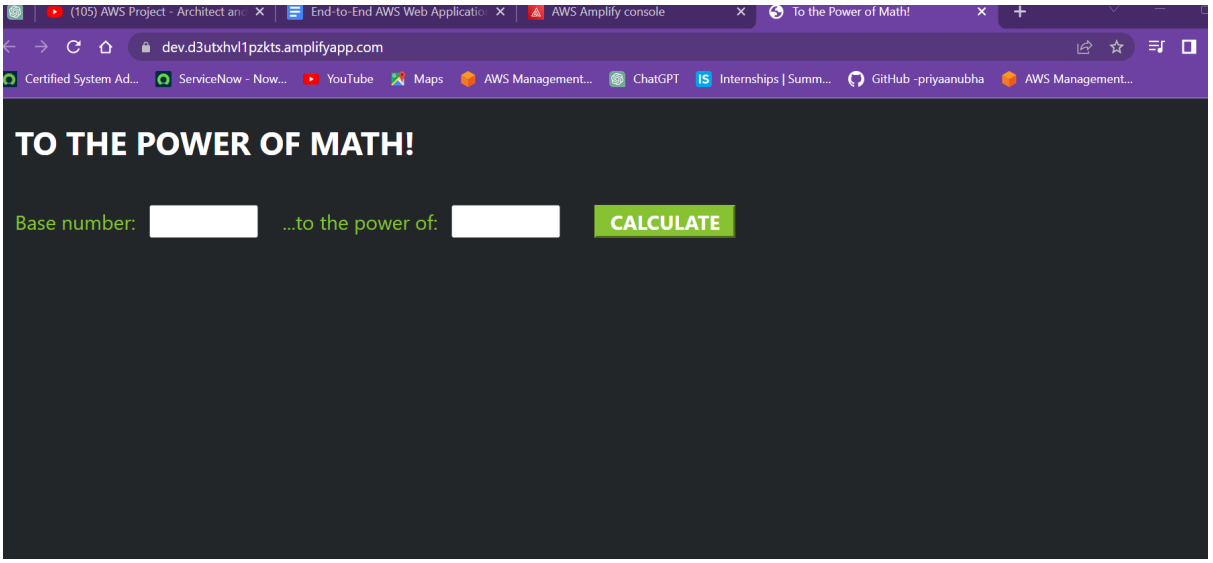
End-to-End AWS Web Application - creation and deployment on AWS



Here, I have created and deployed my first AWS Web application “TO THE POWER OF MATH!”

Services used in this project:
1: AWS Amplify - AWS Amplify is a development platform provided by Amazon Web Services (AWS) that enables developers to build full-stack web and mobile applications quickly and easily. It simplifies the process of developing and deploying applications by providing a set of tools, services, and libraries.

2: AWS Lambda - AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). It allows you to run your code without provisioning or managing servers. With Lambda, you can execute your application code in response to events, such as changes to data in an Amazon S3 bucket, updates to a DynamoDB table, or HTTP requests.

3: Amazon API Gateway - Amazon API Gateway is a fully managed service provided by Amazon Web Services (AWS) that allows you to create, deploy, and manage APIs (Application Programming Interfaces) for your applications. It acts as a gateway between your backend services and your client applications, providing a secure and scalable interface for accessing your API.

4: Amazon DynamoDB - Amazon DynamoDB is a fully managed NoSQL database service provided by Amazon Web Services (AWS). It is designed to provide fast and predictable performance at any scale, making it suitable for applications that require low-latency data access.

5: AWS identity & Access Management - AWS Identity and Access Management (IAM) is a web service provided by Amazon Web Services (AWS) that allows you to manage user access and permissions for your AWS resources. IAM enables you to securely control access to AWS services and resources within your AWS account.

Steps to create our Web App
1: Create website using HTML and deployed on AWS using AWS Amplify
2: Involve the maths functionality using AWS Lambda
3. Public Endpoint URL that invokes lambda function for our WebApp when accessing that using Amazon API Gateway.
4: Store/return the maths result using AWS DynamoDB
5: Handle permission using AWS IAM

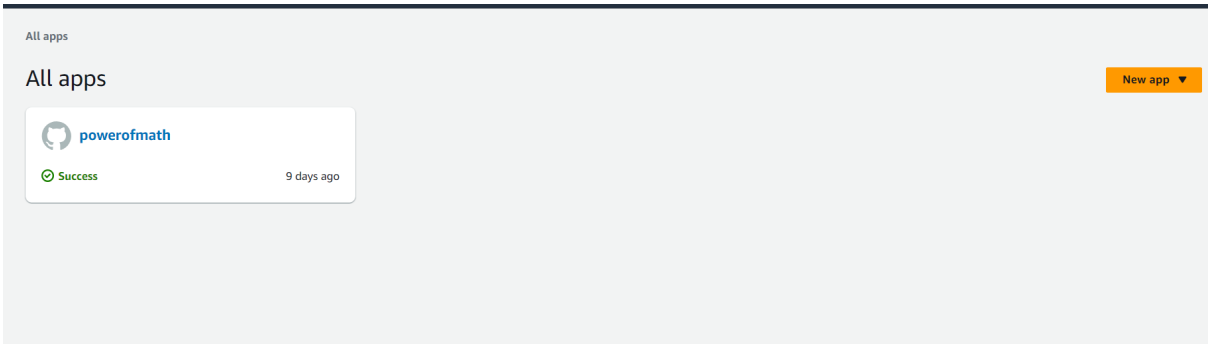
1: Created website using HTML and deployed on AWS using AWS Amplify

A> I have created index.html file using html codes as below:

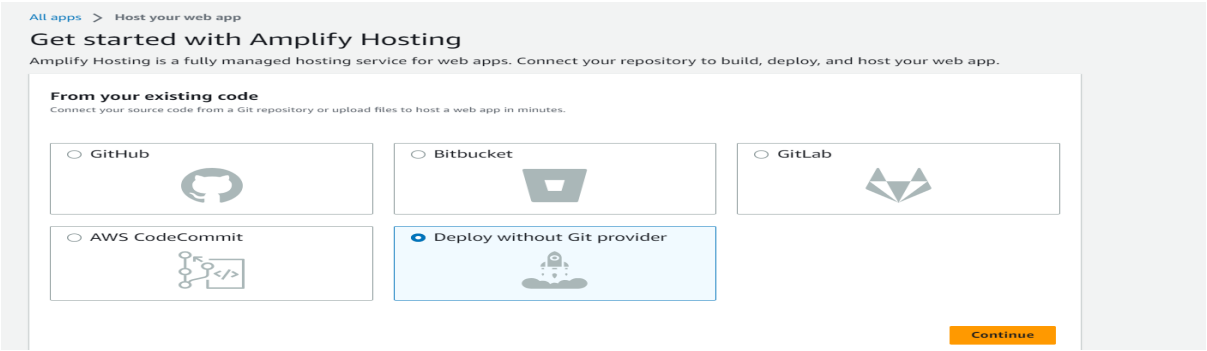
```
 <!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>To the Power of Math!</title>
</head>

<body>
   To the Power of Math!
</body>
</html>
```

And ZIP it up.

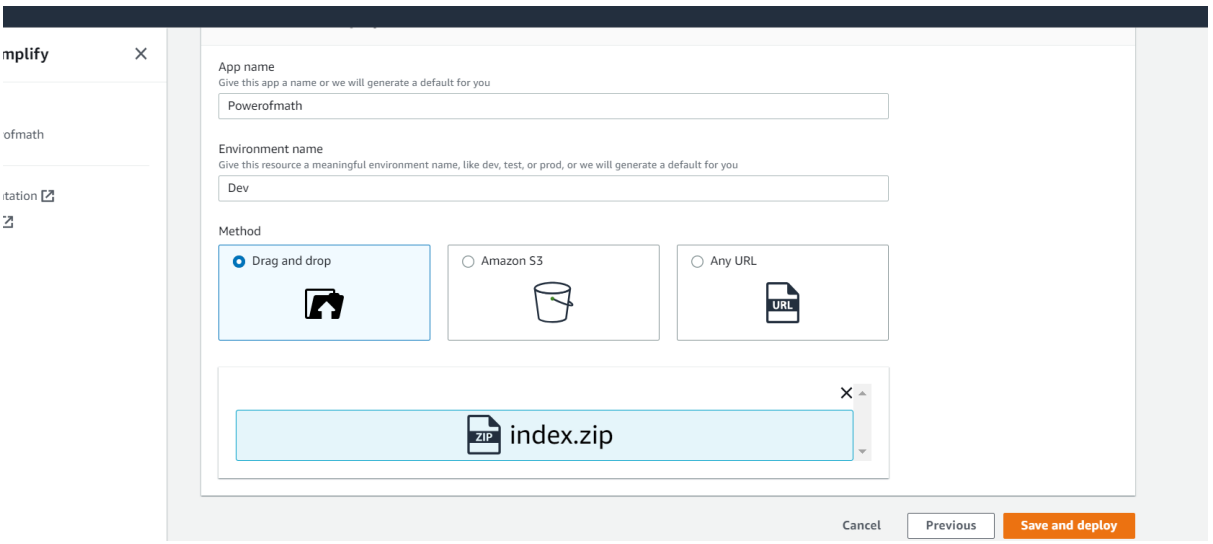
Now go to AWS Amplify



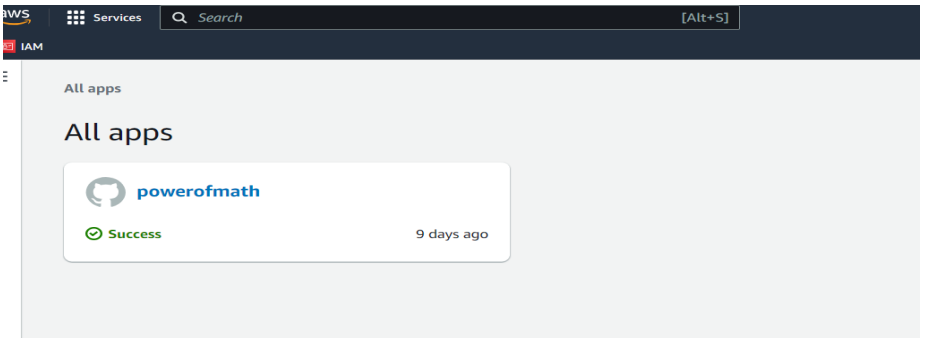
Click on New App -> Host web app



Select Deploy without Git provider and continue



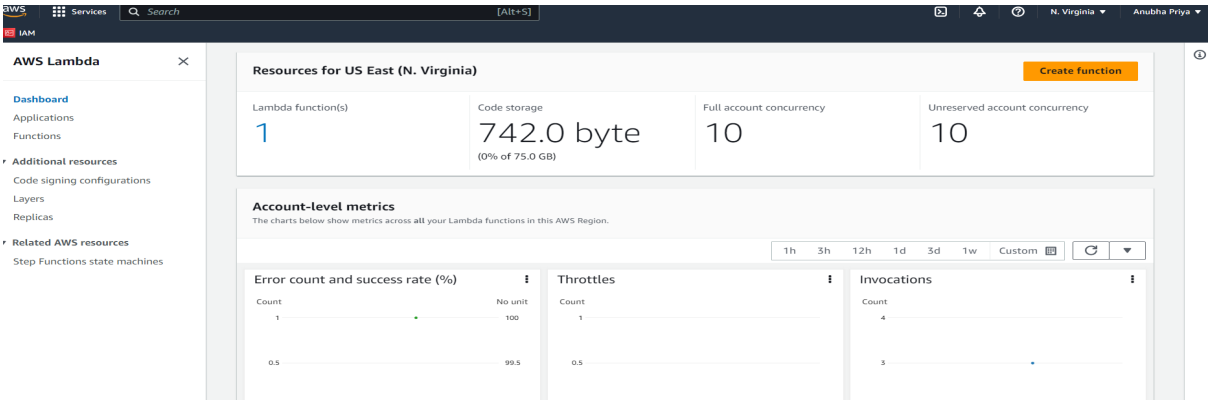
Give name to your app “Powerofmath”
Environment name : Dev // development
And drag and drop your HTML Website
After that click on save and deploy

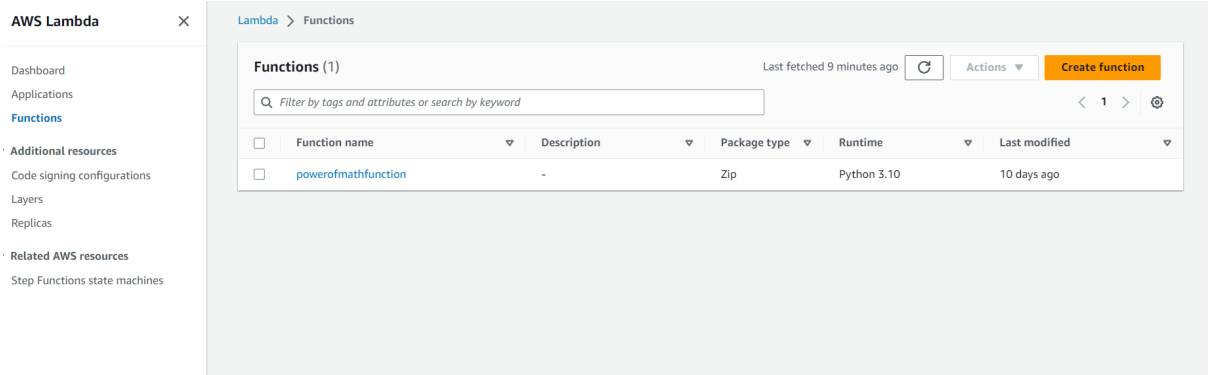


Here my web app “powerofmath” deployed successfully on AWS using AWS amplify.

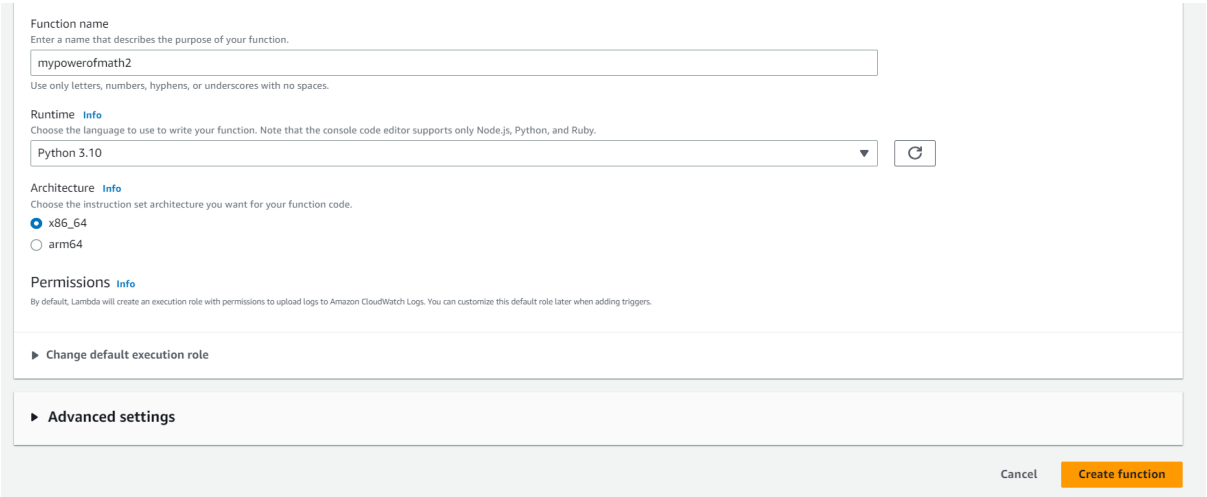
2: Involve the maths functionality using AWS Lambda

Now, we have to create lambda function (Code that run upon some triggers serverlessly)that will be triggered when user will try to access our “Powerofmath”

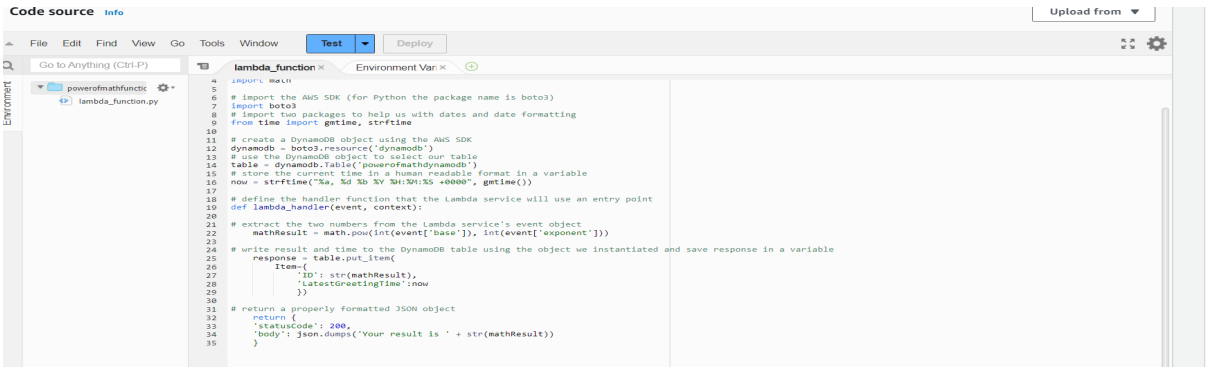




Create function



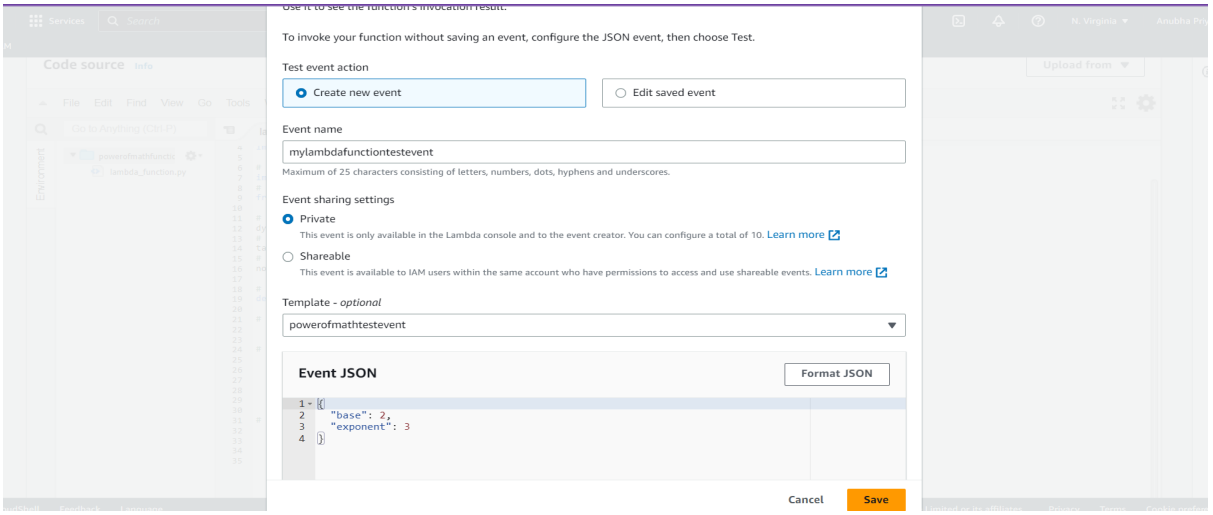
Function created now, go to code source of lambda function and paste the below code:



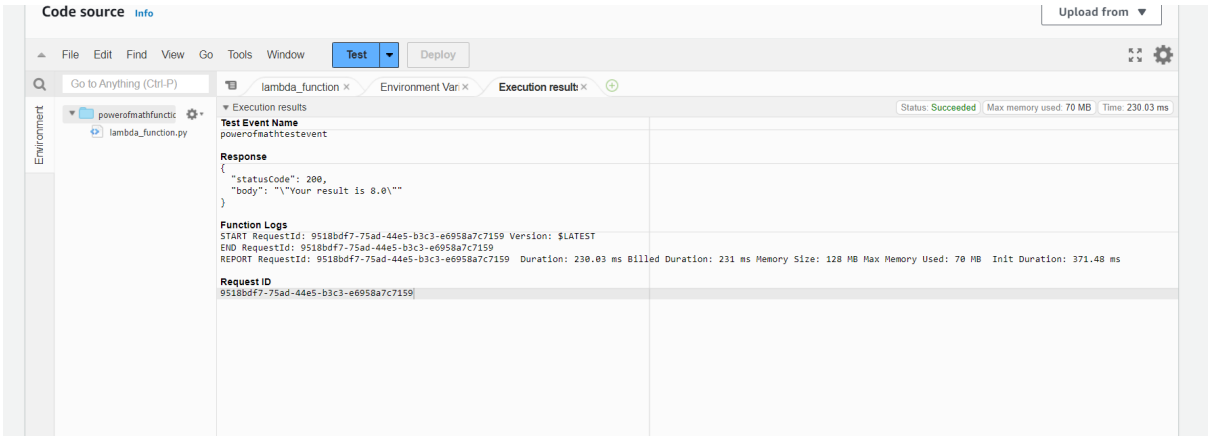

```
# import the JSON utility package
import json
# import the Python math library
import math

# define the handler function that the Lambda service will use an entry point
def lambda_handler(event, context):

# extract the two numbers from the Lambda service's event object
    mathResult = math.pow(int(event['base']), int(event['exponent']))   //our maths result
```

```
# return a properly formatted JSON object
return {
'statusCode': 200,
'body': json.dumps('Your result is ' + str(mathResult))
}
```

Copy pasted above code my lambda function
Ctrl+S
Click on deploy
Now click on test and create test event to test our lambda function



Save test event and click on test

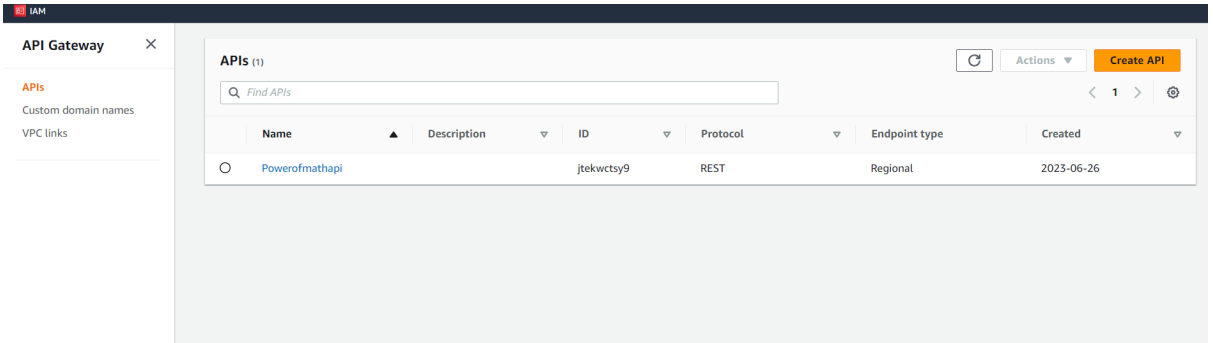


Result is 8 that is correct hence our lambda function is working as expected.

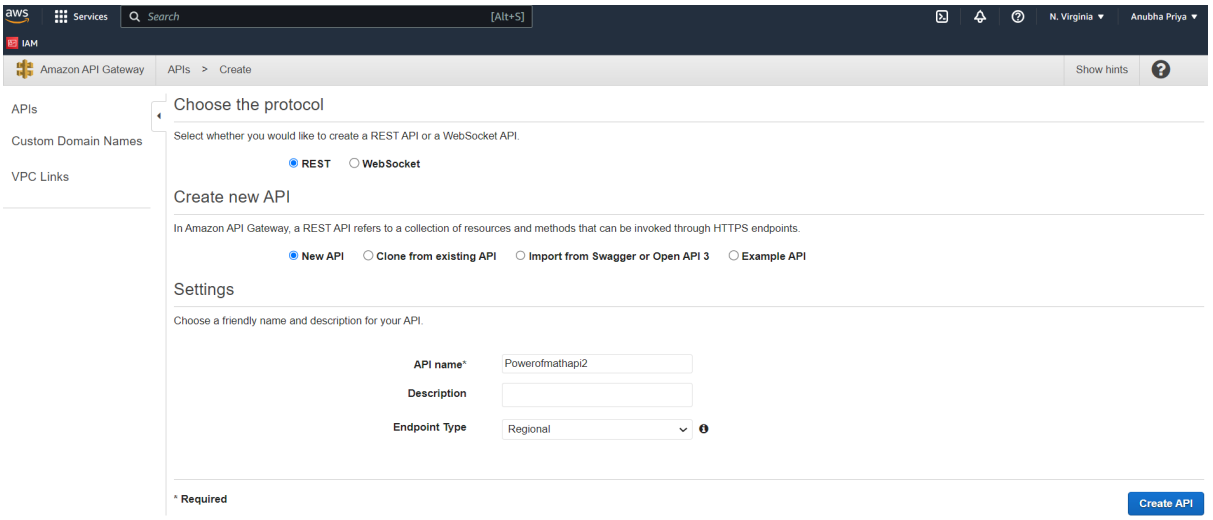
3. Public Endpoint URL that invokes lambda function for our WebApp when accessing that using Amazon API Gateway.

Now we need one public endpoint URL which invokes our lambda function for our webApp when users are trying to access it and that we will create using Amazon API Gateway.

Now go to API (Application and programming gateway) Gateway - used to build HTTP,REST and websocket API

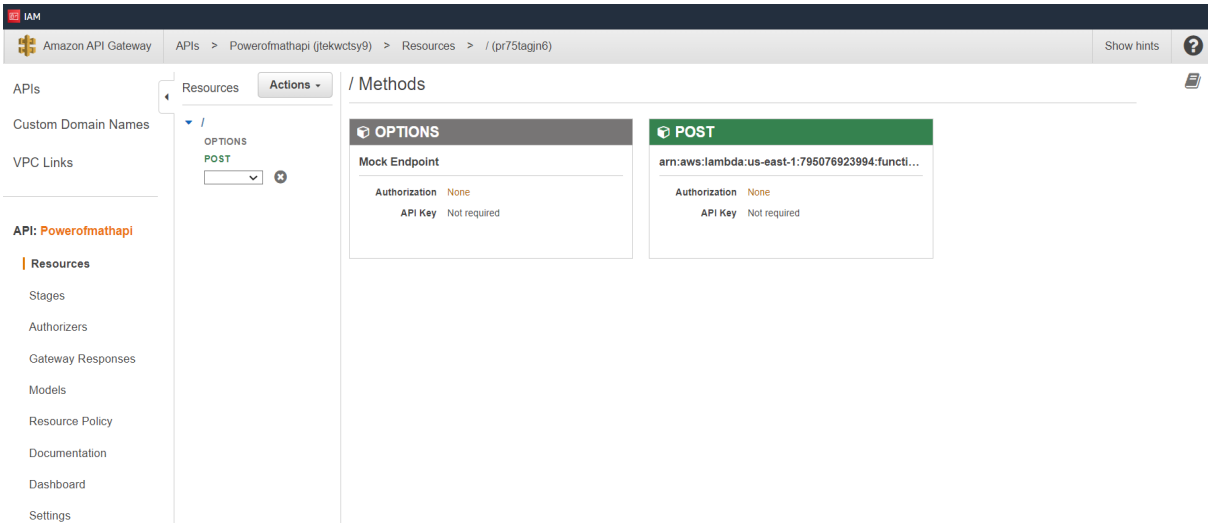


Go to REST API

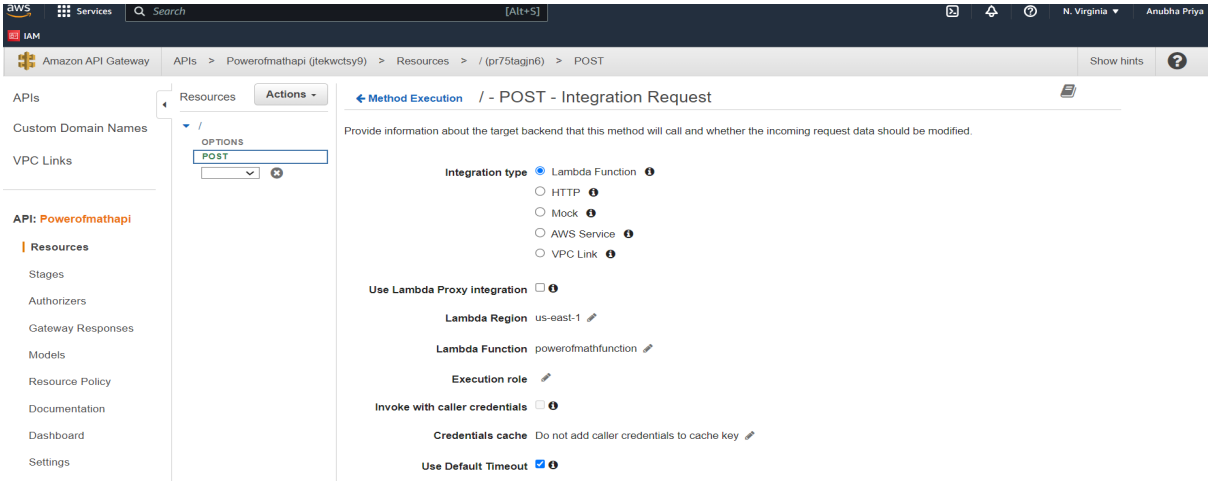


Click on create API

Now, click on API created, select resources -> select "/" ->Actions -> create method -> select POST ->refresh on green colour icon next to post.

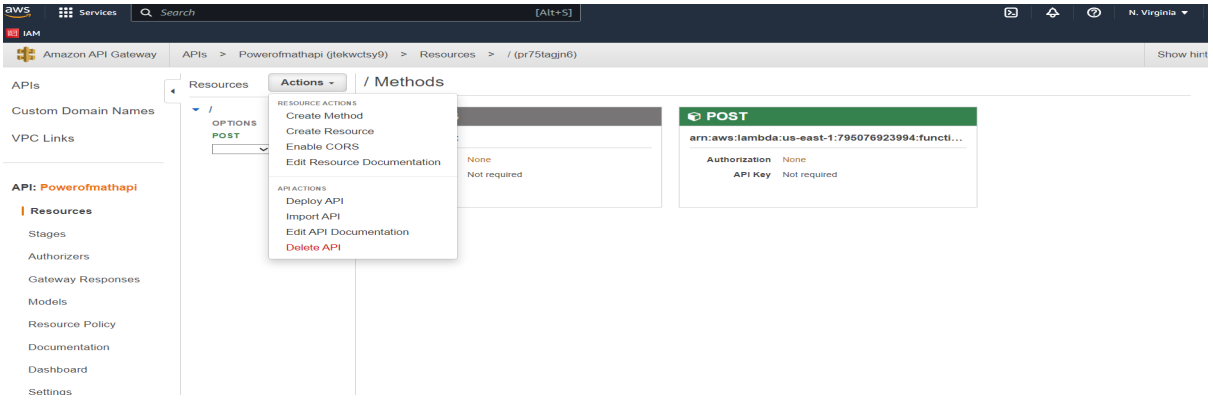


Now, we have to do integration of our lambda function to out API method

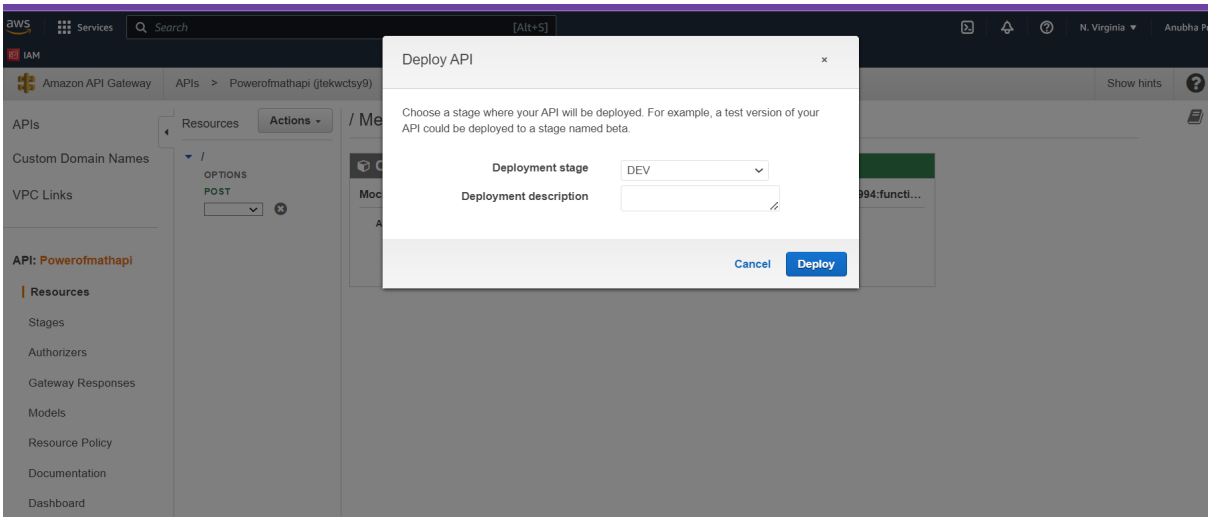


Integration type: Lambda function
And give name of our function : powerofmathfunction
Here, we have given permission to our API Gateway to invoke our lambda function.

Now, We have to enable CORS for cross origin resource sharing so that our API,Lambda and all resources will be able to work smoothly without any origin issues.

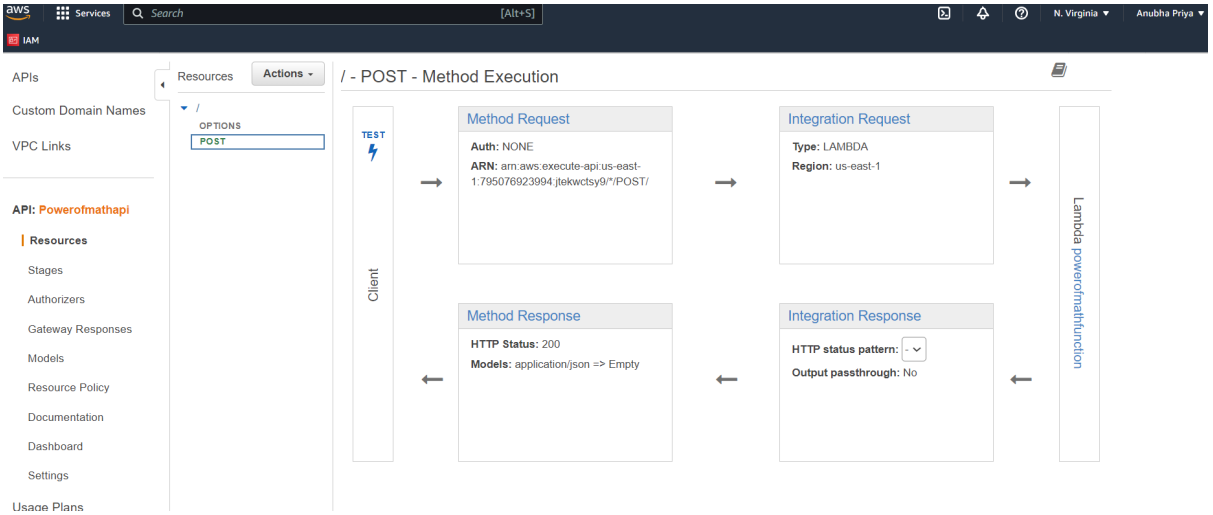


Now, We have to deploy our API. Select “/” -> Actions -> Deploy API -> Set stage ->Dev

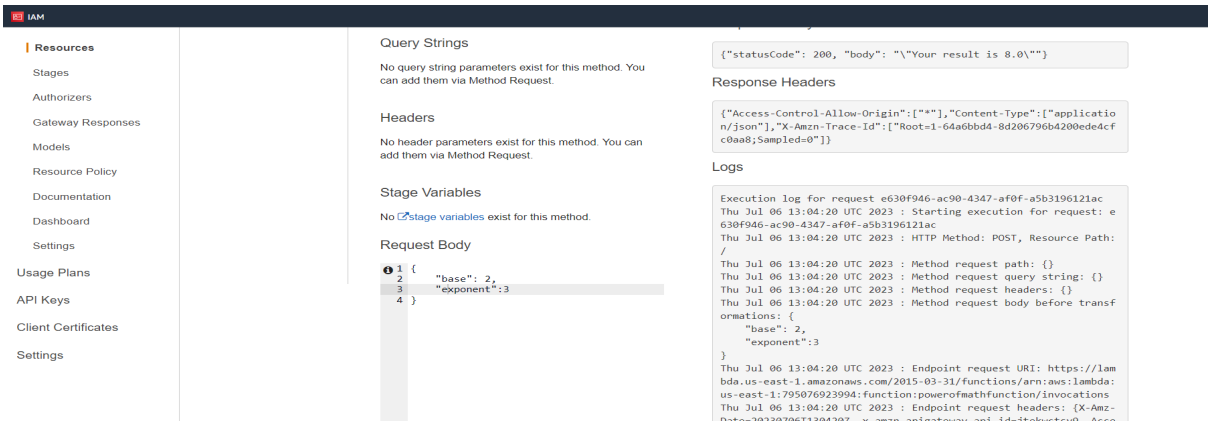


Deploy and copy Invoke URL handy for future use.

And, here is the workflow of API



By clicking on Test symbol we can create one test event for our API Gateway



Here, our result is 8.0 which is as expected.

4: Store/return the maths result using AWS DynamoDB

Now we have to create our DynamoDB to store our result as a database.

Go to DynamoDB (No SQL database) ->create table

Table Name : Powerofmath
Partition Key : ID

Leave all details as it is and create a table.

IAM

Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

DynamoDB > Tables > Create table

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.

Table name
This will be used to identify your table.
powerofmath2
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

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.
ID | String
1 to 255 characters and case sensitive.

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.

Table settings

- (selected) **Default settings** — The fastest way to create your table. You can modify these settings now or after your table has been created.
- **Customize settings** — Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

IAM

Table settings

- (selected) **Default settings** — The fastest way to create your table. You can modify these settings now or after your table has been created.
- **Customize settings** — Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Table class	DynamoDB Standard	Yes
Deletion protection	Off	Yes

Tags

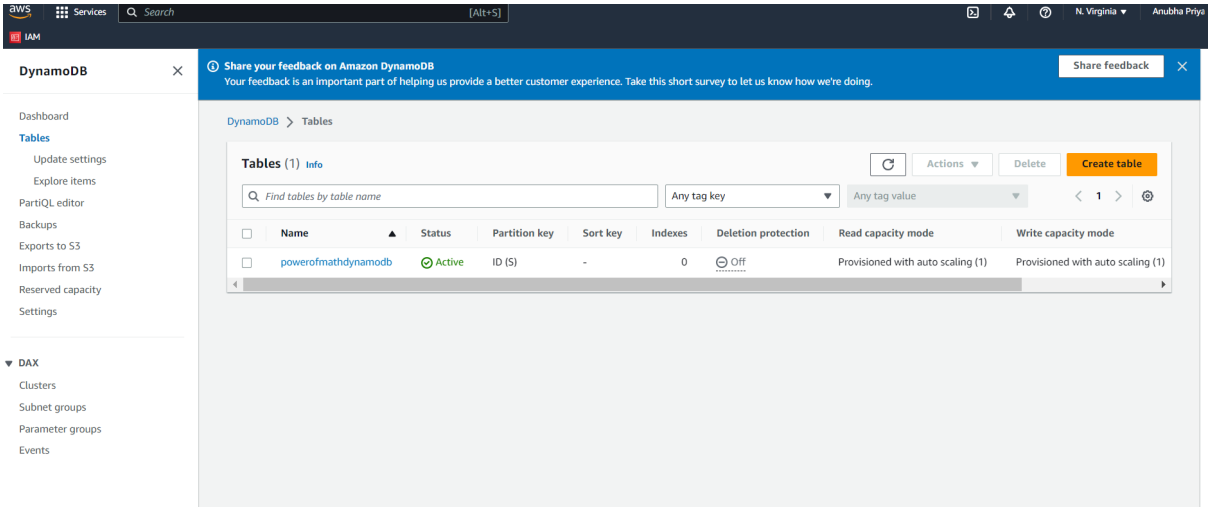
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

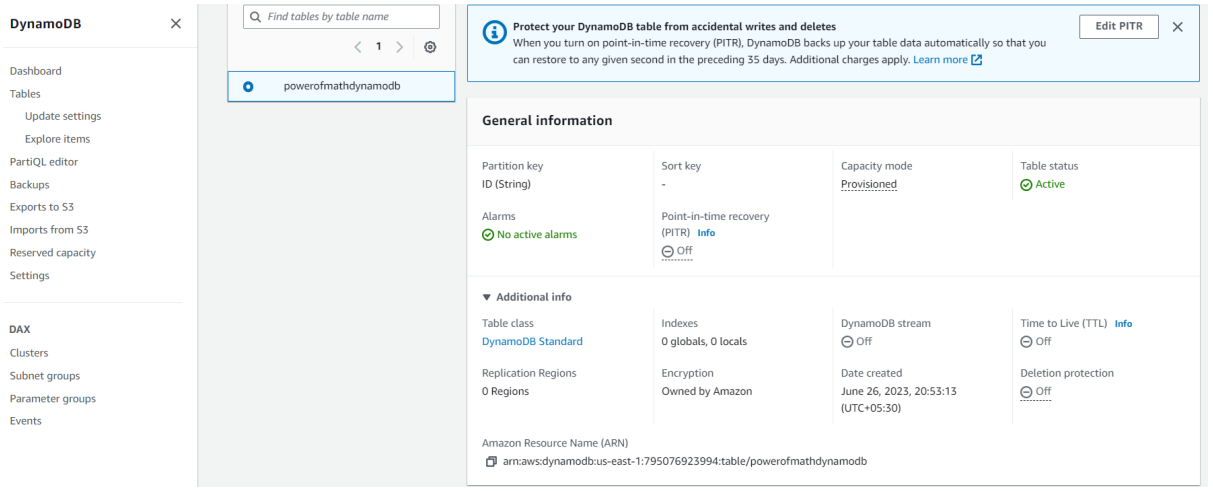
Add new tag

You can add 50 more tags.

Cancel | Create table



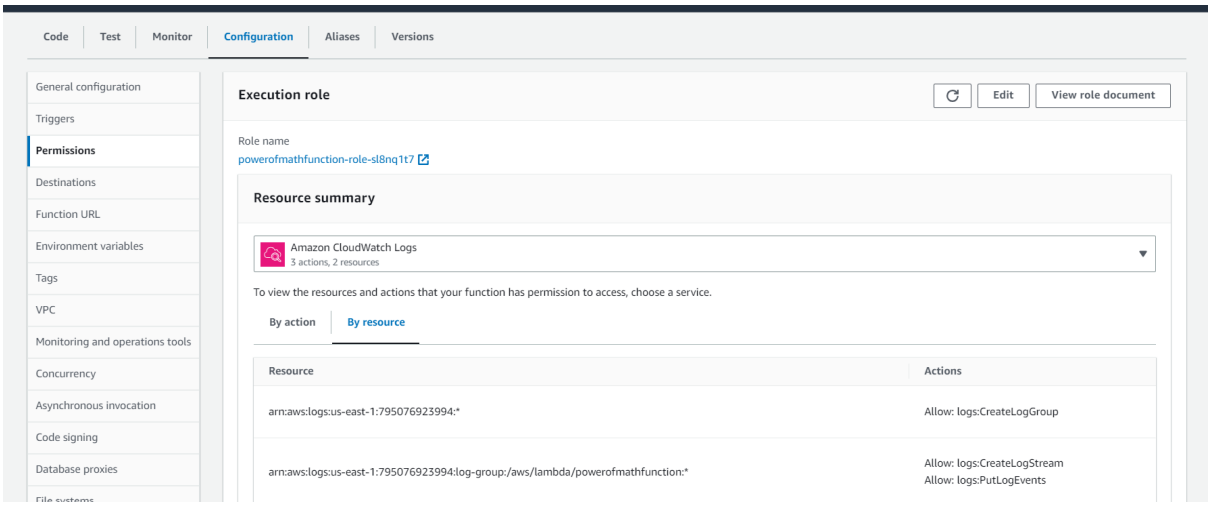
Here our DynamoDB has been created . Open it under general additional information copy Amazon resource name (ARN) for future use.

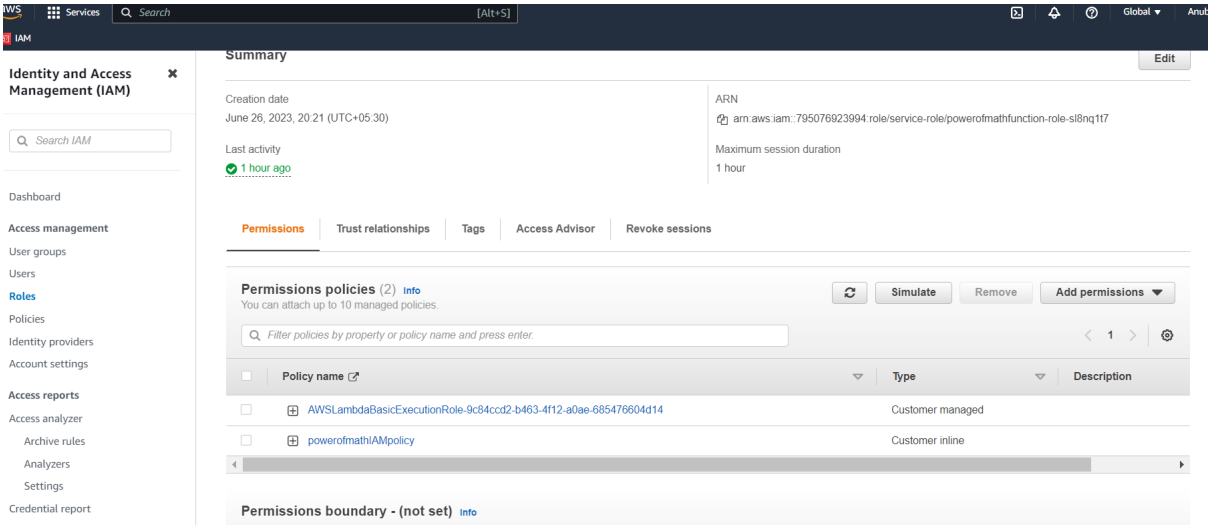


5: Handle permission using AWS IAM

Now, we have to set up IAM permission for our Lambda function.

Go to Lambda function -> configuration -> Permissions -> click on role name -> it will take us to IAM page.





Now we have to add permissions policies here -> add permissions -> create inline policy -> JSON and paste below code:

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Sid": "VisualEditor0",
                        "Effect": "Allow",
                        "Action": [
                                "dynamodb:PutItem",
                                "dynamodb:DeleteItem",
                                "dynamodb:GetItem",
                                "dynamodb:Scan",
                                "dynamodb:Query",
                                "dynamodb:UpdateItem"
                        ],
                        "Resource":
"arn:aws:dynamodb:us-east-1:795076923994:table/powerofmathdynamodb"
                }
        ]
}
```

// for resource name have pasted my dynamoDB ARN

Then review the policy and have given it the name DynamoDB Policy and created it.

—----------

Now we have to update our Lambda JSON code with the IAM changes with the below JSON code:

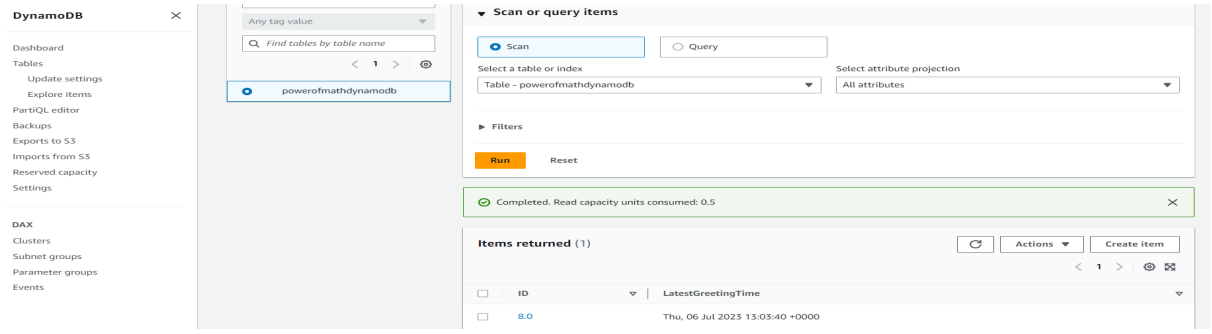
```
# import the JSON utility package
import json
# import the Python math library
import math

# import the AWS SDK (for Python the package name is boto3)
import boto3
# import two packages to help us with dates and date formatting
from time import gmtime, strftime

# create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# use the DynamoDB object to select our table
table = dynamodb.Table('PowerOfMathDatabase')
# store the current time in a human readable format in a variable
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

# define the handler function that the Lambda service will use an entry point
def lambda_handler(event, context):

# extract the two numbers from the Lambda service's event object
    mathResult = math.pow(int(event['base']), int(event['exponent']))

# write result and time to the DynamoDB table using the object we instantiated and save
response in a variable
    response = table.put_item(
        Item={
            'ID': str(mathResult),
            'LatestGreetingTime':now
            })

# return a properly formatted JSON object
    return {
    'statusCode': 200,
    'body': json.dumps('Your result is ' + str(mathResult))
    }
```

After changing code again click on deploy to make it deployed with the new changes in our lambda function.

Test it again using the same test function

—----------

Now go to the DynamoDB table and click on explore table items -> it will show our items returned using the lambda function.



Now we have to connect our Amplify to our API Gateway so all frontend and backend will be connected and hence for that we have to update our HTML codes.

Open index.html in notepad++ and copy paste all these commands:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>To the Power of Math!</title>
   <!-- Styling for the client UI -->
   <style>
   h1 {
      color: #FFFFFF;
      font-family: system-ui;
                  margin-left: 20px;
      }
         body {
      background-color: #222629;
      }
   label {
      color: #86C232;
      font-family: system-ui;
      font-size: 20px;
      margin-left: 20px;
                  margin-top: 20px;
      }
    button {
      background-color: #86C232;
                  border-color: #86C232;
                  color: #FFFFFF;
      font-family: system-ui;
      font-size: 20px;
                  font-weight: bold;
      margin-left: 30px;
                  margin-top: 20px;
                  width: 140px;
```

```
        }
            input {
        color: #222629;
        font-family: system-ui;
        font-size: 20px;
        margin-left: 10px;
                    margin-top: 20px;
                    width: 100px;
        }
    </style>
    <script>
        // callAPI function that takes the base and exponent numbers as parameters
        var callAPI = (base,exponent)=>{
            // instantiate a headers object
            var myHeaders = new Headers();
            // add content type header to object
            myHeaders.append("Content-Type", "application/json");
            // using built in JSON utility package turn object to string and store in a variable
            var raw = JSON.stringify({"base":base,"exponent":exponent});
            // create a JSON object with parameters for API call and store in a variable
            var requestOptions = {
                method: 'POST',
                headers: myHeaders,
                body: raw,
                redirect: 'follow'
            };
            // make API call with parameters and use promises to get response
            fetch("YOUR API GATEWAY ENDPOINT", requestOptions)
            .then(response => response.text())
            .then(result => alert(JSON.parse(result).body))
            .catch(error => console.log('error', error));
        }
    </script>
</head>
<body>
    <h1>TO THE POWER OF MATH!</h1>
            <form>
        <label>Base number:</label>
        <input type="text" id="base">
        <label>...to the power of:</label>
        <input type="text" id="exponent">
        <!-- set button onClick method to call function we defined passing input values as
parameters -->
        <button type="button"
onclick="callAPI(document.getElementById('base').value,document.getElementById('expone
nt').value)">CALCULATE</button>
    </form>
</body>
```

</html>

—----------

///// here, callAPI(document.getElementById('base').value,document.getElementById('exponent').value)">CALCULATE</button>

Will actually call our API upon entering base and exponent value into our WebApp.

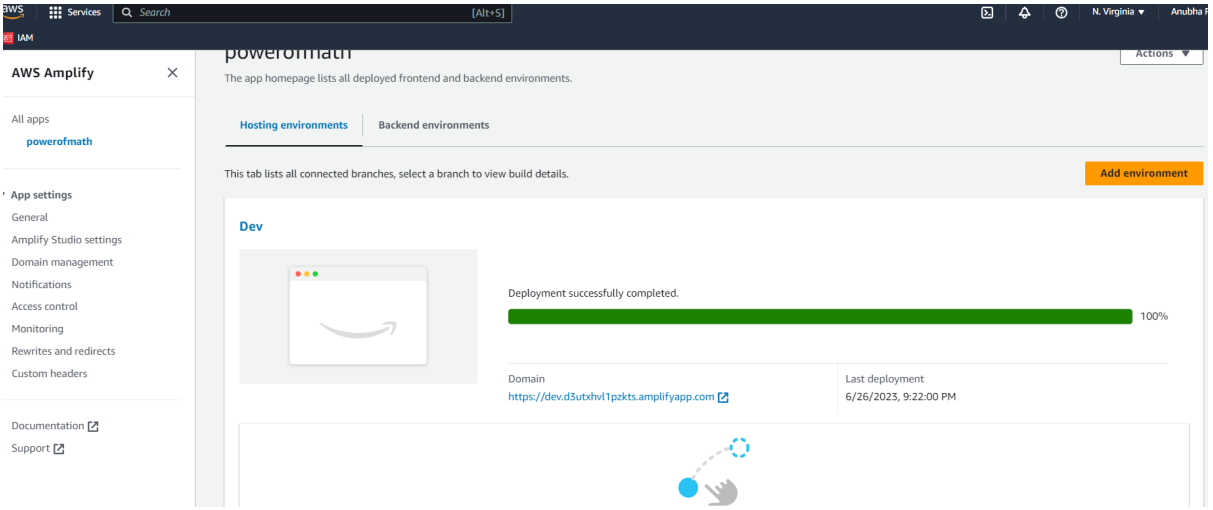
//////// fetch("YOUR API GATEWAY ENDPOINT", requestOptions) /// pasted my API Gateway URL.

```
</style>
<script>
    // callAPI function that takes the base and exponent numbers as parameters
    var callAPI = (base,exponent)=>{
        // instantiate a headers object
        var myHeaders = new Headers();
        // add content type header to object
        myHeaders.append("Content-Type", "application/json");
        // using built in JSON utility package turn object to string and store in a variable
        var raw = JSON.stringify({"base":base,"exponent":exponent});
        // create a JSON object with parameters for API call and store in a variable
        var requestOptions = {
            method: 'POST',
            headers: myHeaders,
            body: raw,
            redirect: 'follow'
        };
        // make API call with parameters and use promises to get response
        fetch("https://jtekwctsy9.execute-api.us-east-1.amazonaws.com/DEV", requestOptions)
        .then(response => response.text())
        .then(result => alert(JSON.parse(result).body))
        .catch(error => console.log('error', error));
    }
</script>
</head>
```

Save all the new HTML codes and again ZIP the HTML file.

—----------

Now, We again have to redeploy our HTML ZIP file to our AWS Amplify with the final changes.

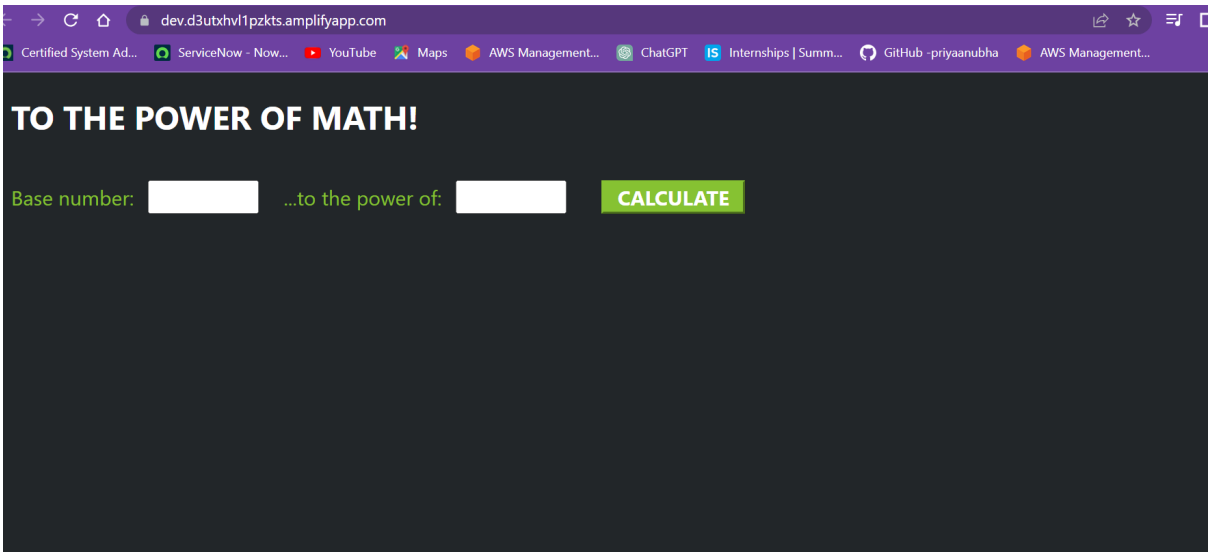


Deployment Domain for our WebApp:

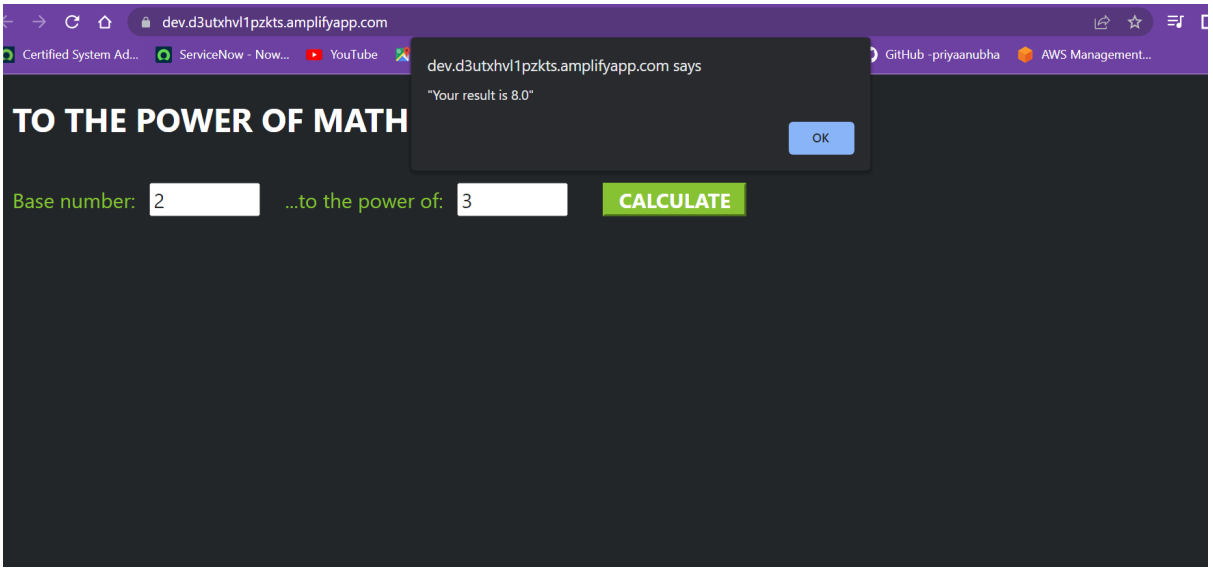
https://dev.d3utxhvl1pzkts.amplifyapp.com

Here is the final output:

1>



2>



Given, Base number : 2
Exponent : 3

Clicked on calculate
Result is 8.0

For reference:
URL : https://dev.d3utxhvl1pzkts.amplifyapp.com/