | AWS = require("aws-sdk");
3  | DDB = new AWS.DynamoDB({
4  |   apiVersion: "2012-08-10",
5  |   region: "us-east-1"
6  | });
7  | file_DATA_ARR = require("./file-data.json");
8
9
10 function addNewItemFromJSON(){
11   console.log("All items now removed, re-seeding now");
12   var
13     file = {},
```

```
bash "ip-172-31-78-213.x" Immediate seed_files_table.js - Stop +
```

```
Run Command: seed_files_table.js

Debugger listening on ws://127.0.0.1:15454/1c914e35-1ccc-479b-8db0-6fc3d7438437
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
All items now removed, re-seeding now
num_items_left initial 36
num_items_left 11
num_items_left 0
(node:9224) NOTE: We are formalizing our plans to enter AWS SDK for JavaScript (v2) into maintenance mode in 2023.

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

Process exited with code: 0
```

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**.

The screenshot shows the AWS DynamoDB 'files' table details page and the 'Items summary' table.

DynamoDB > Tables > files

Tables (1)

General information

- Partition key: file_id (Number)
- Sort key: -
- Capacity mode: Provisioned
- Table status: Active

Items summary

Items returned (36)

file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
115	Design/ungbfwt5r8m...	http://res.cloudin...	101	bezос design 7 descri...	bezос design 7
117	Design/oe0veqe4kgo2...	http://res.cloudin...	101	bezос design 9 descri...	bezос design 9
122	Design/qvv1q2hqiy7of...	http://res.cloudin...	102	elon design 5 descripti...	elon design 5
130	Design/e31gn63logrvf...	http://res.cloudin...	103	nadella design 4 descri...	nadella design 4
110	Design/c7nxlcqswax...	http://res.cloudin...	101	bezос design 2 descrip...	bezос design 2
131	Design/ql5y4oxuwi8kz...	http://res.cloudin...	103	nadella design 5 descri...	nadella design 5
113	Design/g2ccc8ysnwnd...	http://res.cloudin...	101	bezос design 5 descrip...	bezос design 5
118	Design/g5whwoqiwmc...	http://res.cloudin...	102	elon design 1 descripti...	elon design 1

Part 5- Setup Lambda

Lambda in AWS

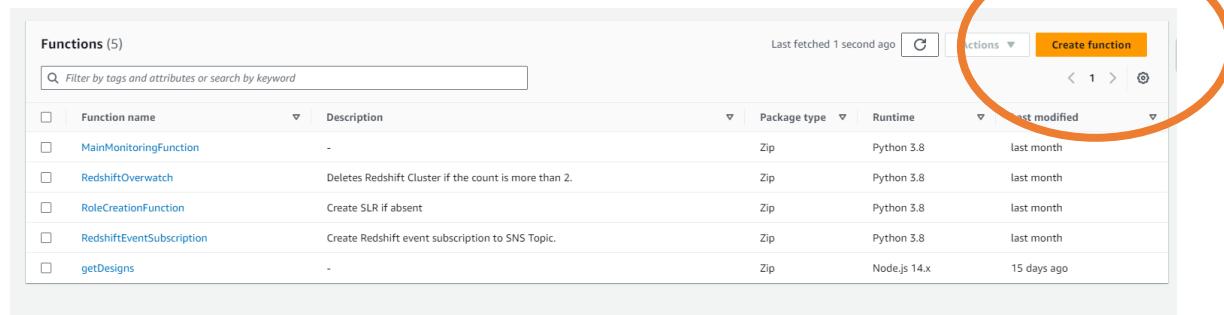
AWS Lambda is a **serverless compute service** that allows you to run code in response to events without managing servers. Its

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.

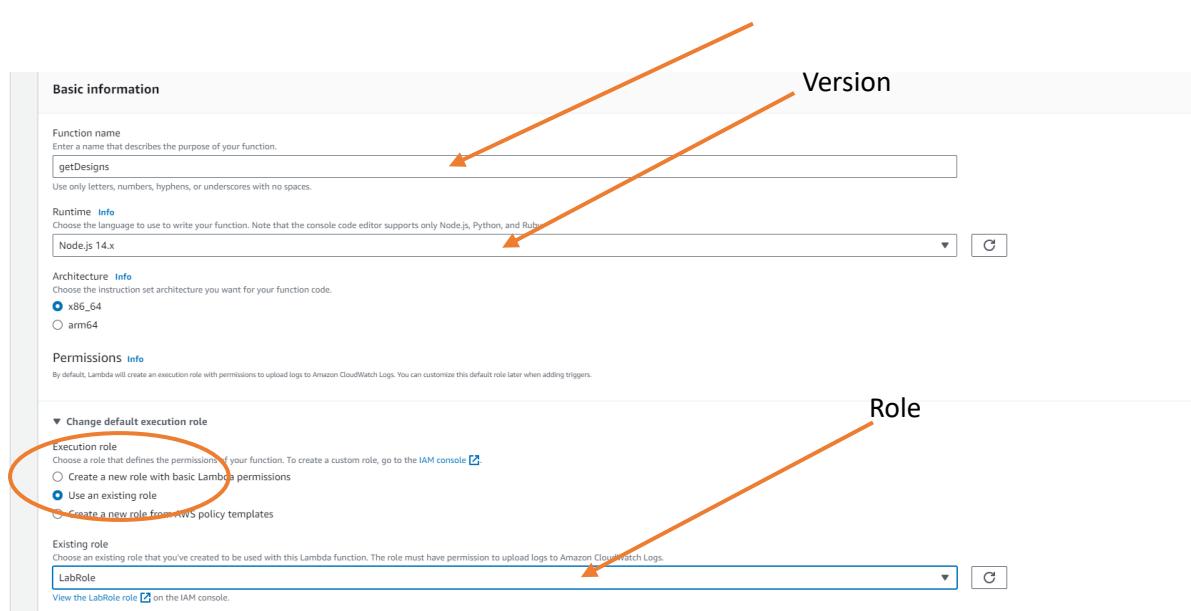


The screenshot shows the AWS Lambda Functions dashboard. At the top, there is a search bar labeled 'Filter by tags and attributes or search by keyword'. Below it is a table with columns: 'Function name', 'Description', 'Package type', 'Runtime', and 'Last modified'. The table lists five functions: 'MainMonitoringFunction', 'RedshiftOverwatch', 'RoleCreationFunction', 'RedshiftEventSubscription', and 'getDesigns'. In the top right corner of the dashboard, there is a prominent orange 'Create function' button. This button is circled in orange in the screenshot.

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

Name



The screenshot shows the 'Create New Lambda Function' wizard. It has several sections:

- Basic information:** Contains fields for 'Function name' (set to 'getDesigns') and 'Runtime' (set to 'Node.js 14.x').
- Version:** A section where the 'Name' field is highlighted.
- Permissions:** A section titled 'Change default execution role' where the 'Execution role' dropdown is highlighted. It contains three options: 'Create a new role with basic Lambda permissions' (radio button), 'Use an existing role' (radio button, selected), and 'Create a new role from AWS policy templates'. The 'Use an existing role' option is circled in orange. Below this, a dropdown menu for 'Existing role' is shown, with 'LabRole' selected and highlighted with a blue border.
- Role:** A section where the 'LabRole' option is highlighted with a blue border.

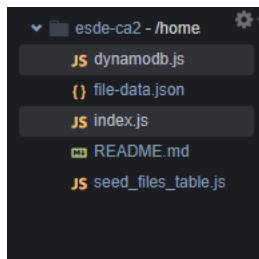
The 'Create Function' button is located at the bottom right of the form.

Check that your function has been created

Functions (5)						Last fetched 1 second ago	Actions	Create function
Function name	Description	Package type	Runtime	Last modified				
MainMonitoringFunction	-	Zip	Python 3.8	last month				
RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	last month				
RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	last month				
RedshiftEventSubscription	Create Redshift event subscription to SNS Topic.	Zip	Python 3.8	last month				
getDesigns	-	Zip	Node.js 14.x	15 days ago				

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

```
voclabs:~/environment $ zip function.zip index.js dynamodb.js
  adding: index.js (deflated 68%)
  adding: dynamodb.js (deflated 56%)
voclabs:~/environment $
```

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 Lambda function.

```
voclabs:~/environment $ aws lambda update-function-code --function-name getDesigns --zip-file fileb://function.zip
{
    "LastUpdateStatus": "InProgress",
    "FunctionName": "getDesigns",
    "LastModified": "2023-07-25T20:30:15.000+0000",
    "RevisionId": "73e18f09-81b1-4f92-b4ae-fa36d925617c",
    "LastUpdateStatusReason": "The function is being created.",
    "MemorySize": 128,
    "State": "Active",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::637825447844:role/LabRole",
    "Timeout": 3,
    "Handler": "index.handler",
    "Runtime": "nodejs14.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "uvZX8o+VAw8bN0oTEbT9s1NIaCIjQfq8NSkW6d0My8=",
    "Description": "",
    "LastUpdateStatusReasonCode": "Creating",
    "CodeSize": 1422,
    "FunctionArn": "arn:aws:lambda:us-east-1:637825447844:function:getDesigns",
    "PackageType": "Zip"
}
voclabs:~/environment $
```

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

The screenshot shows the AWS Lambda function editor interface. The top navigation bar includes 'Code source' and 'Info' buttons, followed by 'Test' and 'Deploy' buttons. Below the navigation is a search bar and a file tree on the left labeled 'Environment'. The main area displays the 'index.js' code for a Lambda function named 'getDesigns'. The code uses the AWS SDK's dynamodb module to query a 'files' table. It handles both string and file ID queries, constructs a projection expression, and handles the response with CORS headers. A red circle highlights the file tree on the left.

```
1  // Import the AWS SDK for Node.js
2  // and the code sample
3  var AWS = require('aws-sdk');
4
5  exports.handler = async function(event, context, callback){
6    if (event.queryStringParameters.fid || event.queryStringParameters.fileName) {
7      if (event.fid) {
8        var fileId = parseInt(event.fid);
9      } else {
10        var fileId = parseInt(event.queryStringParameters.fid);
11      }
12      var region = "us-east-1";
13      var tableName = "files";
14      var expr_attr_values = {"$fileId": fileId};
15      var proj_expr = "$file_id,cloudinary_url,design_title,design_description";
16      await dynamoDBQuery(region, tableName, expr_attr_values, key_cond_exp, proj_expr)
17      .then(data => {
18        console.log("Successfully got item from dynamodb.query")
19        var responseCode = 200;
20        var jsonResult = {'filedata': data.Items[0]};
21        let response = {
22          statusCode: responseCode,
23          body: JSON.stringify(jsonResult),
24          headers: {
25            'Access-Control-Allow-Headers': ["Content-Type,user"],
26            'Access-Control-Allow-Origin': "*",
27            'Access-Control-Allow-Methods': "OPTIONS,POST,GET"
28          }
29        }
30        console.log("response: " + JSON.stringify(response))
31        callback(null, response);
32      })
33      .catch(error => {
34        console.log(`There has been a problem with your fetch operation: ${error.message}`);
35        var responseCode = 500;
36      });
37    }
38  }
```

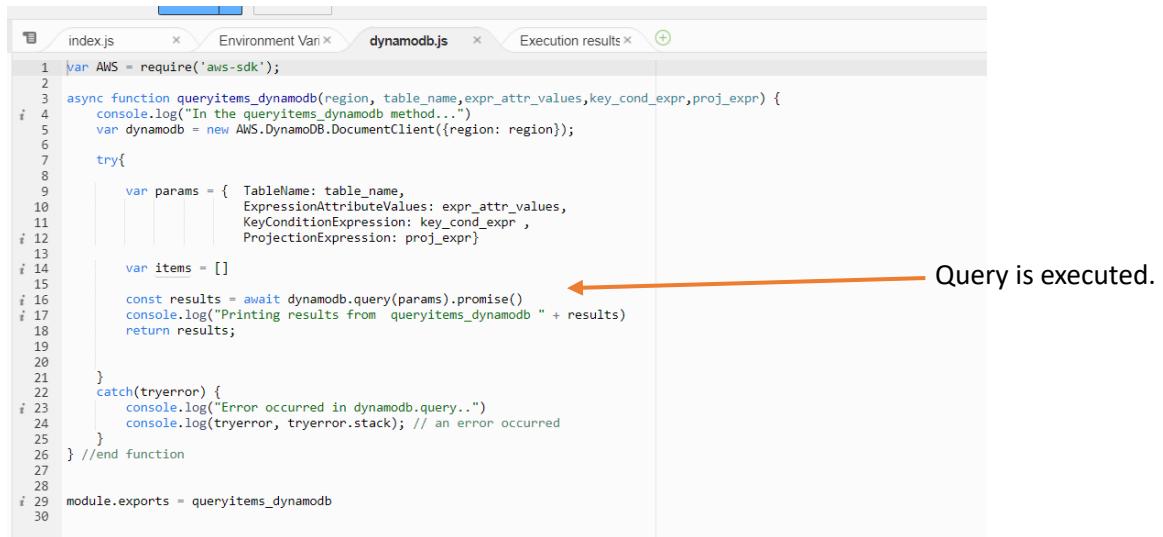
Index.js explanation

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.

```
1 var dynamodbQuery = require('dynamodb');
2
3 exports.handler = async function(event, context, callback){
4     if (event.fid || (event.queryStringParameters && event.queryStringParameters.fid)) {
5         if (event.fid)
6             var fileId = parseInt(event.fid);
7         else
8             var fileId= parseInt(event.queryStringParameters.fid);
9         var region = "us-east-1"
10        var table_name = "files"
11        var expr_attr_values = { ":fileId": fileId }
12        var key_cond_expr = "file_id=:fileId"
13        var proj_expr = "file_id,cloudinary_url,design_title,design_description"
14        await dynamodbQuery(region, table_name,expr_attr_values,key_cond_expr,proj_expr)
15        .then(data => {
16            console.log("Successfully got items from dynamodb.query")
17            var responseCode = 200;
18            var jsonResult = {'filedata': data.Items[0]}
19            let response = {
20                statusCode: responseCode,
21                body: JSON.stringify(jsonResult),
22                headers: {
23                    "Access-Control-Allow-Headers" : "Content-Type,user",
24                    "Access-Control-Allow-Origin": "*",
25                    "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
26                }
27            }
28            console.log("response: " + JSON.stringify(response))
29            callback(null, response);
30        })
31        .catch(error => {
32            console.log('There has been a problem with your fetch operation: ' + error.message);
33            var responseCode = 500;
34
35            let response = {
36                statusCode: responseCode,
37                body: JSON.stringify(error)
38            }
39
40            console.log("response: " + JSON.stringify(response))
41            callback(null, response);
42        });
43    }
44
45 } //end if
```

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 occur**. This function is exported for use in other parts of the code and index.js.



```
index.js Environment Variables dynamodb.js Execution results +  
1 var AWS = require('aws-sdk');  
2  
3 async function queryitems_dynamodb(region, table_name,expr_attr_values,key_cond_expr,proj_expr) {  
4     console.log("In the queryitems_dynamodb method...")  
5     var dynamodb = new AWS.DynamoDB.DocumentClient({region: region});  
6  
7     try{  
8         var params = { TableName: table_name,  
9                         ExpressionAttributeValues: expr_attr_values,  
10                        KeyConditionExpression: key_cond_expr ,  
11                           ProjectionExpression: proj_expr}  
12  
13         var items = []  
14         const results = await dynamodb.query(params).promise()  
15         console.log("Printing results from queryitems_dynamodb " + results)  
16         return results;  
17     }  
18     catch(tryerror) {  
19         console.log("Error occurred in dynamodb.query..")  
20         console.log(tryerror, tryerror.stack); // an error occurred  
21     }  
22 } //end function  
23  
24 module.exports = queryitems_dynamodb  
25  
26  
27  
28  
29  
30
```

Query is executed.

Create test event.

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

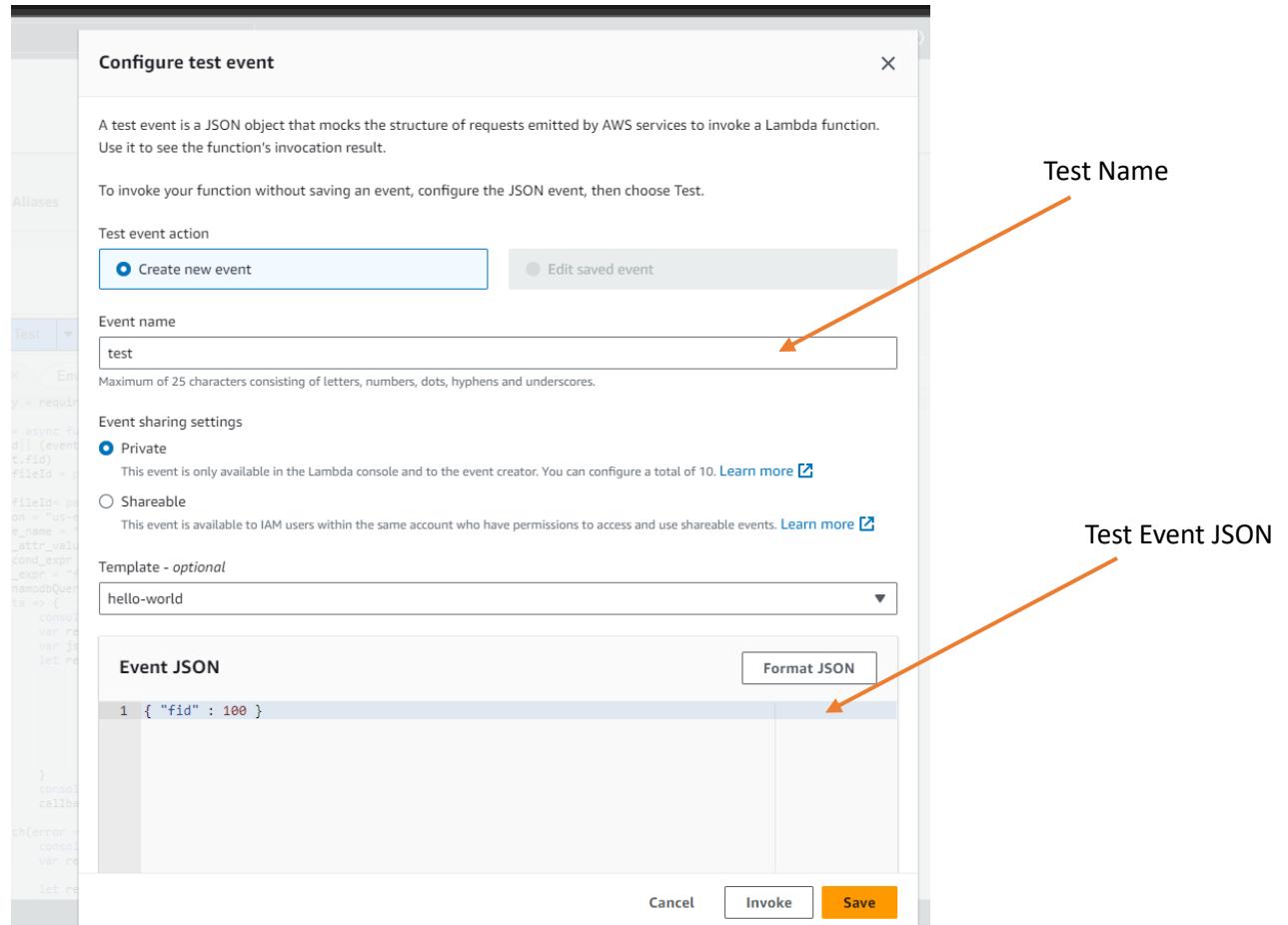
Step 1: Press `Ctrl + Shift + C` in the Lambda code console to open the "Configure test events" panel.

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 , clouddinary url, clouddinary id , created_by_id**

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

Code source [Info](#)

[File](#) [Edit](#) [Find](#) [View](#) [Go](#) [Tools](#) [Window](#) [Test](#) [Deploy](#)

[Upload from](#) [⚙️](#)

Go to Anything (Ctrl-P)

index.js Environment Var Execution result dynamodb.js

Status Succeeded Max memory used: 80 MB Time: 839.00 ms

Execution results

Test Event Name test

Response

```
{
  "statusCode": 200,
  "body": "{\"filedata\":{\"design_title\":\"Kelly design 1\",\"cloudinary_url\":\"http://res.cloudinary.com/dqmyx1lr8/image/upload/v1690310024/Design/enpuktzyafmac9xp4lb.png\",\"design_description\":\"Kelly design 1 description text 1 text 2 text 3 text 4 ....\"},\"file_id\":100}"
}
```

Function Logs

```
START RequestId: e8bcf420-42bf-4d90-ad0f-47eb2b118913 Version: $LATEST
2023-07-25T20:35:34.917Z      e8bcf420-42bf-4d90-ad0f-47eb2b118913  INFO  In the queryItems_dynamodb method...
2023-07-25T20:35:35.659Z      e8bcf420-42bf-4d90-ad0f-47eb2b118913  INFO  Printing results from queryItems_dynamodb [object Object]
2023-07-25T20:35:35.659Z      e8bcf420-42bf-4d90-ad0f-47eb2b118913  INFO  Successfully got items from dynamodb.query
2023-07-25T20:35:35.659Z      e8bcf420-42bf-4d90-ad0f-47eb2b118913  INFO  response: {"statusCode":200,"body":{"filedata":{"design_title":"Kelly design 1","cloudinary_url":"http://res.cloudinary.com/dqmyx1lr8/image/upload/v1690310024/Design/enpuktzyafmac9xp4lb.png"}, "file_id":100}}
END RequestId: e8bcf420-42bf-4d90-ad0f-47eb2b118913
REPORT RequestId: e8bcf420-42bf-4d90-ad0f-47eb2b118913 Duration: 839.00 ms Billed Duration: 840 ms Memory Size: 128 MB Max Memory Used: 80 MB Init Duration: 499.00 ms
```

Request ID e8bcf420-42bf-4d90-ad0f-47eb2b118913

Information retrieved correctly from DynamoDB

```
npuktzyafmac9xp4lb.png\", \"design_description\": \"Kelly design 1 description text 1 text 2 text 3 text 4 ....\", \"file_id\":100}}
```

file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
100	Design/enpuktzyafm...	http://res.cloudin...	100	kelly design 1 descri...	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”.

The screenshot shows the AWS API Gateway dashboard. At the top right, there is a yellow "Create API" button. This button is highlighted with a red rectangular box. Below the button, there is a search bar labeled "Find APIs". The main area displays a table titled "APIs (1)". The table has columns: Name, Description, ID, Protocol, Endpoint type, and Created. There is one entry: "beedesign_API" (Description: "uide6ygynd", Protocol: "REST", Endpoint type: "Regional", Created: "2023-07-26").

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

The screenshot shows the "REST API" creation page. At the bottom right, there are two buttons: "Import" and "Build". The "Build" button is highlighted with a red rectangular box. The page also contains descriptive text about REST APIs and their compatibility with Lambda, HTTP, and AWS Services.

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.

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Import from Swagger or Open API 3 Example API

API Name

Settings

Choose a friendly name and description for your API.

API name*

Description

Endpoint Type

* Required

Create API

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**'.

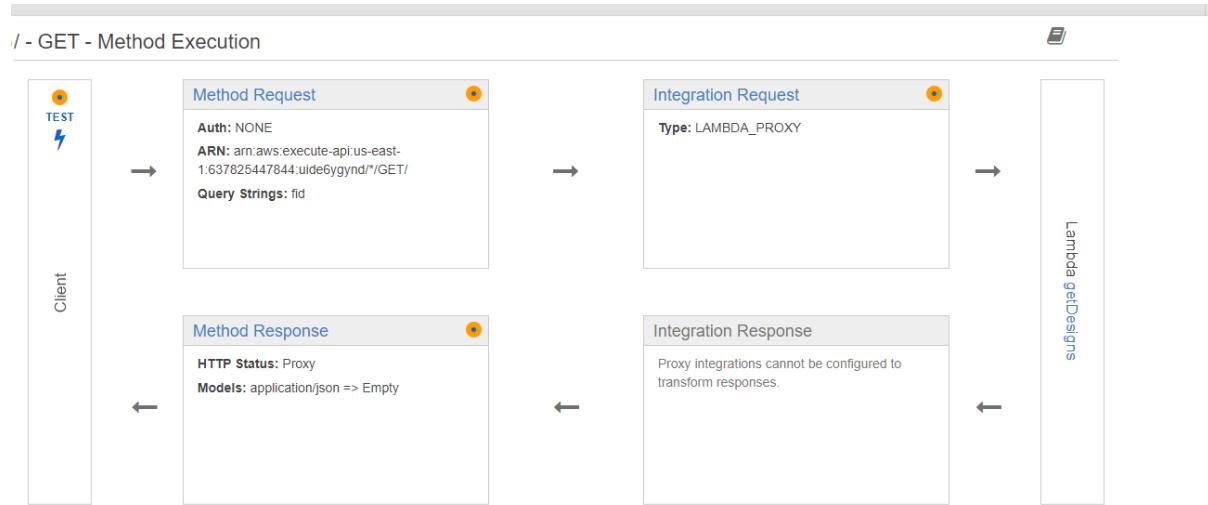
Step 4: Proceed by clicking the 'Save' button.

The screenshot shows the 'GET - Setup' configuration page for an API method. The 'Integration type' is set to 'Lambda Function'. The 'Use Lambda Proxy integration' checkbox is checked. The 'Lambda Region' is set to 'us-east-1'. The 'Lambda Function' input field contains 'getDesigns'. A large orange circle highlights the 'Save' button at the bottom right.

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.

Screenshot of the AWS Lambda Method Request configuration page for the '/ - GET - Method Request'. The page shows settings like Authorization (NONE), Request Validator (NONE), and API Key Required (false). A red arrow points to the 'Name' field in the 'URL Query String Parameters' section, which contains a single entry 'fid'. The 'Required' column has an unchecked checkbox next to 'fid'.

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.

The image consists of two screenshots of the AWS Lambda console. The top screenshot shows the 'Functions' list with five functions: MainMonitoringFunction, RedshiftOverwatch, RoleCreationFunction, RedshiftEventSubscription, and getDesigns. The 'getDesigns' function is highlighted with a red box and an orange arrow points to it with the text 'Click this.'. The bottom screenshot shows the 'Configuration' tab for the 'getDesigns' function. On the left, there's a sidebar with various tabs like Code, Test, Monitor, Triggers, Permissions (which is selected), Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, Database proxies, File systems, and State machines. The main area has tabs for Configuration, Aliases, and Versions. Under 'Configuration', there's a section for 'Execution role' where 'Role name' is set to 'LabRole'. A red circle highlights 'LabRole' with an arrow pointing to it labeled 'Labrole.'

IAM > Roles > LabRole

LabRole

Summary

Creation date: June 27, 2023, 08:31 (UTC+08:00)

Last activity: 9 hours ago

ARN: arn:aws:iam::637825447844:role/LabRole

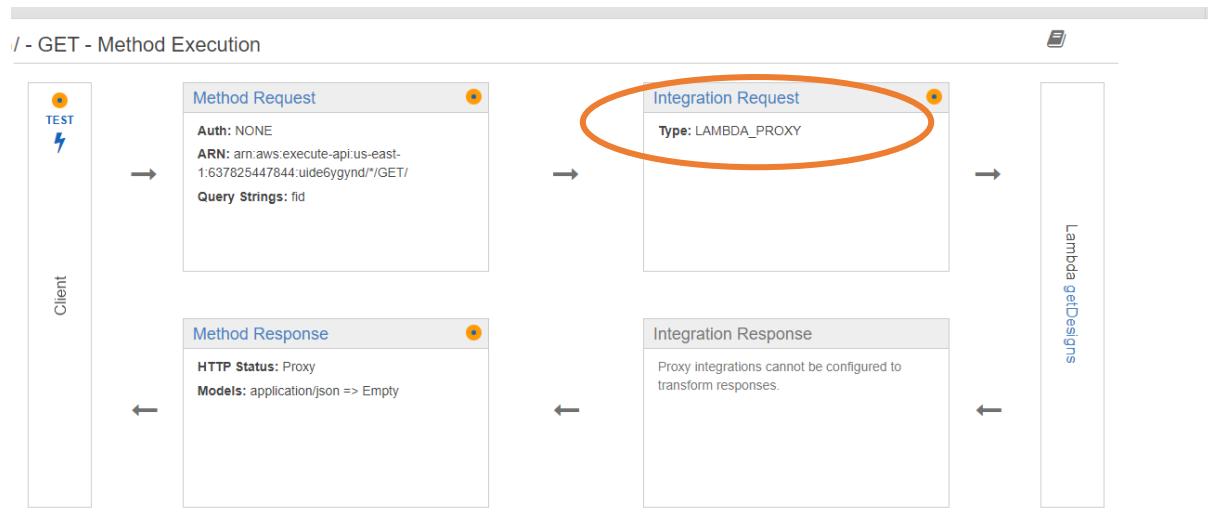
Maximum session duration: 1 hour

Instance profile ARN: arn:aws:iam::637825447844:instance-profile/LabInstanceProfile

Permissions | Trust relationships | Tags (1) | Access Advisor | Revoke sessions

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**.

ay APIs > beedesign_API (uide6ygynd) > Resources > / (kt0nidwno0) > GET

/ - GET - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type: Lambda Function HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration:

Lambda Region: us-east-1

Execution role:

Invoke with caller credentials:

Credentials cache: Do not add caller credentials to cache key

Use Default Timeout:

Enable Cors

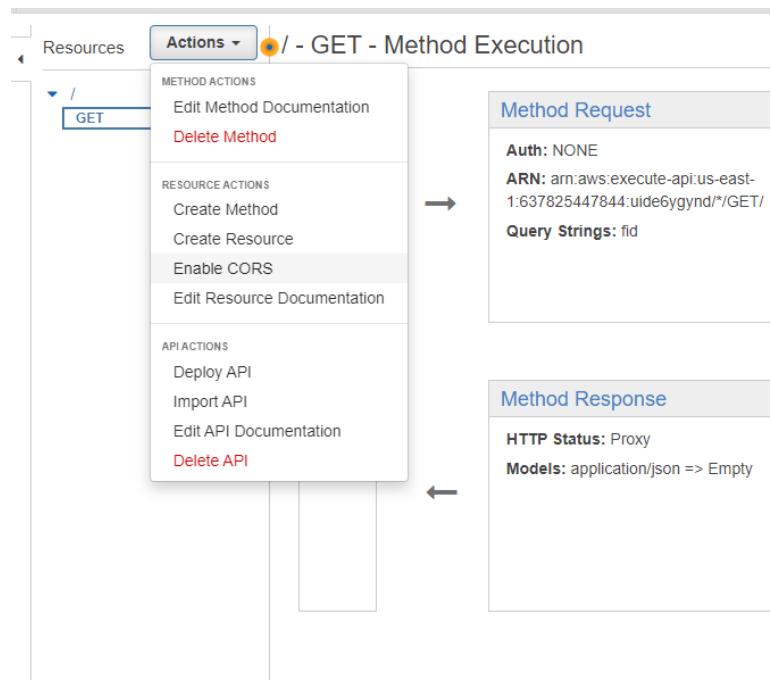
Cors Importance

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.

The screenshot shows the 'Enable CORS' configuration page for an API resource. At the top, there are checkboxes for 'DEFAULT 4XX' and 'DEFAULT 5XX'. Below that, under 'Methods', 'GET' is selected. In the 'Access-Control-Allow-Headers' field, the value 'X-Api-Key,X-Amz-Security-Token,user' is entered. The 'Enable CORS and replace existing CORS headers' button at the bottom is highlighted with an orange oval.

Step 3: Confirm it by pressing “Yes, replace existing values”

The screenshot shows a confirmation dialog box titled 'Confirm method changes'. It contains a message: 'The following modifications will be made to this resource's methods and will replace any existing values. Are you sure you want to continue?'. A bulleted list details the modifications, including creating an OPTIONS method and updating various headers. At the bottom right of the dialog is a blue button labeled 'Yes, replace existing values', which is highlighted with an orange oval.

Cors is successfully enabled.

The screenshot shows a configuration page titled "Enable CORS". On the left, there is a list of successful configuration steps, each preceded by a green checkmark:

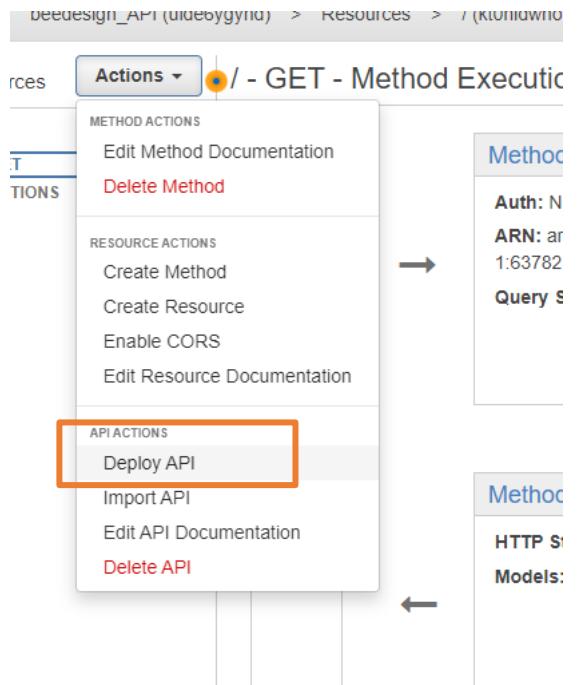
- ✓ Create OPTIONS method
- ✓ Add 200 Method Response with Empty Response Model to OPTIONS method
- ✓ Add Mock Integration to OPTIONS method
- ✓ Add 200 Integration Response to OPTIONS method
- ✓ Add Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin Method Response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

On the right side of the page, there is a small circular icon containing a blue letter "S".

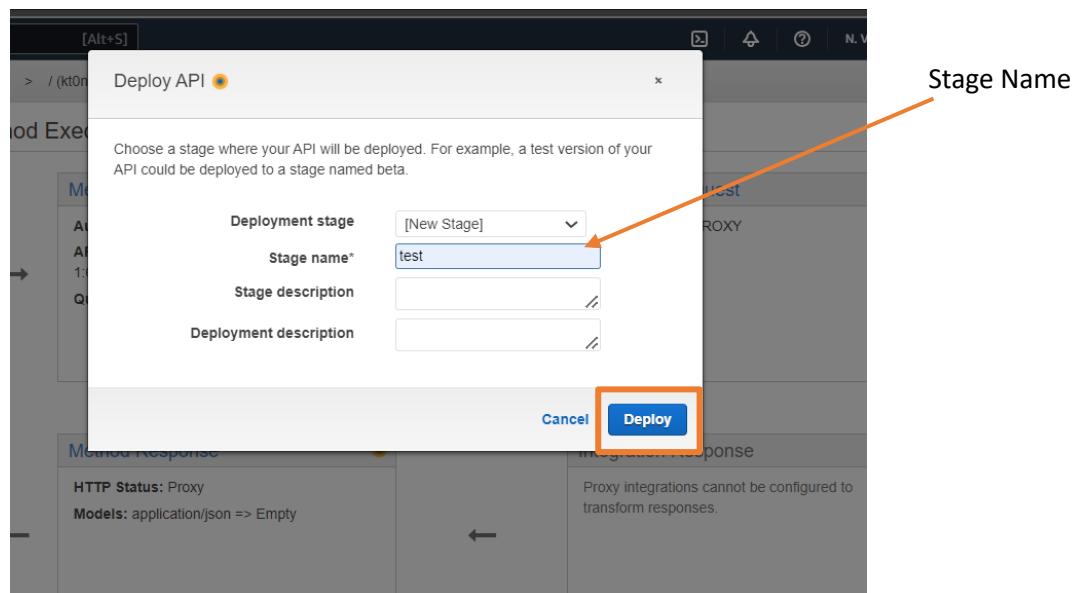
Deploy API

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.

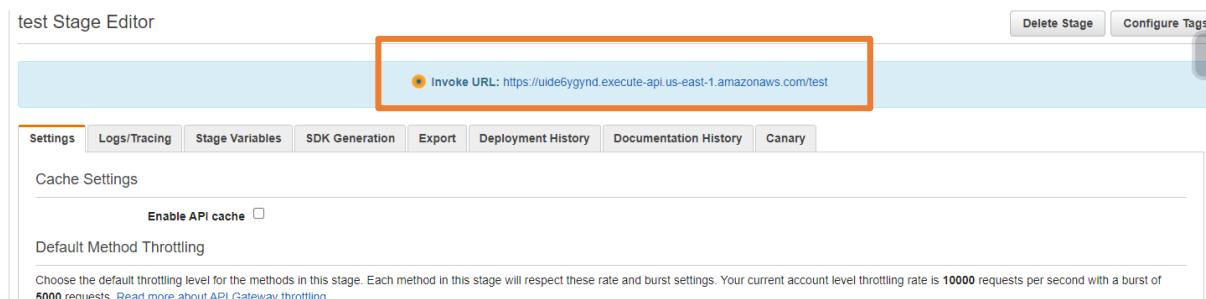
Step 1: Click Actions and click “Deploy API”.



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



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



The screenshot shows the AWS Lambda test stage editor. At the top, there's a button labeled "Invoke URL: https://uide6gyynd.execute-api.us-east-1.amazonaws.com/test". This button is highlighted with an orange rectangle. Below it, there are tabs for "Settings", "Logs/Tracing", "Stage Variables", "SDK Generation", "Export", "Deployment History", "Documentation History", and "Canary". Under the "Settings" tab, there's a section for "Cache Settings" with a checkbox for "Enable API cache". Below that is a section for "Default Method Throttling" with a note about rate and burst settings. A large orange arrow points from the "Invoke URL" button down to the code editor below.

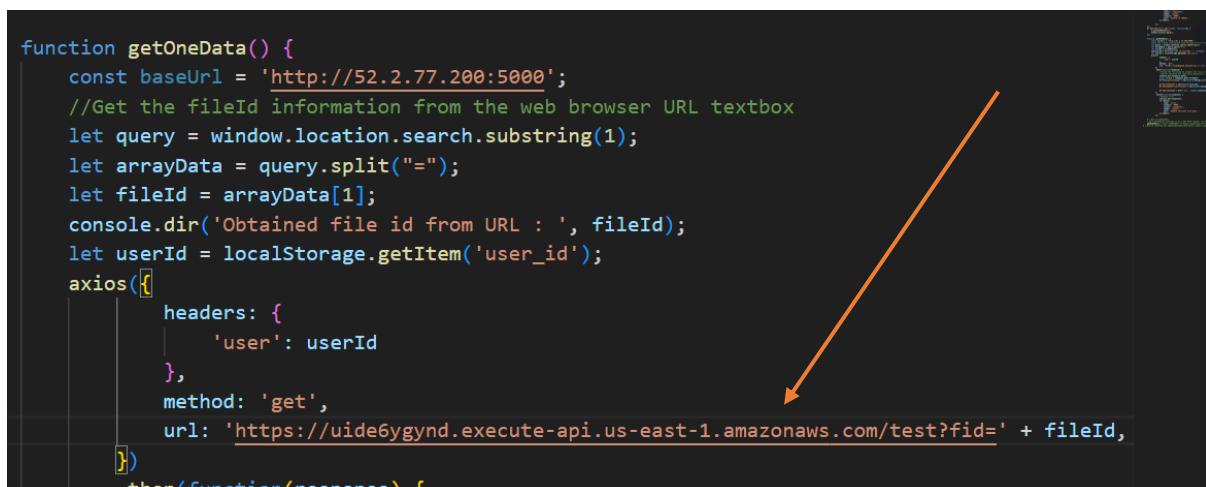
Modify App code to use API endpoint.

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://uide6gyynd.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://uide6gyynd.execute-api.us-east-1.amazonaws.com/test?fid=' + fileId,
  })
  .then(function(response) {
```

A screenshot of a code editor showing a JavaScript file with an axios request. The URL in the request is highlighted with an orange rectangle. An orange arrow points from this highlighted URL down to the code editor below, indicating where to make the modification.

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

C:\...\OneDrive\Desktop\esdeca2-term2\ESDECA2-INSECURECODE-SAKSHI\frontend\public\js				/home/ec2-user/BeeDesignApp/frontend/public/js/			
Name	Size	Type	Changed	Name	Size	Changed	Rights
...		Parent directory	3/7/2023 6:58:31 pm	...		25/7/2023 6:44:55 pm	rwx-r-x--
user_manage_submiss...	10 KB	JavaScript Source F...	13/8/2023 3:03:02 pm	admin_check_user_sub...	9 KB	8/8/2023 3:14:08 pm	rw-r-f--
update_design.js	5 KB	JavaScript Source F...	13/8/2023 3:47:19 pm	admin_manage_user.js	10 KB	8/8/2023 2:57:09 pm	rw-r-f--
submit_design.js	3 KB	JavaScript Source F...	11/8/2023 9:13:13 pm	admin_update_user.js	4 KB	8/8/2023 3:26:56 pm	rw-r-f--
register.js	3 KB	JavaScript Source F...	26/7/2023 2:41:22 am	axios.js	14 KB	3/7/2023 10:58:31 am	rw-r-f--
profile.js	2 KB	JavaScript Source F...	26/7/2023 2:41:22 am	global.js	1 KB	3/7/2023 10:58:31 am	rw-r-f--
manage_invite.js	3 KB	JavaScript Source F...	26/7/2023 2:41:22 am	login.js	3 KB	8/8/2023 11:27:18 am	rw-r-f--
login.js	3 KB	JavaScript Source F...	8/8/2023 7:27:18 pm	manage_invite.js	3 KB	25/7/2023 6:41:22 pm	rw-r-f--
global.js	1 KB	JavaScript Source F...	3/7/2023 6:58:31 pm	profile.js	2 KB	25/7/2023 6:41:22 pm	rw-r-f--
axios.js	14 KB	JavaScript Source F...	3/7/2023 6:58:31 pm	register.js	3 KB	25/7/2023 6:41:22 pm	rw-r-f--
admin_update_user.js	4 KB	JavaScript Source F...	8/8/2023 11:26:56 pm	submit_design.js	3 KB	11/8/2023 1:13:13 pm	rw-r-f--
admin_manage_user.js	10 KB	JavaScript Source F...	8/8/2023 10:57:09 pm	update_design.js	5 KB	13/8/2023 7:47:19 am	rw-r-f--
admin_check_user_sub...	9 KB	JavaScript Source F...	8/8/2023 11:14:08 pm	user_manage_submiss...	10 KB	13/8/2023 7:03:02 am	rw-r-f--

Modify DynamoDB

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

DynamoDB > Tables > files

Tables (1)

Any tag key Any tag value

Find tables by table name

files

Actions ▾ Explore table items

Overview Indexes Monitor Global tables Backups Exports and streams Additional settings

Protect your DynamoDB table from accidental writes and deletes

When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. [Learn more](#)

Edit PITR X

General information

Partition key file_id (Number)

Sort key -

Capacity mode Provisioned

Table status Active

Alarms [No active alarms](#)

Point-in-time recovery (PITR) [Info](#)

Off

Additional info

Before

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

Items returned (36)

file_id	clouddinary_file_id	clouddinary_url	created_by_id	design_description	design_title
100	Design/enpuktyzyafm...	http://res.cloudin...	100	kelly design 1 descriptor...	kelly design 1
101	Design/eksobbd5alhqy...	http://res.cloudin...	100	kelly design 2 descriptor...	kelly design 2
102	Design/nibdm6lz7zkc5...	http://res.cloudin...	100	kelly design 3 descriptor...	kelly design 3
103	Design/gg1dkckv23io...	http://res.cloudin...	100	kelly design 4 descriptor...	kelly design 4
104	Design/acnabsmorgqix...	http://res.cloudin...	100	kelly design 5 descriptor...	kelly design 5
105	Design/lrzcqxehcitxrgy...	http://res.cloudin...	100	kelly design 6 descriptor...	kelly design 6

Actions ▾ Create item

DynamoDB > Explore items: files > Edit item

Edit item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Attributes		Type	Add new attribute ▾
Attribute name	Value	Type	
file_id - Partition key	100	Number	
cloudinary_file_id	Design/enpuktzyafmac9xp41b	String	<button>Remove</button>
cloudinary_url	http://res.cloudinary.com/dqmyxllr8/image/upload/v1690310024/Design/enpuktzyafmac9xp41b.png	String	<button>Remove</button>
created_by_id	100	Number	<button>Remove</button>
design_description	kelly design 1 description text 1 text 2 text 3 text 4	String	<button>Remove</button>
design_title	Test Design 1	String	<button>Remove</button>

After changing design title in DynamoDB. Cancel Save changes

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

file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
100	Design/enpuktzyafm...	http://res.cloudin...	100	kelly design 1 descrip...	Test Design 1

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/
```

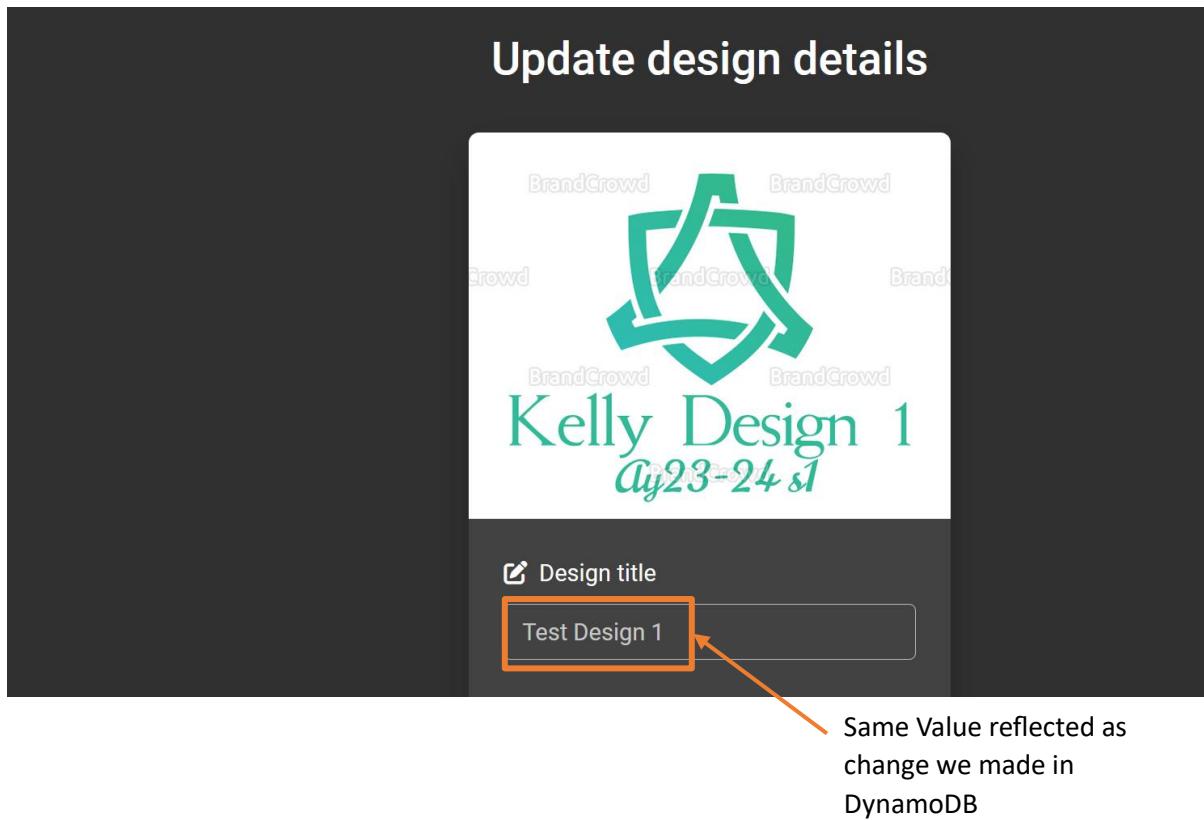
Frontend folder

```
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

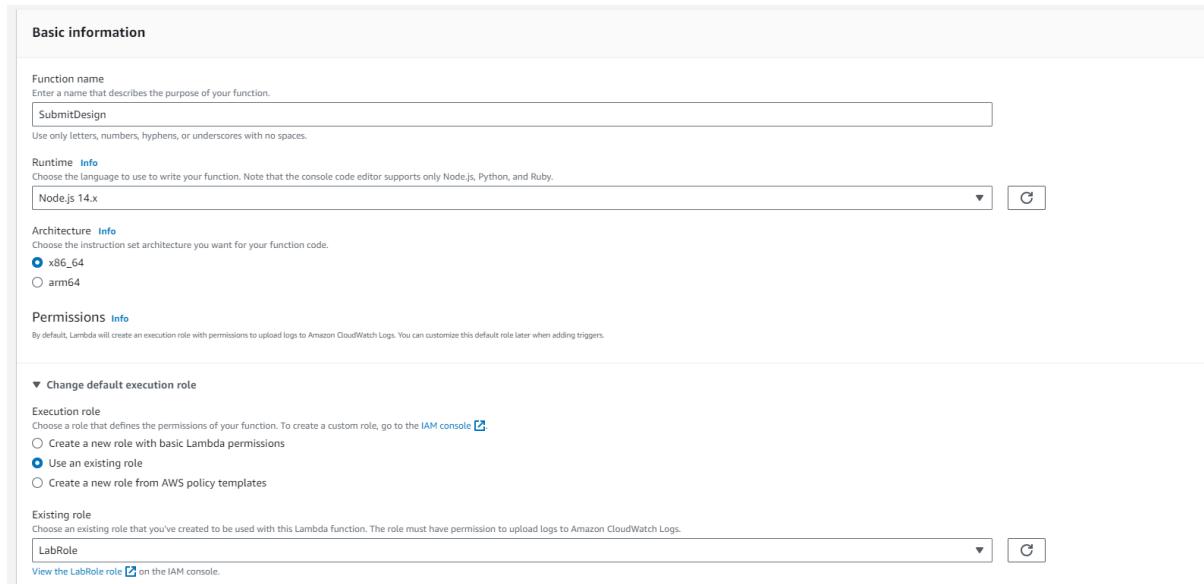
Submit Design (Post)

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

```
EXPLORER ...  
LAMDA  
> node_modules  
( package-lock.json  
( package.json  
> OUTLINE  
> TIMELINE  
> NPM SCRIPTS
```

```
1 {  
2   "name": "lambda",  
3   "version": "1.0.0",  
4   "lockfileVersion": 2,  
5   "requires": true,  
6   "packages": {  
7     "": {  
8       "name": "lambda",  
9       "version": "1.0.0",  
10      "license": "ISC",  
11      "dependencies": {  
12        "aws-sdk": "2.1427.0",  
13        "cloudinary": "^1.40.0"  
14      }  
15    }  
16  }  
17}
```

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 clouddinary

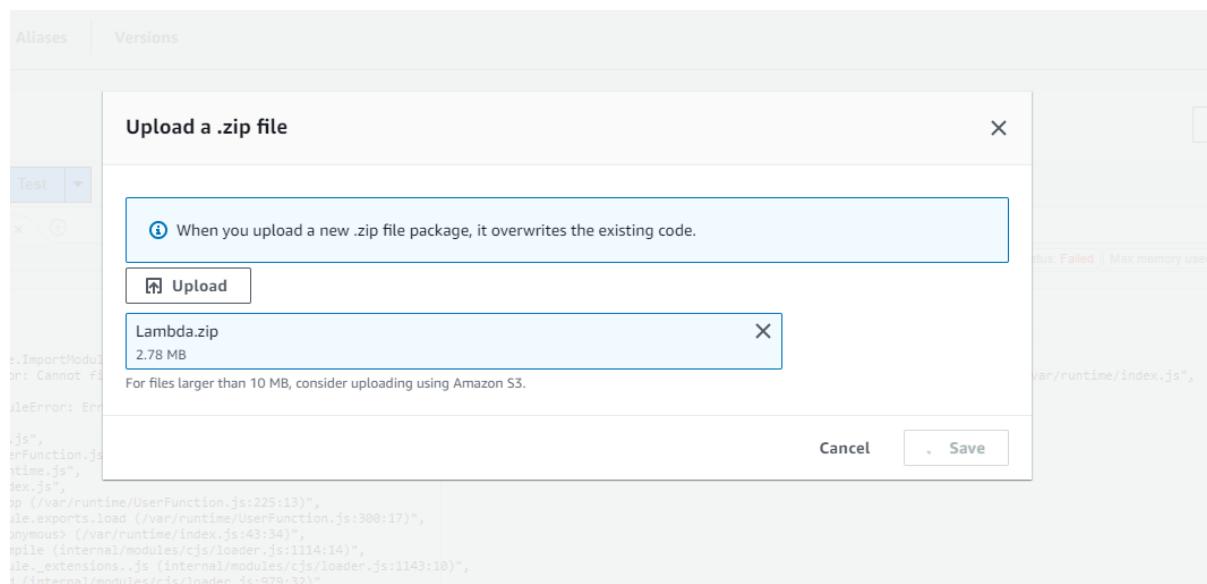
```
C:\Users\sakshi singh\OneDrive\Desktop\Lambda> npm install clouddinary
up to date, audited 37 packages in 2s

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

found 0 vulnerabilities
```

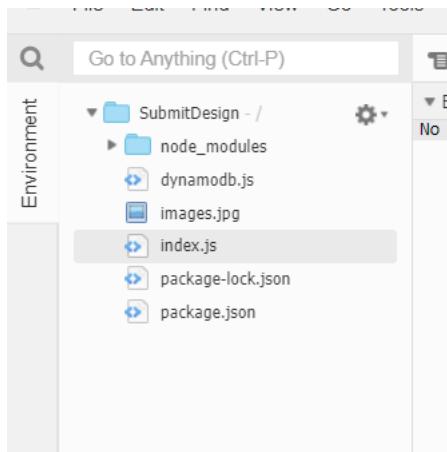
Step 5: Zip the file

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 clouddinary is necessary to access clouddinary 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.

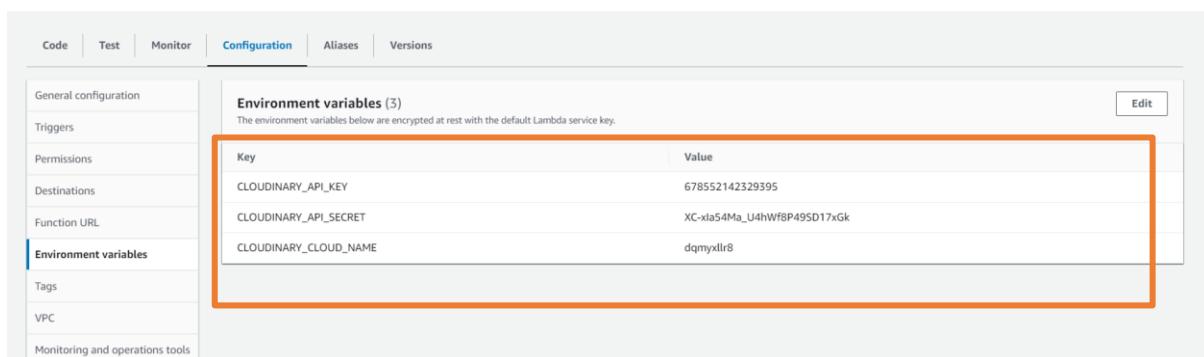
Step 1 : Copy the Clouddinary ids to a notepad .

```
CLOUDINARY_CLOUD_NAME=dqmyxllr8
CLOUDINARY_API_KEY=678552142329395
CLOUDINARY_API_SECRET=XC-xIa54Ma_U4hWf8P49SD17xGk
```

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 values



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. **generateNew fileId**: Calculates a new file ID.
3. **getLast fileId**: Retrieves the last file ID from DynamoDB.

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

```
1 const AWS = require('aws-sdk');
2
3 AWS.config.update({
4   region: 'us-east-1' // Update with your desired AWS region
5 });
6
7 const dynamodb = new AWS.DynamoDB.DocumentClient();
8 const tableName = 'files';
9
10 async function createFileData(imageURL, publicId, userId, designTitle, designDescription) {
11   const new fileId = await generateNew fileId(); // Generate a new file_id
12
13   const params = {
14     TableName: tableName,
15     Item: {
16       file_id: new fileId,
17       cloudinary_file_id: publicId,
18       cloudinary_url: imageURL,
19       created_by_id: userId,
20       design_title: designTitle,
21       design_description: designDescription
22     }
23   };
24
25   try {
26     await dynamodb.put(params).promise();
27   } catch (error) {
28     throw new Error('Error creating file data: ' + error.message);
29   }
30 }
31
32 async function generateNew fileId() {
33   const last fileId = await getLast fileId();
34   return last fileId + 1;
35 }
36
37 async function getLast fileId() {
38   let last fileId = 0;
39
40   const params = {
41     TableName: tableName,
42     ProjectionExpression: 'file_id'
43   };
44
45   try {
46     const result = await dynamodb.scan(params).promise();
47     const items = result.Items;
48
49     if (items.length > 0) {
50       last fileId = Math.max(...items.map(item => item.file_id));
51     }
52
53     return last fileId;
54   } catch (error) {
55     throw new Error('Error getting last file_id: ' + error.message);
56   }
57 }
58
59 module.exports = {
60   createFileData,
61 }
```

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.

```
index.js      dynamodb.js      Execution results      ENV
1  const AWS = require('aws-sdk');
2  const cloudinary = require('cloudinary').v2;
3  const { createFileData } = require('./dynamodb');
4
5  const cloudinaryCloudName = process.env.CLOUDINARY_CLOUD_NAME;
6  const cloudinaryAPIKey = process.env.CLOUDINARY_API_KEY;
7  const cloudinaryAPISecret = process.env.CLOUDINARY_API_SECRET;
8
9  cloudinary.config({
10    cloud_name: cloudinaryCloudName,
11    api_key: cloudinaryAPIKey,
12    api_secret: cloudinaryAPISecret,
13    upload_preset: 'upload_to_design',
14  });
15
16 exports.handler = async function(event, context) {
17
18  const requestBody = JSON.parse(event.body);
19  const { designTitle, designDescription, userId } = event.queryStringParameters;
20  const file = requestBody.file; // Assuming the body contains a JSON payload with the file information
21
22  const uploadResult = await cloudinary.uploader.upload(file.path,
23    { upload_preset: 'upload_to_design' },
24  );
25
26  const imageURL = uploadResult.url;
27  const publicId = uploadResult.public_id;
28
29  await createFileData(imageURL, publicId, userId, designTitle, designDescription);
30
31  return {
32    statusCode: 200,
33    body: JSON.stringify({ message: 'Update completed successfully.', imageURL, event: event }),
34    headers: {
35      'Access-Control-Allow-Headers': 'Content-Type,User',
36      'Access-Control-Allow-Origin': '*',
37      'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',
38    },
39  };
40} catch (error) {
41  console.error('Error:', error);
42}
43
44 return {
45  statusCode: 500,
46  body: JSON.stringify({ status: 'fail', message: error.message, event: event }),
47  headers: {
48    'Access-Control-Allow-Headers': 'Content-Type,User',
49    'Access-Control-Allow-Origin': '*',
50    'Access-Control-Allow-Methods': 'OPTIONS,POST,GET,PUT',
51  },
52};
53
54};
```

Annotations with arrows:

- An arrow points from the text "Cloudinary Credentials" to the configuration block at lines 9-14.
- An arrow points from the text "Data from Json Body event" to the assignment of `requestBody` at line 18.
- An arrow points from the text "Upload to cloudinary and get URL" to the call to `cloudinary.uploader.upload` at line 22.

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.

```
// To the server-side API when the #submitButton element fires the click event.
$("#submitButton").on("click", function (event) {
  event.preventDefault();
  const baseUrl = "http://52.2.77.200:5000";
  let userId = localStorage.getItem('userId');
  let designTitle = $('#designTitleInput').val();
  let designDescription = $('#designDescriptionInput').val();
  let file = document.getElementById('fileInput').files[0];
  let token = localStorage.getItem('token');

  // Create the JSON payload
  let jsonData = {
    designTitle: designTitle,
    designDescription: designDescription,
    userId: userId,
    file: {
      path: `./${file.name}` // Use the selected file's name
    }
  };

  axios({
    method: 'post',
    url: "https://uide6gyynd.execute-api.us-east-1.amazonaws.com/testS?designTitle=${encodeURIComponent(designTitle)}&designDescription=${encodeURIComponent(designDescription)}&userId=${userId}&token=${token}",
    data: jsonData, // Use JSON payload
    timeout: 5000,
    headers: {
      'user': userId,
      'Content-Type': 'application/json',
      'authorization': 'Bearer ' + token,
      // Set Content-Type to JSON
    }
  })
    .then(response =>
      console.log(`Success! Response: ${response.data}`);
    )
    .catch(error =>
      console.error(`Error: ${error.message}`);
    );
});
```

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, let's check the item in our DynamoDB files table. There are currently 40 items in the files table.

Items returned (40)					
#	cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
1	Design/enpuktyzyaf...	http://res.cloudin...	100	Works again api	Update Test 2
2	Design/eksobbd5alhqy...	http://res.cloudin...	100	kelly design 2 ...	kelly design 2
3	Design/nibdm6lz7zkc5...	http://res.cloudin...	100	kelly design 3 descript...	kelly design 3
4	Design/gg1dkckv23io...	http://res.cloudin...	100	kelly design 4 descript...	kelly design 4

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

Submit New Design

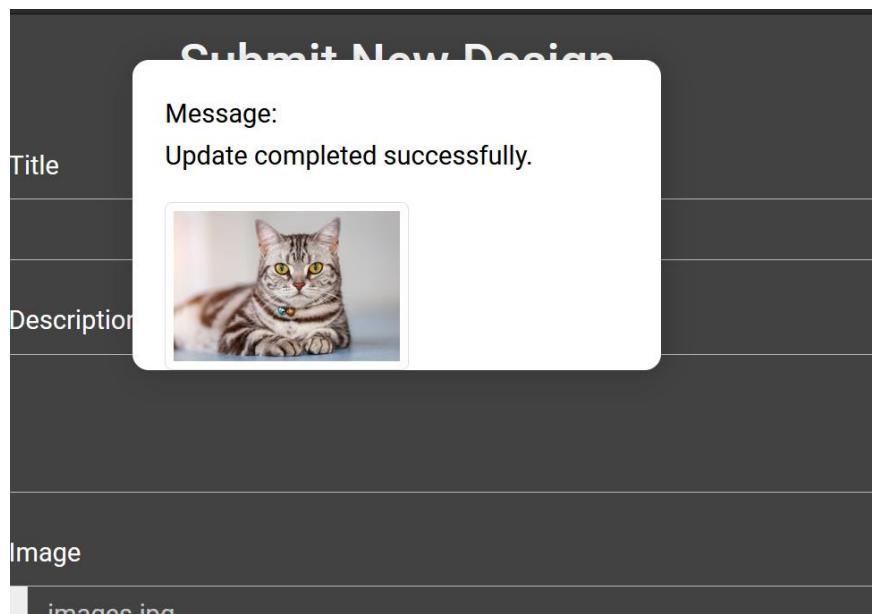
Design Title
ss

Design Description
ss

Design Image
Browse... images.jpg

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.

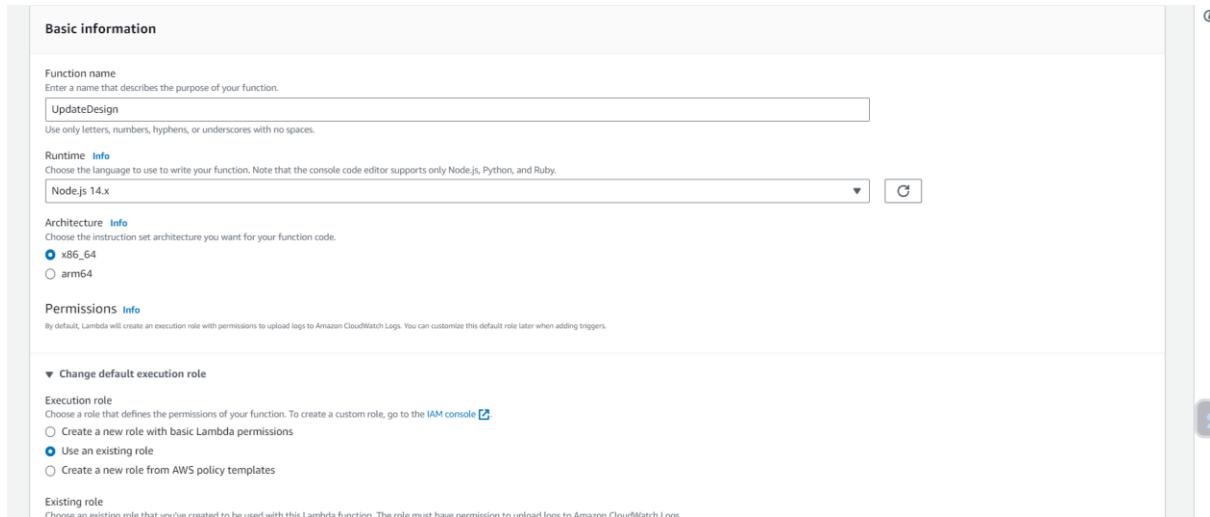
Items returned (41)										
	file_id (Number)	▲	cloudinary_file_id	▼	cloudinary_url	▼	created_by_id	▼	design_description	▼
<input type="checkbox"/>	100		Design/enpuktyzyafm...		http://res.cloudin...		100		Works again api	
<input type="checkbox"/>	101		Design/eksobbd5alhqy...		http://res.cloudin...		100		kelly design 2 descrip...	
<input type="checkbox"/>	102		Design/nibdm6lz7zkc5...		http://res.cloudin...		100		kelly design 3 descrip...	
<input type="checkbox"/>	140		Design/o1sn7lvffn6mf...		http://res.cloudin...		100		konny	
<input checked="" type="checkbox"/>	141		Design/nzd1hcwrbrazf...		http://res.cloudin...		100		ss	

Update Design (Put)

This function is for users to update their design information.

Create Lambda Function

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

```
1 var AWS = require('aws-sdk');
2
3 async function updateItemDynamoDB(region, table_name, fileId, design_title, design_description) {
4     var dynamodb = new AWS.DynamoDB.DocumentClient({ region });
5
6     var params = {
7         TableName: table_name,
8         Key: { "file_id": fileId },
9         UpdateExpression: "set design_description = :description, design_title = :title",
10        ExpressionAttributeValues: {
11            ":description": design_description,
12            ":title": design_title
13        }
14    };
15
16    try {
17        await dynamodb.update(params).promise();
18        console.log("Update successful.");
19    } catch (error) {
20        console.log("Error updating item: ", error);
21        throw error;
22    }
23
24}
25 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 fid;
    var design_title;
    var design_description;
    if (event.fid && event.designTitle && event.designDescription) {
      fid = event.fid;
      design_title = event.designTitle;
      design_description = event.designDescription;
    } else {
      fid = 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, fid, 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-Agent",
          "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-Agent",
          "Access-Control-Allow-Origin": "*", // Allow requests from any origin
          "Access-Control-Allow-Methods": "OPTIONS,POST,GET"
        }
      };
      console.log("response: " + JSON.stringify(response));
      return response;
    });
}

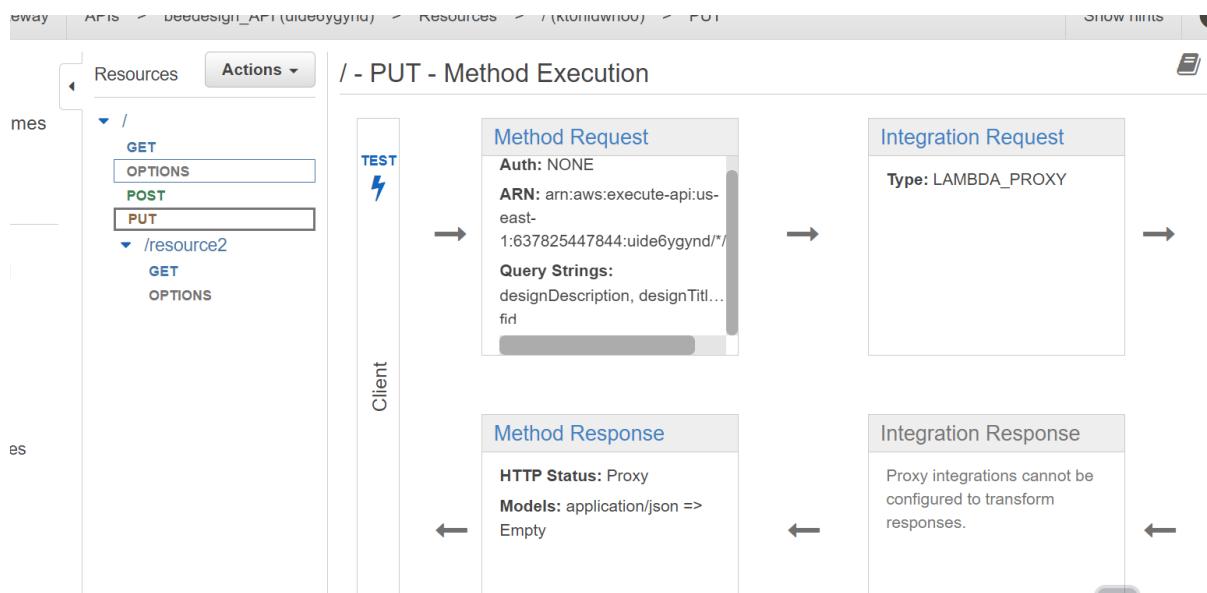
// Create the response with CORS headers for the 400 Bad Request case
let response = {
  statusCode: 400,
  body: JSON.stringify({ message: "Missing required parameters." }),
  headers: {
    "Access-Control-Allow-Headers": "Content-Type,User-Agent",
    "Access-Control-Allow-Origin": "*", // Allow requests 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.



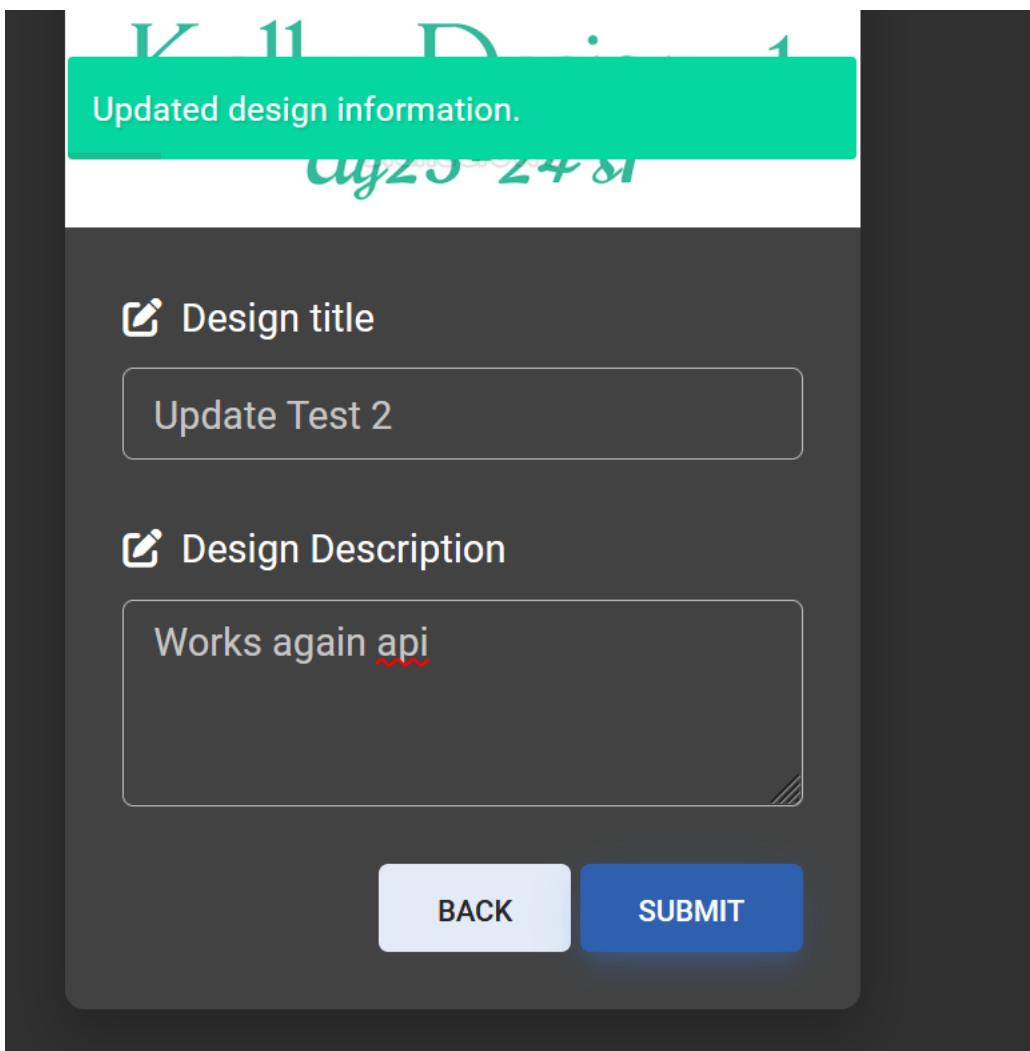
Modified Code for App Update_Design.js
Step 2: Updated api url with query parameters

```
method: 'put',
url: 'https://uide6gynd.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.

cloudinary_file_id	cloudinary_url	created_by_id	design_description	design_title
Design/enpuktyzyafm...	http://res.cloudin...	100	Works again api	Update Test 2

Manage User Submission (Get)

Create Lambda function.

Step 1: create Lambda function Mange Design

Lambda > Functions > ManageDesign

ManageDesign

Throttle Copy ARN Actions ▾

Function overview Info

ManageDesign

Layers (0)

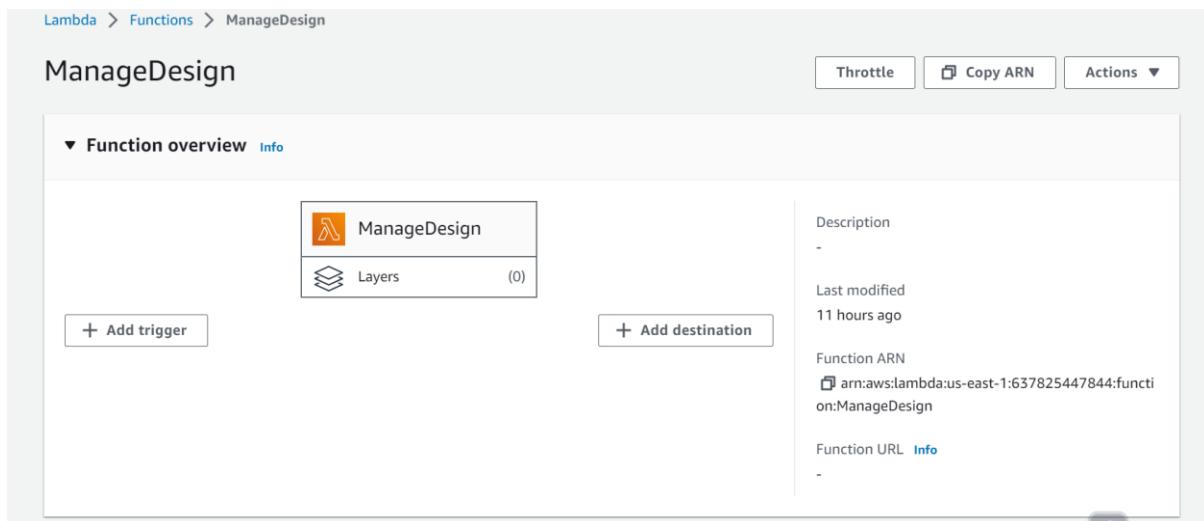
+ Add trigger + Add destination

Description -

Last modified 11 hours ago

Function ARN arn:aws:lambda:us-east-1:637825447844:function:ManageDesign

Function URL Info -



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 });

  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',
        },
      },
    ];
  }
}

```

28:27 1a

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

New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource

Resource Name*

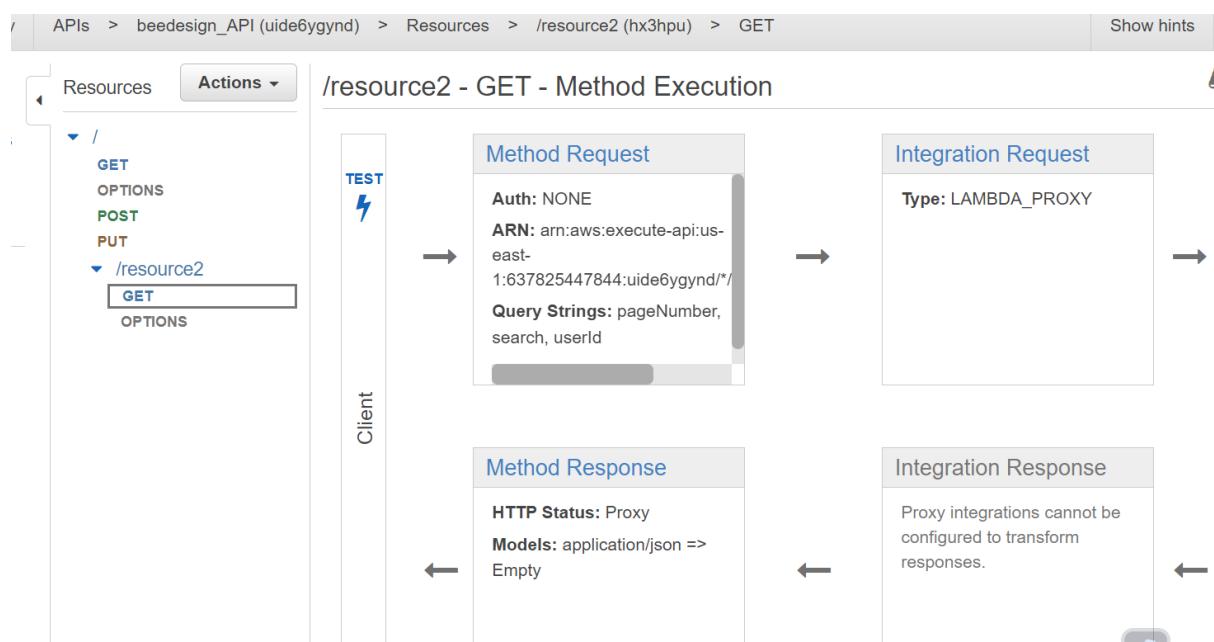
Resource Path*

You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS

* Required Cancel **Create 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://52.27.200:5000';
    let userId = localStorage.getItem('user_id');
    let pageNumber = $(event.target).text().trim();
    let searchInput = $('#searchInput').val();

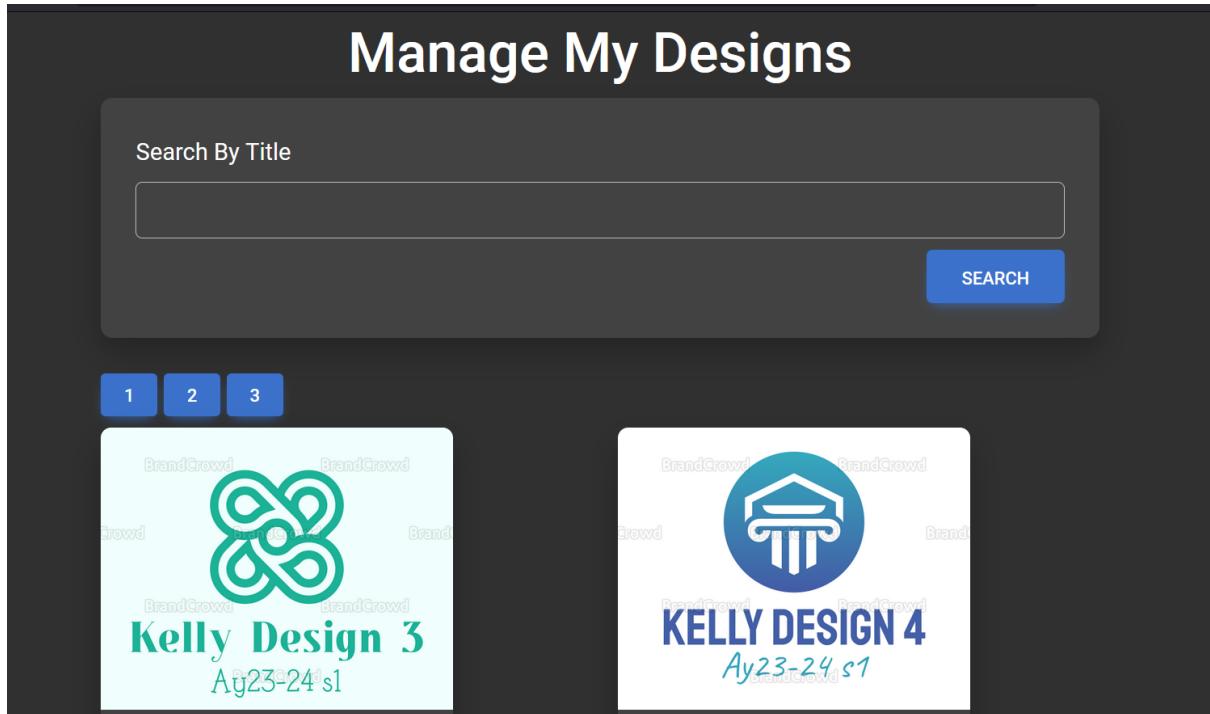
    let requestData = [
        pageNumber,
        userId: parseInt(userId),
        search: searchInput
    ]

    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.filidata;
        const totalNumOfRecords = response.data.total_number_of_records;
    })
}

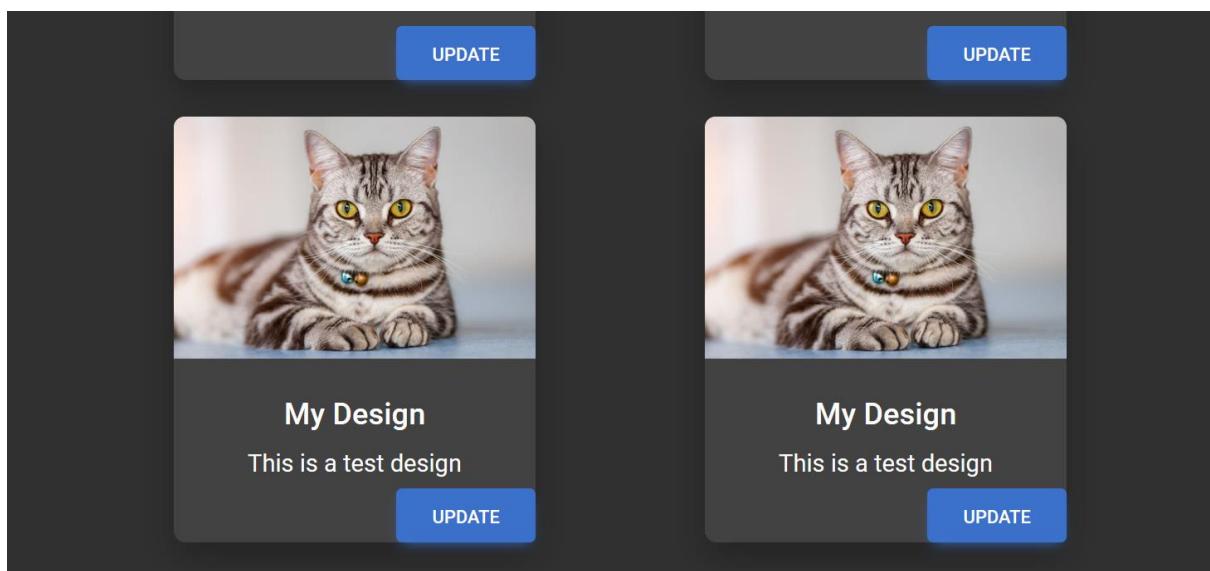
```

Testing Application

Step 1: Log in as Kelly, and press "Search"



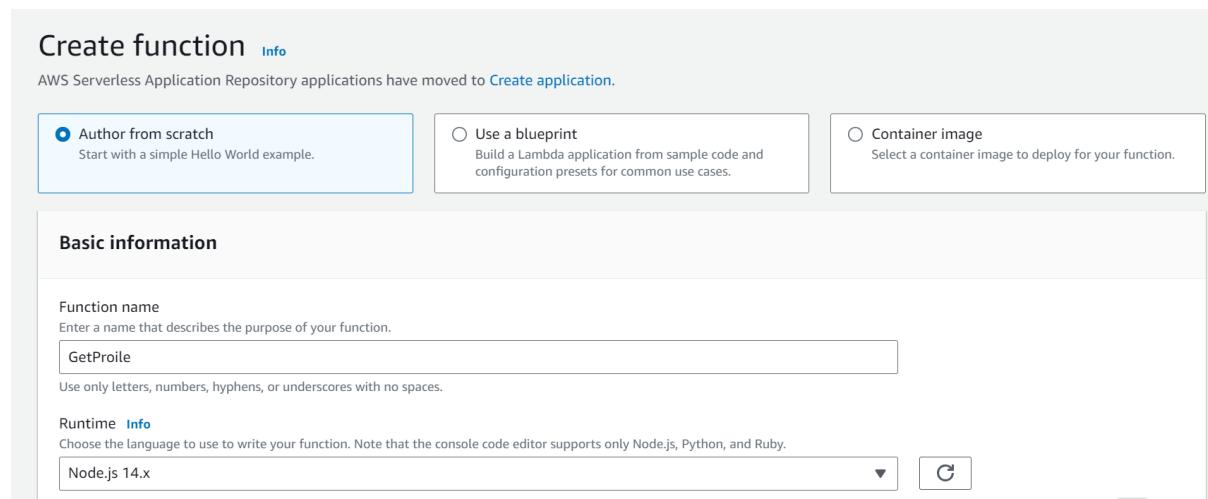
Step 2: You can verify that its retrieving from DynamoDb as it also retives the new submisions from the Submit Design we have done earlier.



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.

```
1 var AWS = require('aws-sdk');
2
3 async function queryitems_dynamodb(region, table_name, expr_attr_values, key_cond_expr, proj_expr) {
4     console.log("In the queryitems_dynamodb method..")
5     var dynamodb = new AWS.DynamoDB.DocumentClient({ region });
6
7     try {
8         var params = {
9             TableName: table_name,
10            ExpressionAttributeValues: expr_attr_values,
11            KeyConditionExpression: key_cond_expr,
12            ProjectionExpression: proj_expr
13        }
14
15        var items = []
16
17        const results = await dynamodb.query(params).promise()
18        console.log("Printing results from queryitems_dynamodb " + results)
19        return results;
20
21    } catch (tryerror) {
22        console.log("Error occurred in dynamodb.query..")
23        console.log(tryerror, tryerror.stack); // an error occurred
24    }
25} //end function
26
27
28 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.

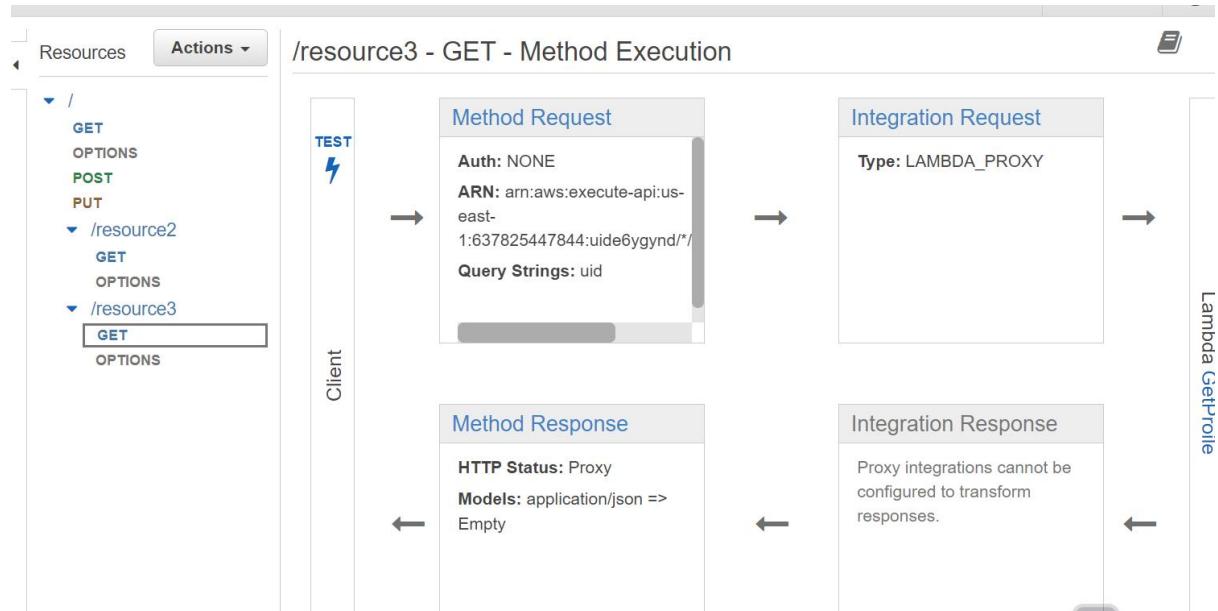


The image shows a code editor interface with two tabs: "index.js" and "Environment". The "index.js" tab contains the following code:

```
exports.handler = async function (event, context, callback) {
  try {
    console.log("Received event:", JSON.stringify(event));
    if (event.uid || event.queryStringParameters.uid) {
      var userId = event.uid || parseInt(event.queryStringParameters.uid);
      console.log("UserId:", userId);
      var region = "us-east-1";
      var table_name = "users";
      var expr_attr_values = {":userId": userId};
      var key_cond_expr = "user_id = :userId";
      var proj_expr = "user_id, fullname, email, role_id";
      console.log("Calling dynamodbquery...");
      const data = await dynamodbQuery(region, table_name, expr_attr_values, key_cond_expr, proj_expr);
      console.log("Successfully retrieved items from user table:", JSON.stringify(data));
      var responseCode = 200;
      var jsonResult = { 'userdata': data.Items[0] };
      console.log(jsonResult);
      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);
    } else {
      console.log("No user ID found in the request.");
      var responseCode = 400;
      let response = {
        statusCode: responseCode,
        body: JSON.stringify({ error: "No user ID provided." }),
        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.error("An error occurred:", error);
    var responseCode = 500;
    let response = {
      statusCode: responseCode,
      body: JSON.stringify({ error: "Internal Server Error", responseData: response.data }),
      headers: {
        "Access-Control-Allow-Headers": "Content-Type, user",
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Methods": "OPTIONS, POST, GET"
      }
    };
  }
};
```

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);
```

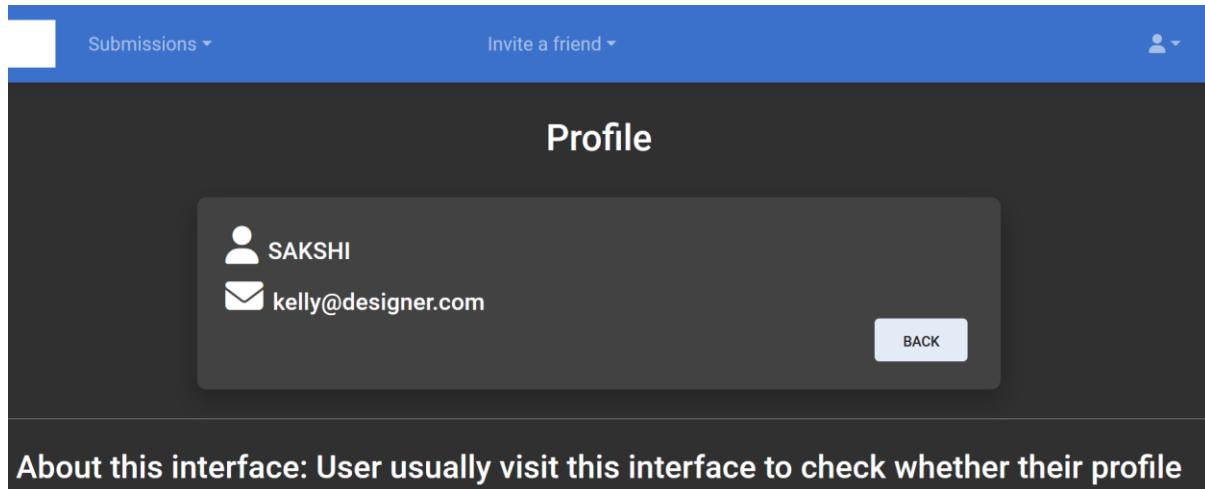
Testing application

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

	104	Albert@ad...	Albert	1	\$2D\$10
	100	kelly@desi...	SAKSHI	2	\$2b\$10
	108	jacob@desi...	jacob	2	\$2b\$10

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.



The screenshot shows a mobile application interface titled "Profile". At the top, there is a blue header bar with three items: "Submissions ▾", "Invite a friend ▾", and a user icon with a dropdown arrow. Below the header, the word "Profile" is centered in a large white font. The main content area has a dark gray background and contains two entries: a user icon followed by the name "SAKSHI" and an envelope icon followed by the email address "kelly@designer.com". In the bottom right corner of this content area, there is a small white button labeled "BACK".

About this interface: User usually visit this interface to check whether their profile

MySQL RDS security

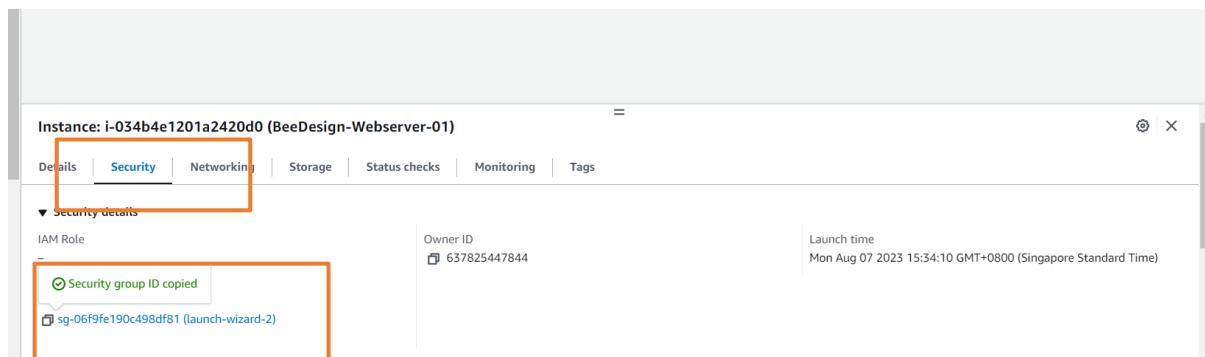
Ensuring only the EC2 instance running NODEJS code can have access to RDS

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.**

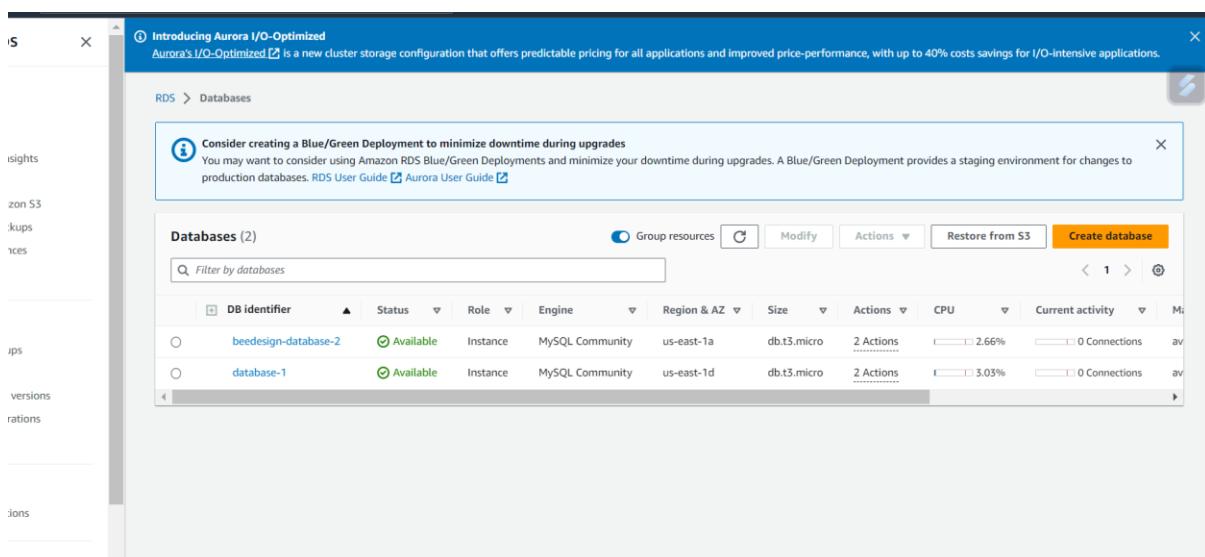
This configuration ensures that the Bee Design Website on the our EC2 instance can communicate with the RDS instance while maintaining a secure network environment.

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

RDS > Databases > beedesign-database-2

beedesign-database-2

Summary

DB identifier: beedesign-database-2 CPU: 2.71% Status: Available Class: db.t3.micro

Role: Instance Current activity: 0 Connections Engine: MySQL Community Region & AZ: us-east-1a

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Connectivity & security

Endpoint & port Networking Security

Endpoint: beedesign-database-2.c2obknd2twss.us-east-1.rds.amazonaws.com Availability Zone: us-east-1a VPC security groups: default (sg-043d70babda98751d) Active

Port: 3306 VPC: vpc-0c050c2288d0d14c6 Publicly accessible: Yes Subnet group: default-vpc-subnet-group-3306db11dr6 Certificate authority: Info

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

sg-043d70babda98751d - default

Details | **Inbound rules** | Outbound rules | Tags

You can now check network connectivity with Reachability Analyzer Run Reachability Analyzer X

Inbound rules (2)

Filter security group rules

C Manage tabs 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

The screenshot shows the 'Edit inbound rules' page. It displays a table of rules with columns for Security group rule ID, Type, Protocol, Port range, Source, and Description. Four new rules have been added:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-02c8f6569a816b801	MySQL/Aurora	TCP	3306	Custom	0.0.0.0/0
sgr-0b8f2ff226668a391	All traffic	All	All	Custom	sg-043d70babda98751d
-	Custom TCP	TCP	3001	Custom	sg-06f9fe190c498df81
-	Custom TCP	TCP	5000	Custom	sg-06f9fe190c498df81

The last two rows (port 3001 and 5000) are highlighted with an orange box. At the bottom right are buttons for 'Cancel', 'Preview changes', and 'Save rules'.

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

The screenshot shows the 'Inbound rules' list. It displays a table of rules with columns for Security group rule ID, IP version, Type, Protocol, and Port range. The last two rules (port 3001 and 5000) are highlighted with an orange box.

Security group rule ID	IP version	Type	Protocol	Port range
gr-07064caedb49ce7fd	-	Custom TCP	TCP	3001
gr-02c8f6569a816b801	IPv4	MySQL/Aurora	TCP	3306
gr-0b8f2ff226668a391	-	All traffic	All	All
gr-0831d17401e496af3	-	Custom TCP	TCP	5000

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 rules of 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?

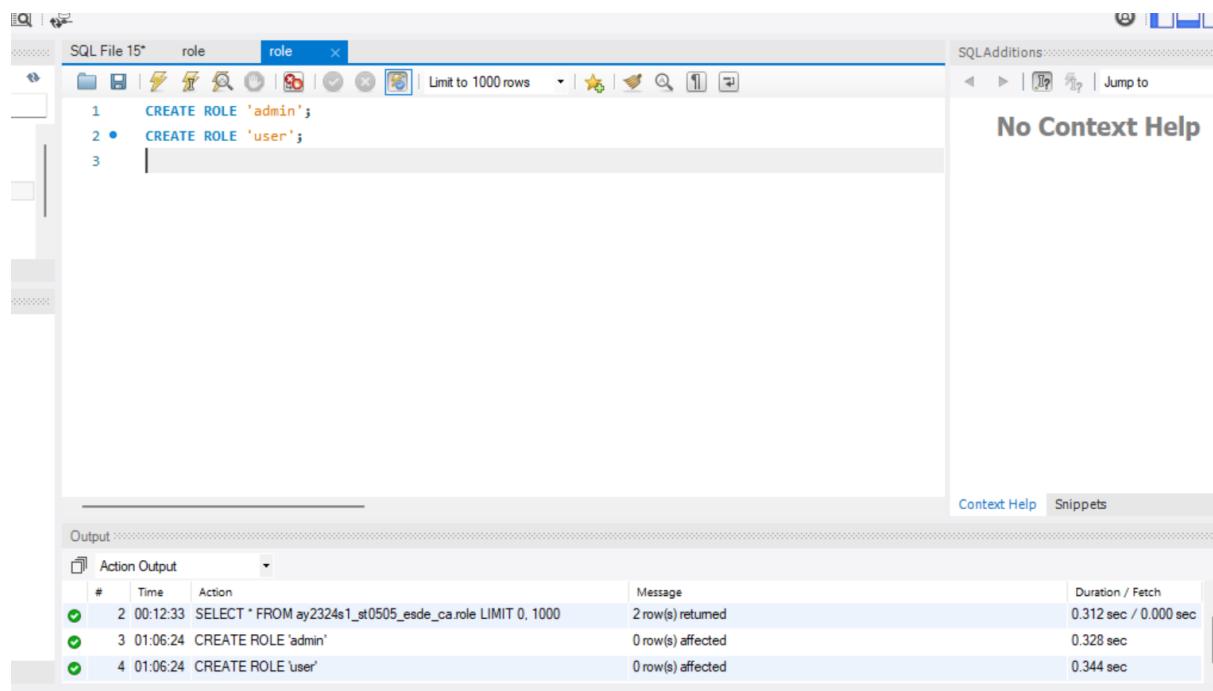
The privileges you granted in MySQL will be enforced at the database level. When you grant privileges to a specific user or role in MySQL, those privileges control what actions that user or role can perform on the specified database objects (e.g., tables, views). These privileges include actions like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



The screenshot shows the MySQL Workbench interface. In the main editor window, there are two SQL statements:

```
1 CREATE ROLE 'admin';
2 • CREATE ROLE 'user';
3 |
```

The second statement is highlighted with a blue dot, indicating it is the current statement being run. Below the editor, the "Output" pane displays the results of the executed statements:

Action	Time	Message	Duration / Fetch
2 00:12:33	SELECT * FROM ay2324s1_st0505_esde_ca.role LIMIT 0, 1000	2 row(s) returned	0.312 sec / 0.000 sec
3 01:06:24	CREATE ROLE 'admin'	0 row(s) affected	0.328 sec
4 01:06:24	CREATE ROLE 'user'	0 row(s) affected	0.344 sec

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.

The screenshot shows a SQL editor interface with a toolbar at the top and a main workspace below. In the workspace, there are two GRANT SELECT statements:

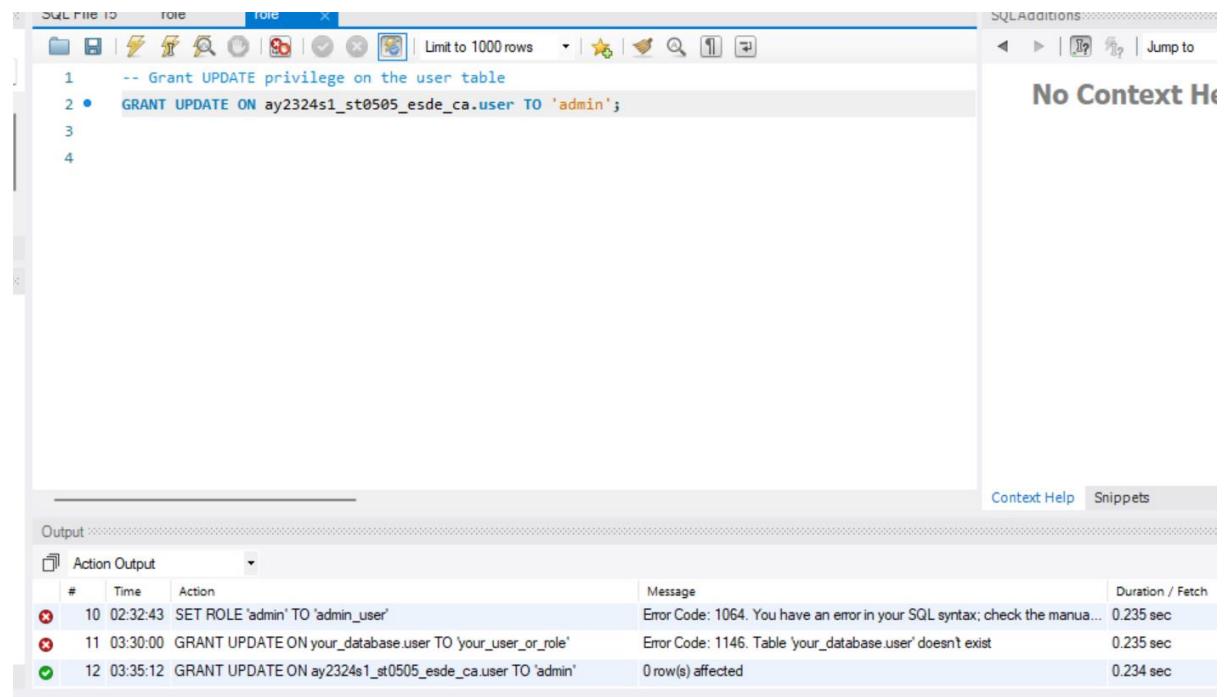
```
1 • GRANT SELECT ON ay2324s1_st0505_esde_ca.user TO 'admin';
2 • GRANT SELECT ON ay2324s1_st0505_esde_ca.role TO 'admin';
3
```

To the right of the statements, a message "No Context" is displayed. Below the workspace is an "Output" section titled "Action Output". It contains a table with the following data:

#	Time	Action	Message	Duration / Fetch
4	01:06:24	CREATE ROLE 'user'	0 row(s) affected	0.344 sec
5	01:13:40	GRANT SELECT ON ay2324s1_st0505_esde_ca.user TO 'admin'	0 row(s) affected	0.296 sec
6	01:13:41	GRANT SELECT ON ay2324s1_st0505_esde_ca.role TO 'admin'	0 row(s) affected	0.328 sec

Step 3: Admin – can update user privilege

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



The screenshot shows the MySQL Workbench interface. In the SQL Editor pane, there is a single line of code:

```
GRANT UPDATE ON ay2324s1_st0505_esde_ca.user TO 'admin';
```

In the Output pane, the results of the execution are shown in a table:

#	Time	Action	Message	Duration / Fetch
10	02:32:43	SET ROLE 'admin' TO 'admin_user'	Error Code: 1064. You have an error in your SQL syntax; check the manual...	0.235 sec
11	03:30:00	GRANT UPDATE ON your_database.user TO your_user_or_role'	Error Code: 1146. Table 'your_database.user' doesn't exist	0.235 sec
12	03:35:12	GRANT UPDATE ON ay2324s1_st0505_esde_ca.user TO 'admin'	0 row(s) affected	0.234 sec

VerifyAdmin function

The middleware and role-based authorization adds an additional layer of security and control. It ensures 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

```
end > src > middlewares > js verifyFn.js > verifyAdminToken > verifyAdminToken > jwt.verify() callback
module.exports.verifyAdminToken = (req, res, next) => {
  //logger.info("verifyTokenUserID middleware called");
  let token = req.headers['authorization'];
  console.log('token: ',req.headers);
  console.log('verifying admin token.....')
  res.type('json');
  if (!token || !token.includes("Bearer")) {
    console.log("Unauthorized access attempt was made, no token");
    res.status(403);
    res.send(`{"Message":"Not Authorized"}`);
  } else {
    console.log("Token found... now trying to decode it...")
    token = token.split('Bearer ')[1];
    jwt.verify(token, config.JWTKey, function (err, decoded) {
      if (err) {
        console.log("Unauthorized access attempt was made, invalid token");
        res.status(403);
        res.send(`{"Message":"Not Authorized"}`);
      } else {
        console.log('decoded id is ', decoded.role_id)
        if(decoded.role_id == 1) [ //1 is admin, 2 is user
          console.log("Token verified");
          req.body.userId = decoded.id;
          // if (req.role )
          next();
          return;
        } else {
          console.log("Unauthorized access attempt was made, not admin token")
          res.status(403);
          res.send(`{"Message":"Not Authorized"}`);
        }
      }
    });
  }
}
```

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.