**ESDE – Cloud Computing**

**Assignment 1**

|  |
| --- |
| **Basic Setup** |

**Register Account in AWS**

1. Register at <https://www.awseducate.com/Application>
2. Select ‘student’ role
3. Fill out student application (use ichat email as your email)
4. After submitting the application, check your ichat email, you will receive an email
5. Verify your email using the email
6. Accept the terms and conditions
7. When they have said your application is under review, check your email
8. If your application is rejected, check your email and follow instructions in the email.

\*Else if your application is accepted, follow the instructions to reset the password to login for the first time.

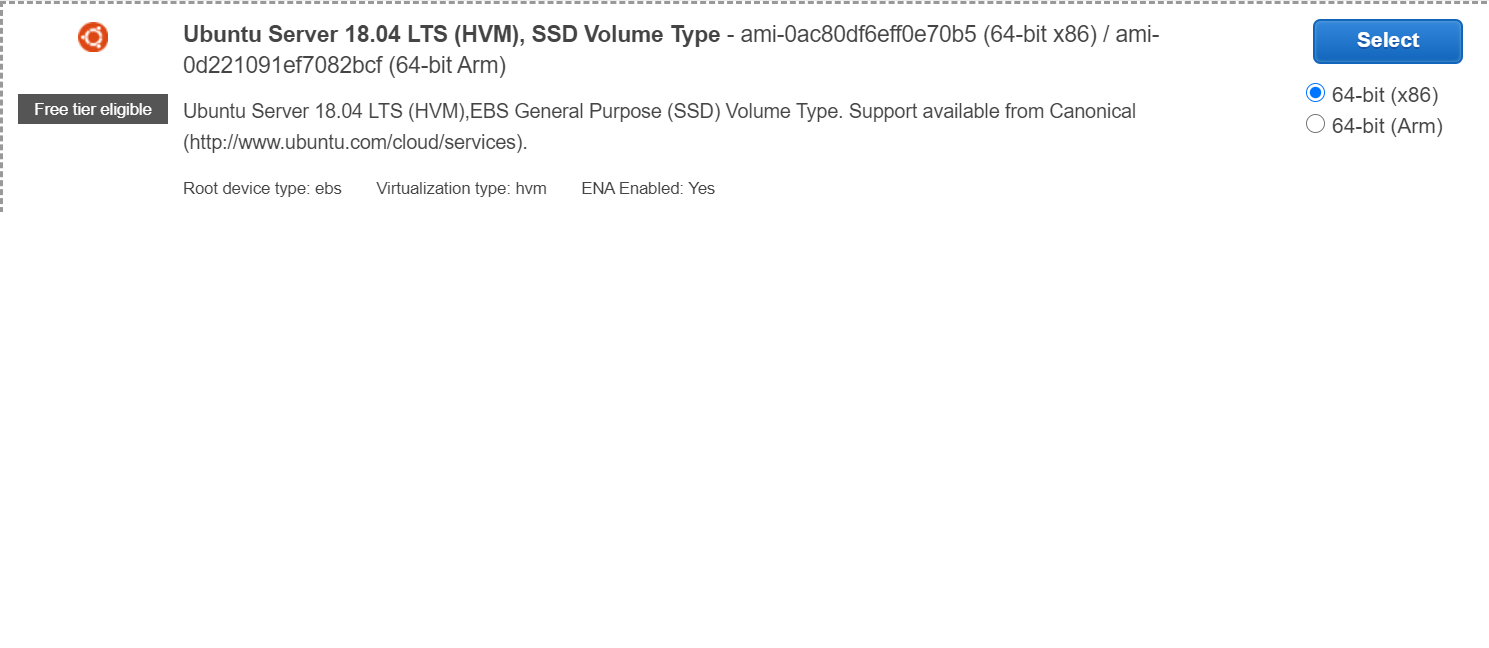
1. After setting your password, login by clicking on ‘AWS Account’
2. Click on the orange button ‘AWS Educate Starter Account’
3. You will be brought to vocareum, click on AWS console
4. Then you will be on the AWS dashboard

**Task 1: For cloud computing, you are to deploy the above application in AWS cloud, using EC2 service, AWS RDS service, S3 services, etc.**

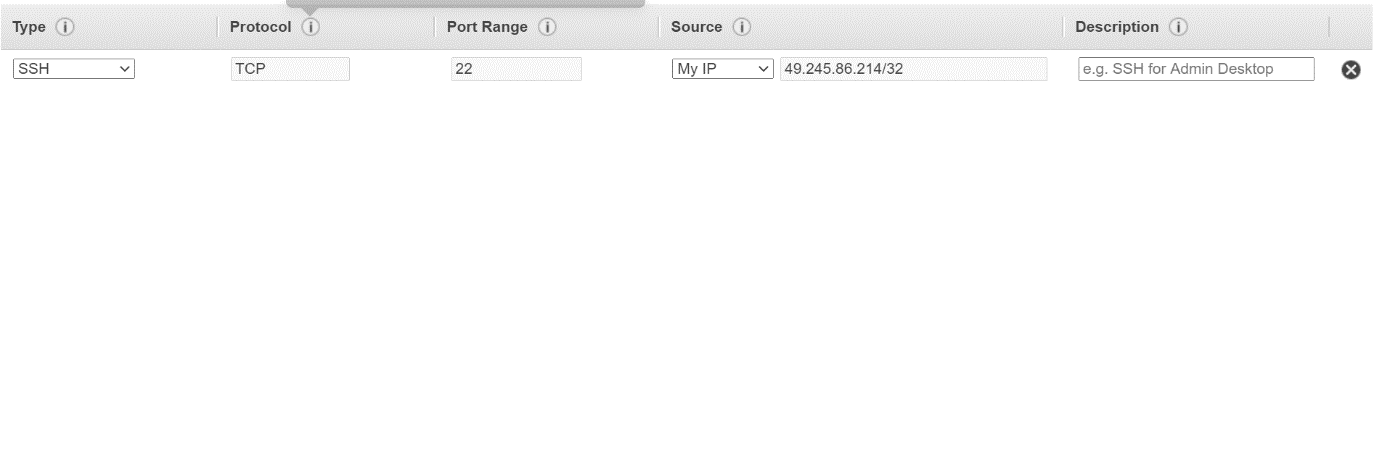
|  |
| --- |
| **Set up Ubuntu 18.04 on AWS EC2 Instance** |

**Create Instance**

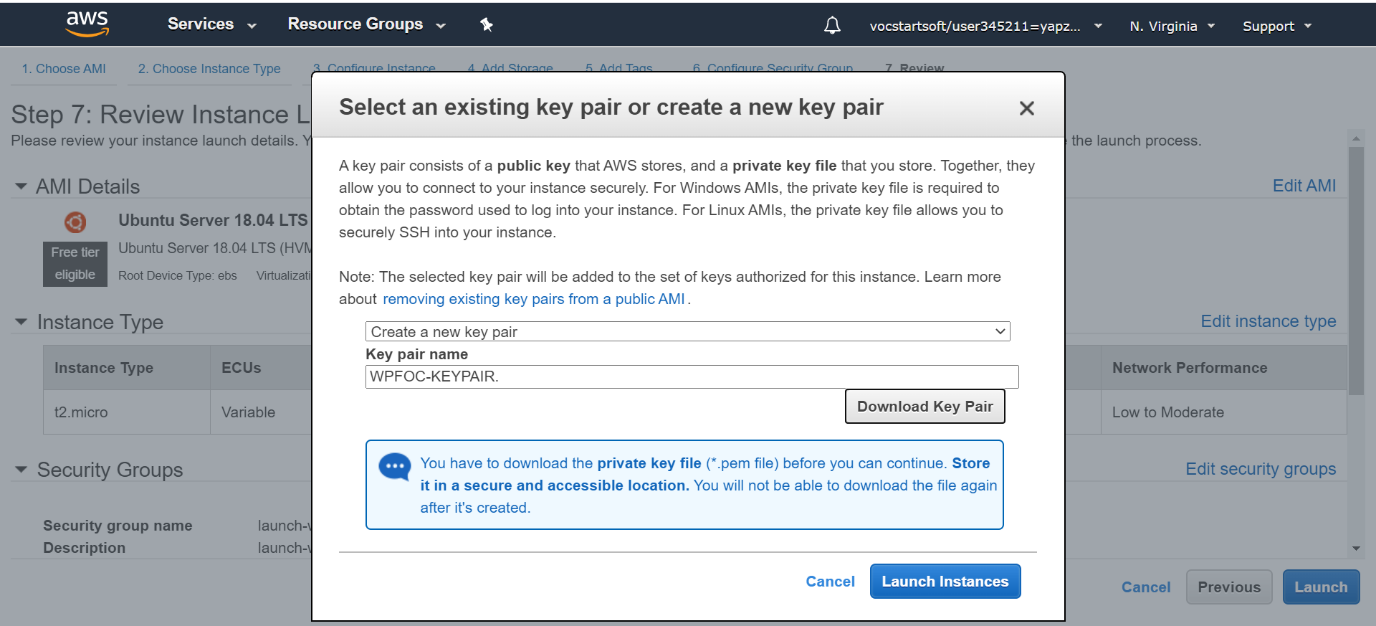
1. Click on services
2. Select EC2, then launch instance
3. In step 1, when choosing community AMIs, select Ubuntu Server 18.04 LTS (HVM), SSD Volume Type 64-bit (x86)

****

1. In step 6, when choosing security group, change the source to ‘My IP’ and click the ‘review and launch’ button and ‘launch’ button.



1. Create a new keypair and save it on your computer as ‘WPFOC-KEYPAIR.’ (keep your keypair in a safe place, do not lose it!)



1. Then, launch instance.

**Connect Created Remote Instance Using Putty**

- we need to generate the downloaded private key (.pem extennsion) to login to the remote instance using putty.

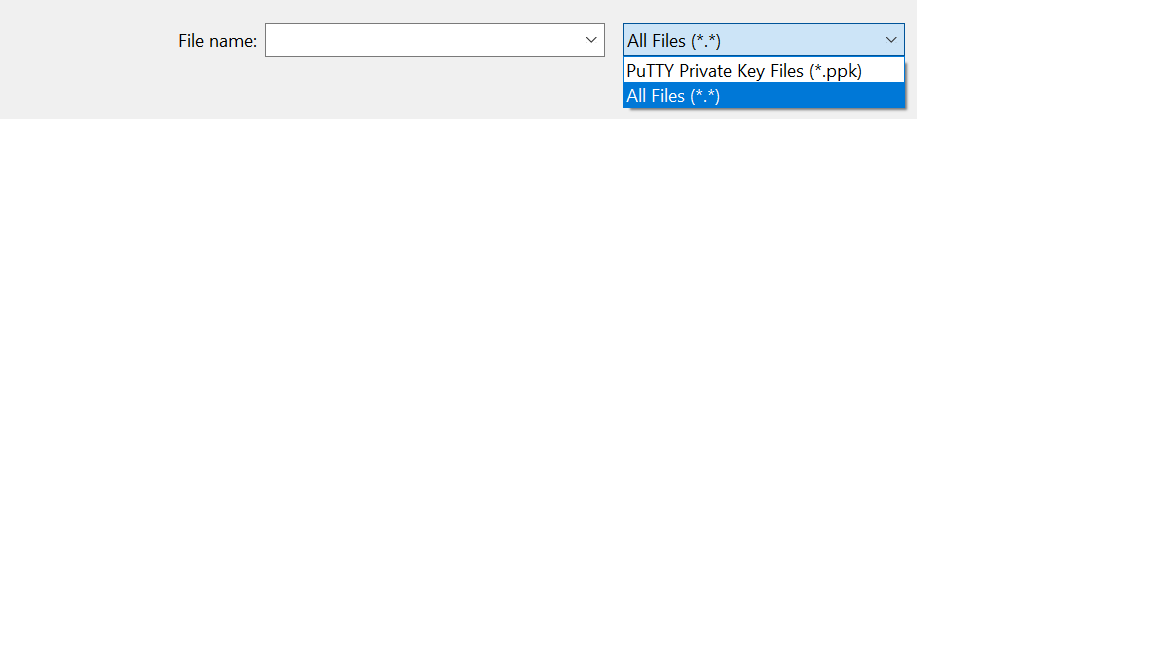
1. Install putty (PuttyGen will also be installed)
2. Open the PuttyGen
3. Under Type of key to generate, choose RSA

Note: If you're using an older version of PuTTYgen, choose SSH-2 RSA

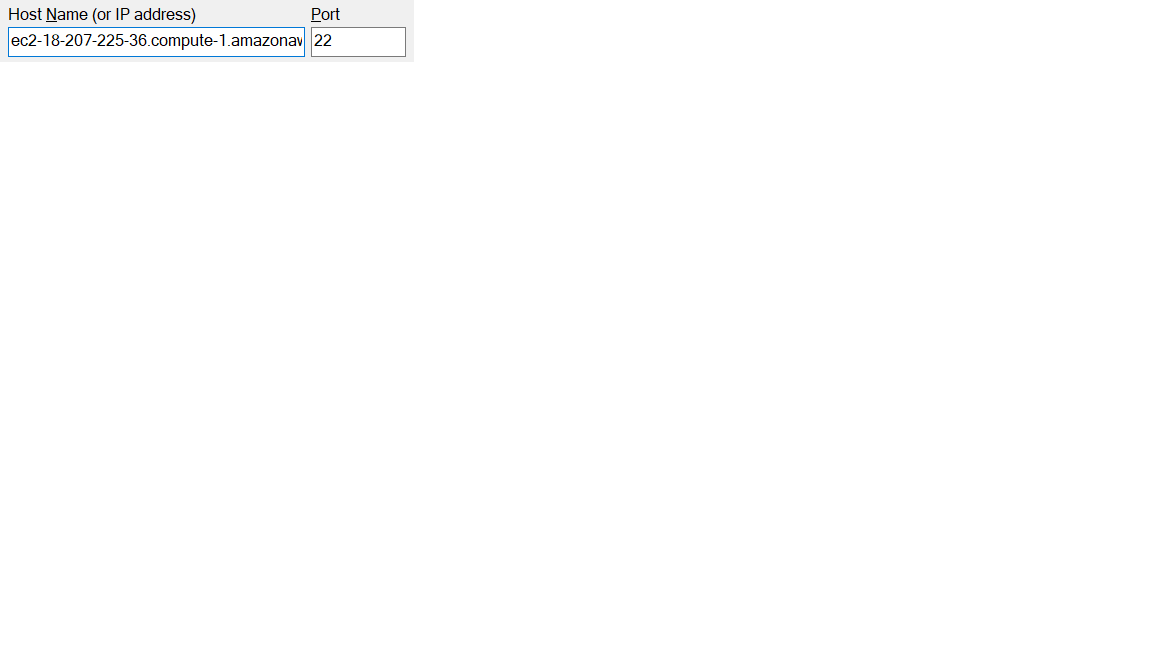


1. Choose Load

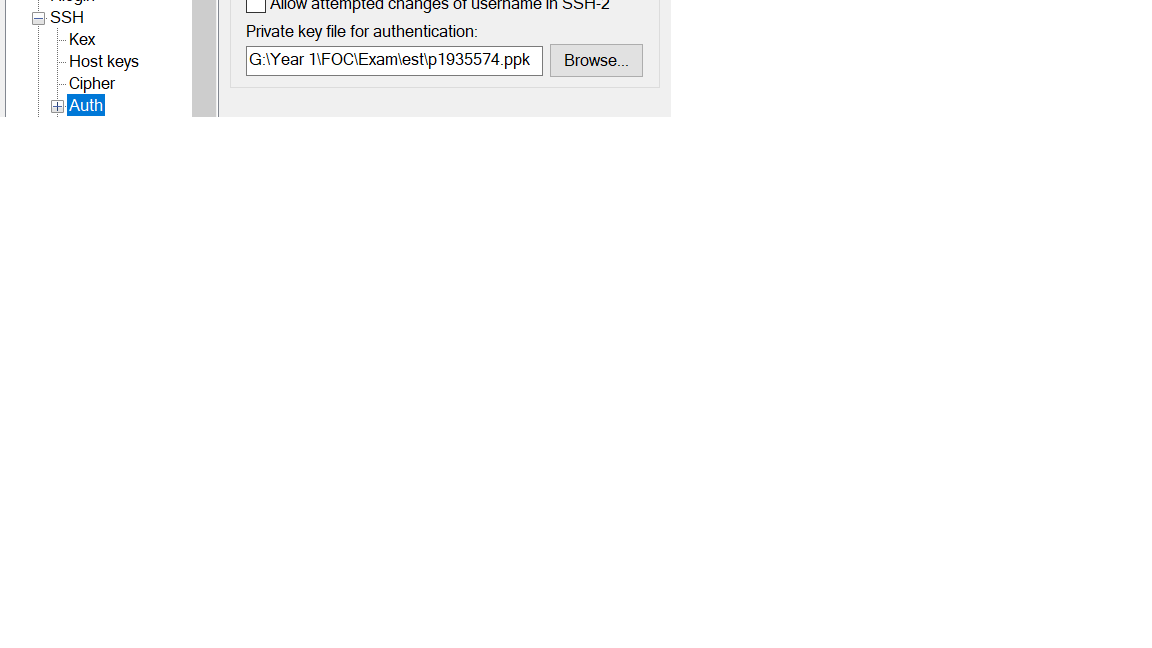
Note: By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.



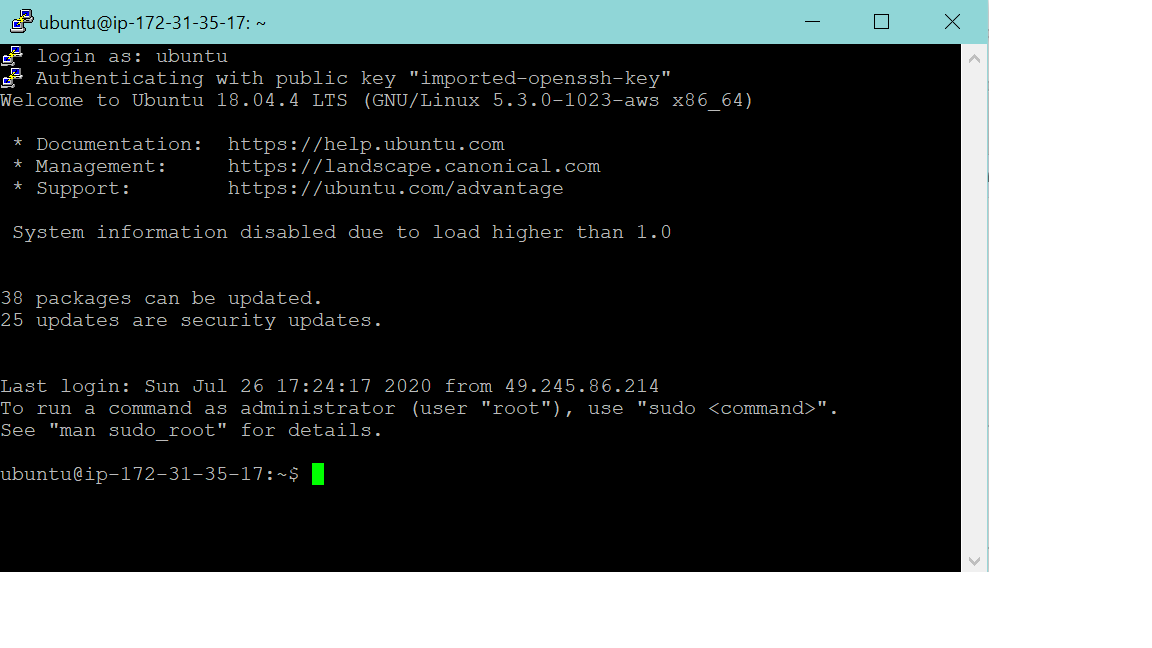
1. Select your .pem file for the key pair that you specified when you launched your instance and choose Open. Choose OK.
2. It will generate the private key. Save private key to a file by selecting “Save private key”, name it as “WPFOC.ppk”
3. Now, open Putty, Select the “Connection” -->SSH -->Auth and then load the .ppk file.



1. Fill the Hostname field with <your public dns>. Also you can save the session by giving it a name and clicking on “Save”.



1. Click on Open, and “yes” to login as “ubuntu” to the server. You are logged in now.

****

**Troubleshoot**

**- How to make putty ssh connection never to timeout when user is idle?**

If you go to your putty settings -> Connection and set the value of "Seconds between keepalives" to 30 seconds this should solve your problem.

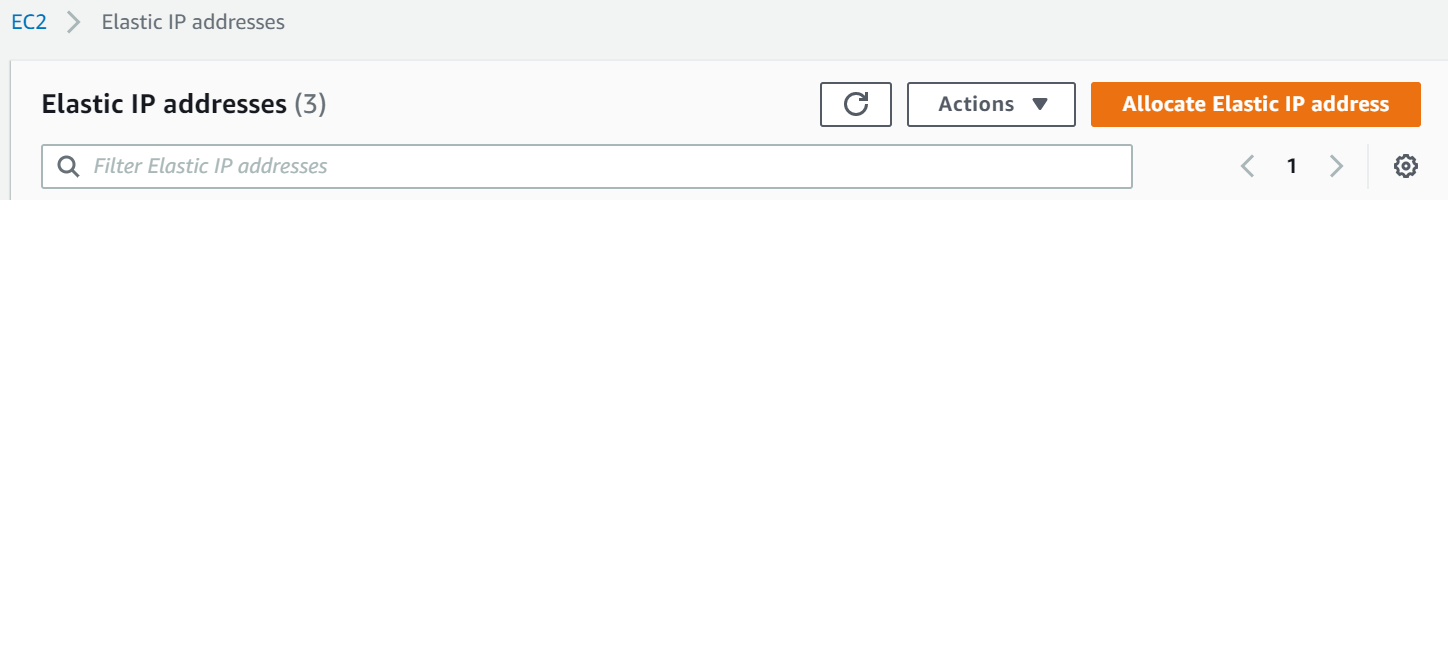
**- How could I get a fixed IP address associated with my instance?**

Get an Elastic IP from AWS. To allocate an Elastic IP address from Amazon's pool of public IPv4 addresses using the console:

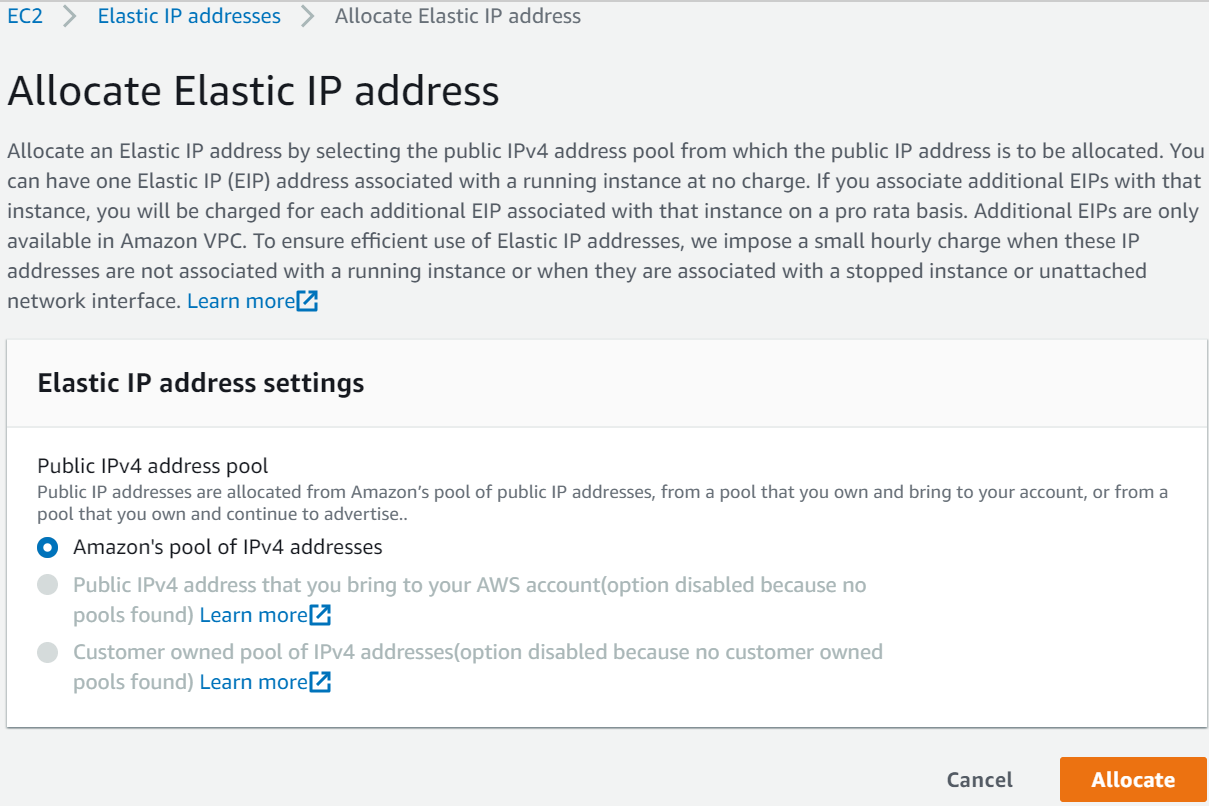
**1.** Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

**2.** In the navigation pane, choose Elastic IPs.

**3.** Choose Allocate new address.



**4.** For IPv4 address pool, choose Amazon pool.



**5.** Choose Allocate and close the confirmation screen.

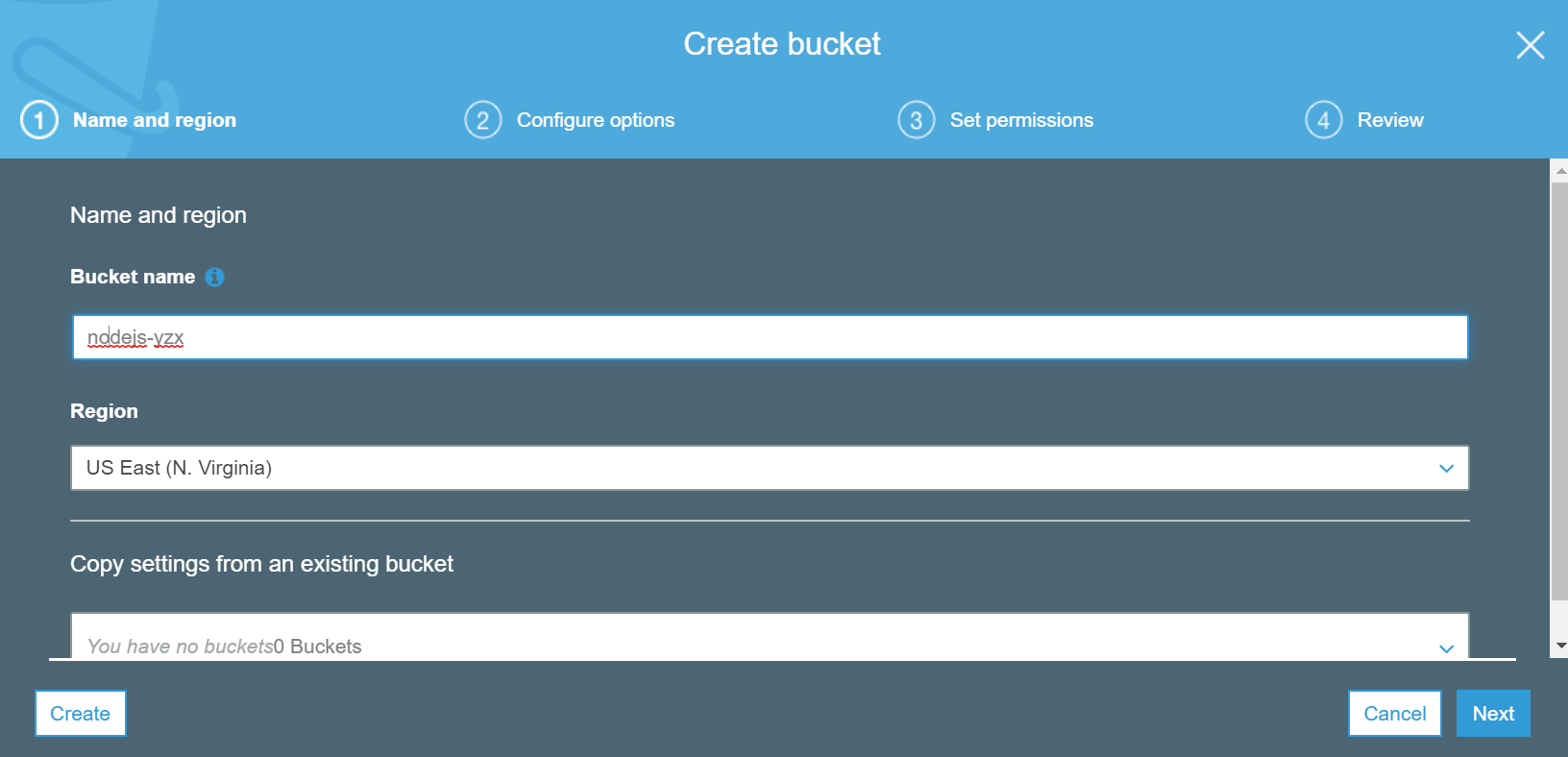
|  |
| --- |
| **Deploying a Node App on Amazon EC2** |

**Using Amazon S3 for your Code Repository**

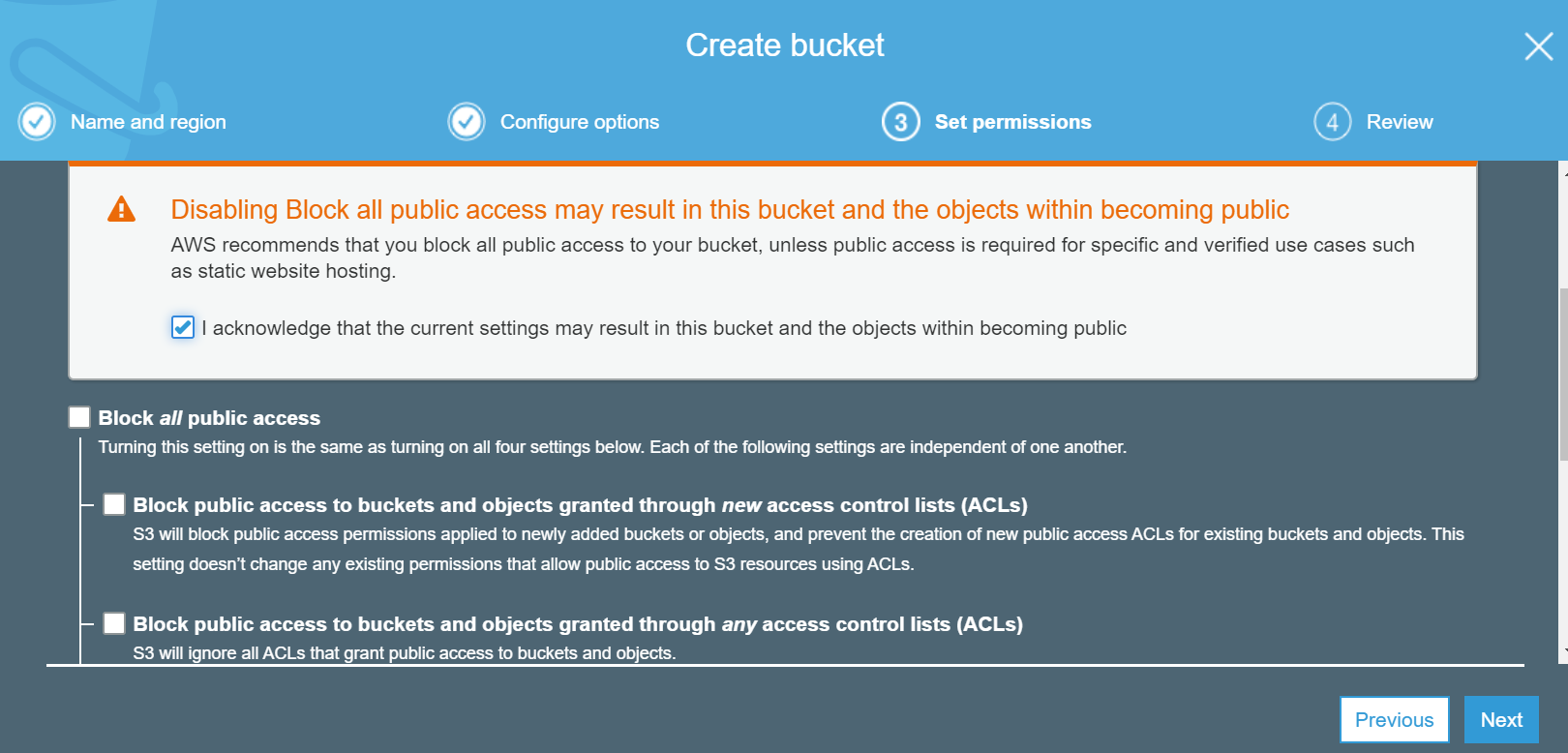
**(For Windows users)**

**1.** Right click on your node folder and select “Compressed (zipped) folder” option in “Send to”.

**2.** Login to your AWS console and select S3. Click on the Create bucket button, enter a unique name: nodejs-[your initials] for your bucket and click on the Next button.

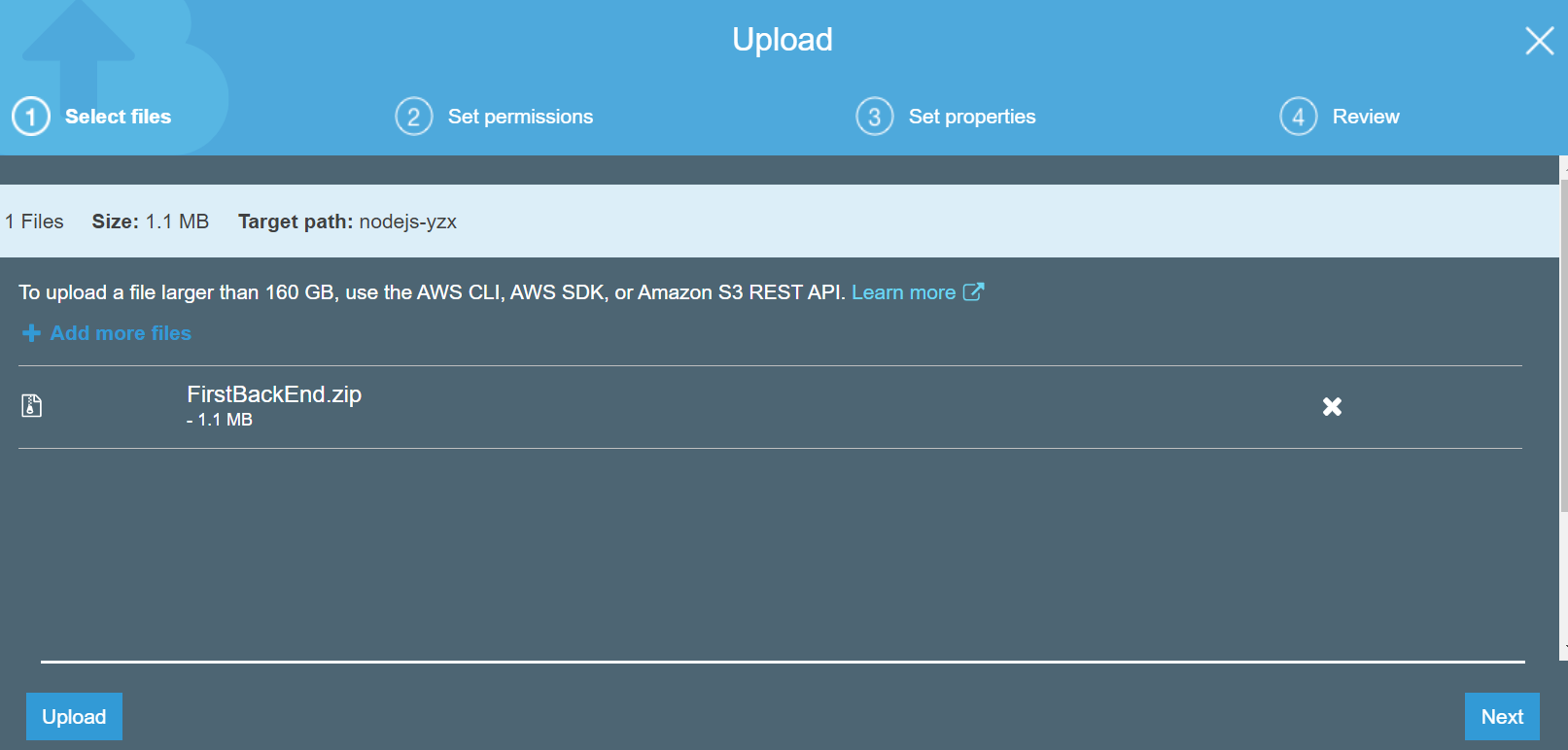


**3.** Click Next button again and uncheck all the checkboxes in Set Permissions tab.

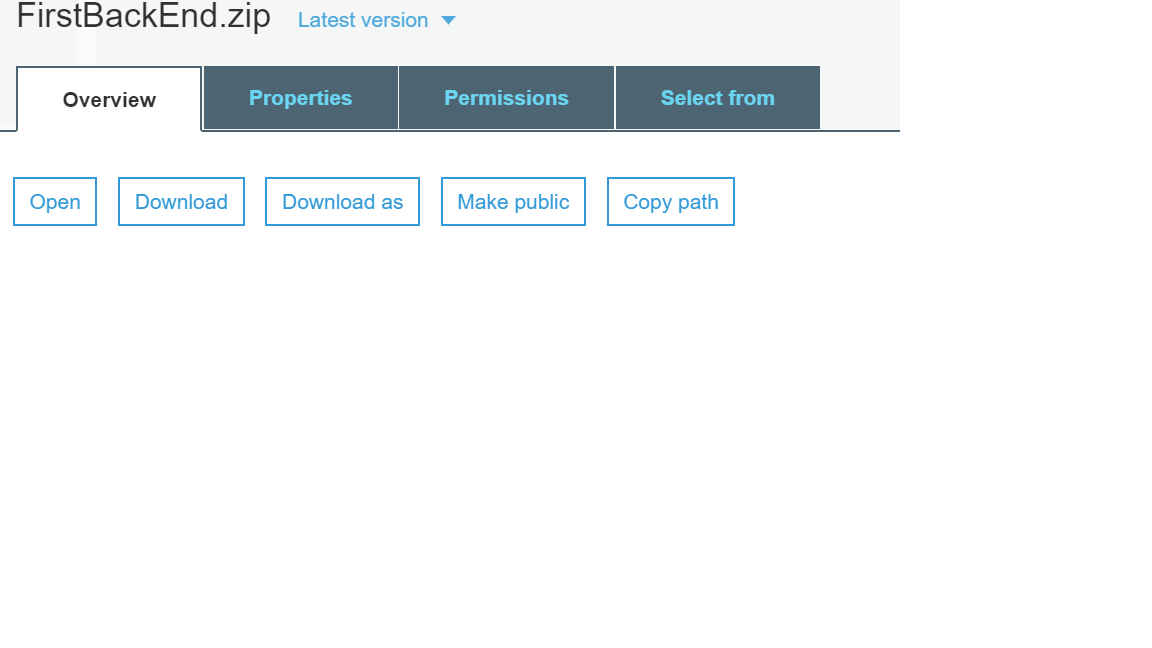
****

**4.** Click Next button and then click Create bucket button.

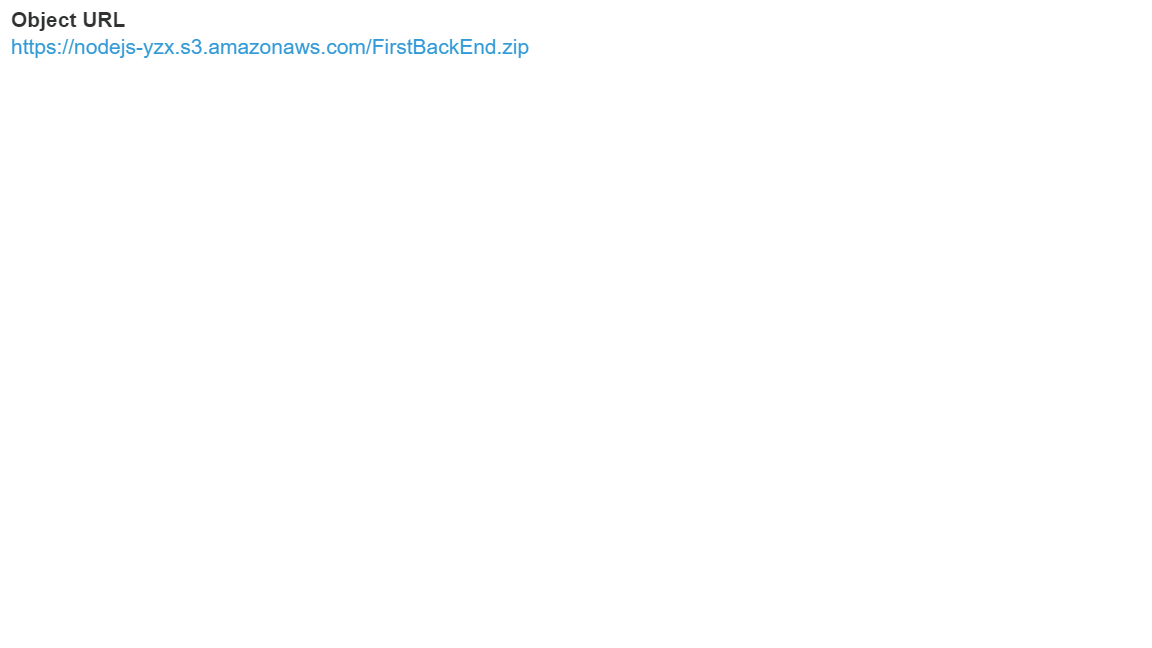
**5.** Select your newly created bucket and upload the project’s zip file to the bucket.



**6.** Click on the zip file and then click on the Make Public button.



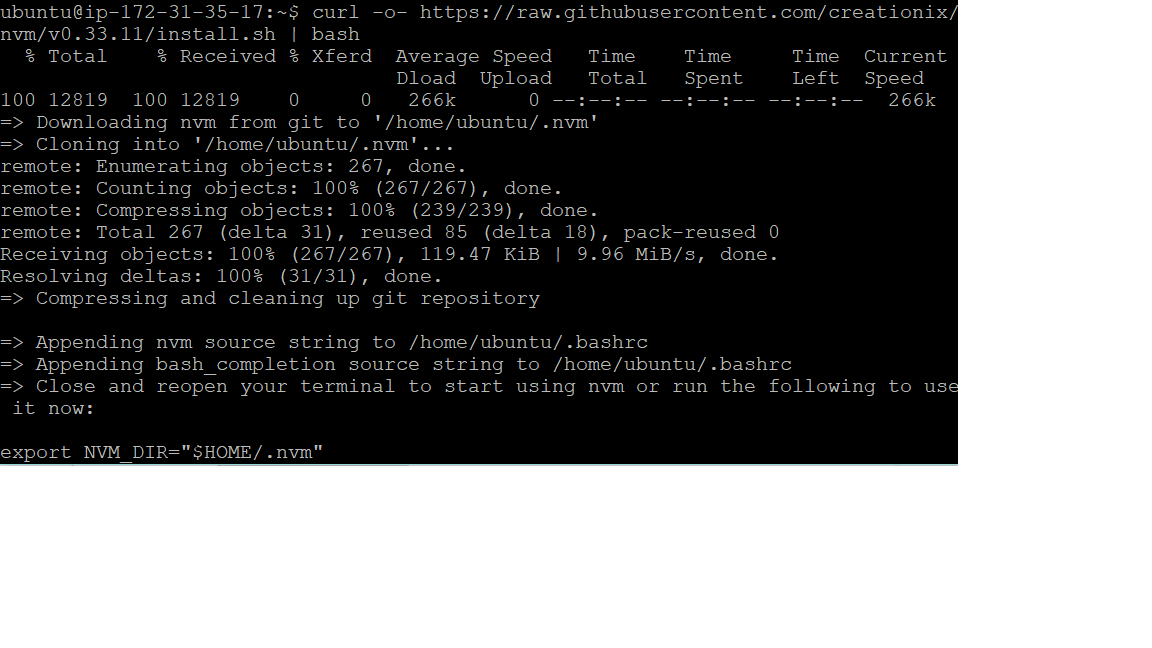
**7.** Record down the endpoint for your source code somewhere

****

****

**Install Node on EC2 Instance**

**1.** Install Node on the instance in your terminal or git bash: **curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash**

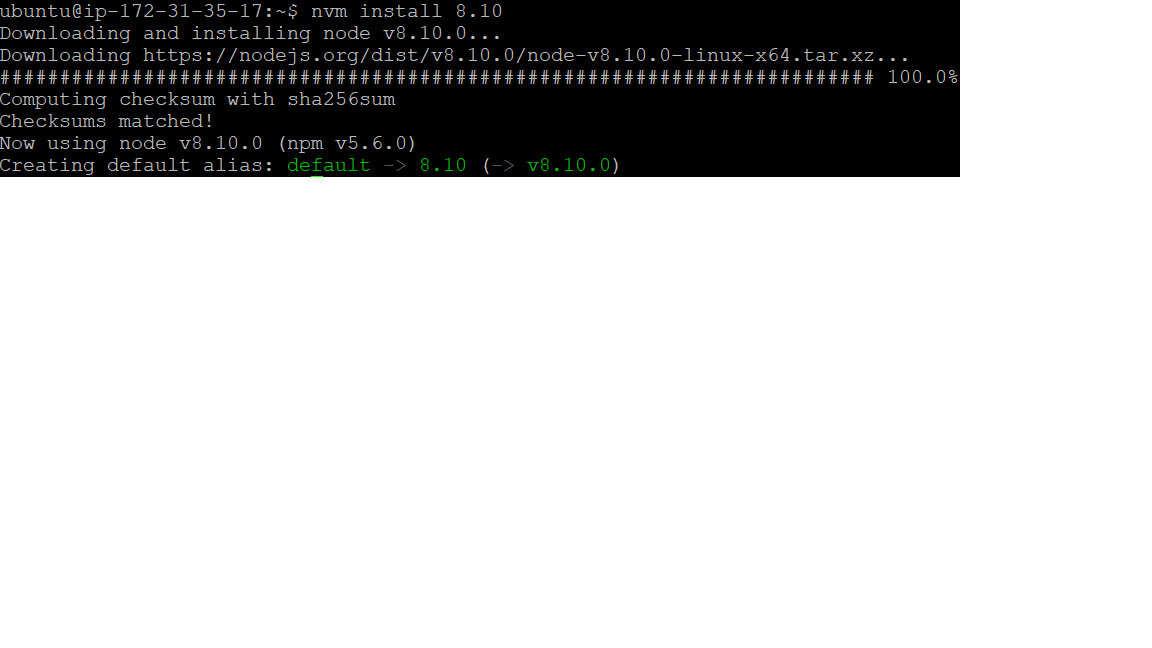


**2.** Close and reopen your terminal to start using nvm

**3.** Use nvm to install Node.js because nvm can install multiple versions of Node.js and allow you to switch between them: **. ~/.nvm/nvm.sh**



**4.** Installing the node version 8.10.: **nvm install 8.10**

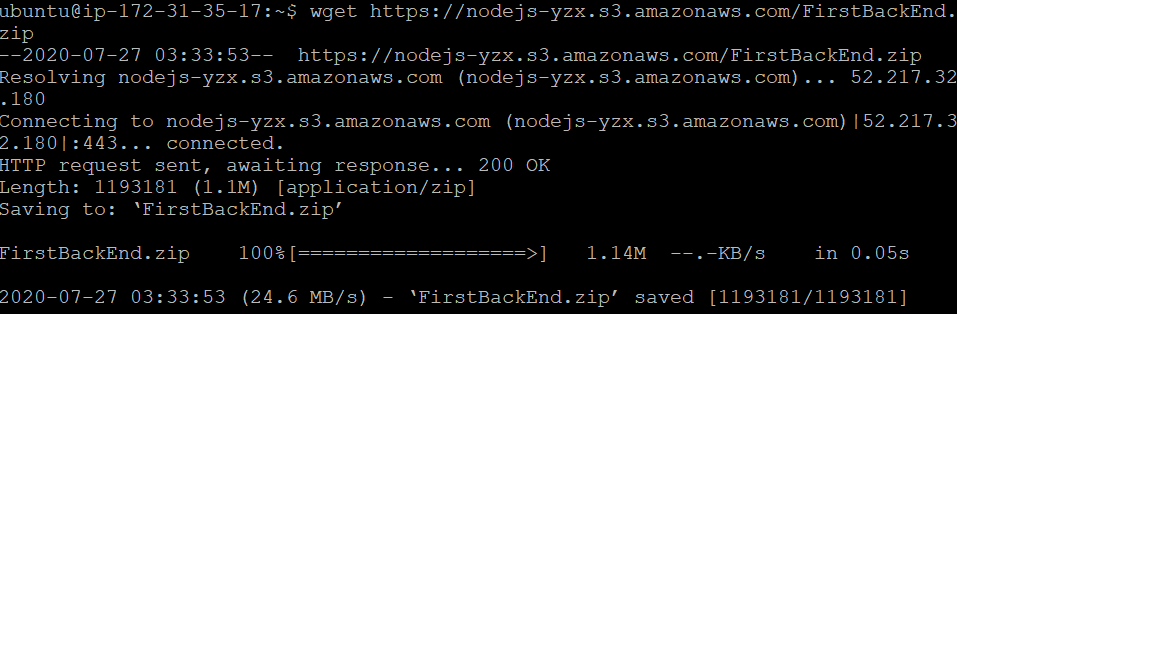
****

**5.** Check your node version: **node -v**

****

**Copy code on your EC2 instance and install dependencies to your home directory(/home/ubuntu)**

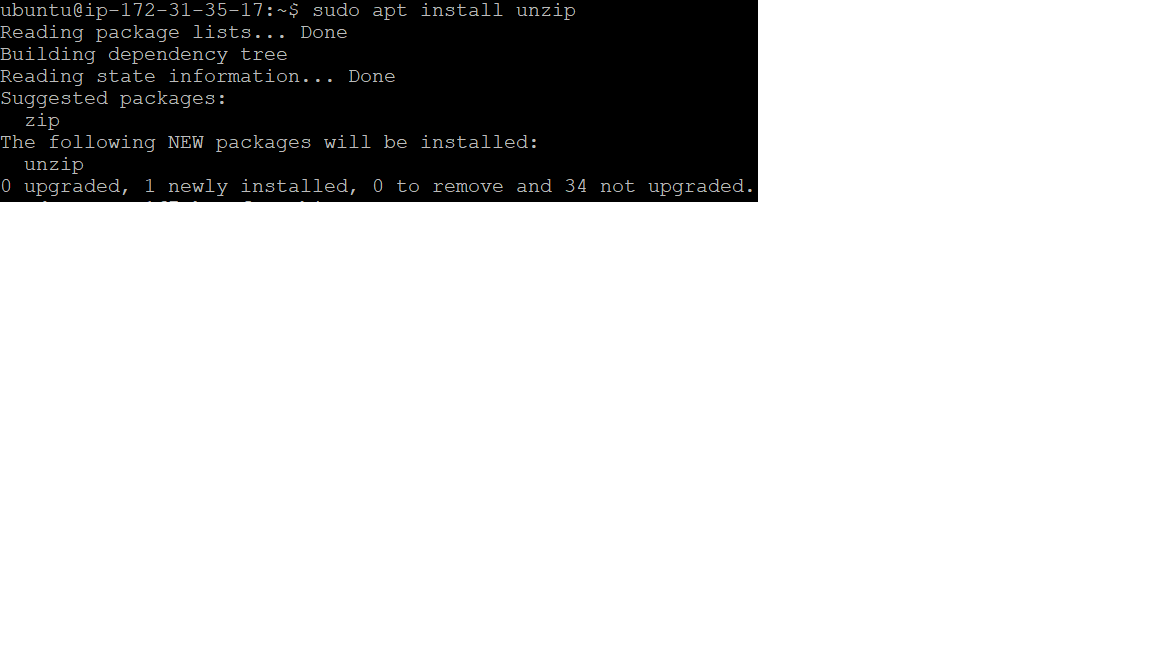
**1.** Get code from S3 repository that was previously noted down: **wget https://nodejs-yzx.s3.amazonaws.com/FirstBackEnd.zip**

****

**2.** Verify the node application is copied to your home directory: **ls**

****

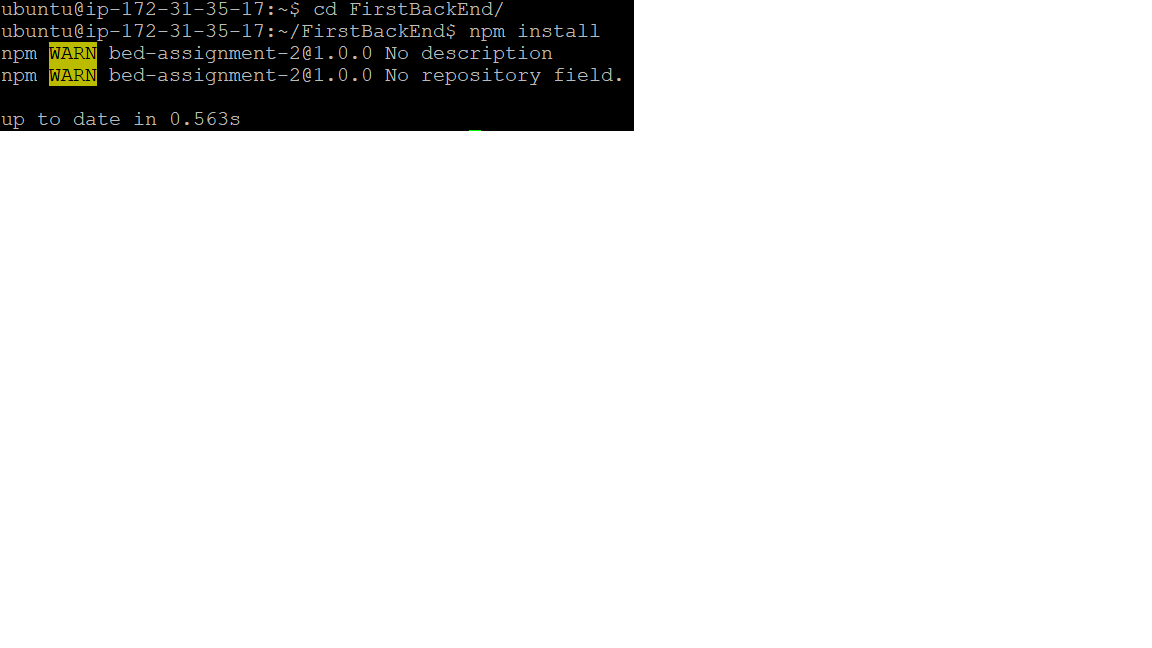
**3.** Install unzip: **sudo apt install unzip**

****

**4.** Decompress the zip file: **unzip FirstBackEnd.zip**

****

**5.** Install Dependencies: **cd FirstBackEnd** and **npm install**

****

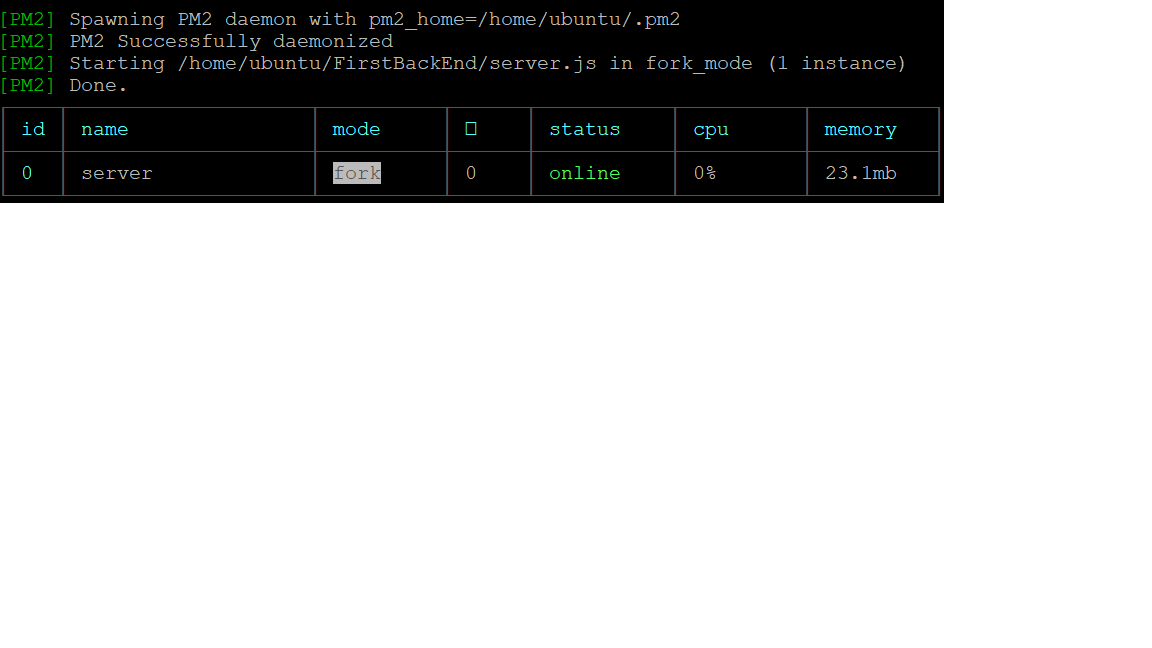
**Start server to run forever**

**1. npm install -g pm2**

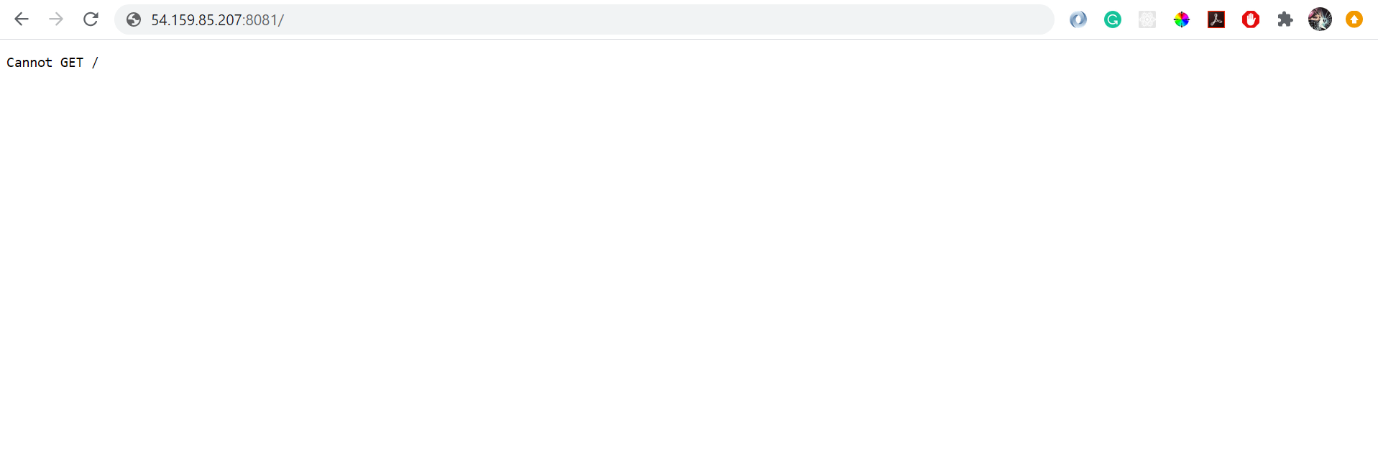
****

**2.** Go to the directory and run: **pm2 start server.js**

****

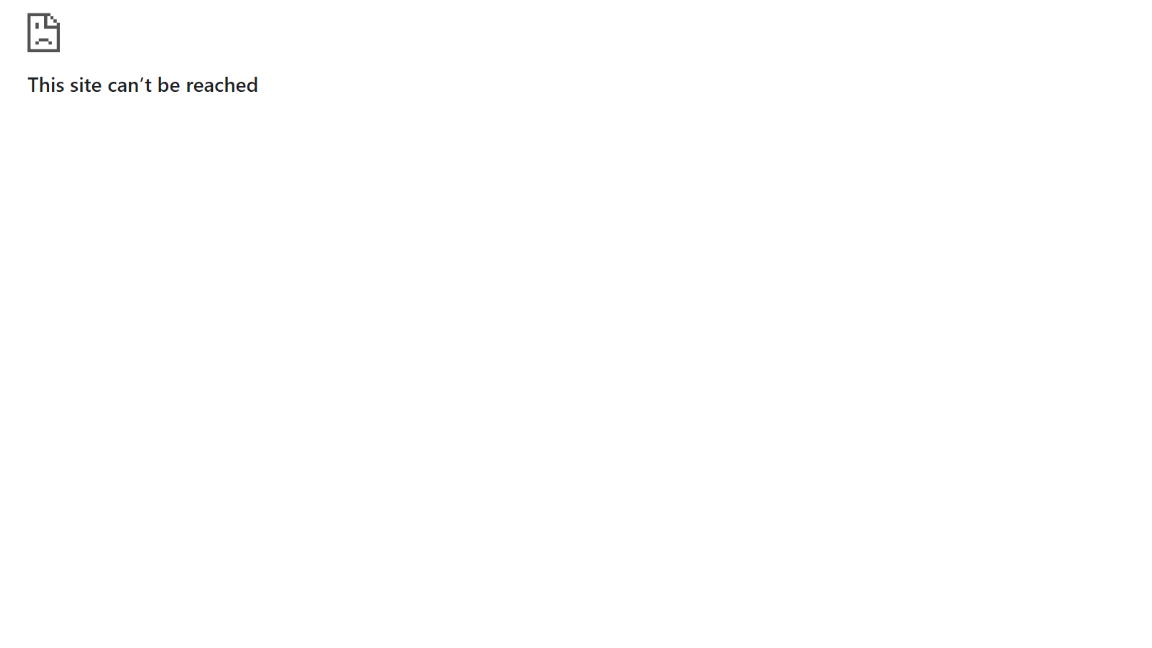
****

**3.** Open in browser: **http://[your public IP address or domain name]:8081**

****

**(if you get this page, mean it works!)**

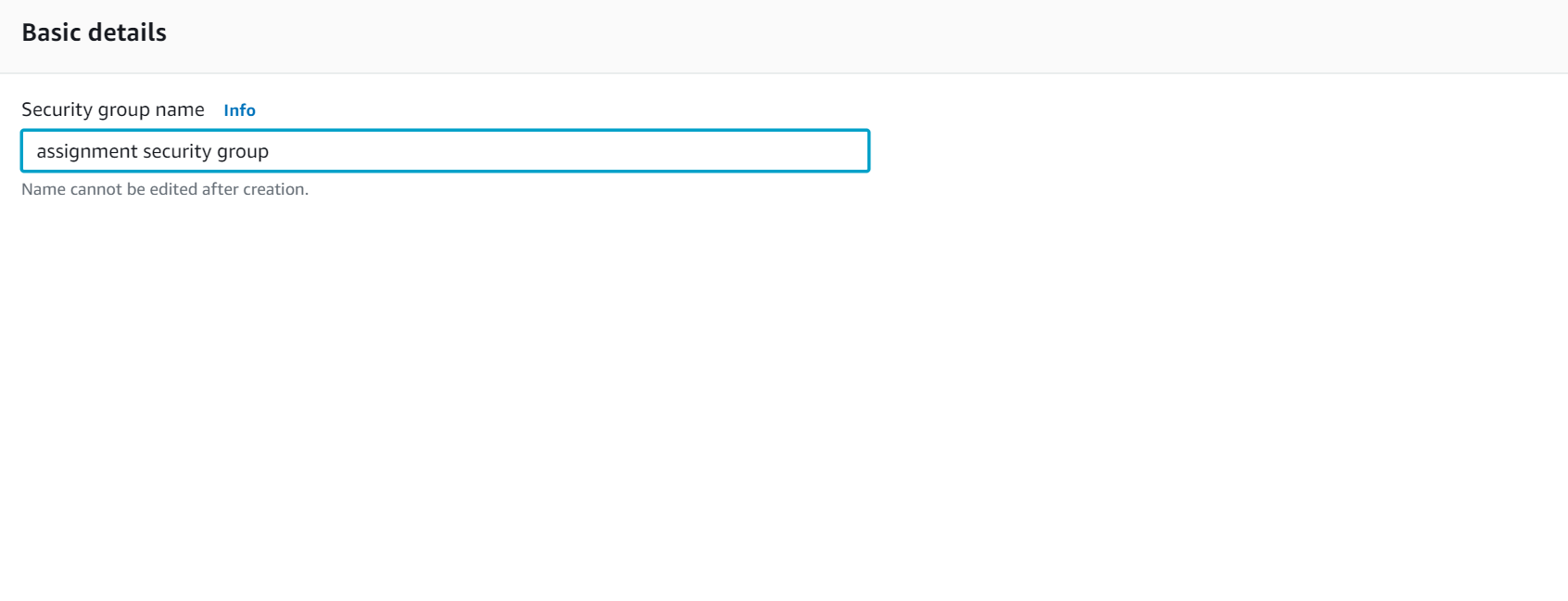
**Troubleshoot**

****

**- Configuring the security group to make sure you open the port on which your Node application will run**

**1.** Under ‘Network & Security’, go to ‘Security Groups’ and create security group

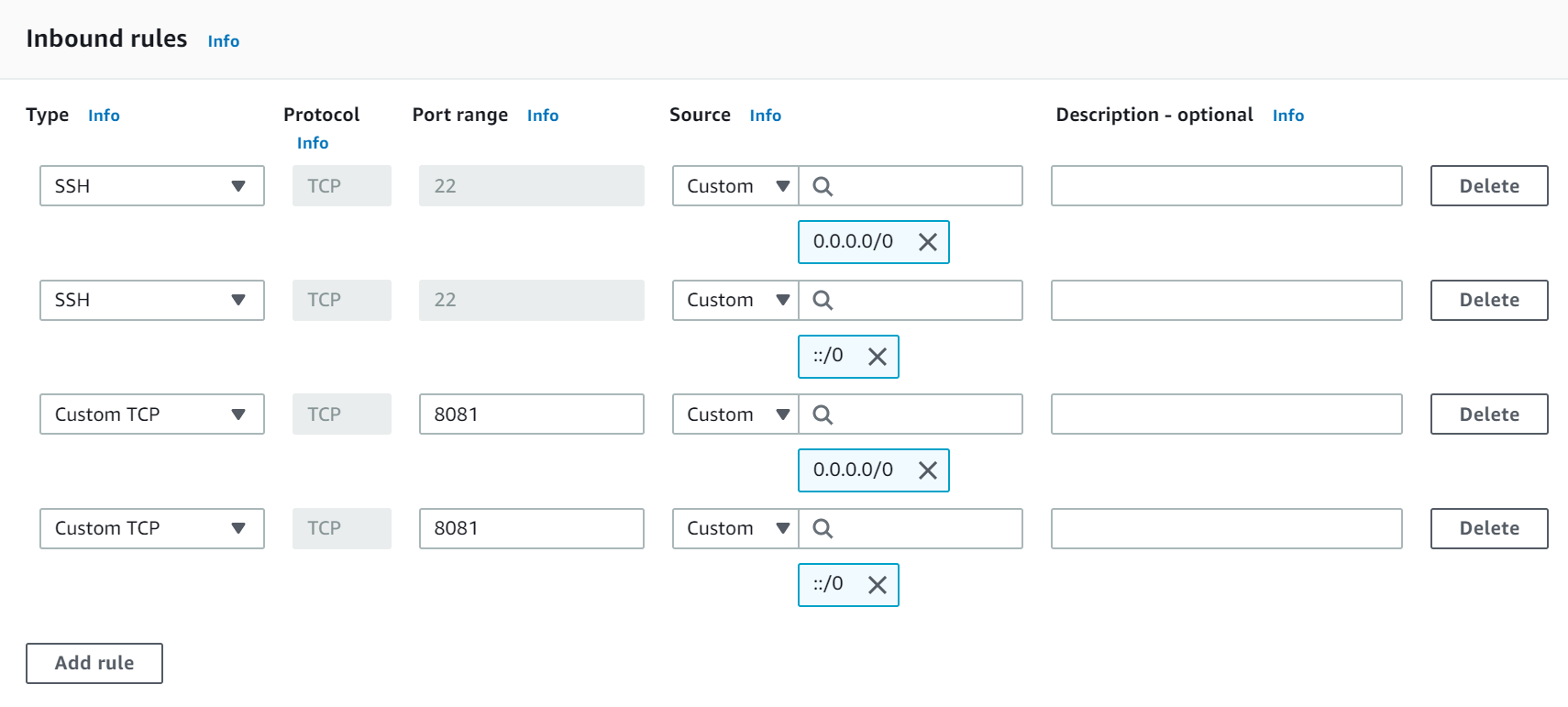
**2.** Add in security group name

****

**3.** Add in inbound rules:

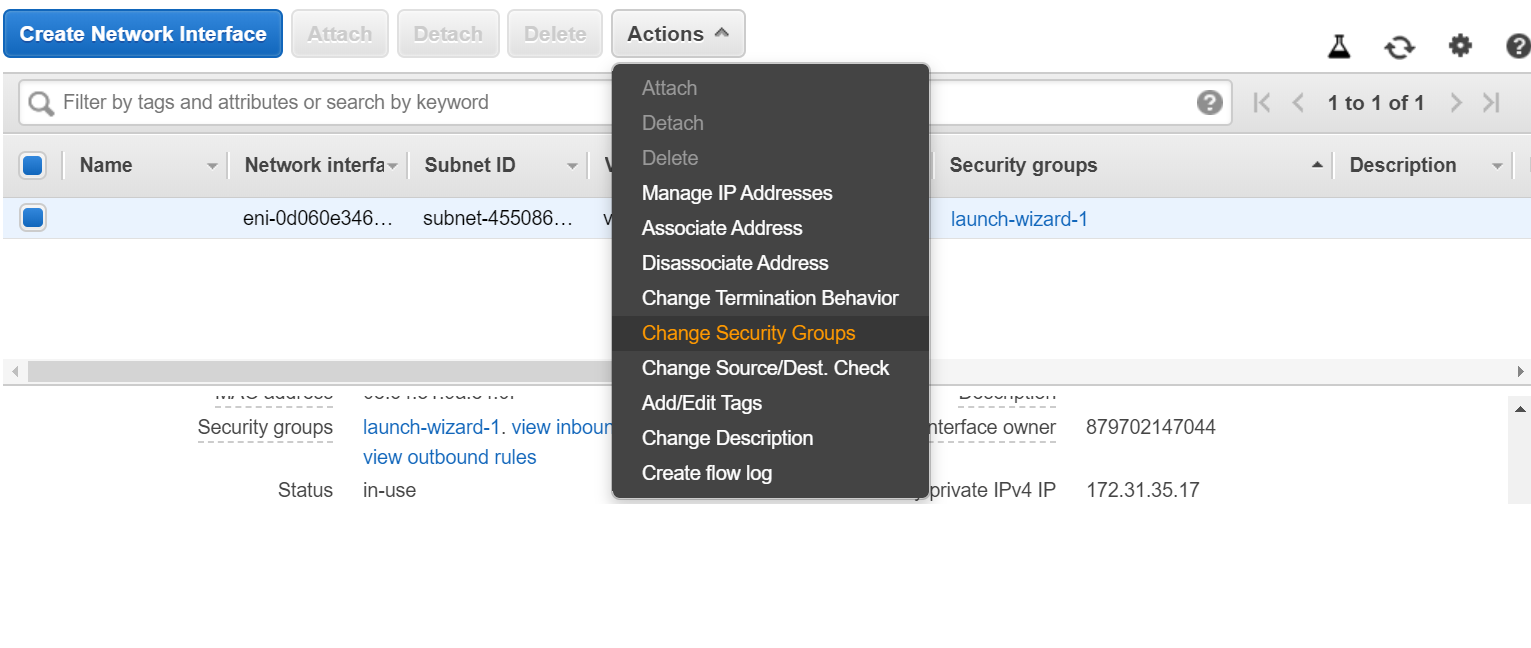
- Custom TCP for port **8081** and custom source **0.0.0.0/0, ::/0**

- SSH for custom source **0.0.0.0/0, ::/0**

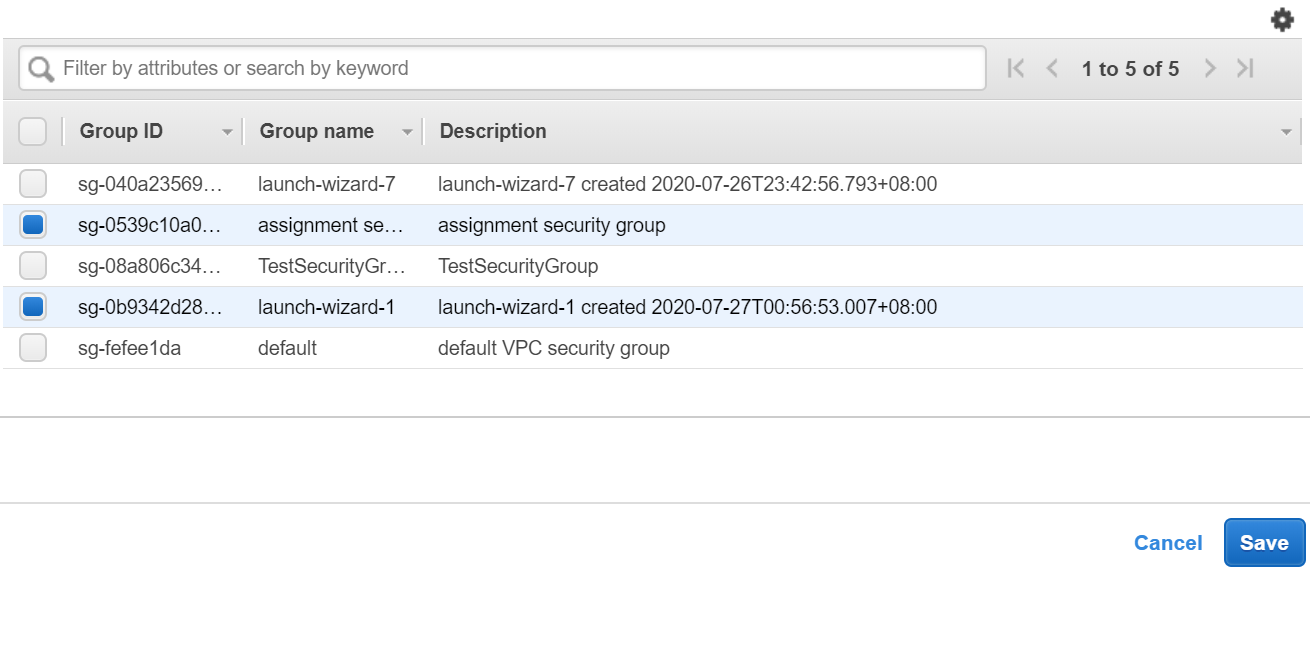


**4.** Under ‘Network & Security’, go to ‘Network Interfaces’

**5.** Select the instance you want and under action, select ‘Change Security Group’



**6.** Check the newly created security group, in this case, ‘assignment security group’.

****

**Configure pm2**

Run: **pm2 startup**

-To freeze a process list via reboot: **pm2 save**

-To restart server: **pm2 restart server**

-To reload server: **$ pm2 reload server**

-To stop server: **$ pm2 stop server**

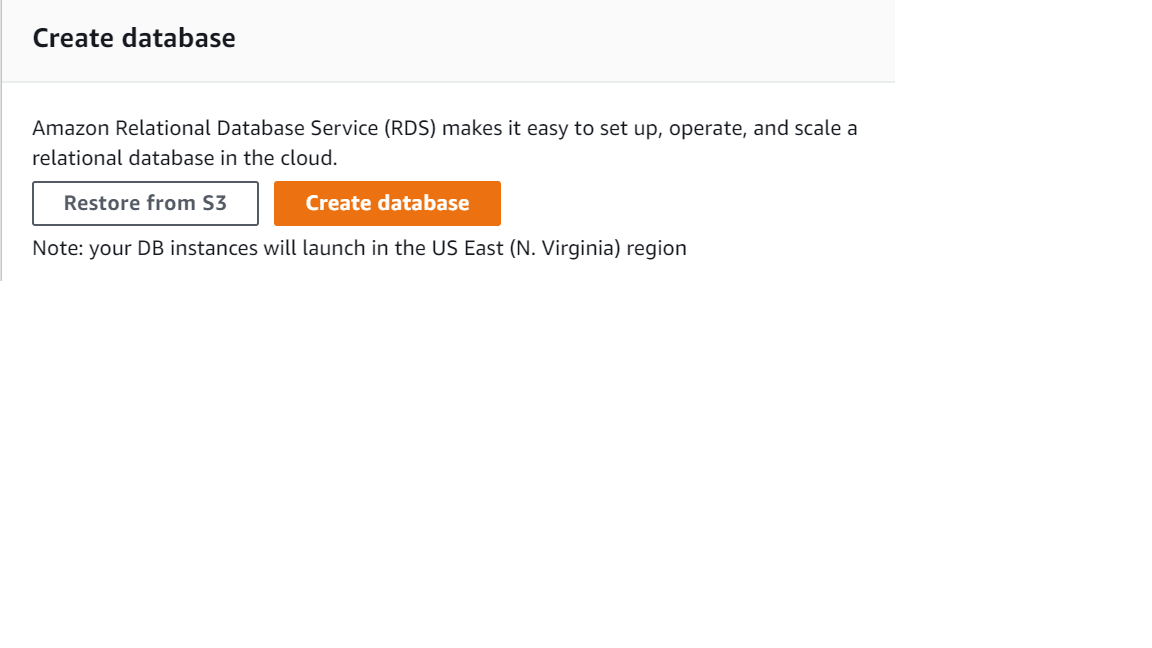
-To delete server: **$ pm2 delete server**

|  |
| --- |
| **Create and Connect to a MySQL Database with Amazon RDS** |

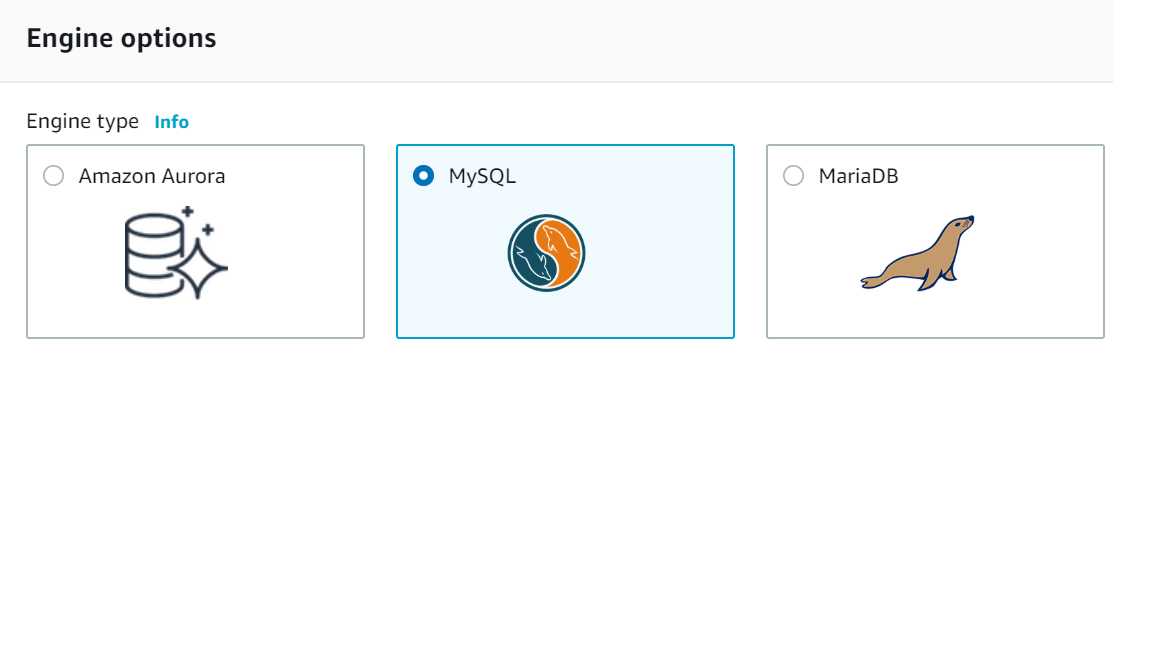
**Created a new Amazon Web Service RDS instance with Amazon RDS**

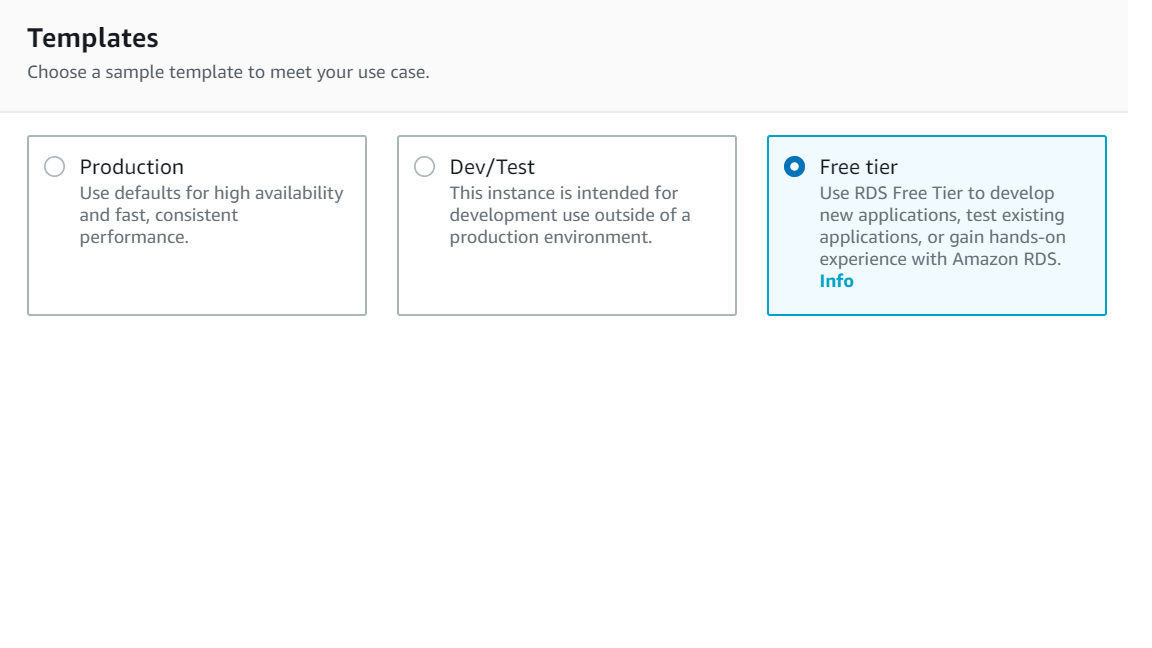
**1.** Under services, click on RDS in Database

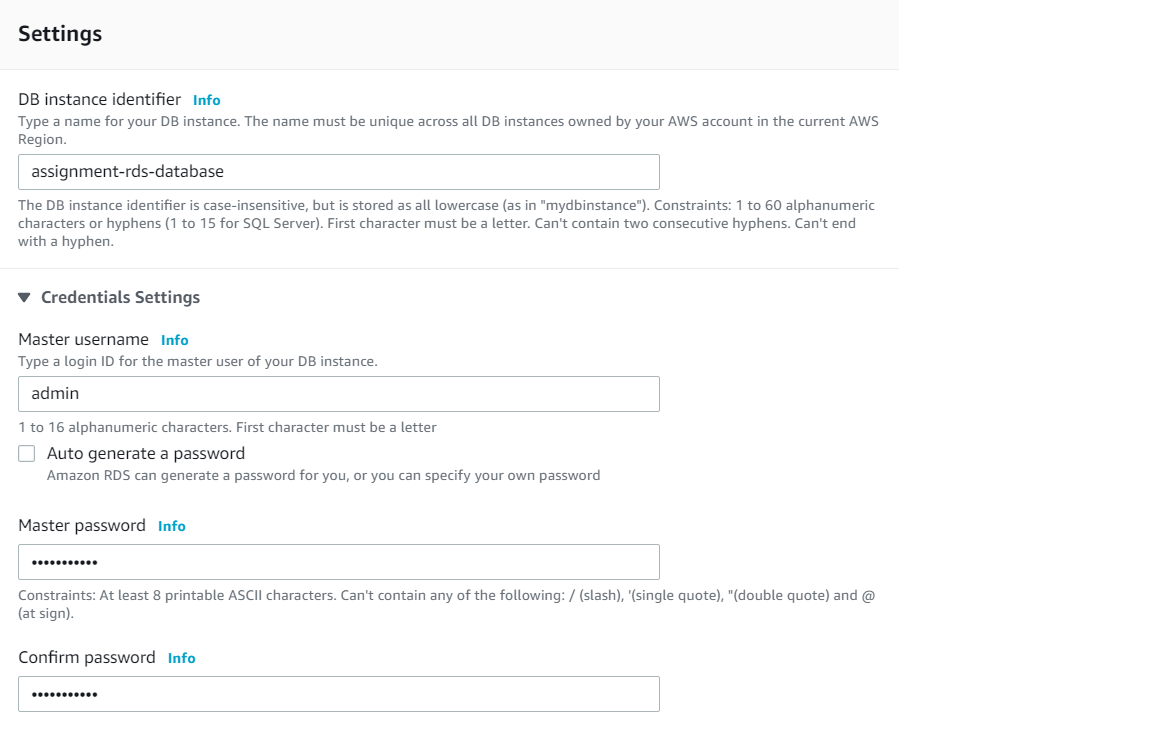
**2.** Click on ‘Create database’

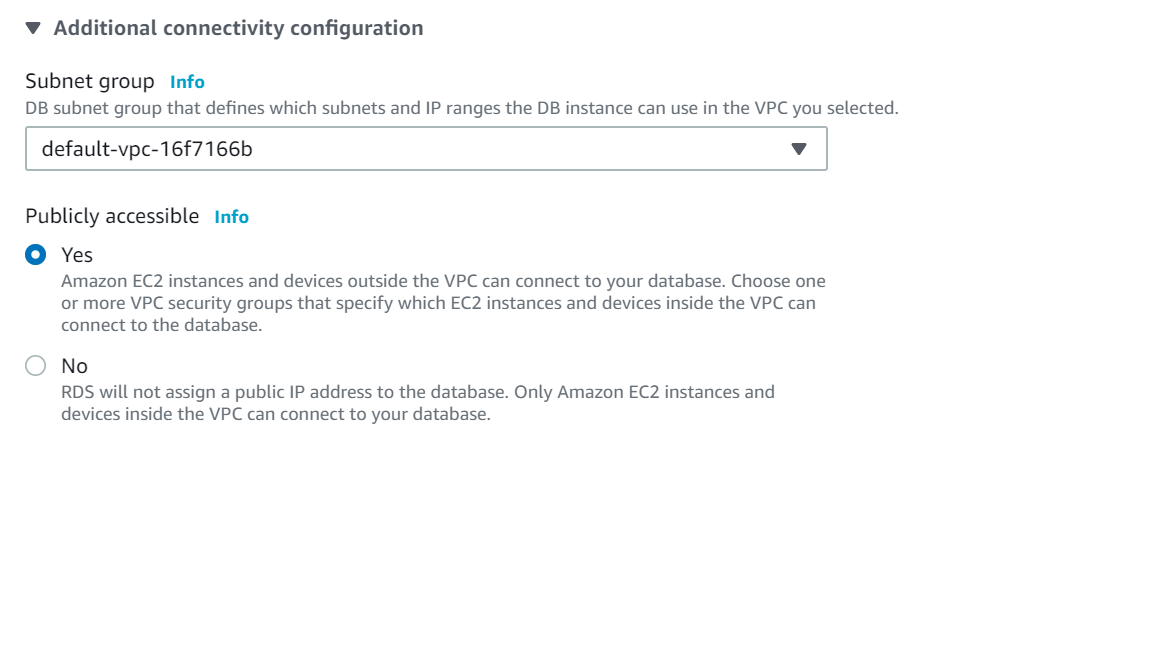


**3.** Keep everything as default, except for the following screenshots:

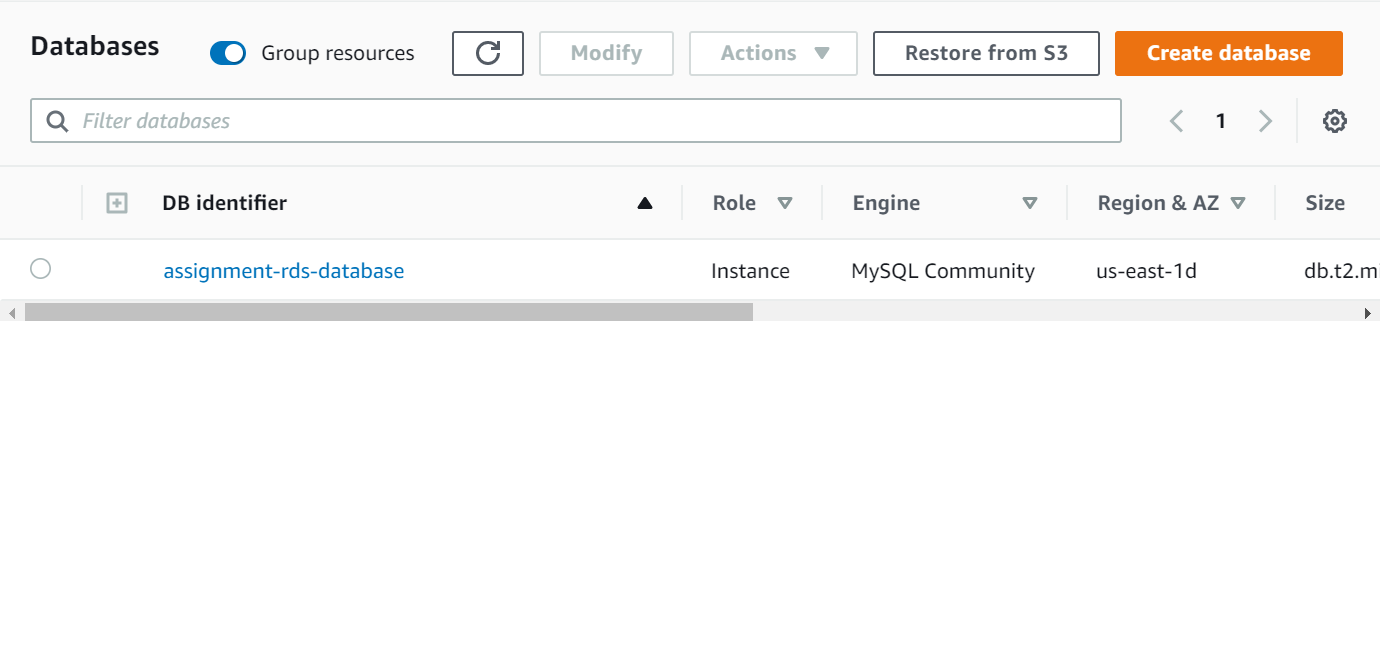
****

****

****

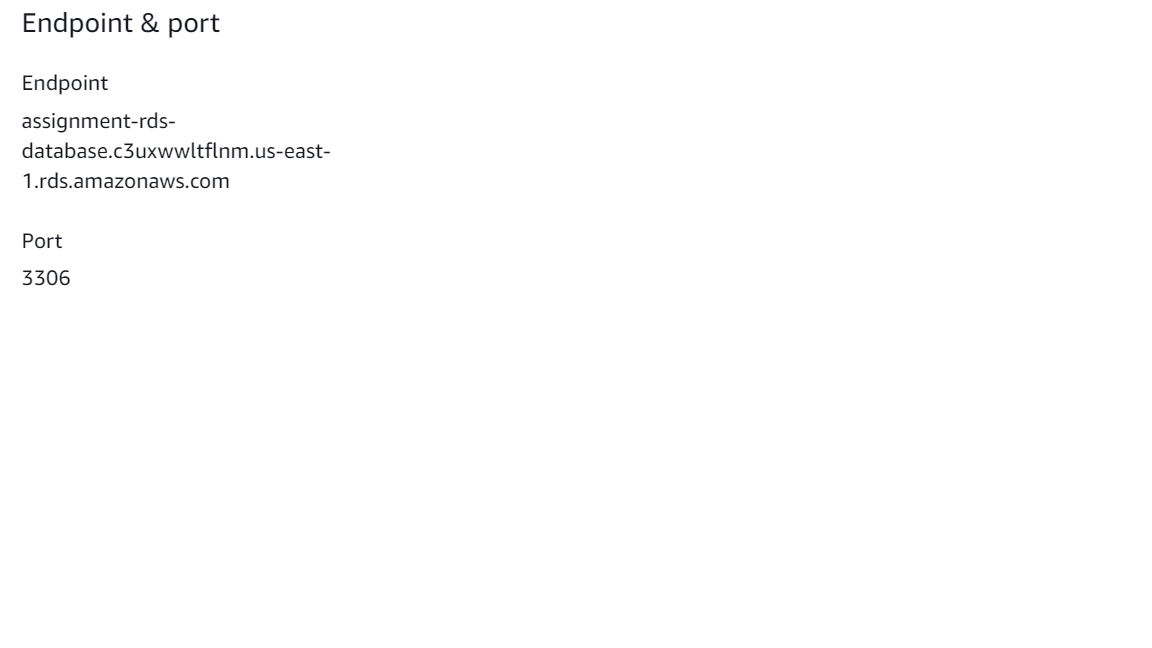
****

**4.** Click on ‘Create database’ button, you should see your database being created



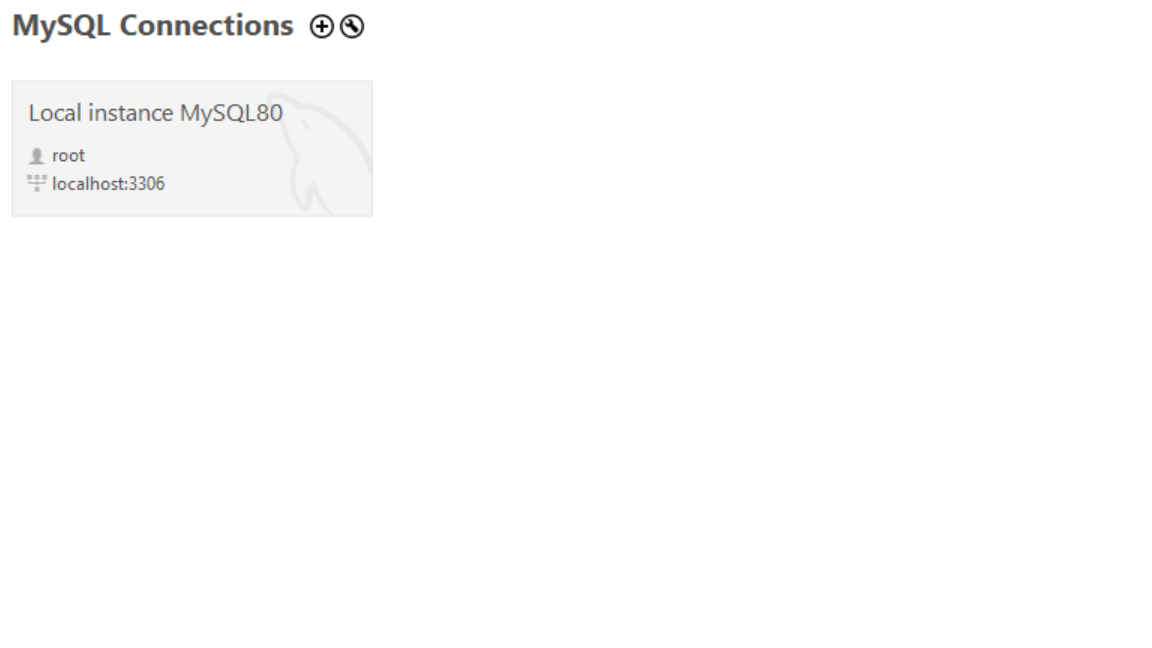
**5.** View information about the database by clicking on the database name

**6.** Under Connectivity & security, note down the endpoint and port number

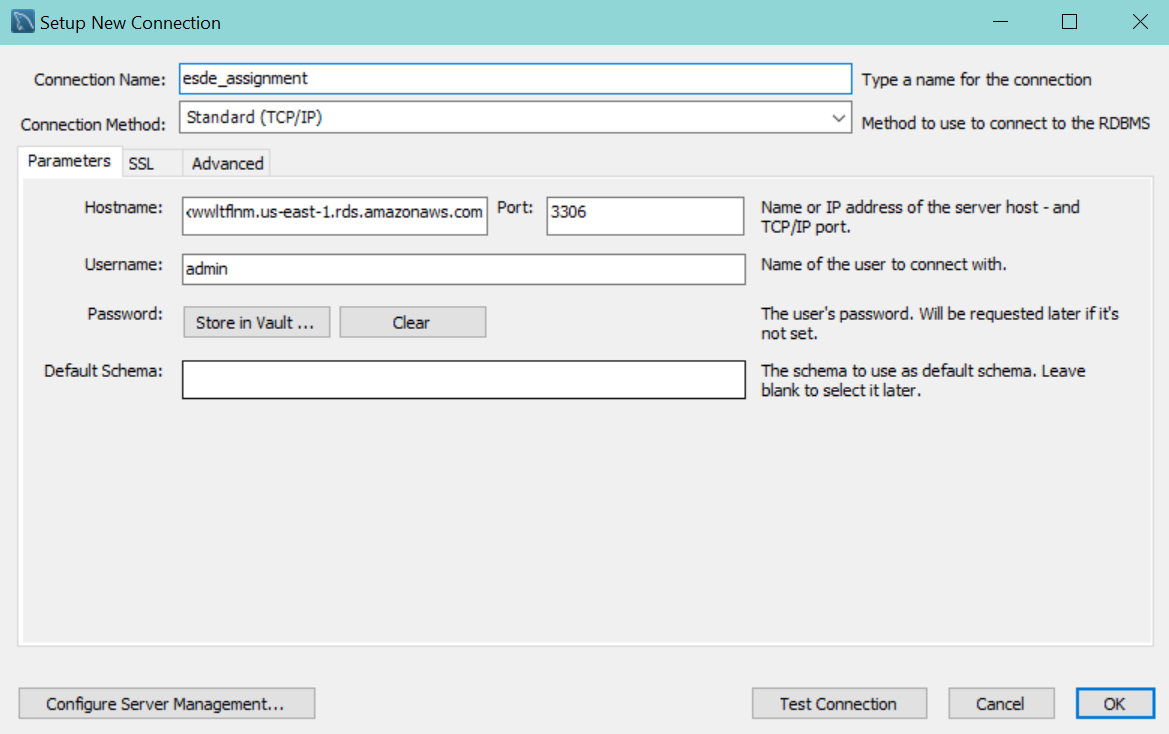


**Connect and send SQL queries remotely to an Amazon Web Service RDS instance using MySQL Workbench**

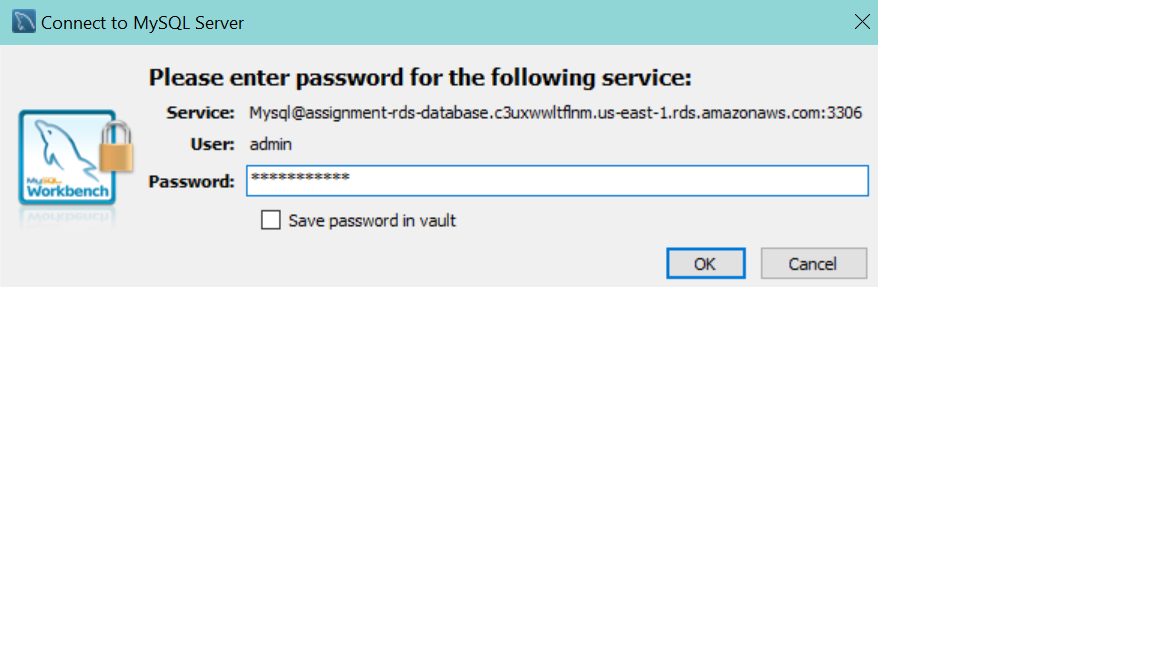
**1.** Launch MySQL Workbench and locate the '+' icon to add a new connection

****

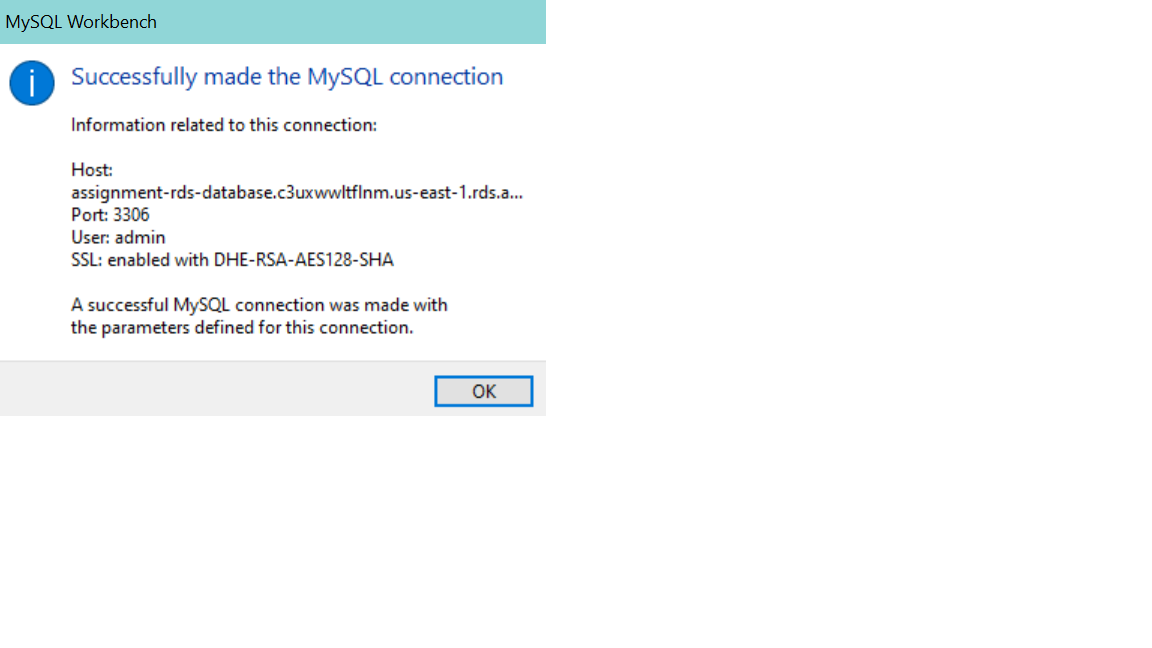
**2.** Fill up the form with the RDS database information that you had recorded down (**endpoint, port and username**)

****

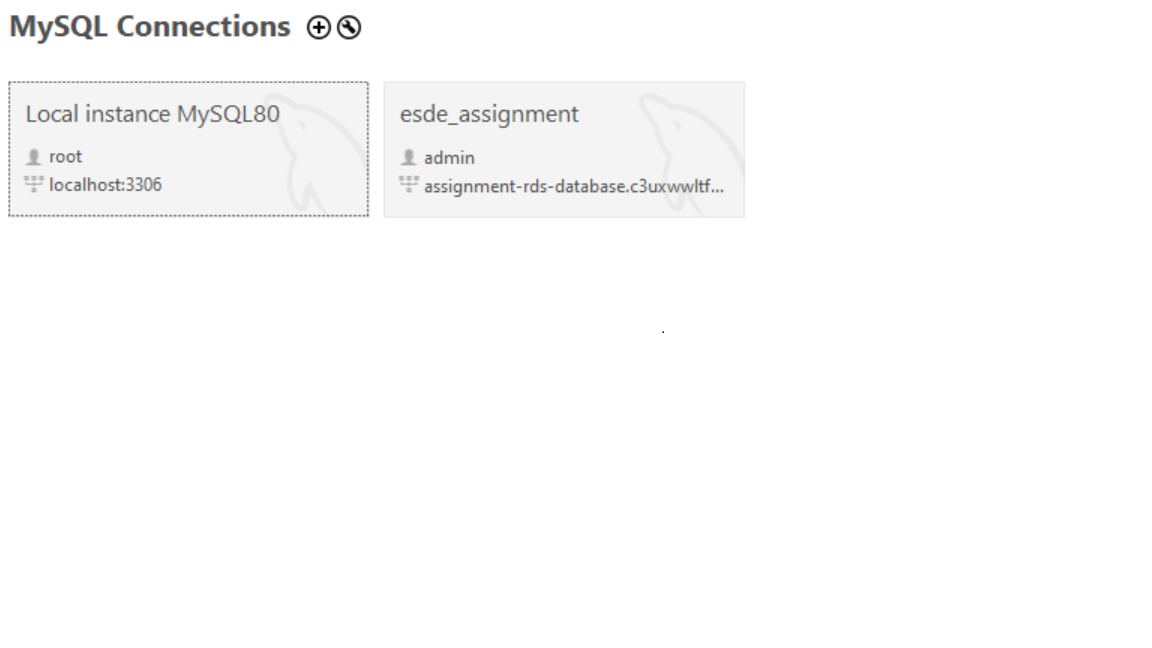
**3.** Click on Test connection, you should be prompted a dialog asking for your MySQL password

****

**4.** Submit your password and you should finally have a notice, stating that the connection is working fine



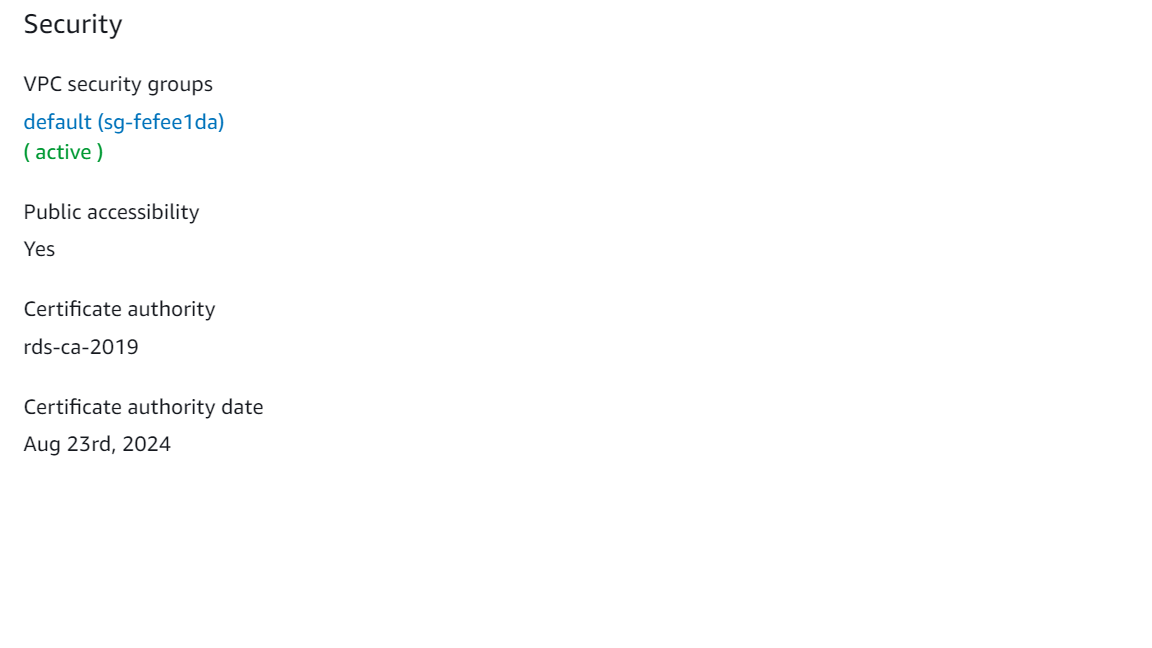
**5.** You should be able to see a new MySQL connection being created



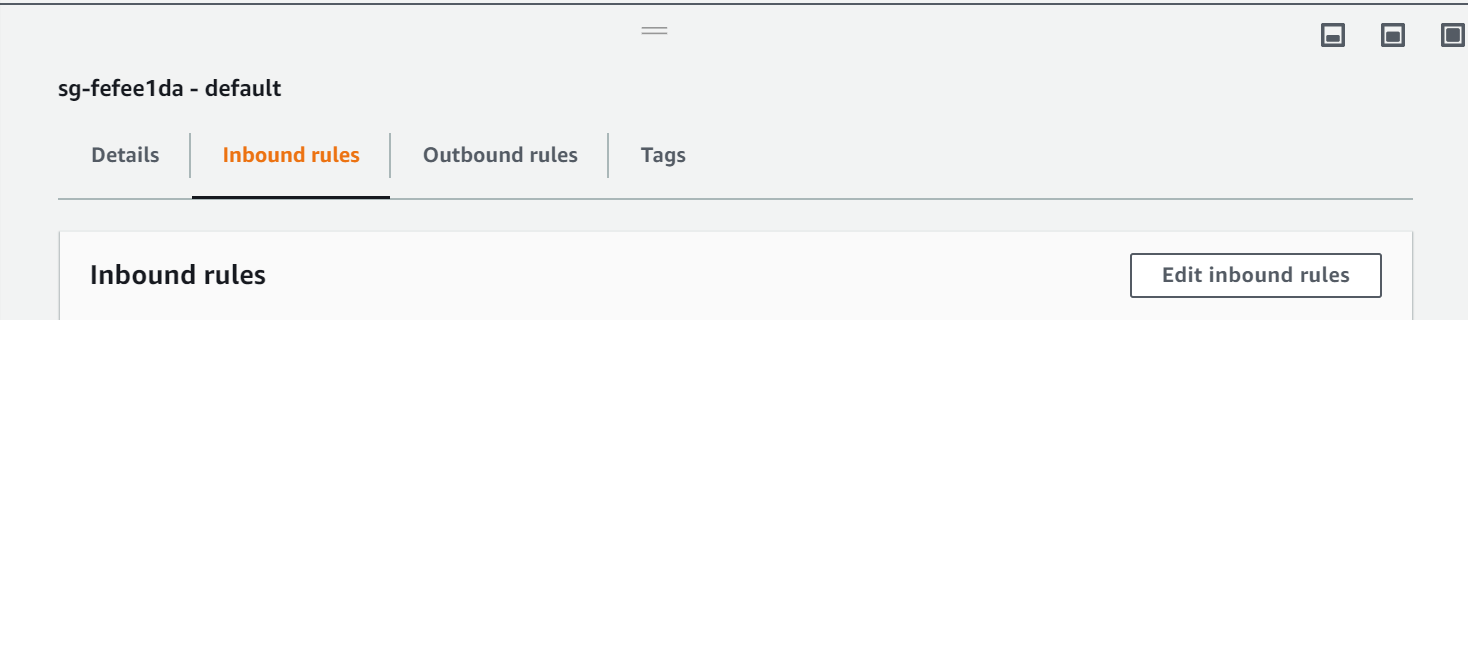
**Troubleshoot**

**- If you got a connection error, check the security group for the database if it allows your application server to connect.**

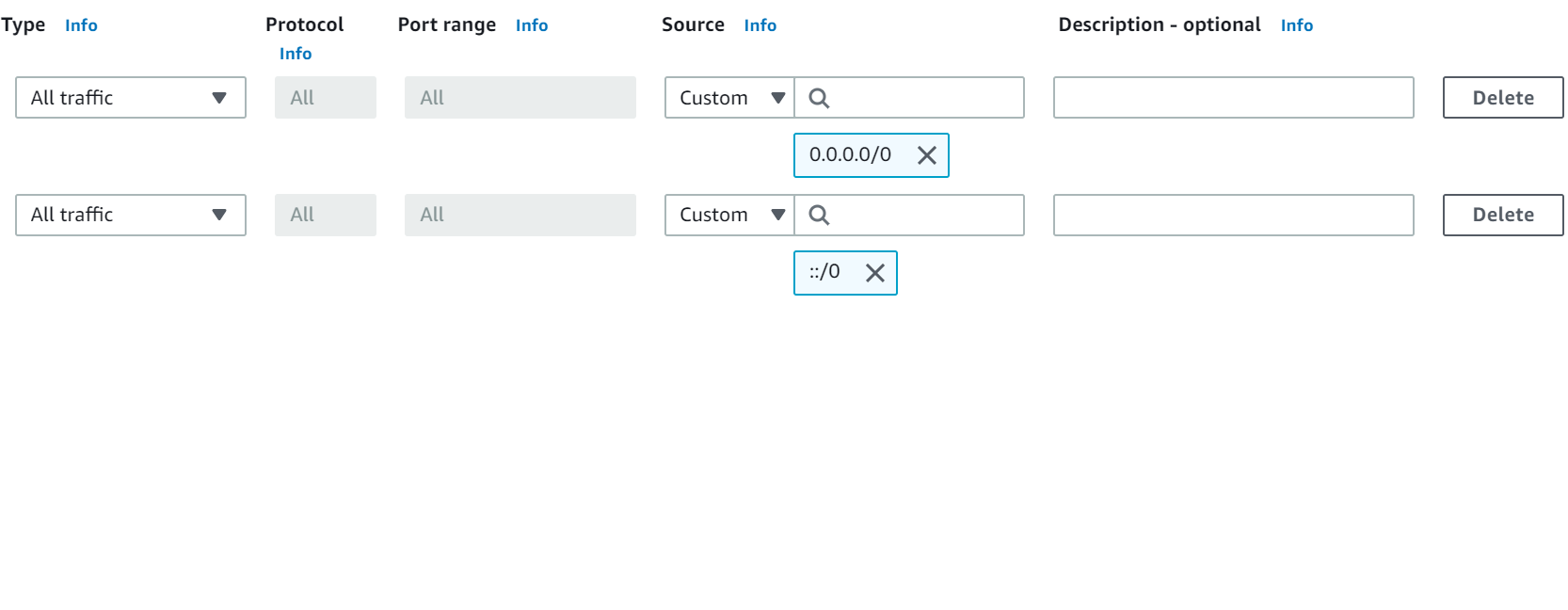
**1.** Under Connectivity & security, click on the VPC security group link



**2.** Click on inbound rules, edit inbound rules



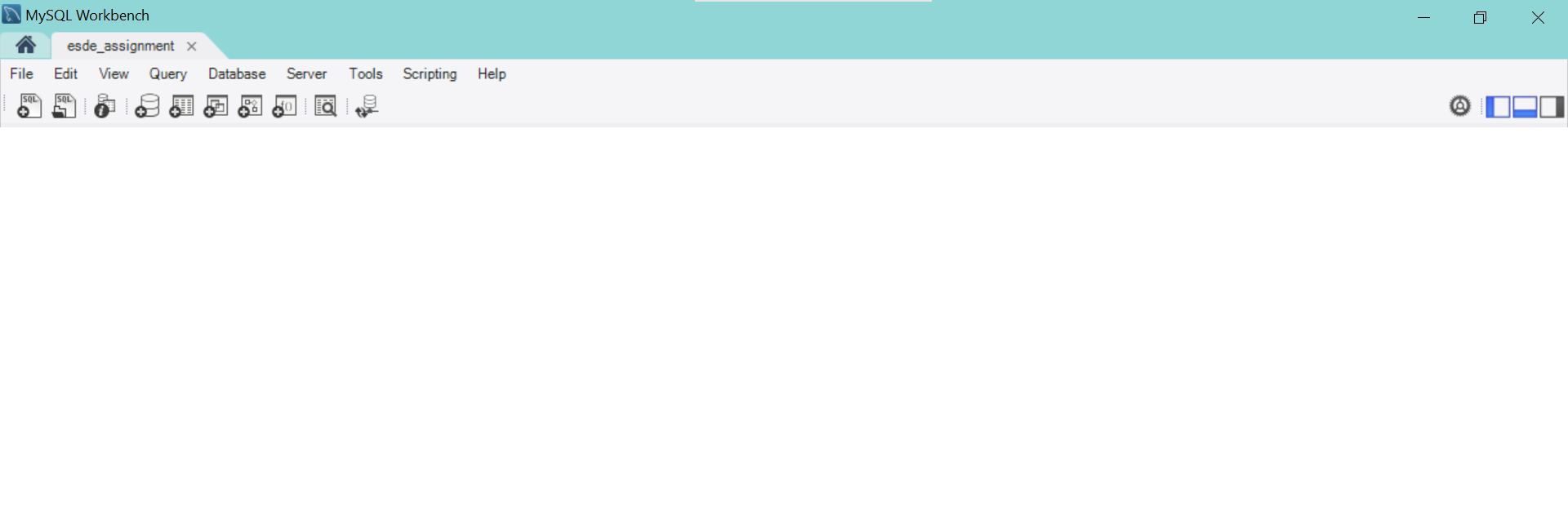
**3.** Add rules: **All traffic**, custom source **0.0.0.0/0,::/0** and save rules.



**4.** Try to connect MySQL with AWS RDS again. The connection should be working now.

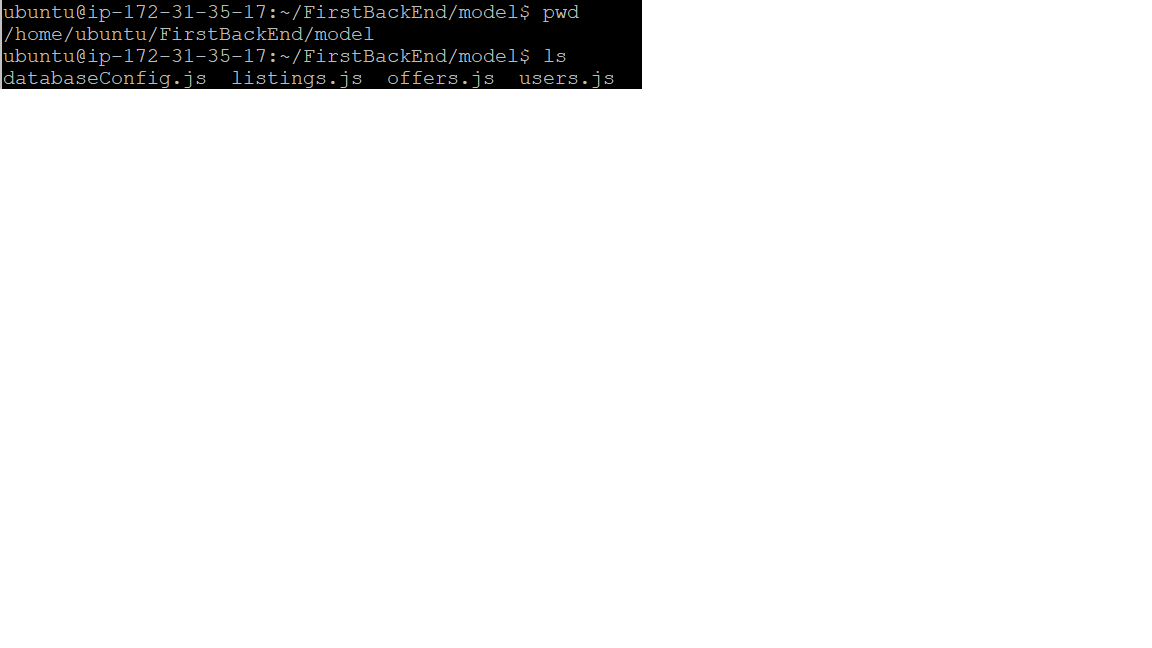
**Load SQL text file into the new connection and configure database connection in EC2**

**1.** Click on the icon to open a SQL script file



**2.** Load the script file (snapsell-v2) into the SQL workbench and execute it

**3.** After the script file has been loaded, open up the Putty terminal and cd into the directory with the databaseConfig.js



**4.** Edit databaseCongif.js by opening the file in the terminal: **vim databaseConfig.js**

****

**5.** Click on the character ‘I’ on the keyboard to start editing.

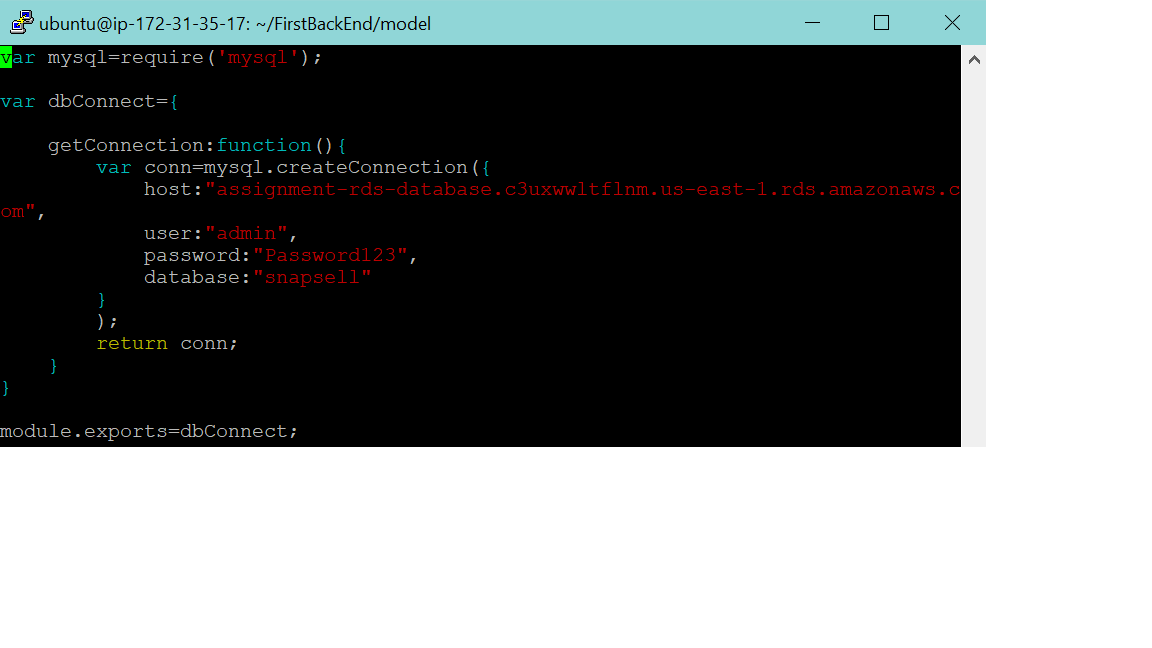
**6.** Edit the following:

- change host to the endpoint of the RDS

- change user to the username set in RDS

- change password to the password set in RDS

- change the database name to the table in MySQL

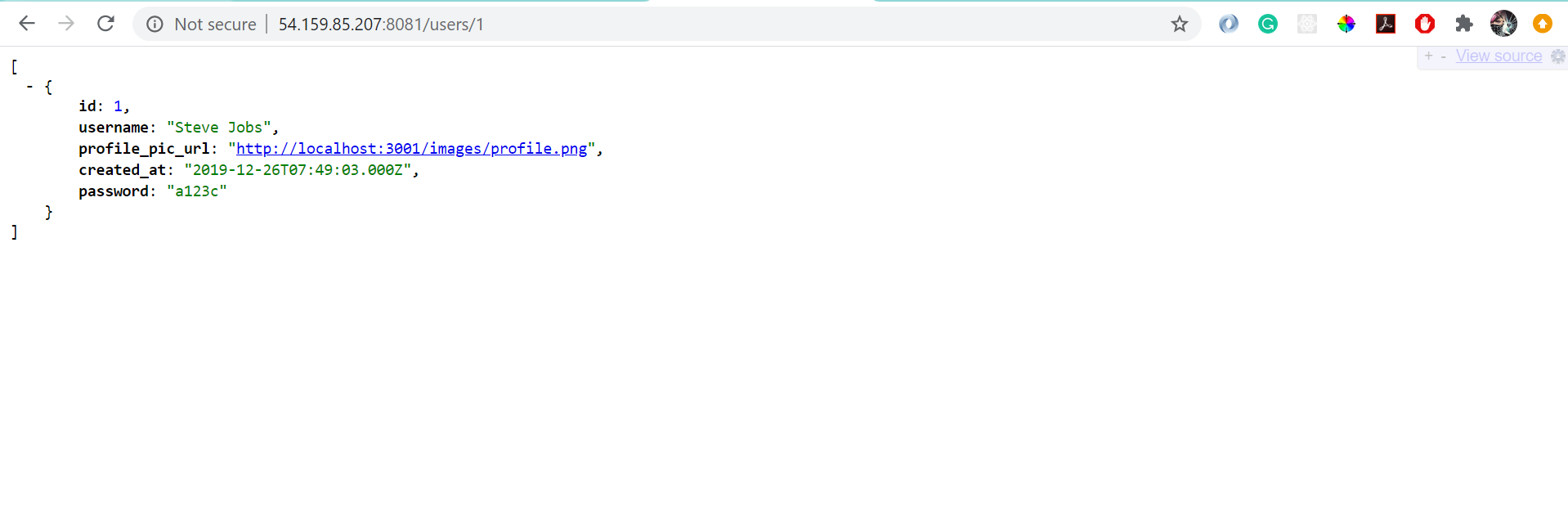


**7.** To save the changes, click on **‘Esc’** key and type **‘:wq!’** and press **‘Enter’**

**8.** Restart the pm2 server: **pm2 restart server**

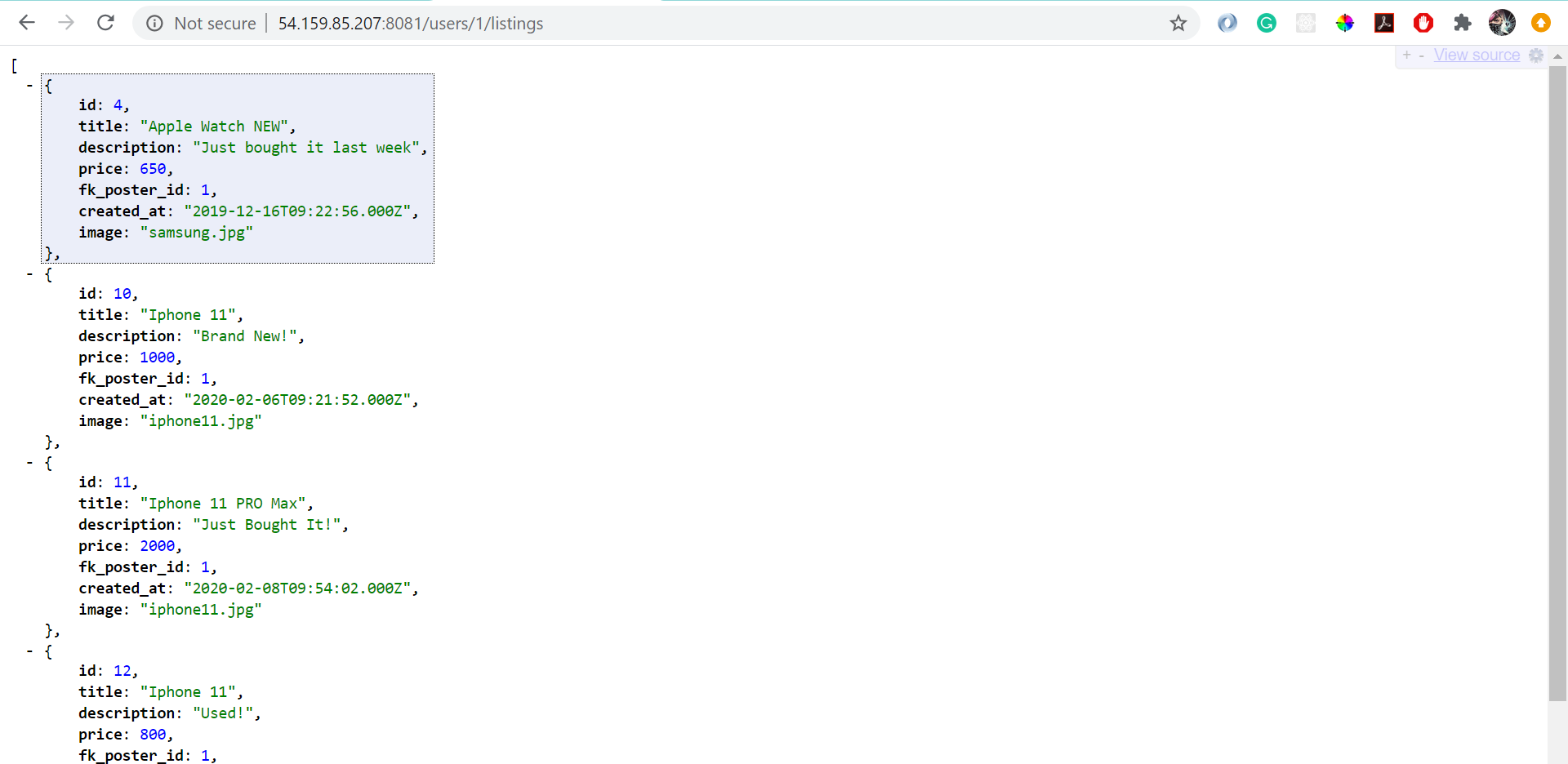
**9.** Open in browser:

**http://[your public IP address or domain name]:8081/users/1**



**(You should be able to successfully retrieve the data in the database)**

**http://[your public IP address or domain name]:8081/users/1/listings**



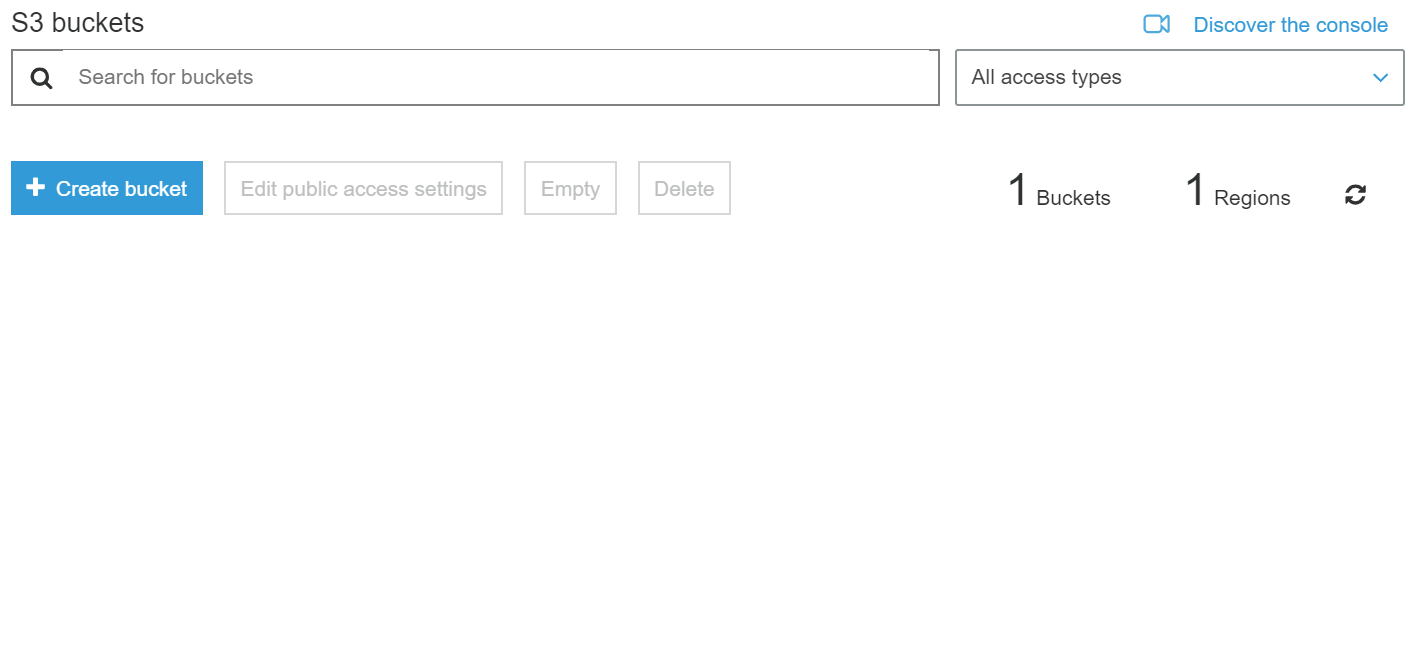
**(You should be able to successfully retrieve the data in the database)**

|  |
| --- |
| **Hosting a Static Website** |

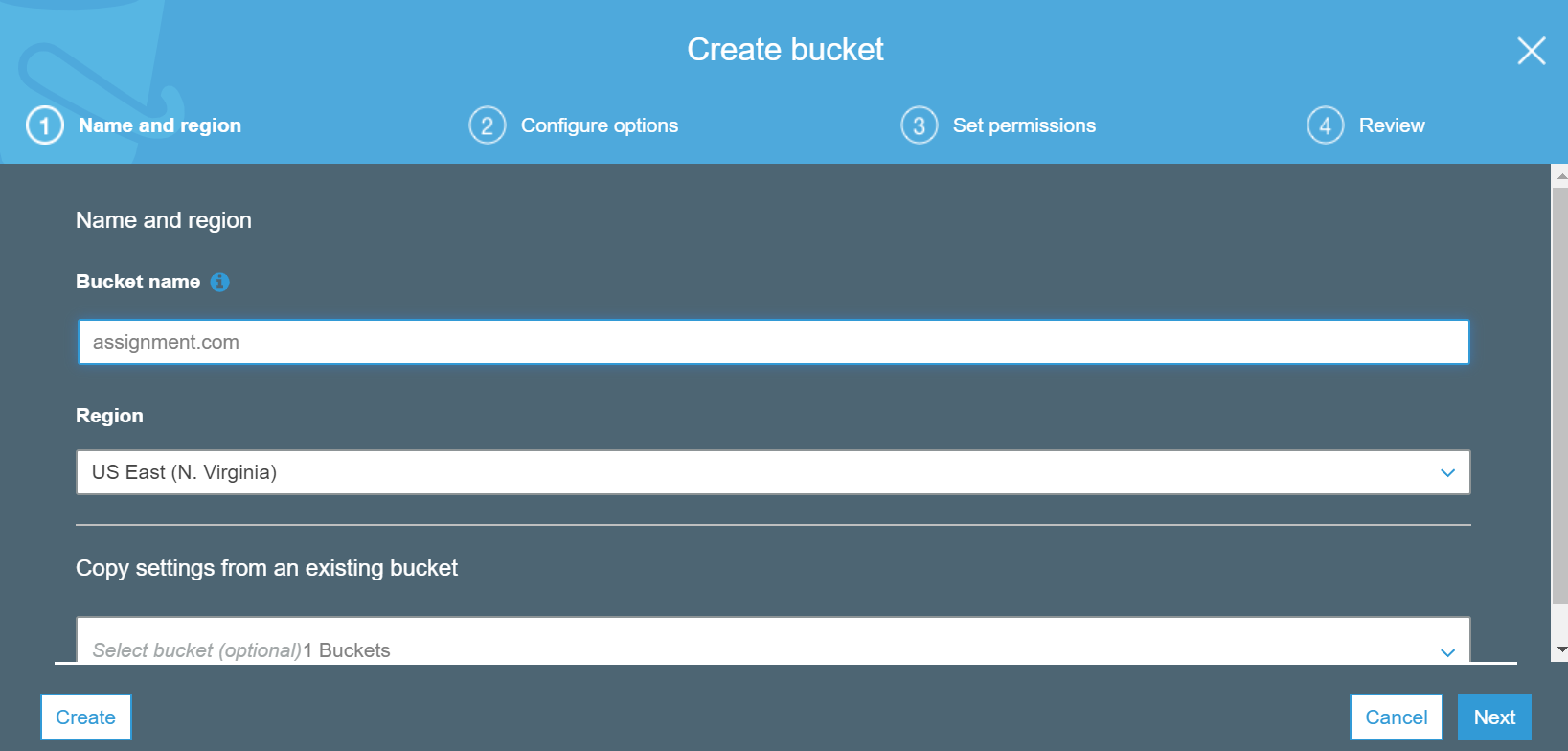
**Create a bucket**

**1.** Under services, click on S3

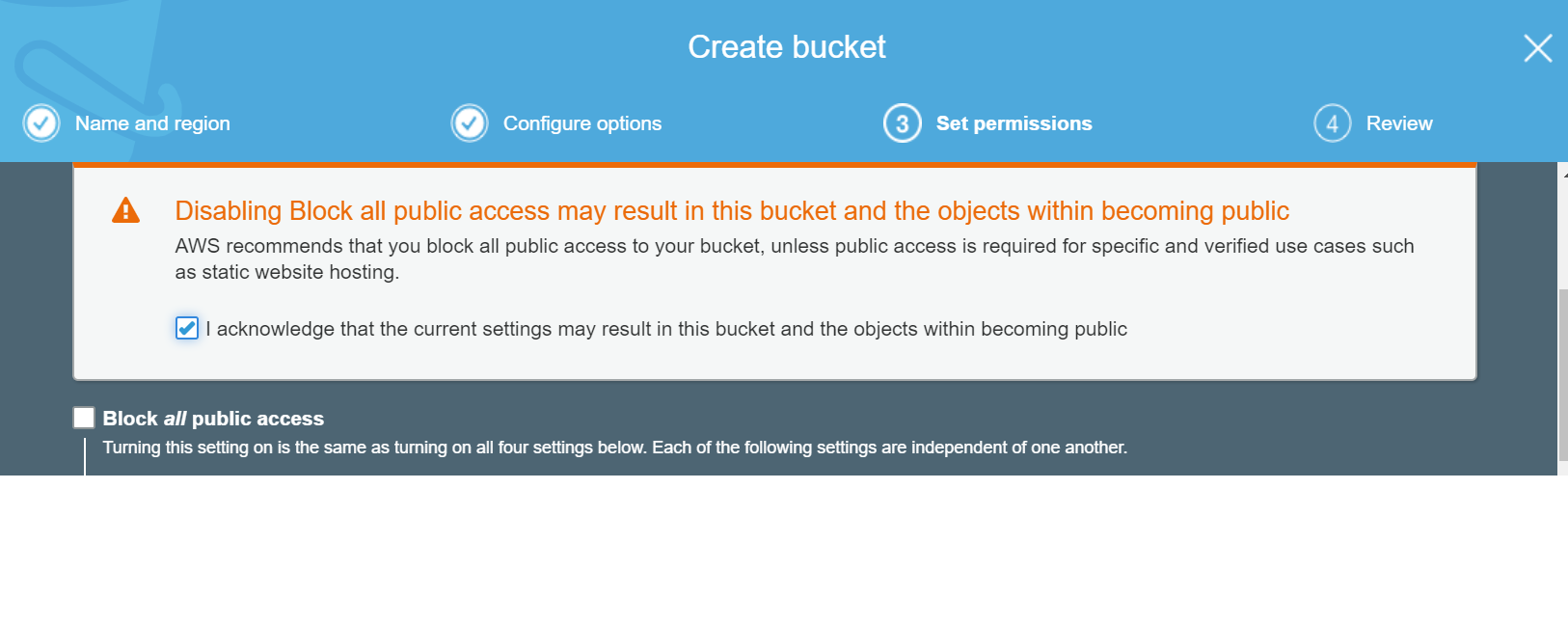
**2.** Click on create bucket



**3. Enter the Bucket name (for example, example.com)**

****

**4. Click Next button again and uncheck all the checkboxes in Set Permissions tab.**

****

**5. Click Next button and then click Create bucket button.**

**Enable static website hosting**

**1.** In the Bucket name list, choose the bucket that you want to use for your static website.

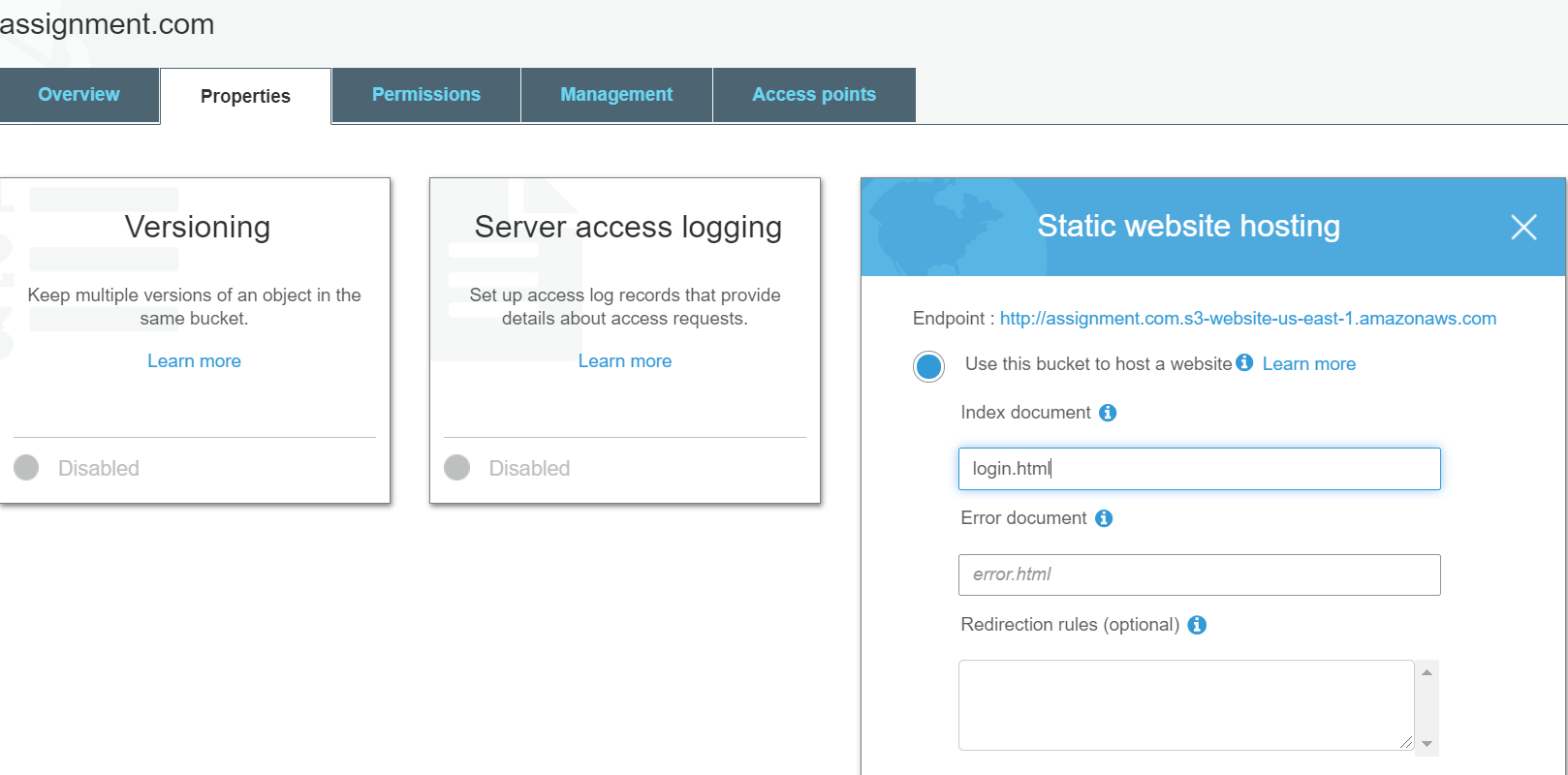
**2.** Choose Properties.

**3.** Choose Static website hosting.

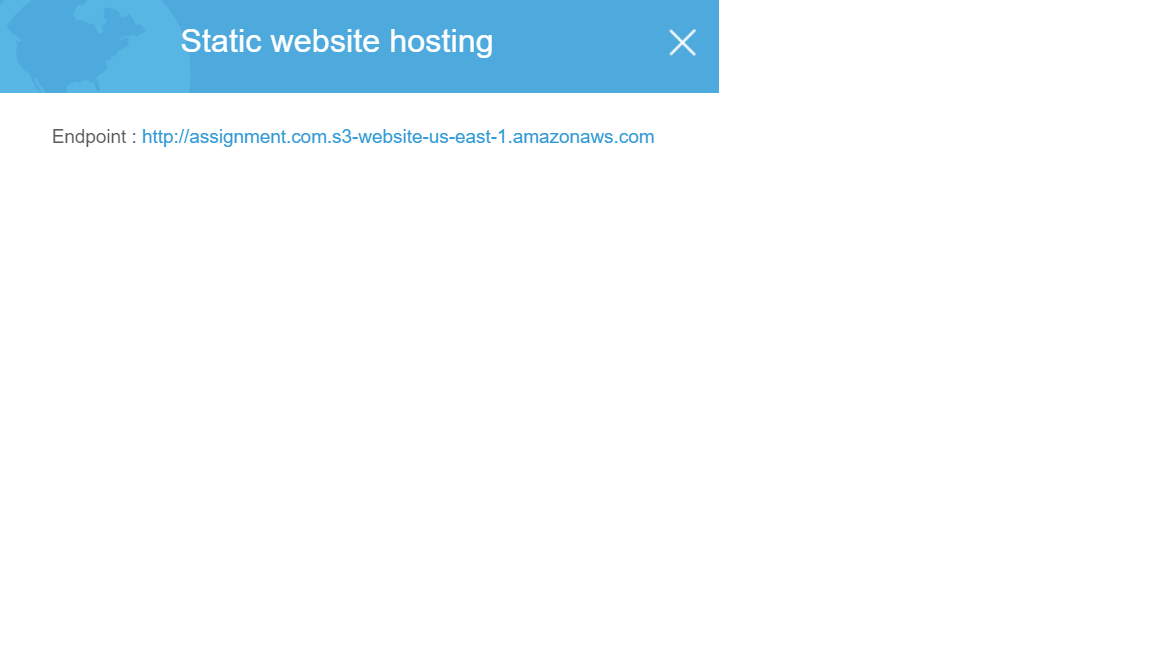
**4.** Choose Use this bucket to host a website.

**5.** Enter the name of your index document.

**6.** The index document name is typically index.html. The index document name is case sensitive and must exactly match the file name of the HTML index document that you plan to upload to your S3 bucket. For more information, see Configuring an index document.



**7.** Under Static website hosting, note the Endpoint: <http://assignment.com.s3-website-us-east-1.amazonaws.com/>



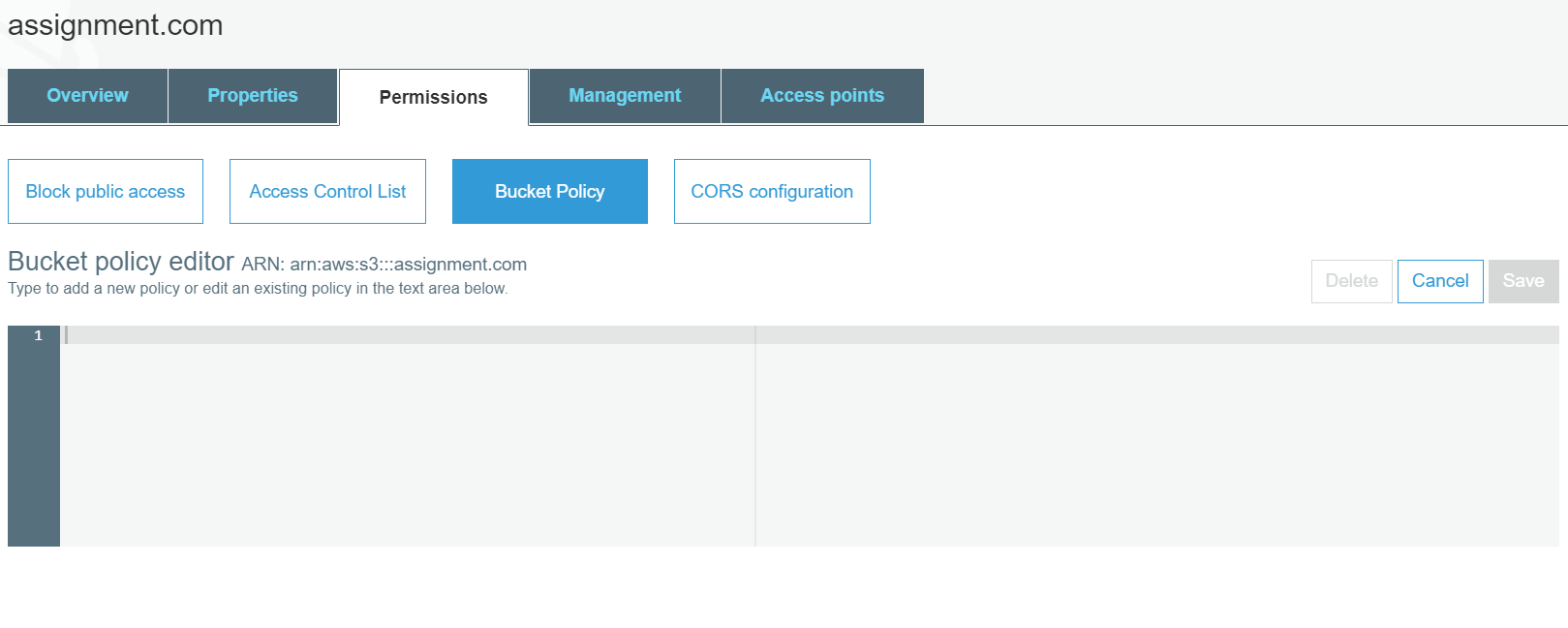
**8.** Click on save

**Add a bucket policy that makes your bucket content publicly available**

**1.** Under Buckets, choose the name of your bucket.

**2.** Choose Permissions.

**3.** Choose Bucket Policy.



**4.** To grant public read access for your website, copy the following bucket policy, and paste it in the Bucket policy editor.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": [

"s3:GetObject"

],

"Resource": [

"arn:aws:s3:::example.com/\*"

]

}

]

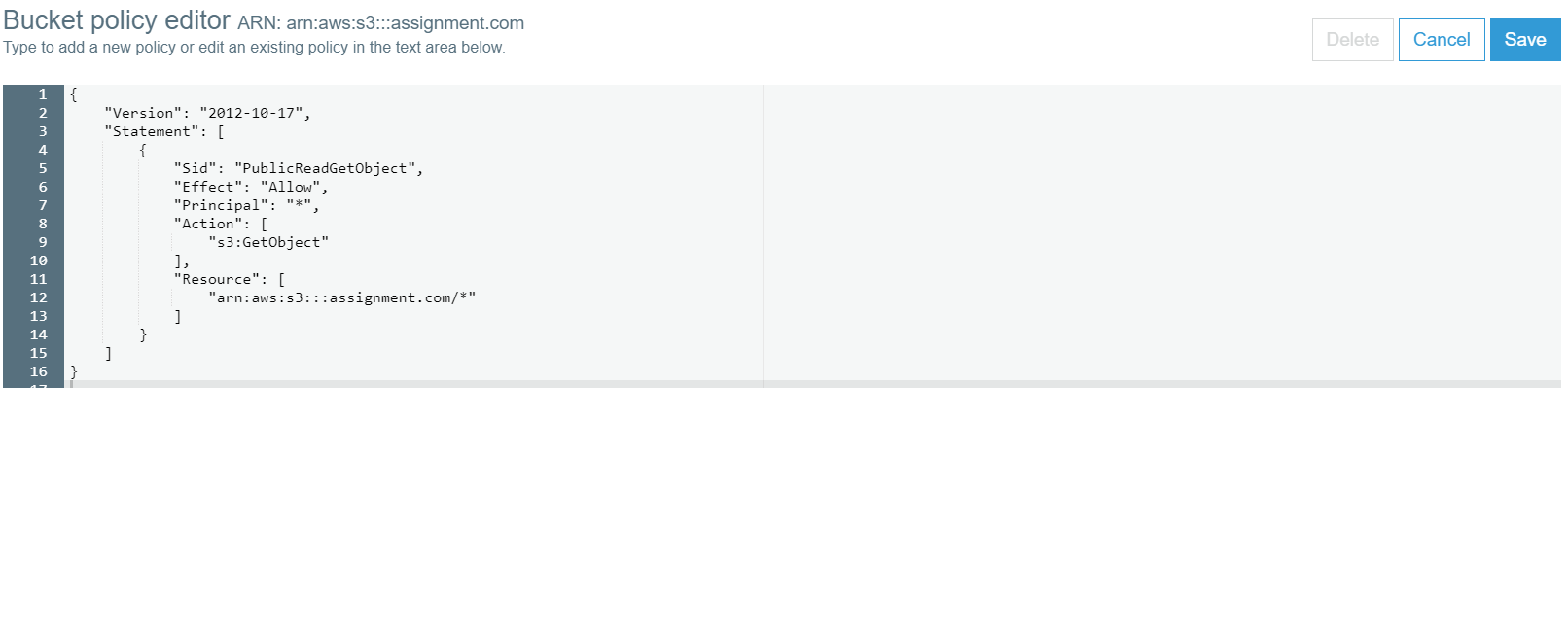
}

**5.** Update the Resource to include your bucket name.

"Resource": [

"arn:aws:s3:::assignment.com/\*"

]



**6.** Click on save

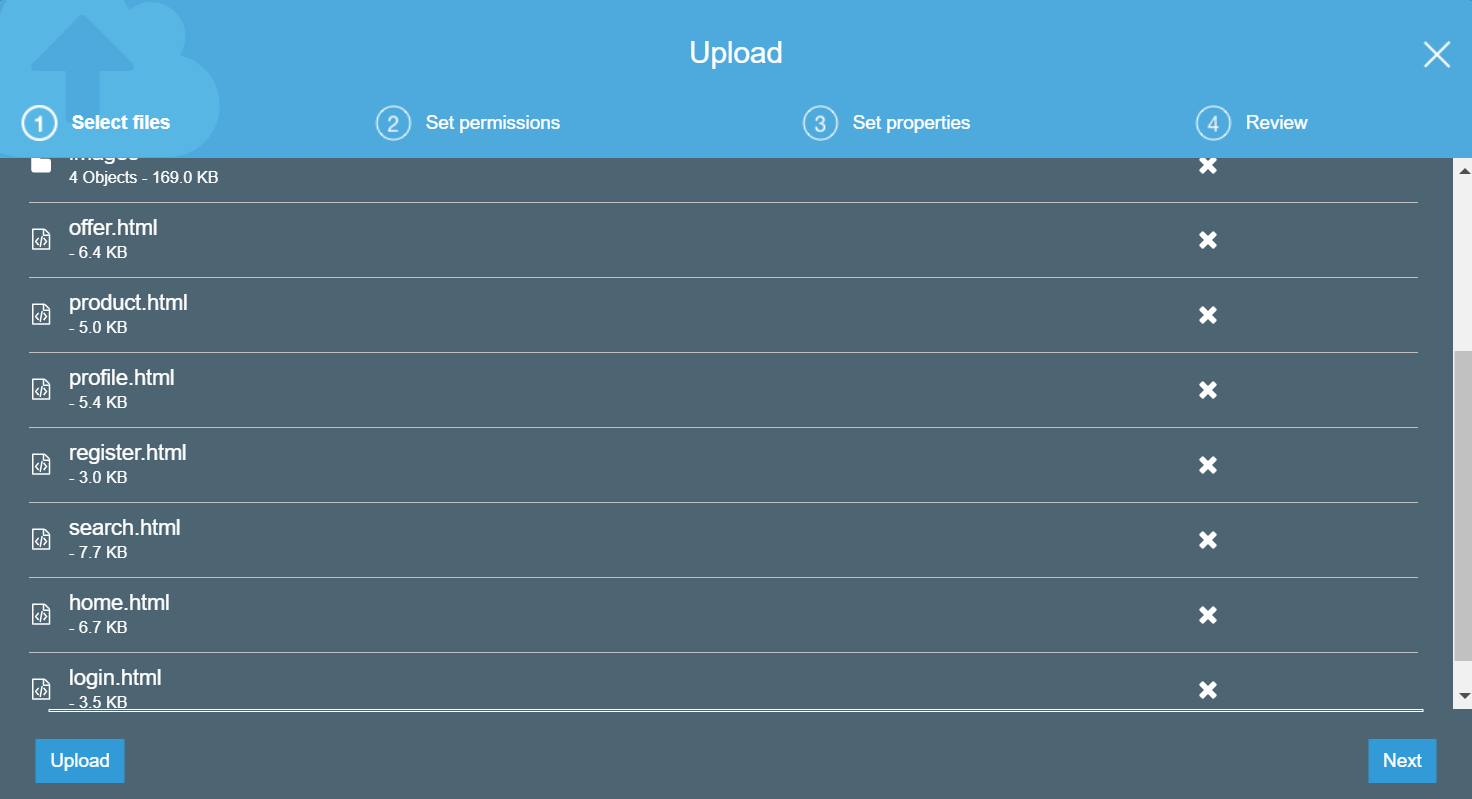
**Configure an index document**

**1.** Configure all the html files by changing the following:

- localhost:8081 -> (IP Address):8081

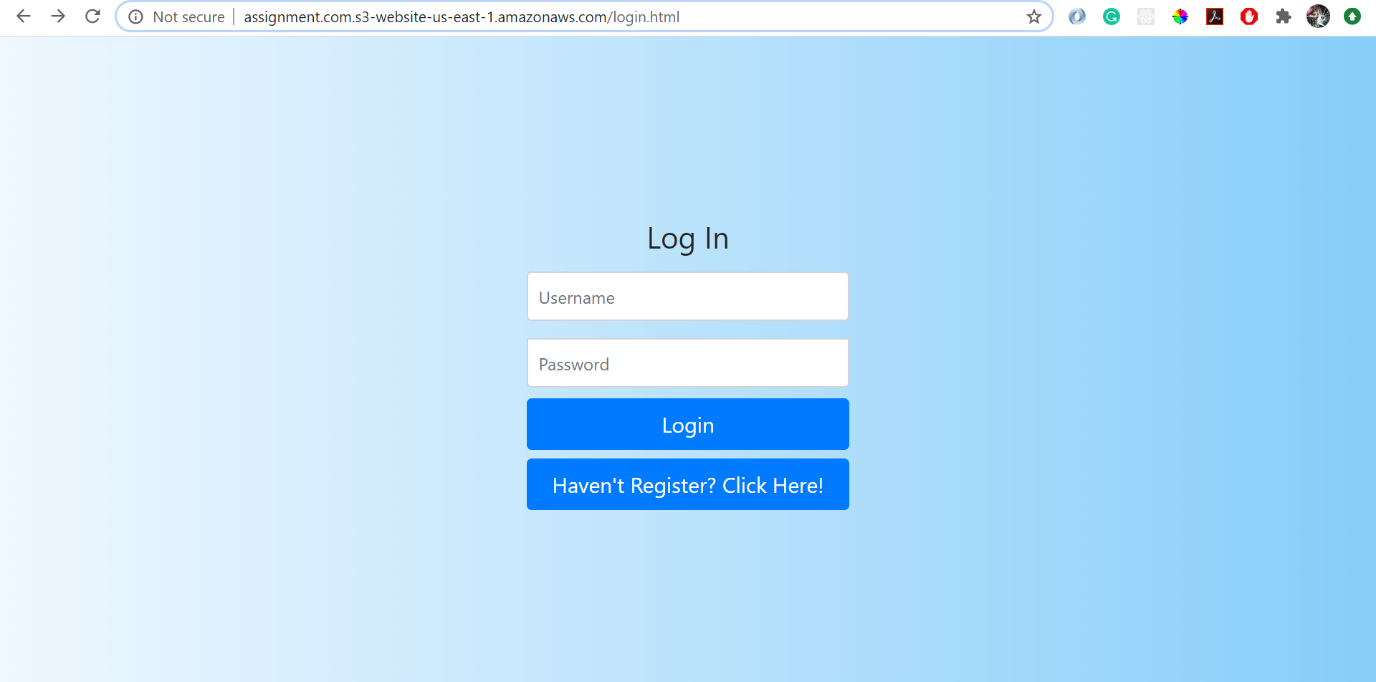
- localhost:3001 -> (S3 BUCKET ENDPOINT URL)

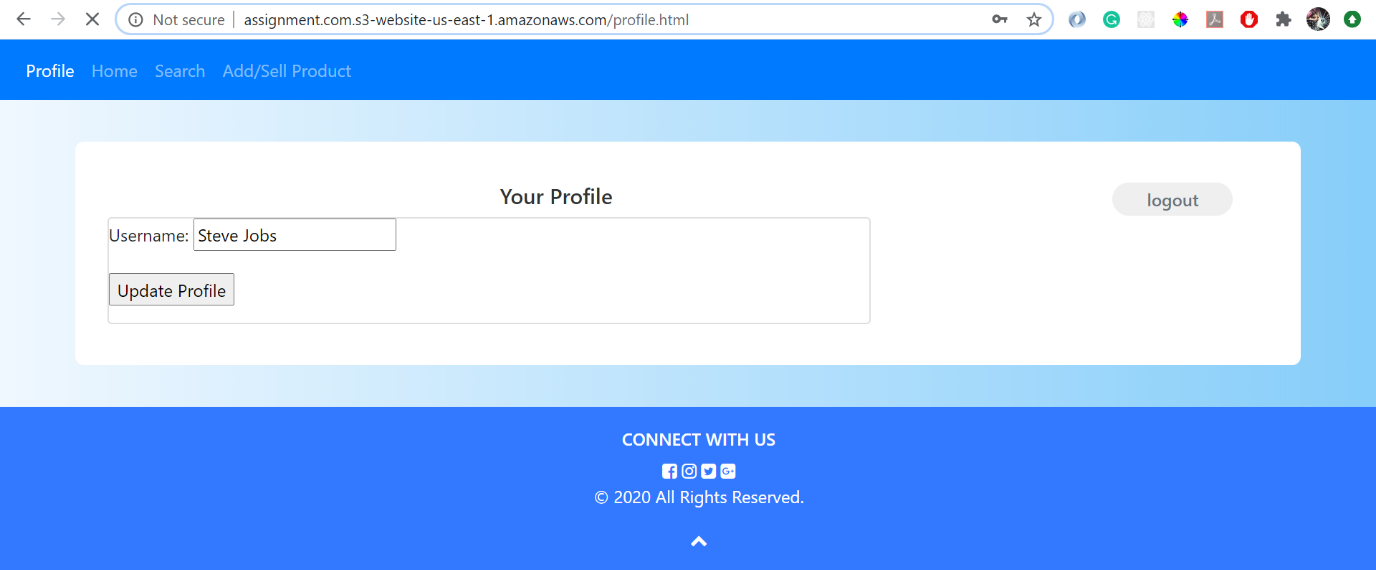
**2.** Upload all the .html files into S3 bucket

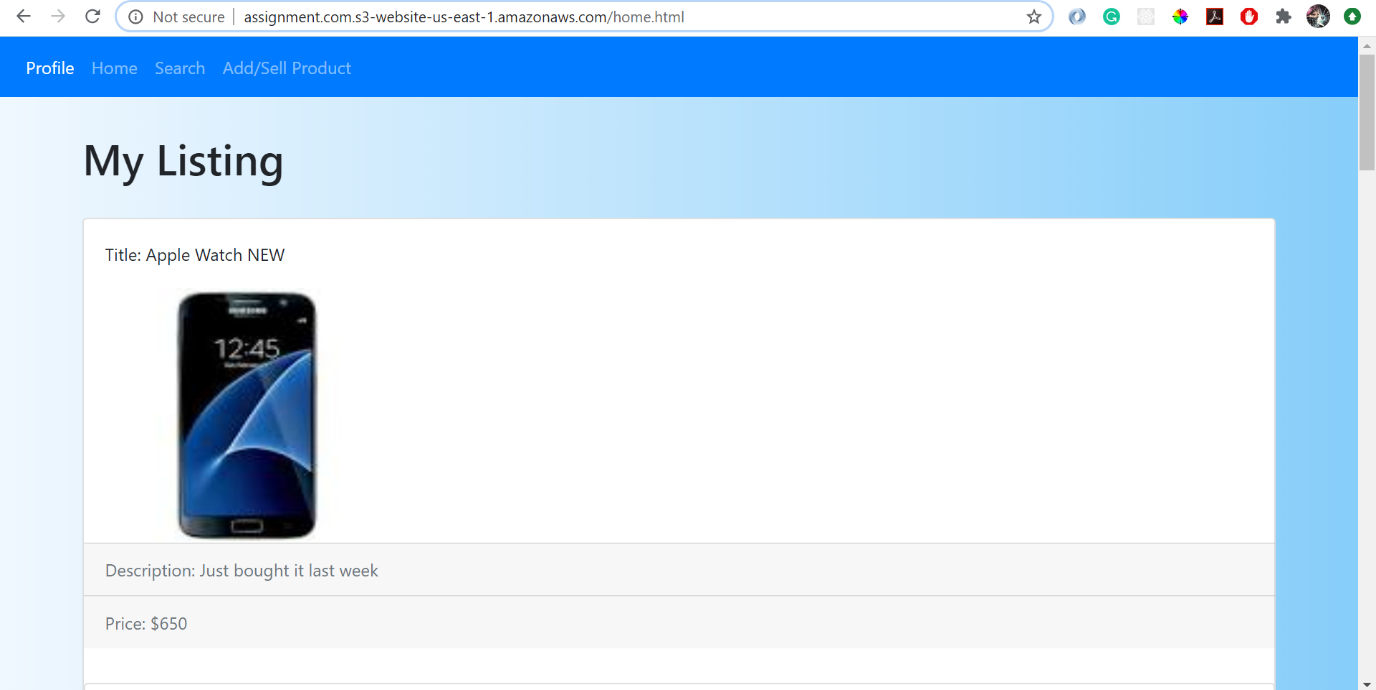


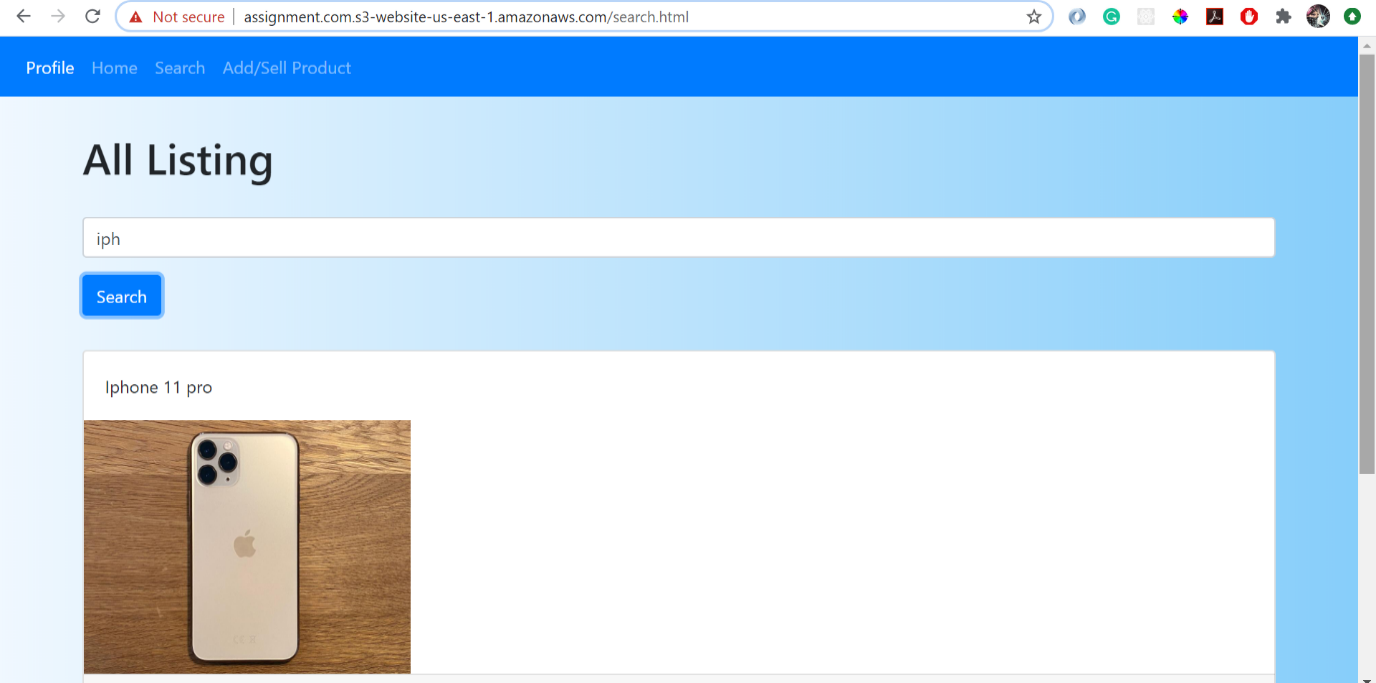
**Test your website endpoint**

**1.** To test your website, enter the website endpoint in your browser that you previously noted down. If your browser displays your .html page, the website was successfully deployed.









**Troubleshoot**

**- 403 Forbidden message**

- You should now see a 403 Forbidden message. This response is expected! This message indicates that your static website is being hosted by Amazon S3, but that the content is private.

- You can make Amazon S3 objects public through two different ways:

- To make either a whole bucket public, or a specific directory in a bucket public, use a bucket policy.

- To make individual objects in a bucket public, use an access control list (ACL).

- It is normally safer to make individual objects public because this avoids accidentally making other objects public. However, if you know that the entire bucket contains no sensitive information, you can use a bucket policy.

- You will now configure the individual objects to be publicly accessible.

- Return to the web browser tab with the Amazon S3 console (but do not close the website tab).

- Select all three objects.

- In the Actions menu, choose Make public.

A list of the three objects is displayed.

- Choose Make public

Your static website is now publicly accessible.

- Return to the web browser tab that has the 403 Forbiddenmessage.

- Refresh the webpage.

You should now see the static website that is being hosted by Amazon S3.

**Task 2: You are to choose one existing Restful service, and redesign/re-implement the service using AWS API gateway, Lambda function and dynamoDB, etc.**

|  |
| --- |
| **Developing with Amazon DynamoDB using AWS SDK** |

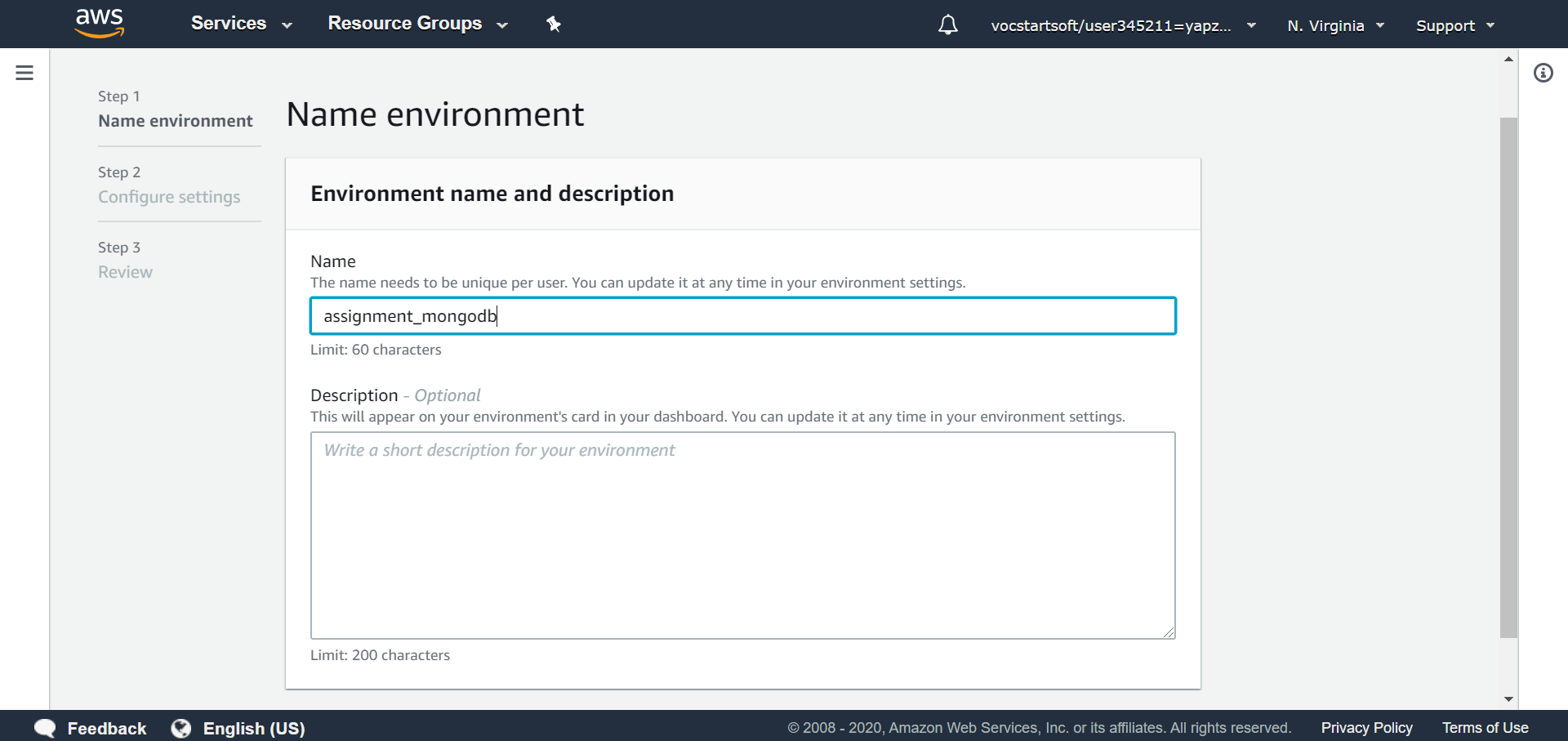
**Create Cloud9 Environment**

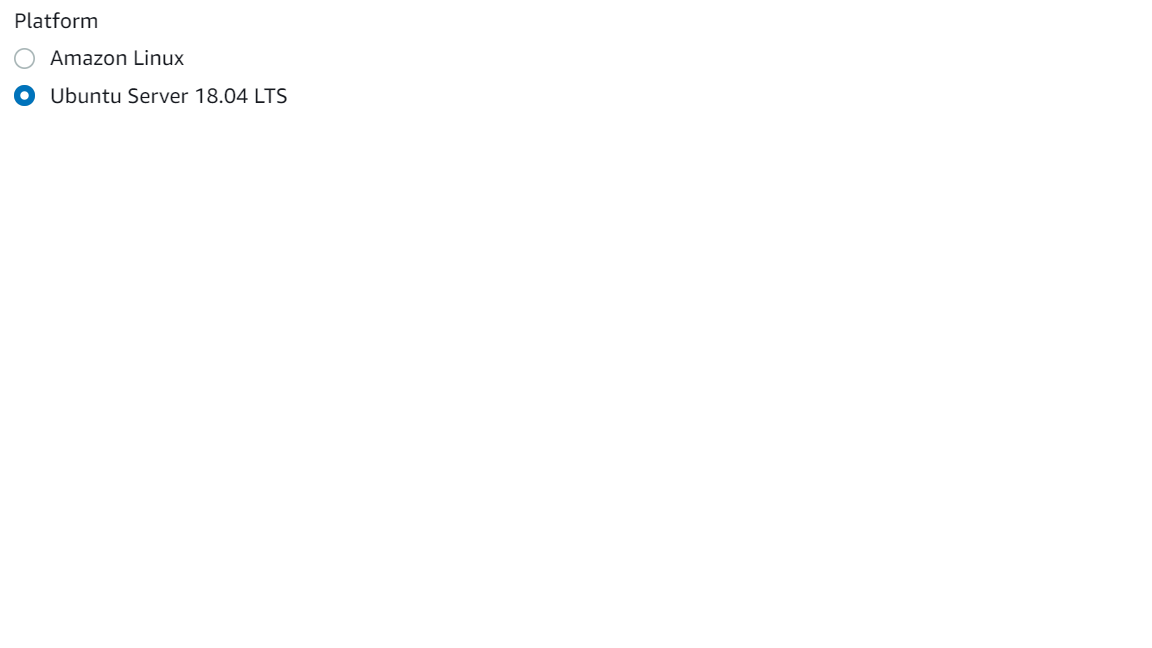
**1.** Under services, go to Cloud9

**2.** Click on create environment



**3.** Keep everything default except to change the name of the assignment and the platform to ‘Ubuntu Server 18.04 LTS’



****

**4.** Click on create button

**5.** To seed your AWS Cloud9 filesystem, go to the AWS Cloud9 bash terminal (at the bottom of the page) and run the following wget command:

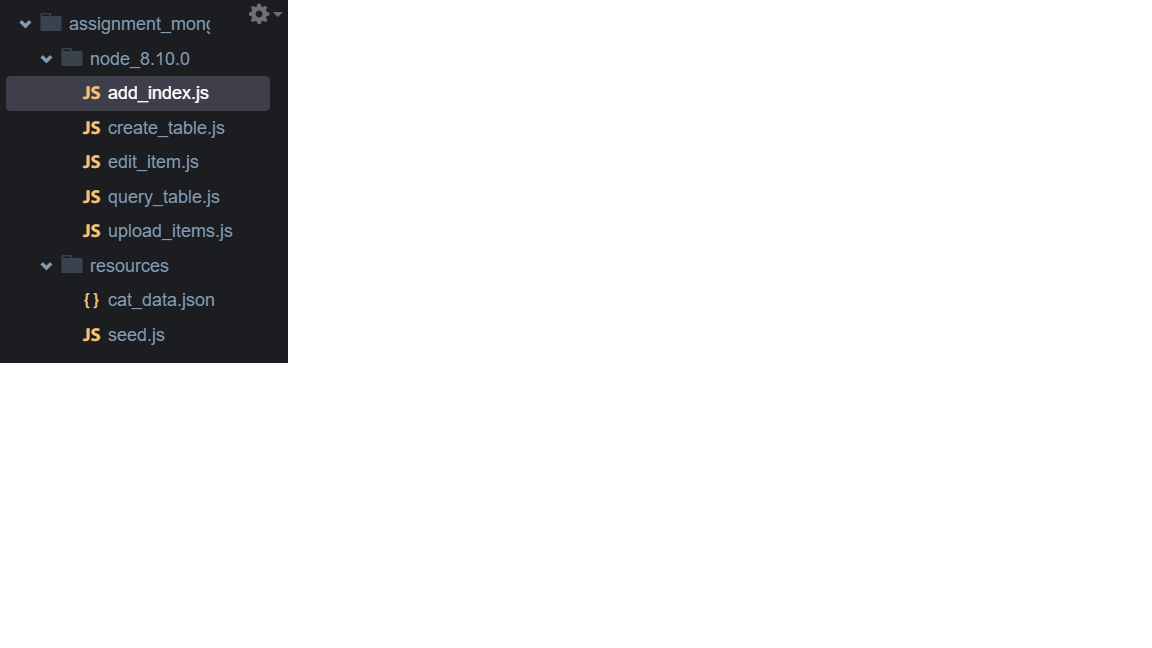
**wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-ILT-TF-200-ACCDEV-1/lab-3-ddb.zip -P /home/ubuntu/environment**

**6.** Unzip the lab-3-ddb.zip file by running the following command: **unzip lab-3-ddb.zip**

**7.** Clean up your environment by deleting README, zip folder, python folder, ruby folder and everything inside node folder except for solutions folder

**8.** Move the files from the solution folder into the node folder and delete away the solution folder which should be empty now

**9.** Your folder should look like this now

****

**10.** Set the terminal path into node\_8.10.0: **cd node\_8.10.0**

**11.** One-Time Initialization and Import: **npm install aws-sdk**

**Use the AWS SDK to Create a DynamoDB Table**

**1.** Open (double-click) the create\_table file and

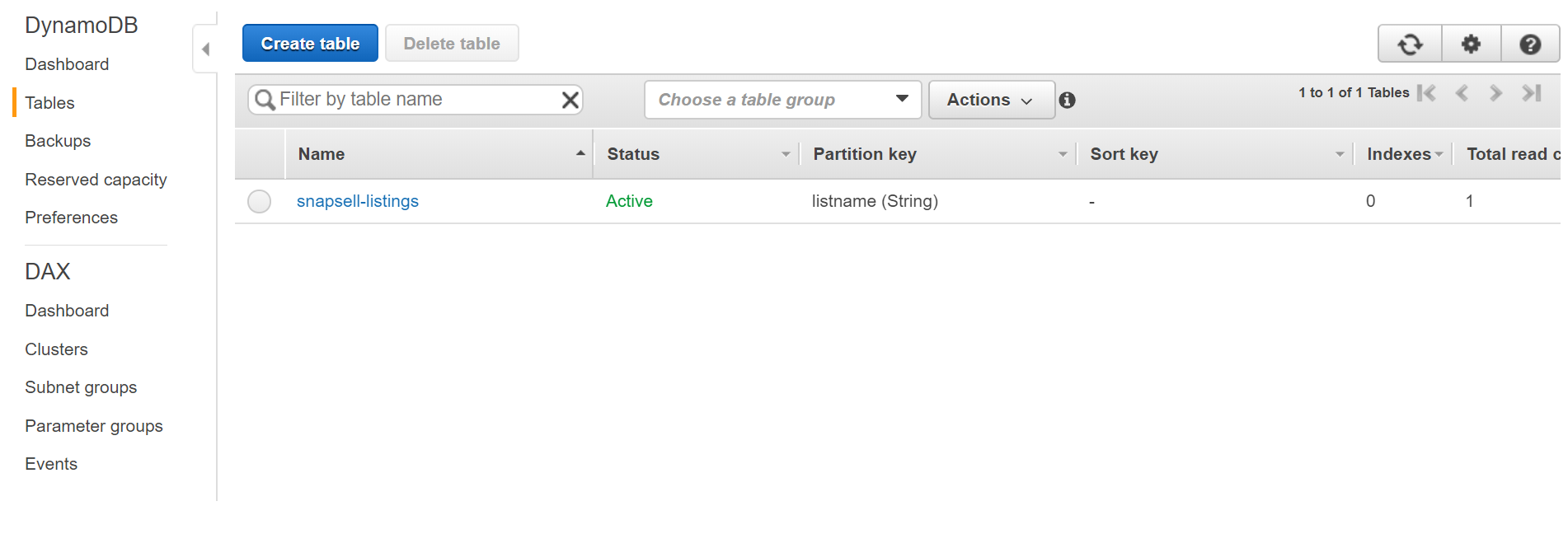
**2.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"), // Bring in the aws-sdk  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  });  (function createADataBaseTable(){  var  params = {  AttributeDefinitions: [{  AttributeName: "listname",  AttributeType: "S"  }],  KeySchema: [{  AttributeName: "listname",  KeyType: "HASH"  }],  ProvisionedThroughput: {  ReadCapacityUnits: 1,  WriteCapacityUnits: 1  },  TableName: "snapsell-listings"  };  DDB.createTable(params, function(err, data){  console.log(err, data);  });  })(); |

**3.** Run **node create\_table.js**



**4.** You should see a table ‘snapsell-listings’ being created

****

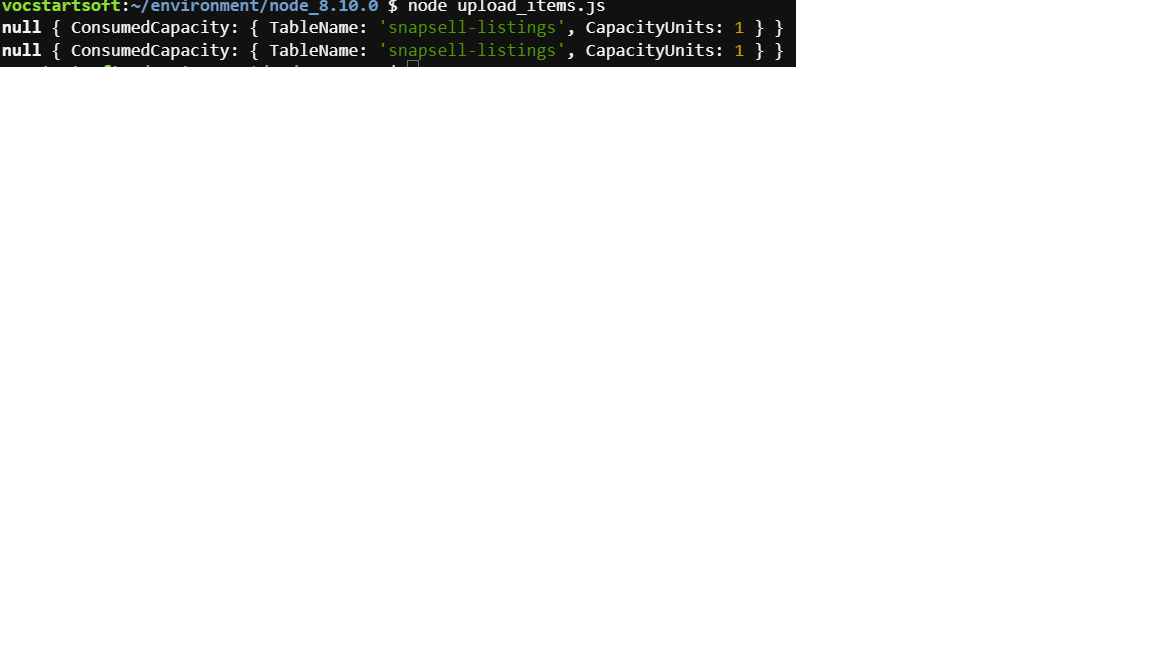
**Use the AWS SDK to Add Items to a DynamoDB Table**

**1.** Open (double-click) the upload\_items file.

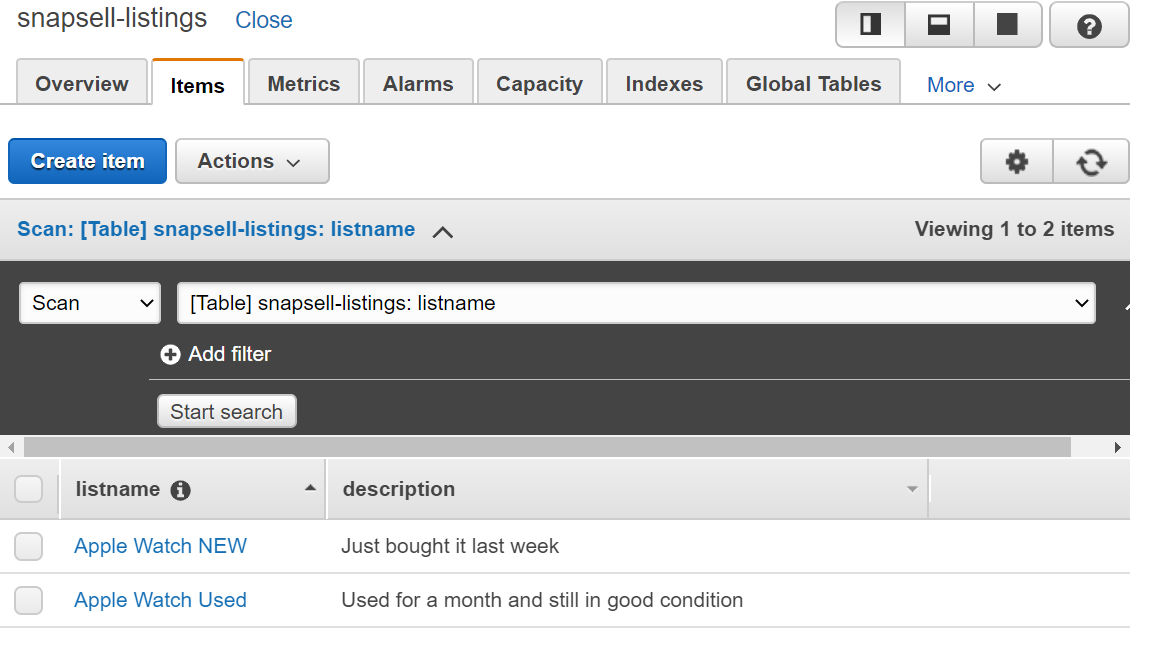
**2.** Change the codes accordingly

|  |  |
| --- | --- |
| var  AWS = require("aws-sdk"),  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  });  (function uploadItemstoDynamoDB(){  var  cat\_1 = {  Item:{  "listname":{  S: "Apple Watch NEW"  },  "description":{  S: "Just bought it last week"  }  },  ReturnConsumedCapacity: "TOTAL",  TableName: "snapsell-listings"  }; | DDB.putItem(cat\_1, function(err, data){  console.log(err, data);  });  var  cat\_2 = {  Item:{  "listname":{  S: "Apple Watch Used"  },  "description":{  S: "Used for a month and still in good condition"  }  },  ReturnConsumedCapacity: "TOTAL",  TableName: "snapsell-listings"  };  DDB.putItem(cat\_2, function(err, data){  console.log(err, data);  });  })(); |

**3.** Run **node upload\_items.js**

****

**4.** You should see items being uploaded in the database table



**Populate the Database**

**1.** In your AWS Cloud9 filesystem, identify the folder called resources. This folder contains two files: cat\_data.json and seed.js.

**2.** In the AWS Cloud9 terminal, navigate to the resources folder from your code folder by running the following command: **cd ~/environment/resources**

**3.** Run the following command: **npm install aws-sdk**

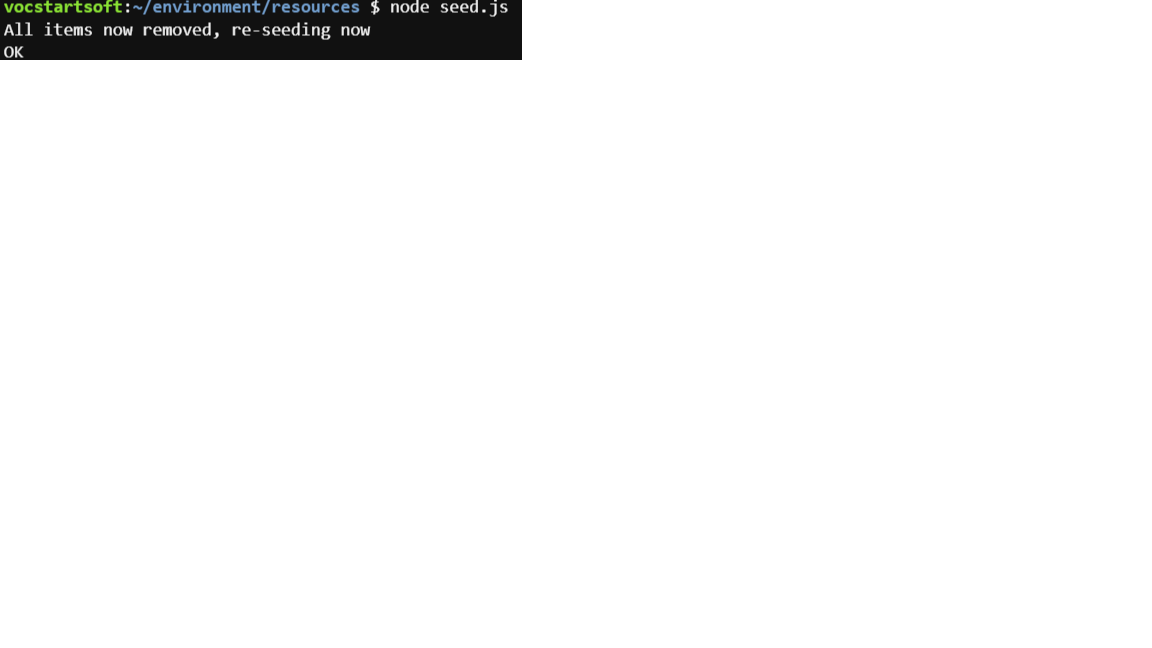
**4.** Change the codes in seed.js accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  }),  CAT\_DATA\_ARR = require("./cat\_data.json");  function addNewItemsFromJSON(){  console.log("All items now removed, re-seeding now");  var  cat = {},  cat\_formatted\_arr = [],  params = {};  for(var i\_int = 0; i\_int < CAT\_DATA\_ARR.length; i\_int += 1){  cat = {  PutRequest: {  Item: {  listname: {  "S": CAT\_DATA\_ARR[i\_int].listname\_str  },  description: {  "S": CAT\_DATA\_ARR[i\_int].description\_str  },  price: {  "N": CAT\_DATA\_ARR[i\_int].price\_num  },  created\_at: {  "S": CAT\_DATA\_ARR[i\_int].created\_at\_str  },  image: {  "S": CAT\_DATA\_ARR[i\_int].image\_str  }  }  }  };  cat\_formatted\_arr.push(cat);  }  params = {  RequestItems: {  "snapsell-listings": cat\_formatted\_arr.reverse()  }  };  DDB.batchWriteItem(params, function(err, data){  if(err){  throw err;  }  console.log("OK");  });  }  function init(){  removeExistingItemsFromDynamo(function(err, data){  if(err){  throw err;  }  addNewItemsFromJSON();  });  }  function removeExistingItemsFromDynamo(cb){  var  params = {  RequestItems: {  "snapsell-listings": [{  DeleteRequest: {  Key: {  listname: {  "S": "Apple Watch NEW"  }  }  }  },{  DeleteRequest: {  Key: {  listname: {  "S": "Apple Watch Used"  }  }  }  }]  }  };    DDB.batchWriteItem(params, function(err, data){  if(err){  throw err;  }  //if empty table this will still work.  //if you run into problems manaully delete the items (not the DB)  //then run this script  cb(null, true);  });  }  init(); |

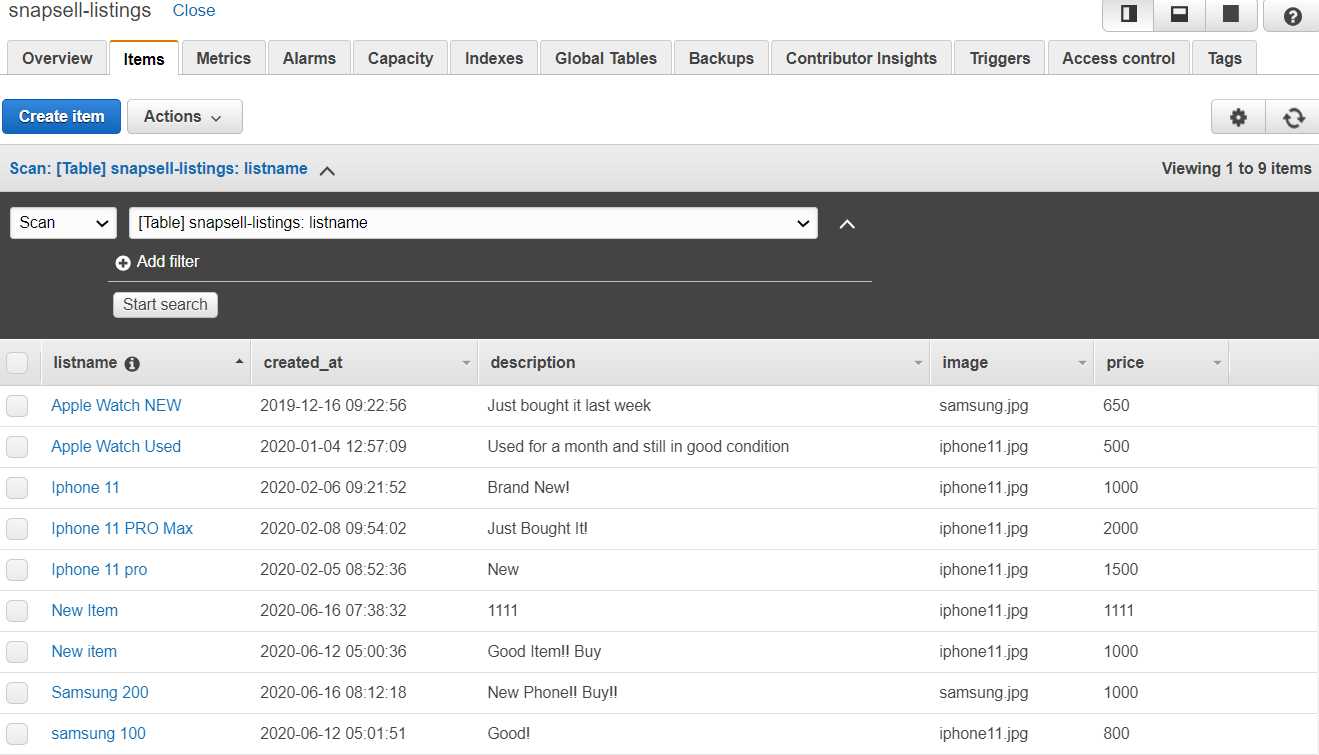
**5.** Change the codes in cat\_data.json accordingly

|  |
| --- |
| [{  "listname\_str": "Apple Watch NEW",  "description\_str": "Just bought it last week",  "price\_num": "650",  "created\_at\_str": "2019-12-16 09:22:56",  "image\_str": "samsung.jpg"  },{    "listname\_str": "Apple Watch Used",  "description\_str": "Used for a month and still in good condition",  "price\_num": "500",  "created\_at\_str": "2020-01-04 12:57:09",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "Iphone 11 pro",  "description\_str": "New",  "price\_num": "1500",  "created\_at\_str": "2020-02-05 08:52:36",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "Iphone 11",  "description\_str": "Brand New!",  "price\_num": "1000",  "created\_at\_str": "2020-02-06 09:21:52",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "Iphone 11 PRO Max",  "description\_str": "Just Bought It!",  "price\_num": "2000",  "created\_at\_str": "2020-02-08 09:54:02",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "New item",  "description\_str": "Good Item!! Buy ",  "price\_num": "1000",  "created\_at\_str": "2020-06-12 05:00:36",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "samsung 100",  "description\_str": "Good!",  "price\_num": "800",  "created\_at\_str": "2020-06-12 05:01:51",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "New Item",  "description\_str": "1111",  "price\_num": "1111",  "created\_at\_str": "2020-06-16 07:38:32",  "image\_str": "iphone11.jpg"  },{    "listname\_str": "Samsung 200",  "description\_str": "New Phone!! Buy!!",  "price\_num": "1000",  "created\_at\_str": "2020-06-16 08:12:18",  "image\_str": "samsung.jpg"  }] |

**6.** Run **node seed.js**



**7.** You should see the items being seeded into the tables of DynamoDB

****

**Use the AWS SDK to Query for an Item in a DynamoDB Table**

**1.** Navigate back to node\_8.10.0 folder

**2.** Open (double-click) the query\_table file

**3.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  });  (function queryTable(){  var  params = {  ExpressionAttributeValues: {  ":name": {  S: "Apple Watch NEW"  }  },  KeyConditionExpression: "listname = :name",  ProjectionExpression: "price",  TableName: "snapsell-listings"  };  DDB.query(params, function(err, data){  if(err){  throw err;  }  console.log(data.Items[0].price.N);  });  })(); |

**4.** Run **node query\_table.js** and you should be able to retrieve the price of a listing

****

**Use the AWS SDK to Create a GSI on a DynamoDB Table**

**1.** Open (double-click) the add\_index file.

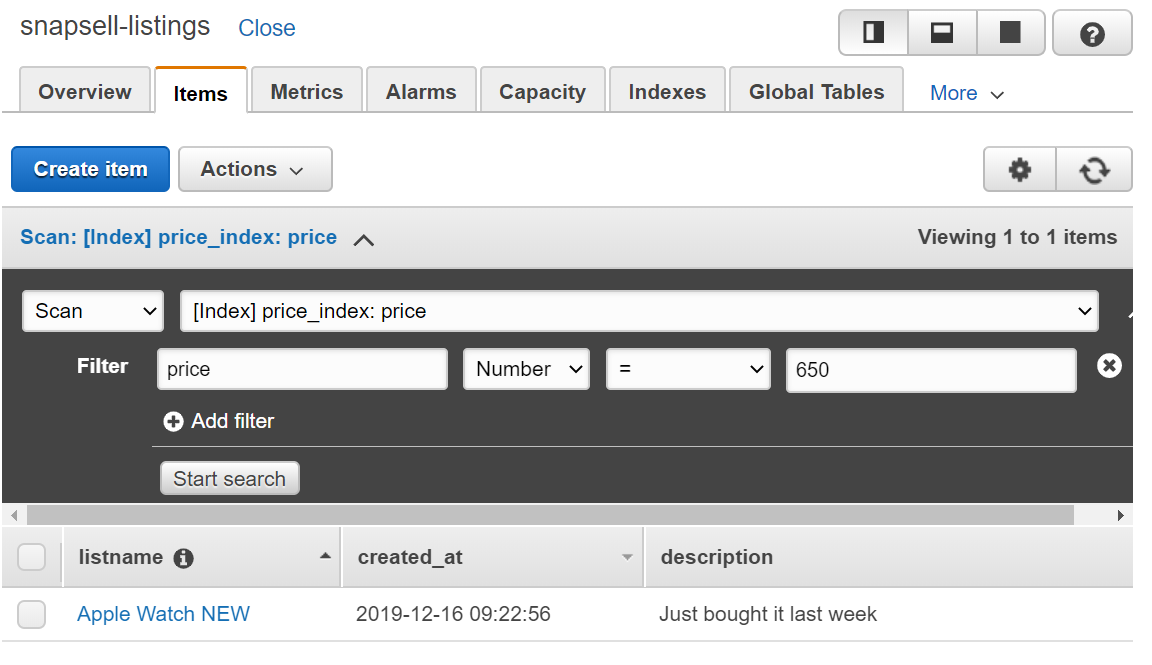
**2.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  });  (function addIndex(){  var  params = {  TableName: "snapsell-listings",  AttributeDefinitions: [{  AttributeName: "price",  AttributeType: "N"  }],  GlobalSecondaryIndexUpdates: [{  Create: {  IndexName: "price\_index",  KeySchema: [{  AttributeName: "price",  KeyType: "HASH"  }],  Projection: {  ProjectionType: "ALL"  },  ProvisionedThroughput: {  ReadCapacityUnits: 1,  WriteCapacityUnits: 1  }  }  }]  };  DDB.updateTable(params, function(err, data){  console.log(err, data);  });  })(); |

**3.** Run **node add\_index.js**

****

**4.** You should be able to search a query.



**Troubleshoot**

**- Error running these command “wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-ILT-TF-200-ACCDEV-1/lab-3-ddb.zip -P /home/ubuntu/environment”**

****

Make sure you are in the right directories: **/home/ubuntu/environment**

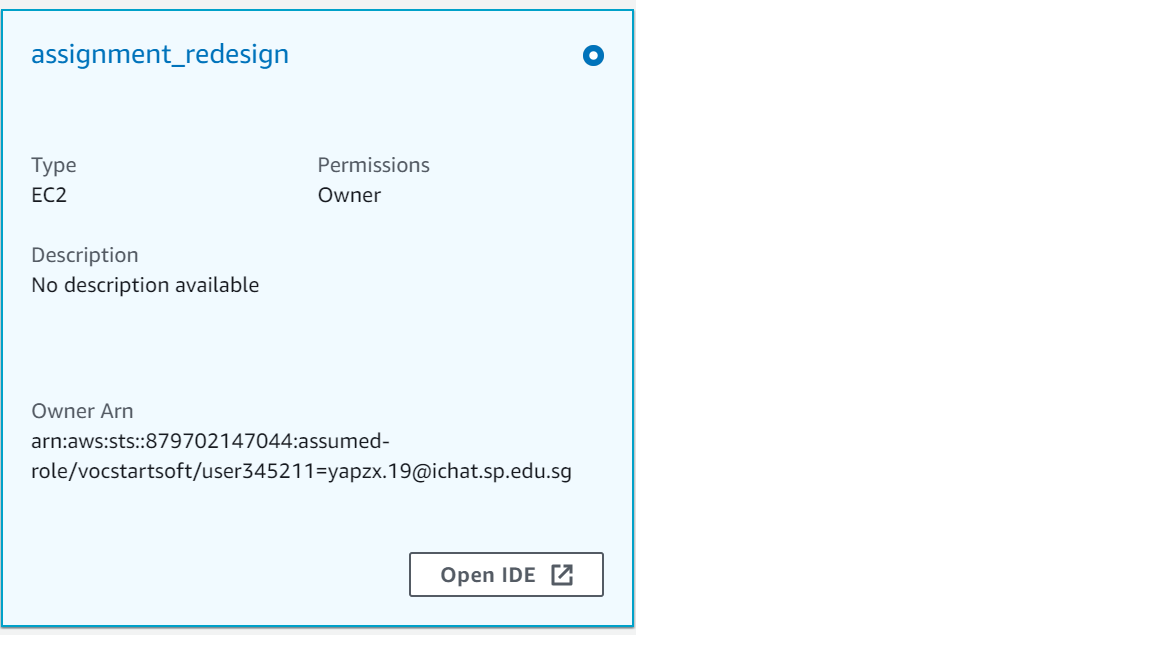
Note: We created Cloud9 using ubuntu and not using Amazon Linux AMI… Thus, if you are

using Ubuntu, default user is ubuntu and if you are using Linux, default user is ec2-user

|  |
| --- |
| **Developing with AWS Lambda and Amazon API Gateway using the AWS SDK** |

**Setup in Cloud9**

**\*Create a new Cloud9 environment called ‘assignment\_redesign’**

****

**1.** To seed your AWS Cloud9 filesystem, go to the AWS Cloud9 bash terminal (at the bottom of the page) and run the following wget command:

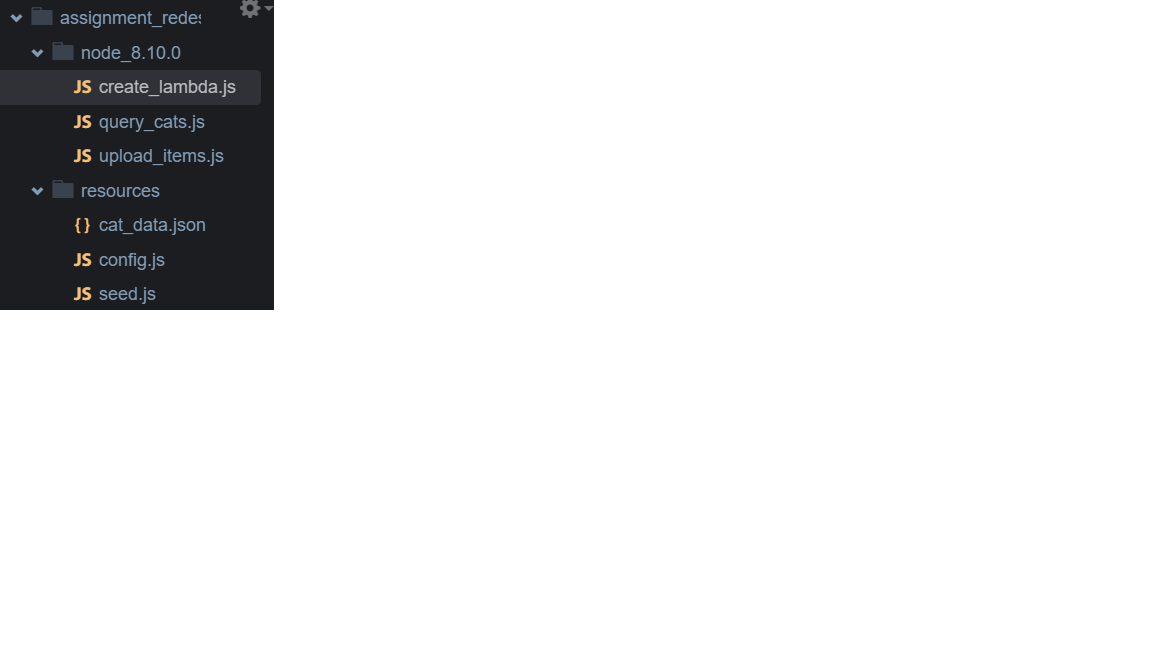
**wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-ILT-TF-200-ACCDEV-1/lab-5-lambda-gateway.zip -P /home/ubuntu/environment**

**2.** Unzip the lab-5-lambda-gateway.zip file: **unzip lab-5-lambda-gateway.zip**

**3.** Clean up your environment by deleting README, zip folder, python folder, ruby folder and everything inside node folder except for solutions folder

**4.** Move the files from the solution folder into the node folder and delete away the solution folder which should be empty now

**5.** Your folder should look like this now



**6.** Set the terminal path into node\_8.10.0: **cd node\_8.10.0**

**7.** One-Time Initialization and Import (update): **npm install aws-sdk**

**Create a new S3 bucket and upload html**

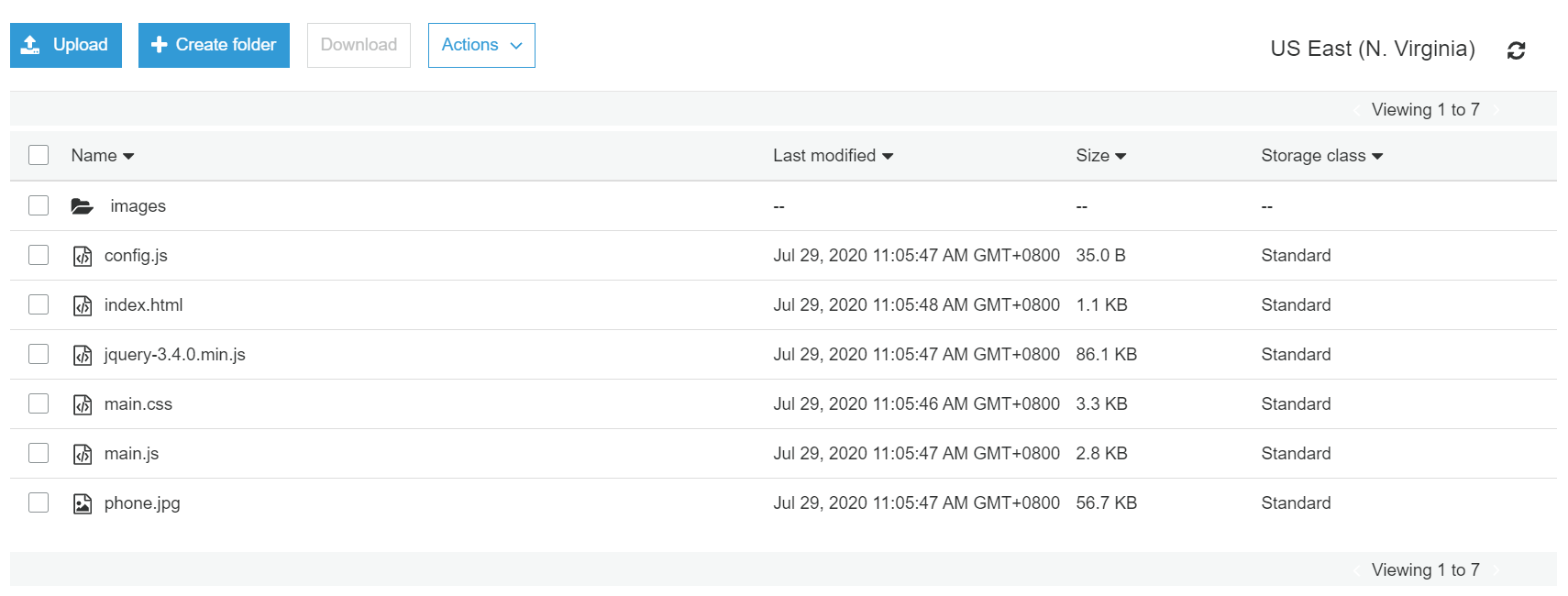
**-**  Name it as ‘redesignfrontend’

**-** Allow all public access

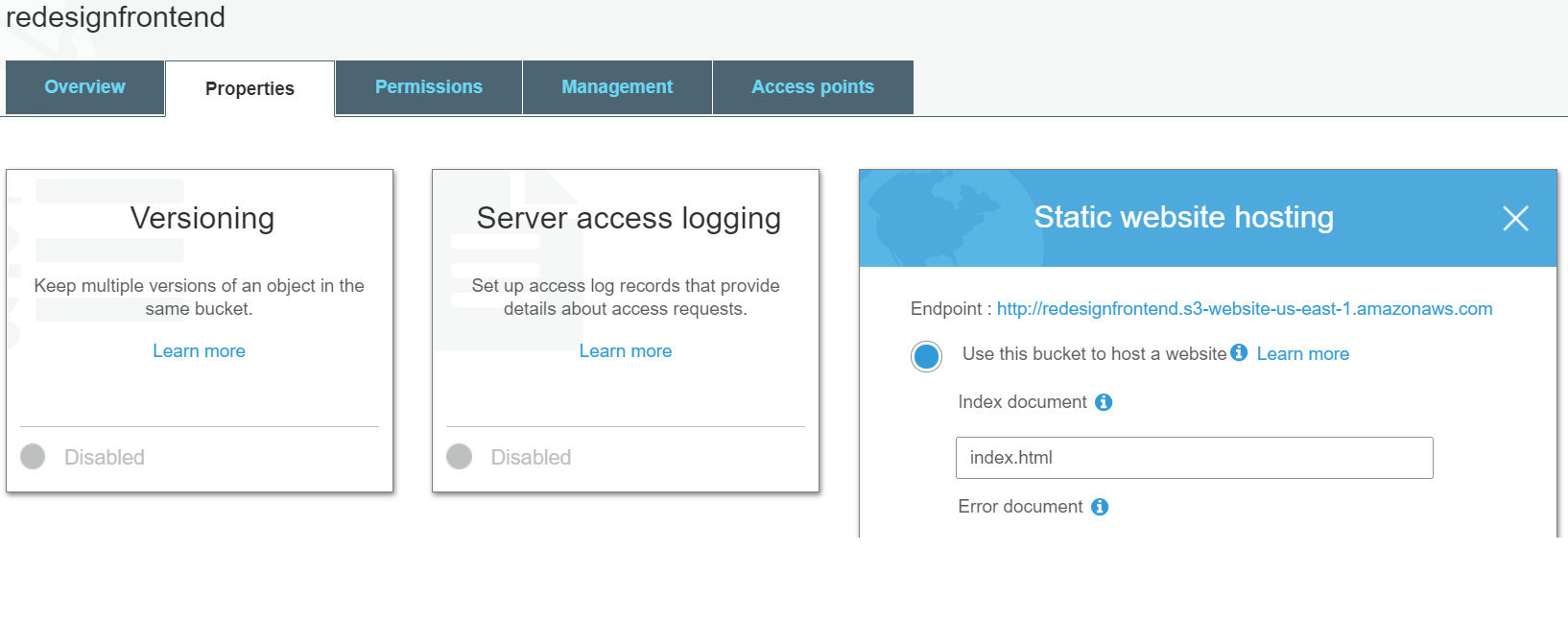
**-** Under ‘Permission’ -> ‘Bucket Policy’, add in the following and click save

|  |
| --- |
| {  "Version": "2012-10-17",  "Statement": [  {  "Sid": "PublicReadGetObject",  "Effect": "Allow",  "Principal": "\*",  "Action": [  "s3:GetObject"  ],  "Resource": [  "arn:aws:s3:::redesignfrontend/\*"  ]  }  ]  } |

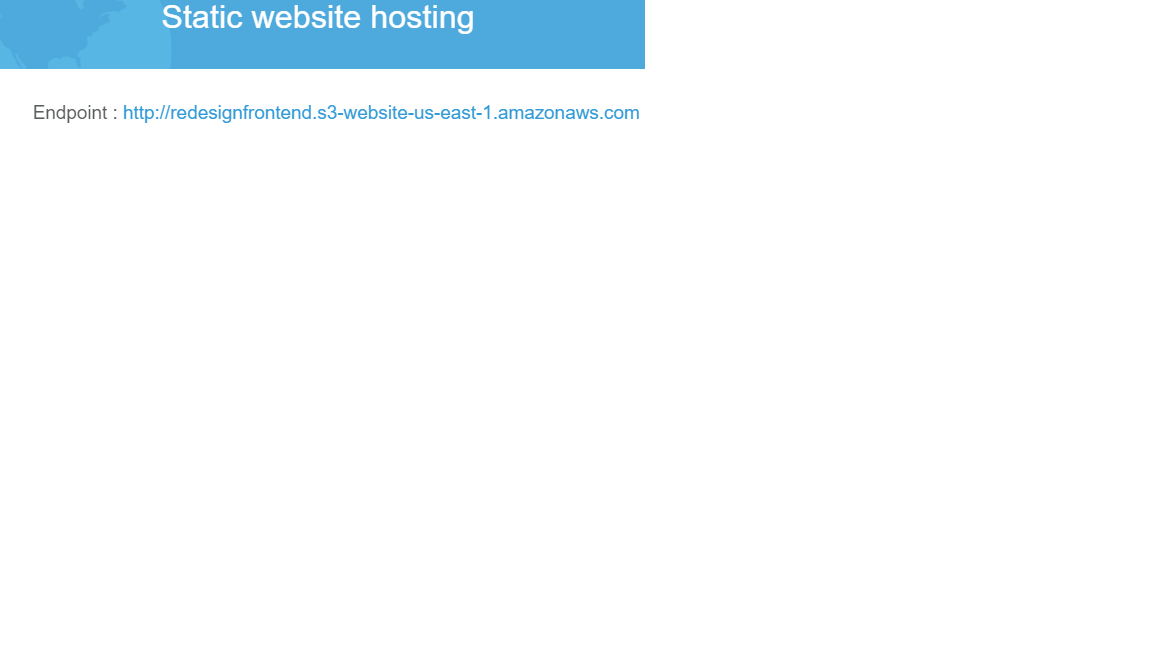
**1.** Upload the documents into the bucket

****

**2.** Under properties, go to ‘static website hosting’ and fill in ‘index.html’ under index documents

****

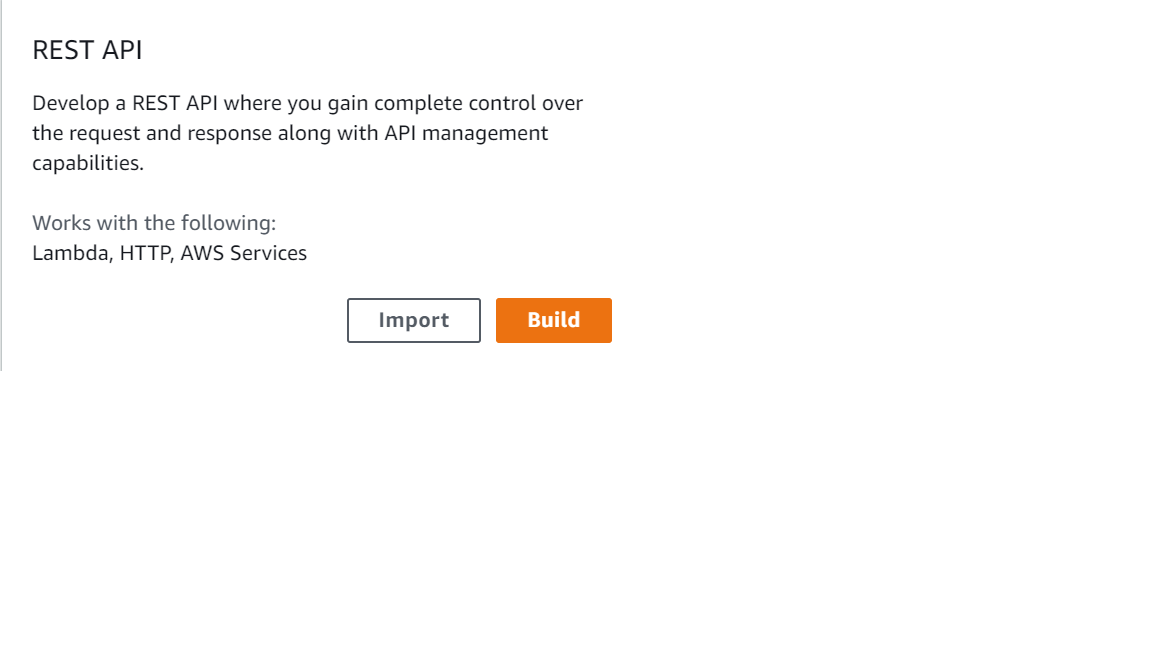
**3.** Note the url endpoint: <http://redesignfrontend.s3-website-us-east-1.amazonaws.com>



**Create a REST Endpoint with Amazon API Gateway**

**1.** Under services, choose API Gateway

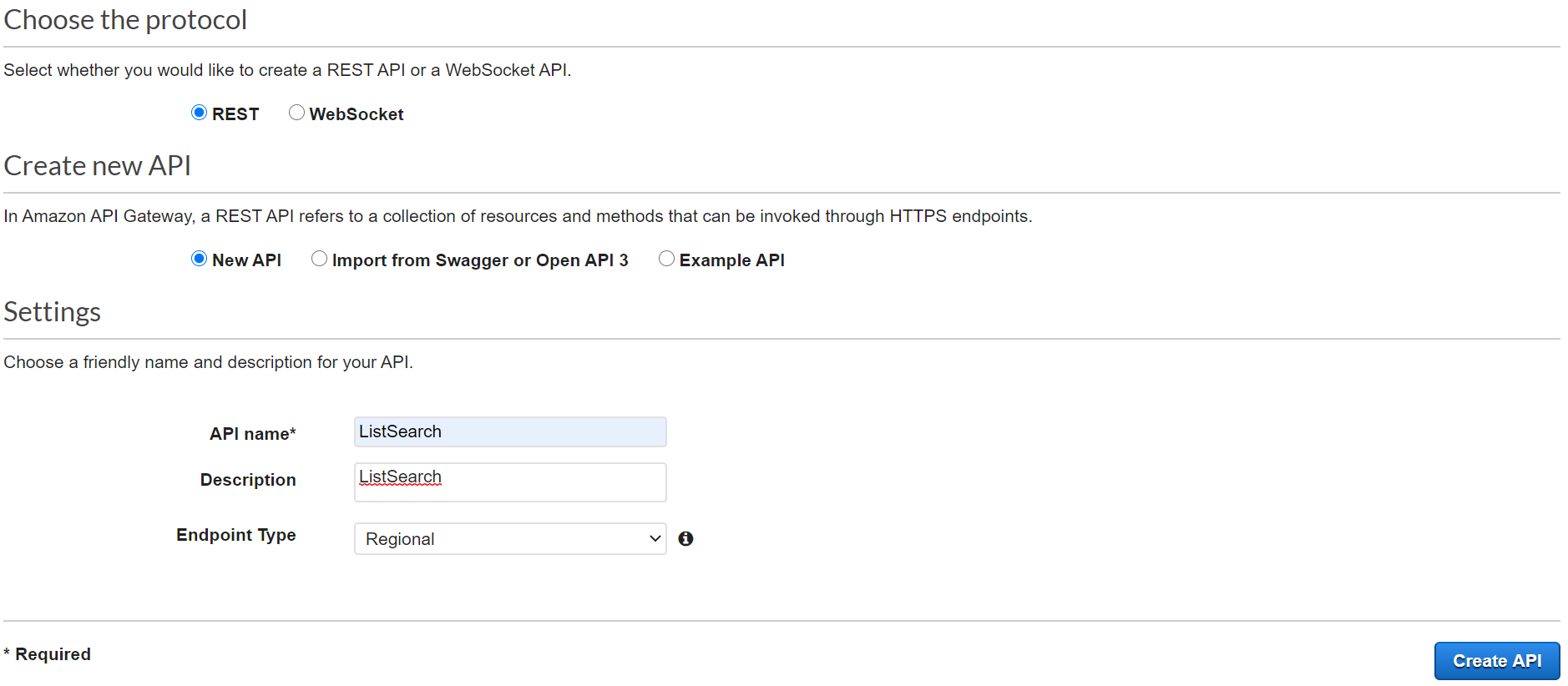
**2.** Create a new API under ‘REST API’ by clicking on build



**3.** Configure the settings as follows:

- Under ‘Create new API’, select **‘New API’**

- Under ‘Settings’, add an **API name (ListSearch)** and **Description (ListSearch)**



**4.** Click on ‘Create API’ button

**5.** Choose Actions and then Create Method.

**6.** Under Resources, go to the dropdown menu under the /, choose POST, and choose the check mark icon.

**7.** For Integration type, choose Mock and then choose Save.

**8.** Choose TEST.

**9.** Scroll down. Under Request Body, enter the following JSON:

{

"listname\_str": "Apple Watch New"

}

**10.** Choose Test.

**11.** The test is returning information successfully, though without any values. You want to create a mock API test and return some static values. You will create a mock API endpoint that always returns two cat items, which you will hardcode in to the API's response. The API will return these items like they came from an Amazon DynamoDB response. You will let your client-side web application handle formatting directly in the website's JavaScript code.

**12.** To go back to the test settings, scroll up and choose Method Execution.

**13.** Choose Integration Response.

**14.** To show the response properties, choose the arrow.

**15.** Choose the arrow for Mapping Templates.

**16.** Under Content-Type, select application/json.

**17.** For General template, select Method Request passthrough.

**18.** Replace all the lines with the following JSON:

|  |
| --- |
| [  {  "listname": {  "S": "Apple Watch Used"  },  "created\_at": {  "S": "2020-01-04 12:57:09"  },  "price": {  "N": "500"  },  "description": {  "S": "Used for a month and still in good condition"  },  "image": {  "S": "iphone11.jpg"  }  },  {  "listname": {  "S": "Iphone 11 pro"  },  "created\_at": {  "S": "2020-02-05 08:52:36"  },  "price": {  "N": "1500"  },  "description": {  "S": "New"  },  "image": {  "S": "iphone11.jpg"  }  }  ] |

**19.** At the bottom of the Mapping Templates section, choose Save.

**20.** At the top of the page, choose Save.

**21.** Scroll back to the top and choose Method Execution again.

**22.** Choose TEST.

**23.** Set the request body as:

{

"listname\_str": " Apple Watch New"

}

**24.** Choose Test.

**25.** You have now created and tested your REST API. Next, you must publish it:

**26.** Choose Actions and under API Actions, select Deploy API.

**27.** On the Deploy API window:

- Deployment stage: Select [New Stage].

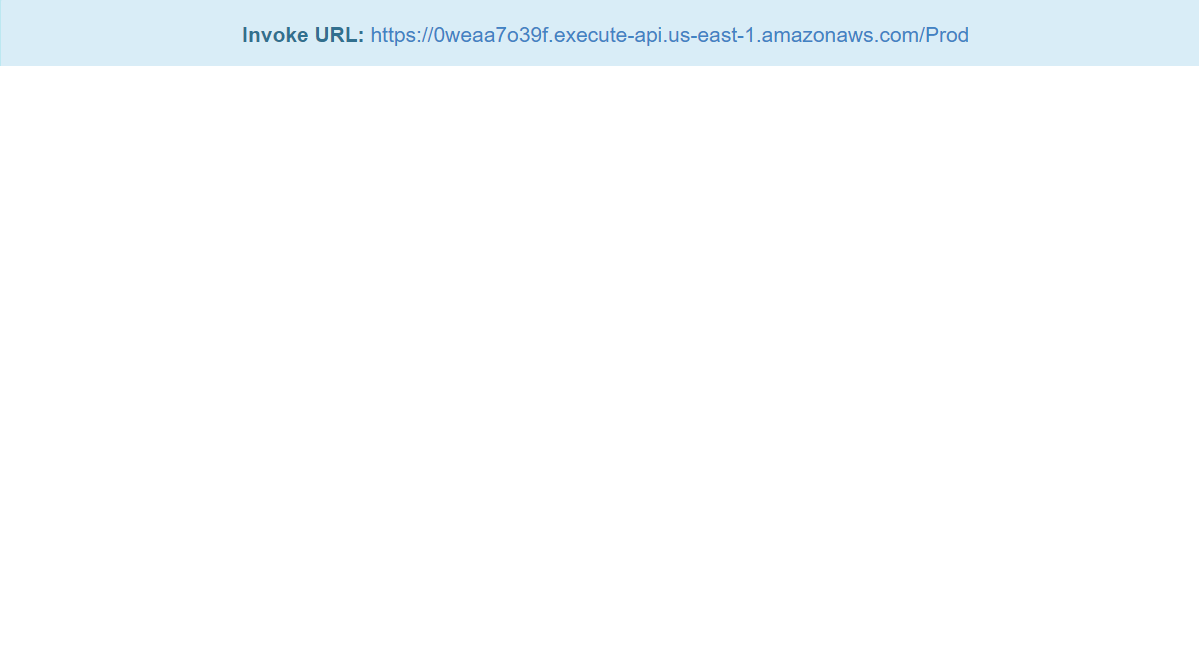
- Stage name: Enter Prod.

- Stage description: Enter Prod.

- Deployment description: Leave this box blank.

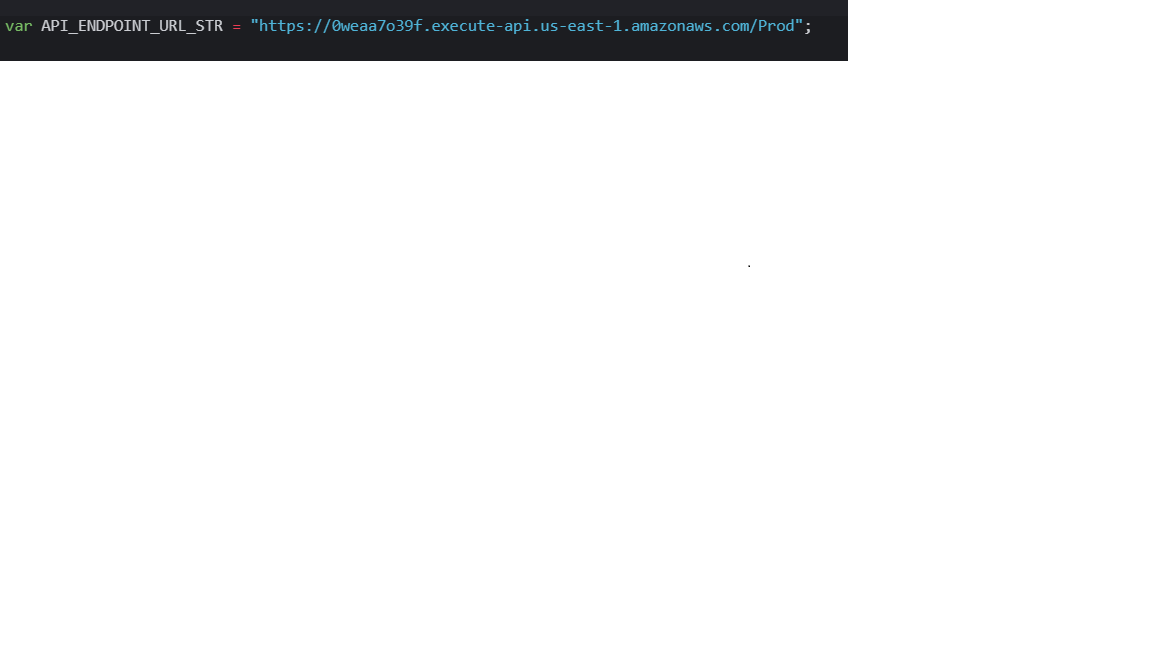
- Choose Deploy. (Ignore any warnings.)

**28.** Note down the URL from the ListSearch API

****

**Connect the New API (Mock) Endpoint to Your Website**

**1.** Open (double-click) the file called config.js and add in the URL API endpoint

****

**2.** Open (double-click) the upload\_items file.

**3.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  S3API = new AWS.S3({  apiVersion: "2006-03-01",  region: "us-east-1"  }),  FS = require("fs"),  bucket\_name\_str = "redesignfrontend";  function uploadItemAsBinary(key\_name\_str, content\_type\_str, bin){  var params = {  Bucket: bucket\_name\_str,  Key: key\_name\_str,  Body: bin,  ContentType: content\_type\_str,  CacheControl: "max-age=0"  };  S3API.putObject(params, function(error, data){  console.log(error, data);  });  }  (function init(){  var  file\_path\_str = "/home/ubuntu/environment/resources/",  file\_name\_str = "config.js",  config\_bin = FS.readFileSync(file\_path\_str + file\_name\_str);  uploadItemAsBinary(file\_name\_str, "text/javascript", config\_bin);  })(); |

**4.** Run **node upload\_items.js**

****

**5.** Navigate back to your Amazon S3 website.

When the webpage opens, it will start to query for all listings. However, you will notice a new error message: Failed due to CORS.

**Enable CORS on Your Endpoint**

**1.** Navigate to the Amazon API gateway console and view your ListSearch API.

**2.** In the left navigation pane, go to Resources.

**3.** Above POST, choose the resource /.

**4.** Choose Actions and select Enable CORS.

**5.** Select the top two check boxes for DEFAULT 4XX and DEFAULT 5XX.

**6.** Leave the other settings at their default settings.

**7.** Choose Enable CORS and replace existing CORS headers.

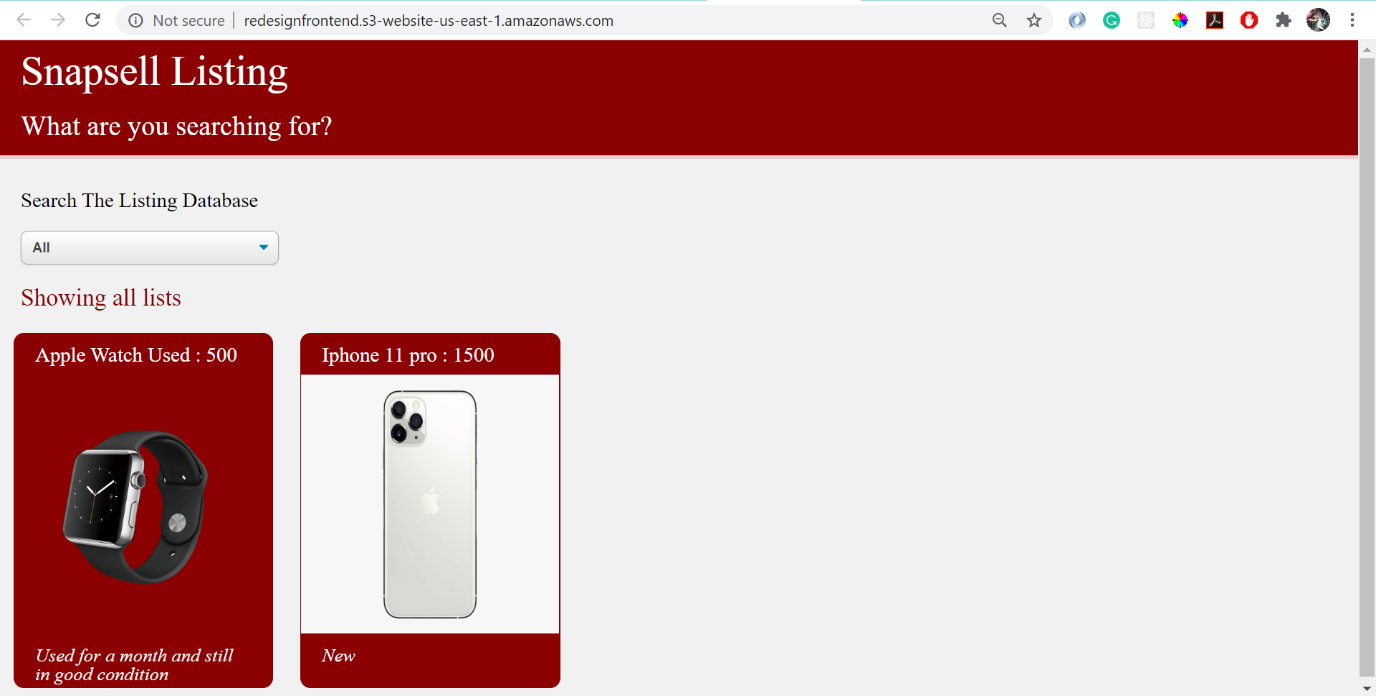
**8.** Choose Yes, replace existing values.

**9.** Choose Actions and then choose Deploy API.

**10.** For Deployment stage, select Prod.

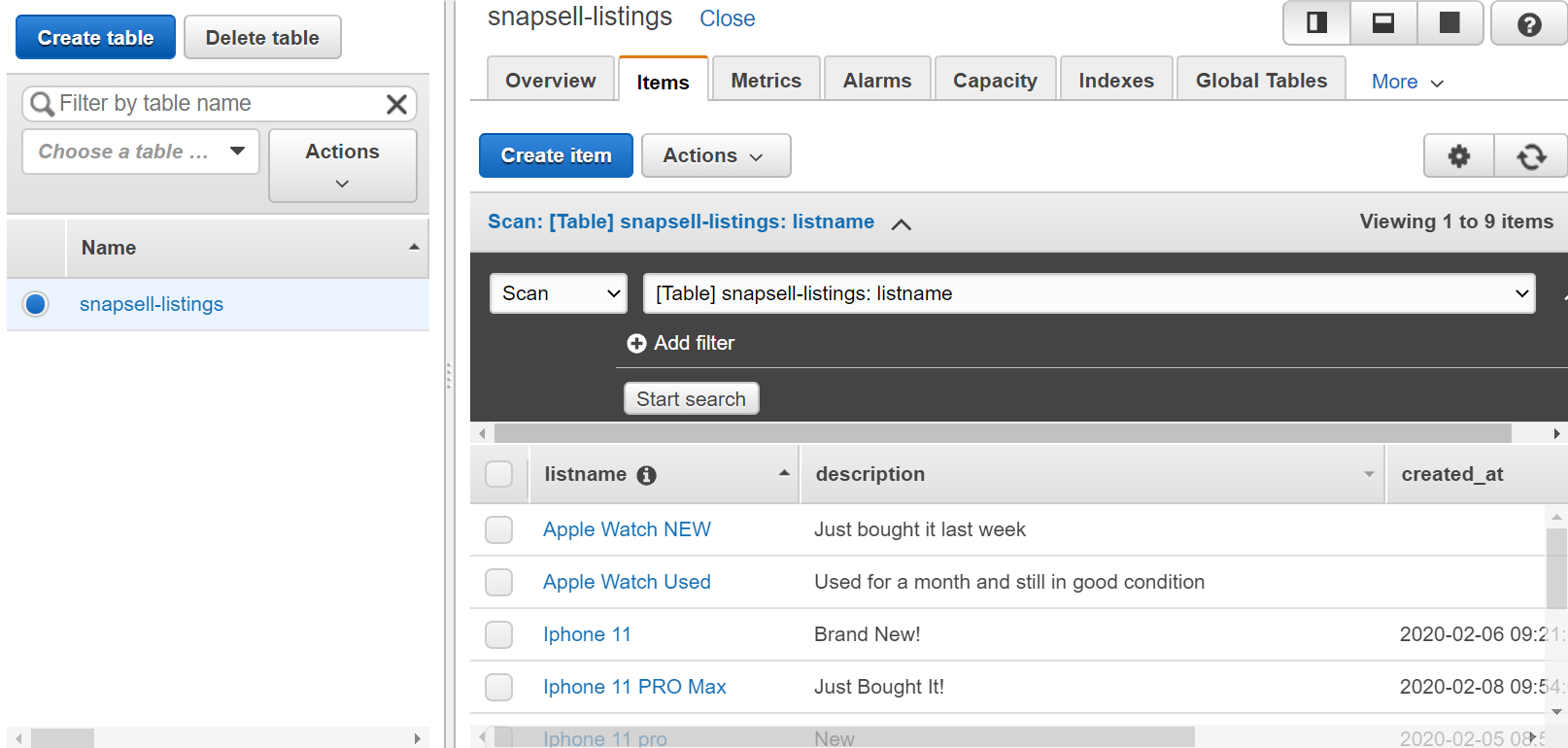
**11.** Choose Deploy to deploy to your prod stage. (Ignore any warnings.)

**12.** Navigate back to your Amazon S3 website and refresh the page. You should see 2 listings being displayed

****

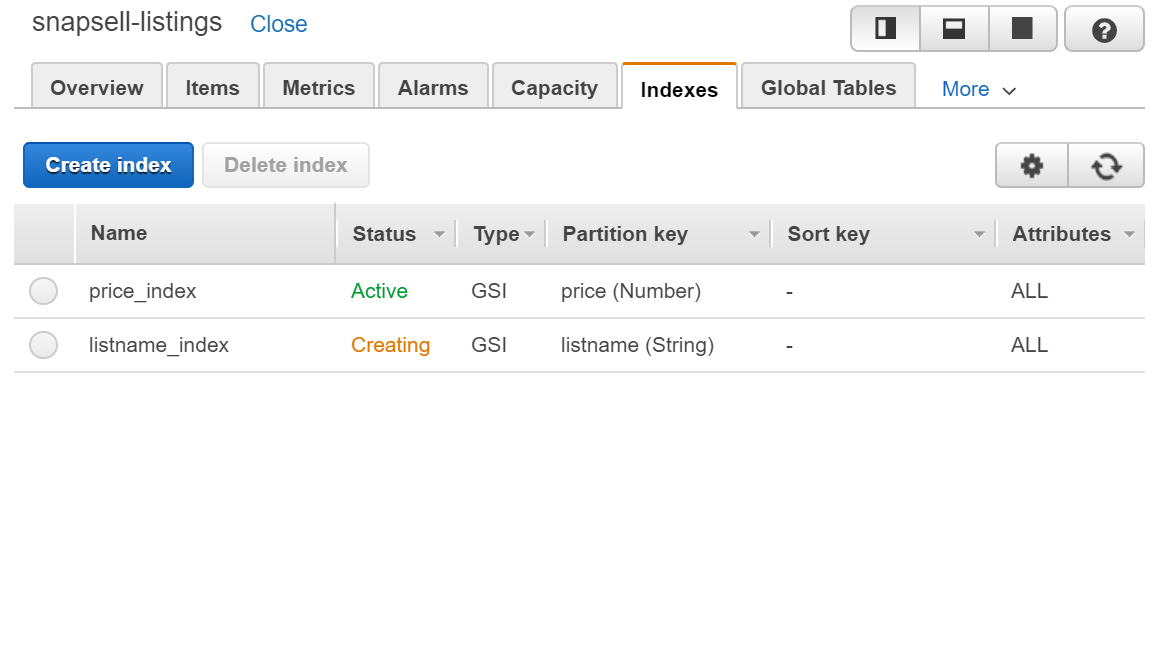
**Populate the Database**

\*Ensure that database was populated from the previous parts



**Use the AWS SDK to Create a Lambda Function that Queries a Database**

\*Following previous parts on adding index, add listname\_index



**1.** Open (double-click) the query\_cats file.

**2.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  DDB = new AWS.DynamoDB({  apiVersion: "2012-08-10",  region: "us-east-1"  });    function queryIndex(listname\_str, cb){  var  params = {  ExpressionAttributeValues: {  ":listname": {  S: listname\_str  }  },  KeyConditionExpression: "listname = :listname",  TableName: "snapsell-listings",  IndexName: "listname\_index"  };  DDB.query(params, function(err, data){  var  cat\_reply\_arr = [];  if(err){  throw err;  }  if(data.Items.length === 0){  return cb(null, []);  }  for(var i\_int = 0; i\_int < data.Items.length; i\_int += 1){  cat\_reply\_arr.push(data.Items[i\_int]);  }  cb(null, cat\_reply\_arr);  });  }    function scanTable(cb){  var  params = {  TableName: "snapsell-listings"  };  DDB.scan(params, function(err, data){  if(err){  throw err;  }  cb(null, data.Items);  });  }    (function init(){  var  listname\_str = "all",  cb = null;  if(process.argv[2] !== undefined){  console.log("Local test for " + process.argv[2]);  listname\_str = process.argv[2];  cb = console.log;  }else{  console.log("Running in lambda");  console.log(event);  cb = callback; //becomes available in lambda  listname\_str = event.listname\_str;  }  if(listname\_str === "All"){  scanTable(cb);  }else{  queryIndex(listname\_str, cb);  }  })();  //}; |

**3.** Run **node query\_cats.js "samsung 100"**

**(You should be able to retrieve the details for ‘samsung 100’**

****

**Use the AWS SDK to Upload Your Lambda Deployment Package**

**1.** Modify the codes in query\_cats.js file

|  |
| --- |
| //At the very top of the query\_cats.js file, add this line:  exports.handler = function(event, context, callback){  …  //At the bottom of the file (after all your code), add this line:  }; |

**2.** To create a package .zip file with your new query\_cats code, run this command in the AWS Cloud9 terminal: **zip website\_api\_code.zip query\_cats.js**

**3.** Open the upload\_items file and modify it slightly as follows and save the file.

- For the file\_path variable, replace the path name with ./ so that it uses this current directory where your code is going to run. (That is, replace /home/ubuntu/environment/resources/ with the string ./)

- For the file\_name variable, replace the string config.js with the string website\_api\_code.zip.

- Replace the string text/javascript with the string application/zip.

**4.** Run **node upload\_items.js**

****

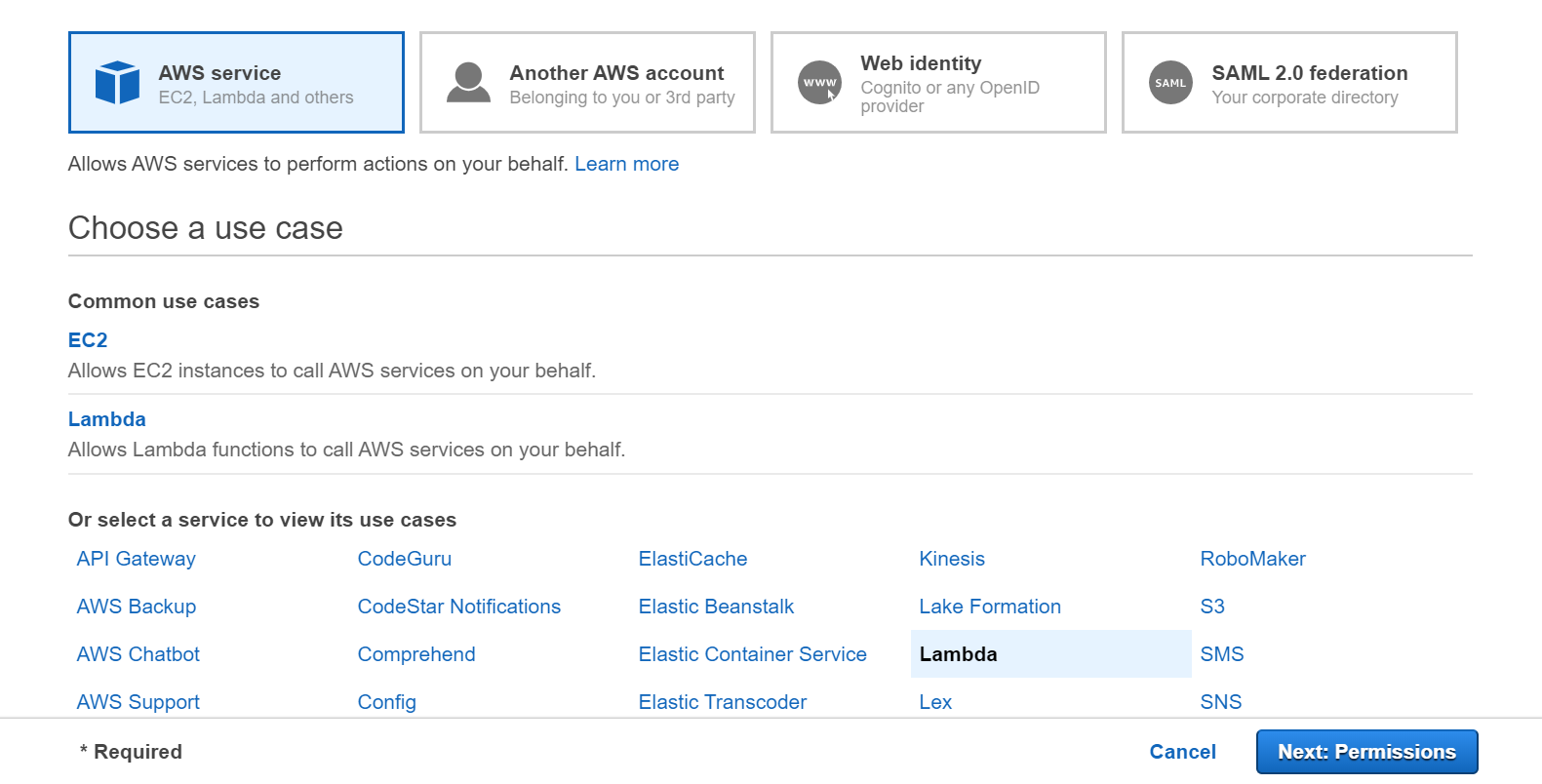
**Use the AWS SDK to Call Your Deployment Package and Configure Your Lambda Function**

**1.** You will need to create IAM role first

**2.** Under services, click on IAM

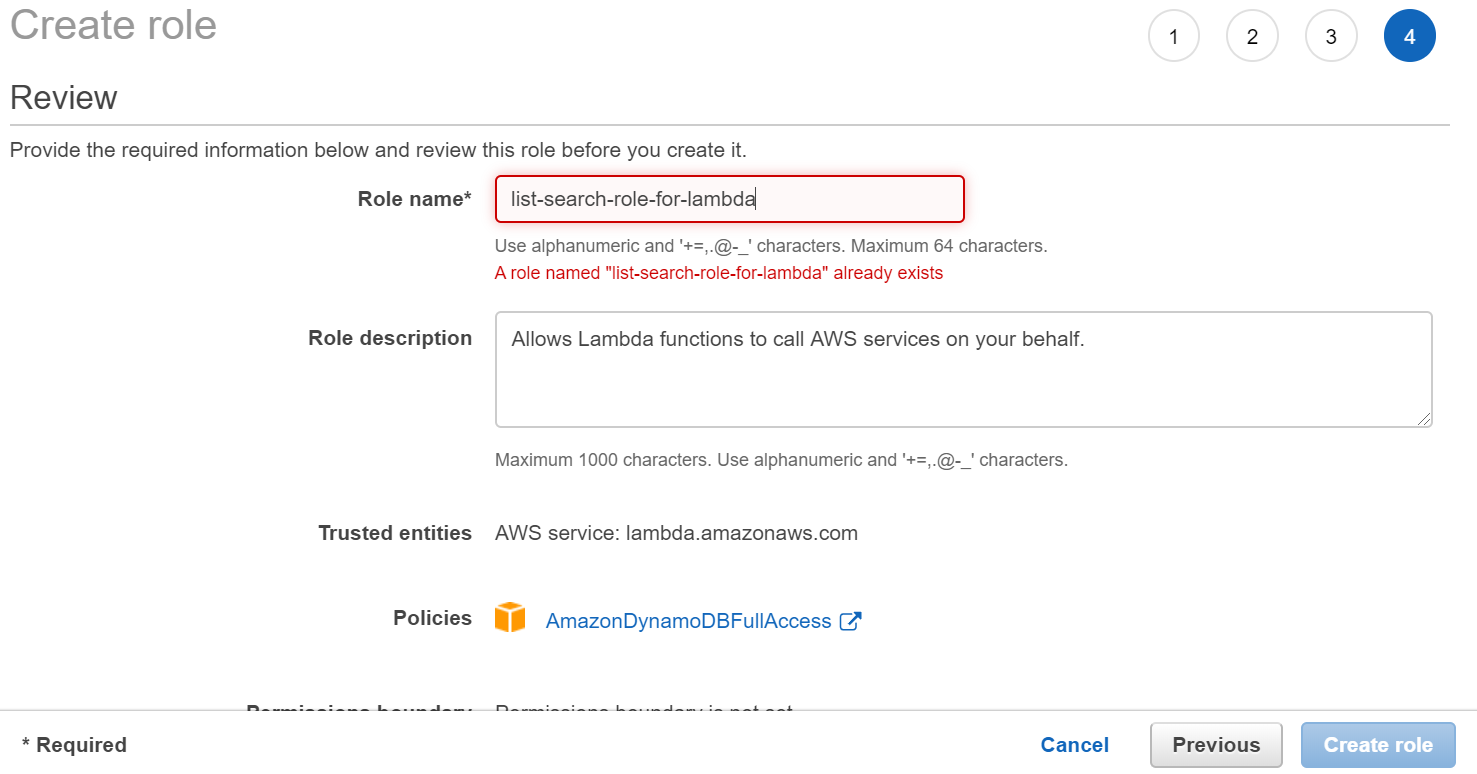
**3.** Under access management, click on roles and create roles

**4.** Select the lambda and click next



**5.** Under filter policies, search **AmazonDynamoDBReadOnlyAccess** and **AWSLambdaExecute** and select them

**6.** Change the role name to: list-search-role-for-lambda

****

**7.** Open (double-click) the create\_lambda file.

**8.** Change the codes accordingly

|  |
| --- |
| var  AWS = require("aws-sdk"),  LAMBDA = new AWS.Lambda({  apiVersion: "2015-03-31",  region: "us-east-1"  });  function createLambdaFromZip(){  var  params = {  Code: {  S3Bucket: "redesignfrontend",  S3Key: "website\_api\_code.zip"  },  Description: "Listing website",  FunctionName: "ListSearch",  Handler: "query\_cats.handler",  MemorySize: 128,  Publish: true,  Role: **"arn:aws:iam::879702147044:role/list-search-role-for-lambda",**  Runtime: **"nodejs12.x"**,  Timeout: 30,  };  LAMBDA.createFunction(params, function(err, data){  console.log(err, data);  });  }  (function init(){  createLambdaFromZip();  })(); |

**9.** Run **node create\_lambda.js**

****

**Replace the REST API Mock Endpoint With Your New Lambda Function**

\*You must now connect your Amazon S3 website to your backend Lambda function.

**1.** From the AWS Management Console, select API Gateway.

**2.** Under APIs, select the ListSearch API

**3.** Under Resources, choose POST and then choose Integration Request.

**4.** For the Integration type, change Mock to Lambda Function.

\*WARNING: Make sure that Use Lambda Proxy Integration is not selected.

**5.** For Lambda Region, choose us-east-1.

**6.** For Lambda Function, enter ListSearch.

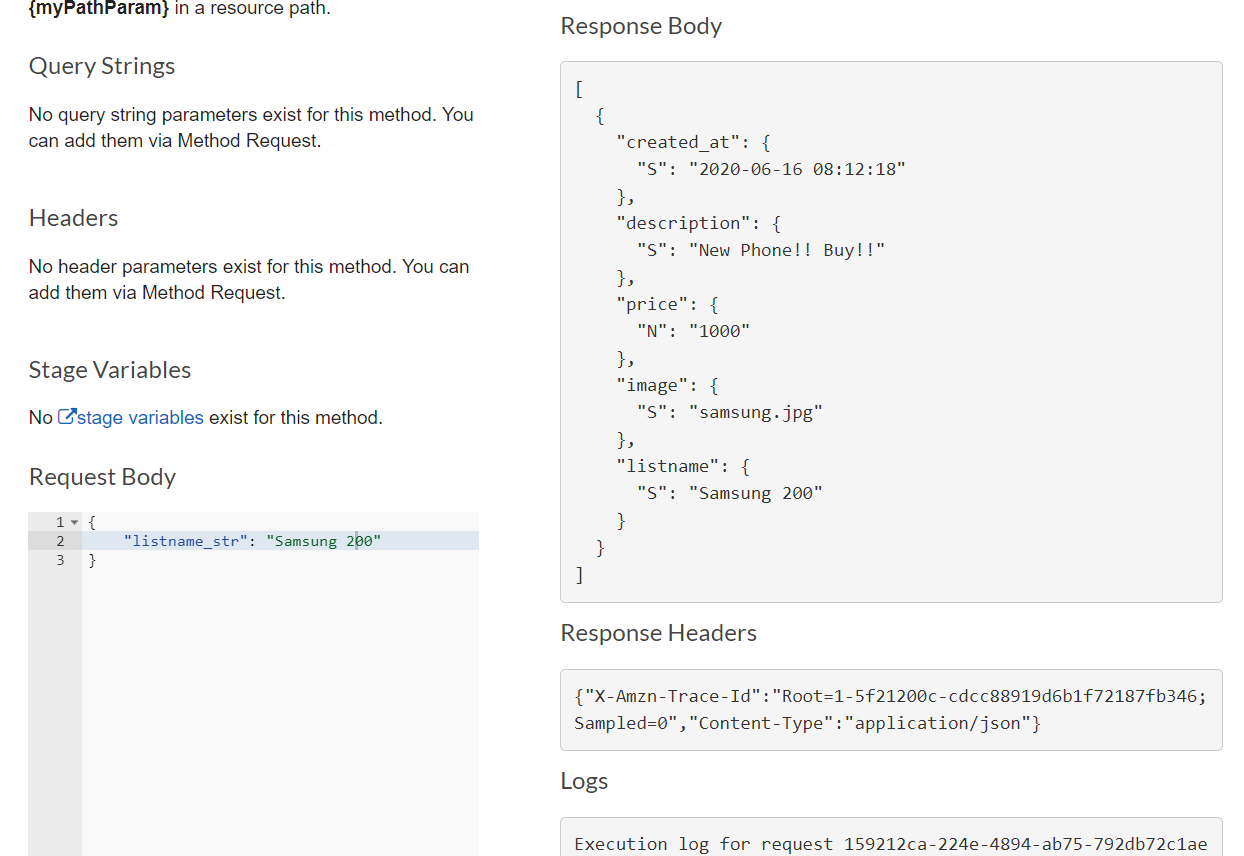
**7.** Leave Use Default Timeout selected.

**8.** Choose Save.

**9.** On the Switch to Lambda integration window, choose OK.

**10.** The console will show a message that you are adding Permissions to Lambda Function. Choose OK.

**11.** Confirm that it works:



**Re-Enable CORS**

**1.** Choose Actions and then choose Enable CORS. Like before, select DEFAULT 4XX and DEFAULT 5XX.

**2.** Choose Enable CORS and replace existing CORS headers.

**3.** On the Confirm method changes window, choose Yes, replace existing values

**4.** Choose Actions again and then choose Deploy API.

**5.** For Deployment stage, select Prod.

**6.** Choose Deploy.

**7.** Navigate back to your Amazon S3 website. Refresh the page to see all listings, and try searching for a listing using the dropdown list to filter your search.



****