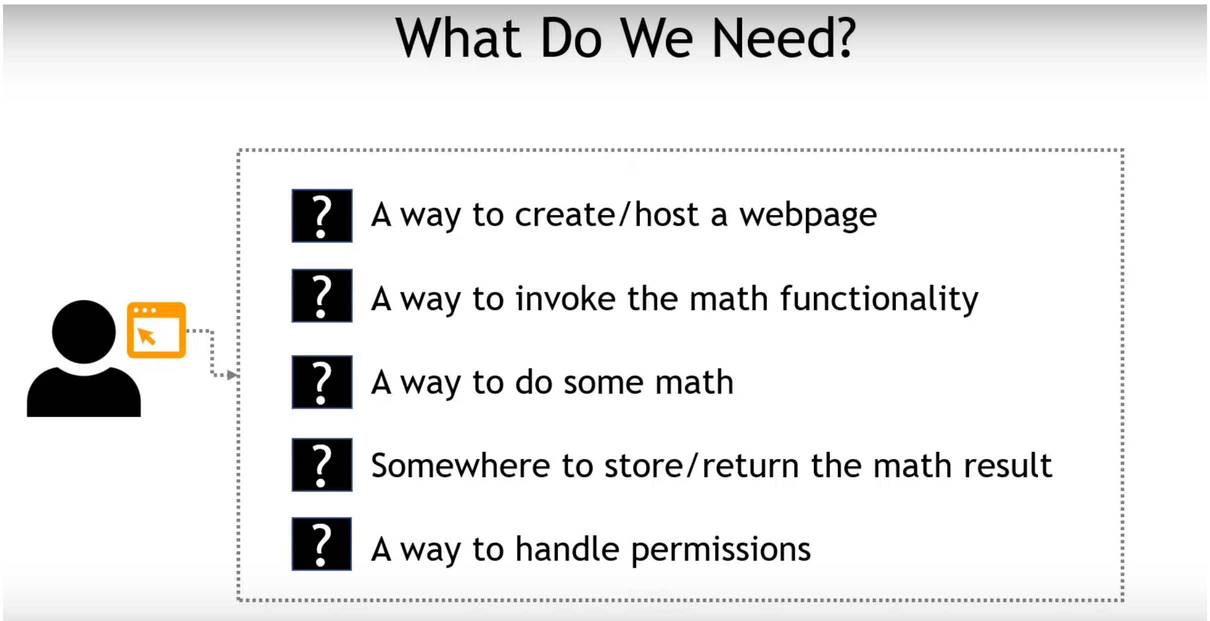
Deploying End-to-End Serverless AWS Web Application using Amplify & DynamoDB

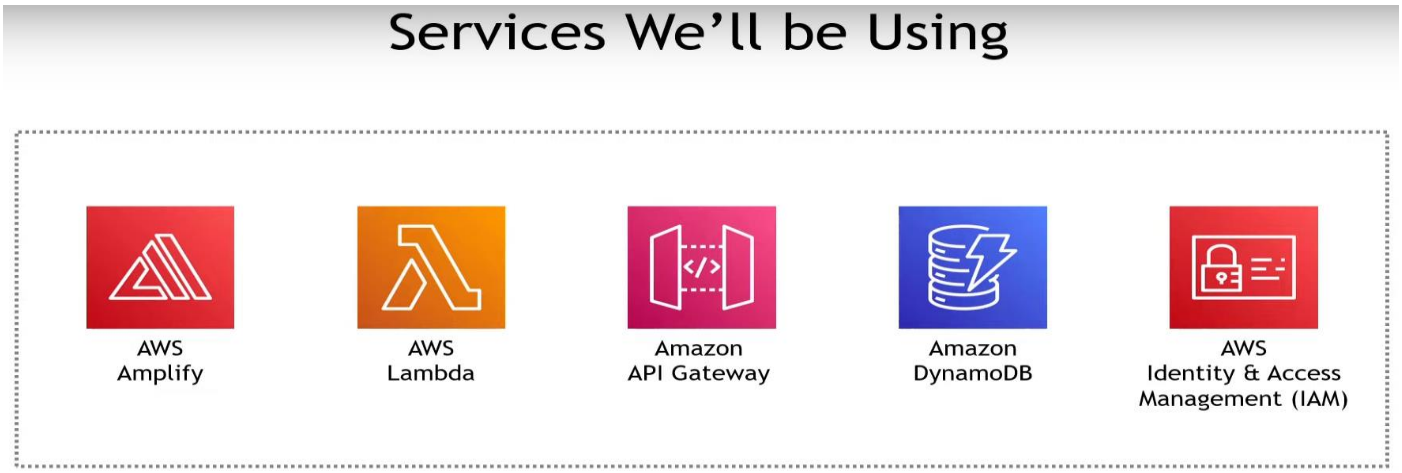
Objectives:



What Do We Need?

- ? A way to create/host a webpage
- ? A way to invoke the math functionality
- ? A way to do some math
- ? Somewhere to store/return the math result
- ? A way to handle permissions

Services we will be using:



Services We'll be Using

AWS Amplify

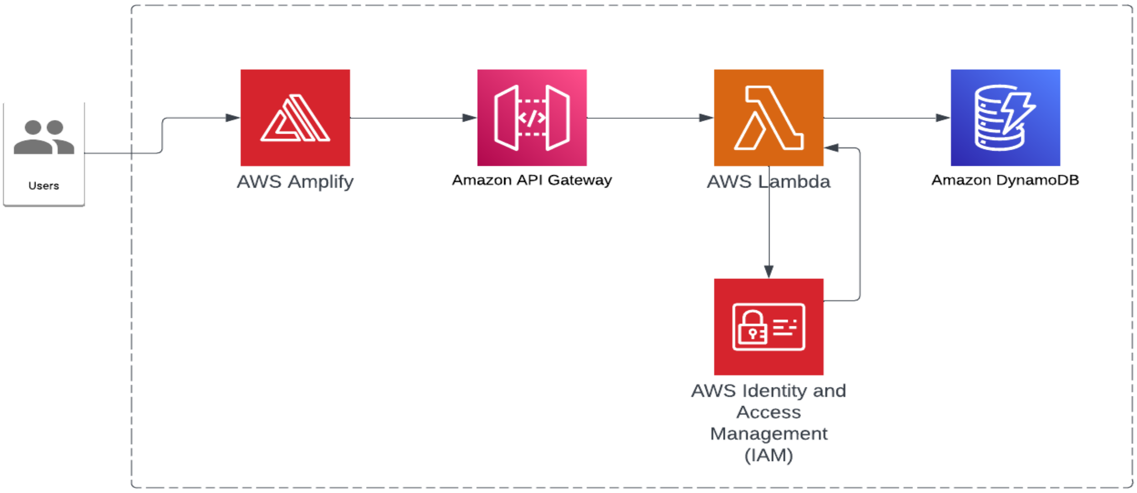
AWS Lambda

Amazon API Gateway

Amazon DynamoDB

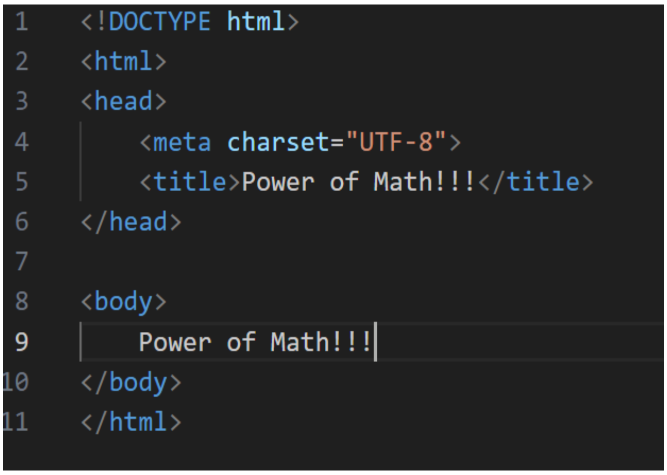
AWS Identity & Access Management (IAM)

Architecture of the Project:

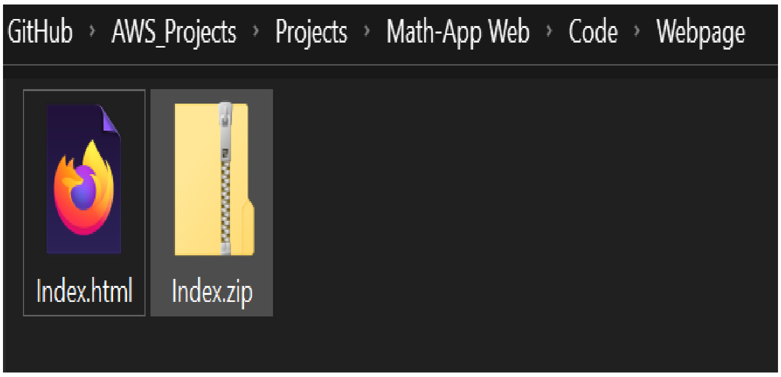


Step1: Creating an index.html file for the webpage

Create a index.html file & enter the simple code. Then zip the code (Refer “index-original.html” from code folder).

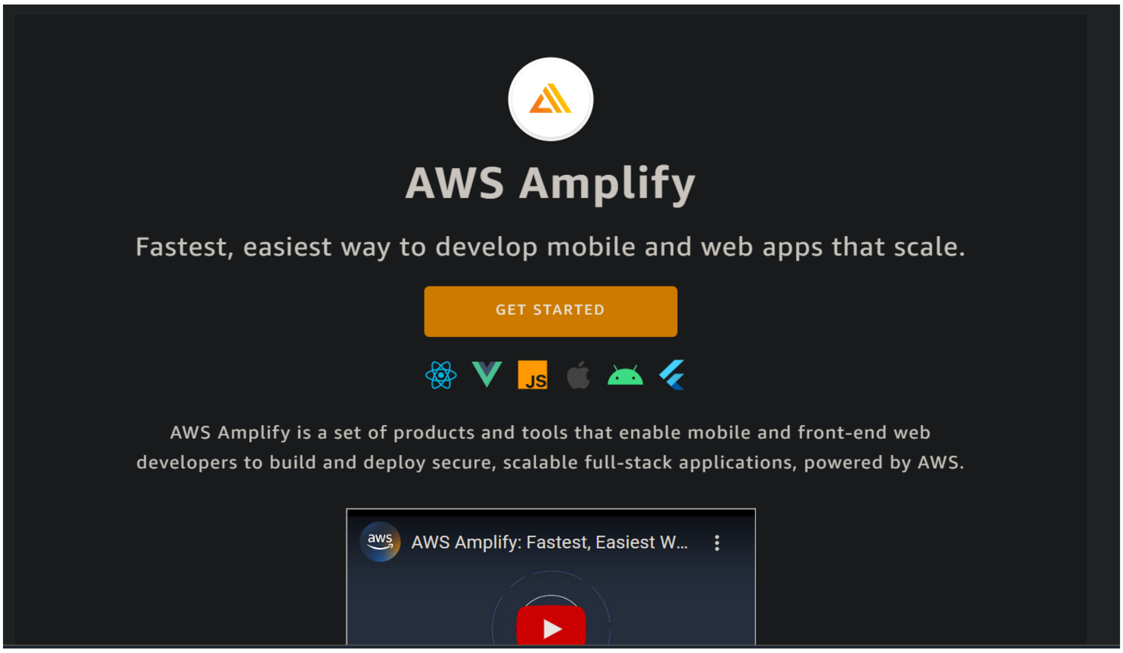


```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Power of Math!!!</title>
</head>

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



GitHub › AWS_Projects › Projects › Math-App Web › Code › Webpage

Step 2: Use AWS Amplify to deploy the code



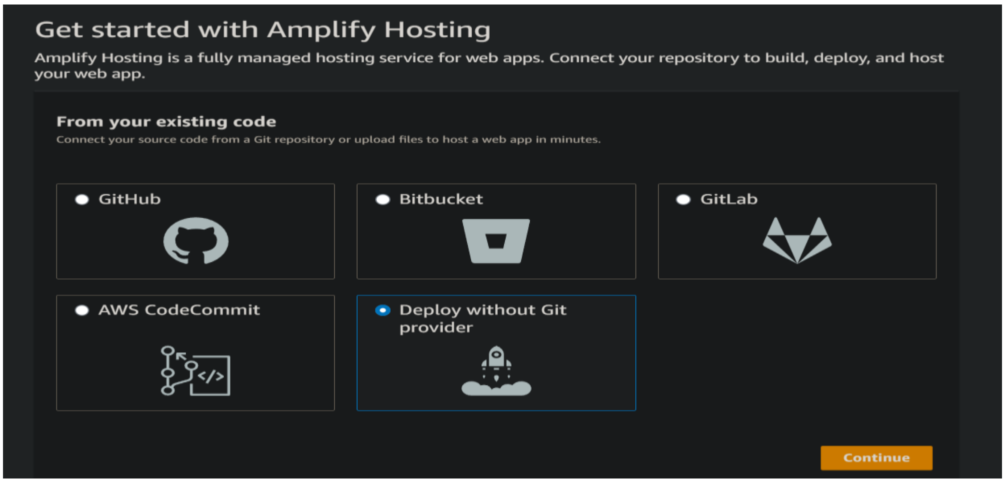
AWS Amplify

Fastest, easiest way to develop mobile and web apps that scale.

AWS Amplify is a set of products and tools that enable mobile and front-end web developers to build and deploy secure, scalable full-stack applications, powered by AWS.

Select Host the web app option.

Then,



Get started with Amplify Hosting

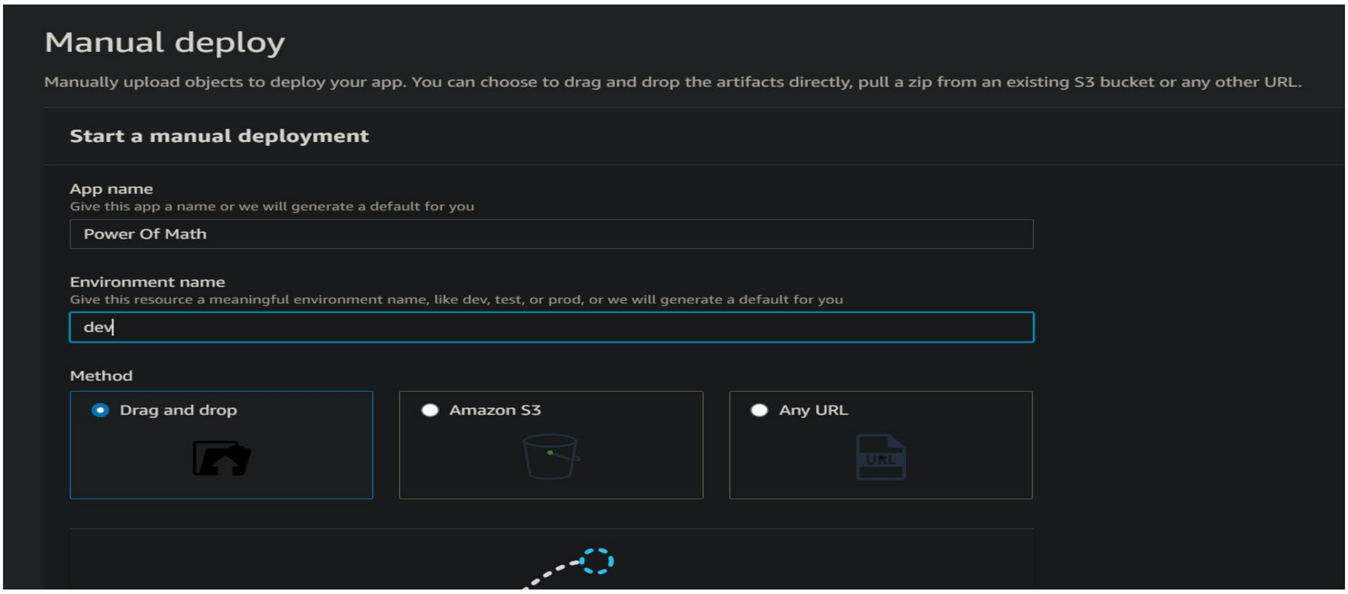
Amplify Hosting is a fully managed hosting service for web apps. Connect your repository to build, deploy, and host your web app.

From your existing code

Connect your source code from a Git repository or upload files to host a web app in minutes.

- ○ GitHub
- ○ Bitbucket
- ○ GitLab
- ○ AWS CodeCommit
- ● Deploy without Git provider

Then,



Manual deploy

Manually upload objects to deploy your app. You can choose to drag and drop the artifacts directly, pull a zip from an existing S3 bucket or any other URL.

Start a manual deployment

App name
Give this app a name or we will generate a default for you
Power Of Math

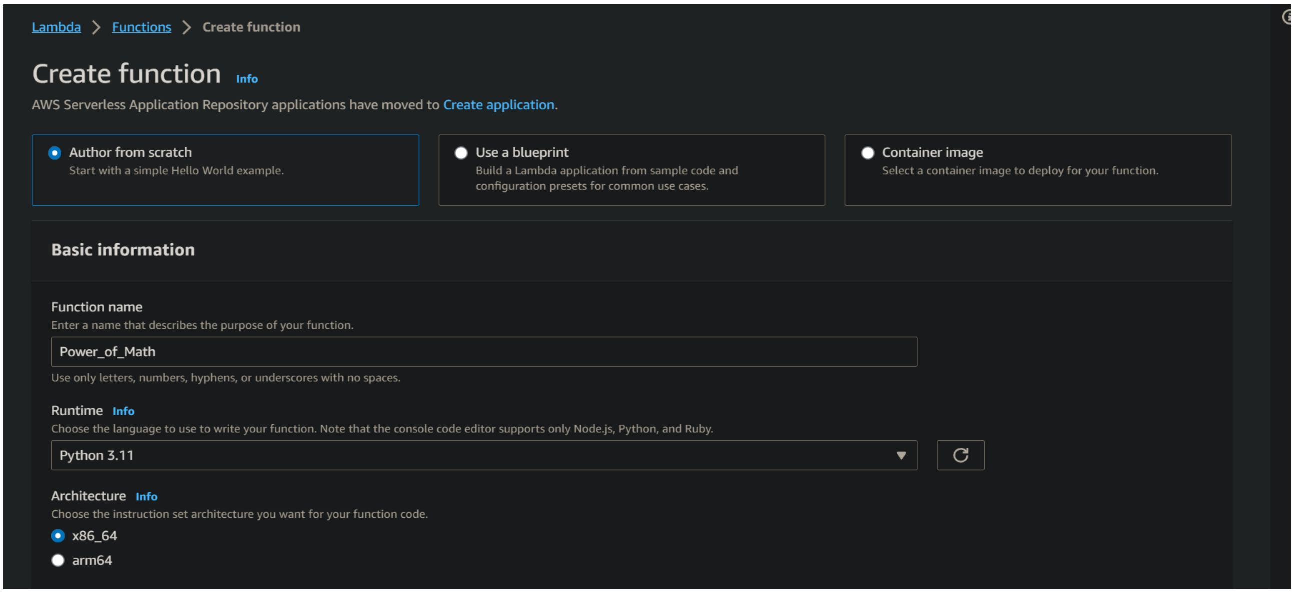
Environment name
Give this resource a meaningful environment name, like dev, test, or prod, or we will generate a default for you
dev

Method
- ● Drag and drop
- ○ Amazon S3
- ○ Any URL

Now the partial frontend hosting of the app is done.

Lets do some maths functions to be triggered upon using Lambda.

Step 2: Creating the Lambda function for math calculations backend



Create function Info

AWS Serverless Application Repository applications have moved to Create application.

- ● Author from scratch — Start with a simple Hello World example.
- ○ Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.
- ○ Container image — Select a container image to deploy for your function.

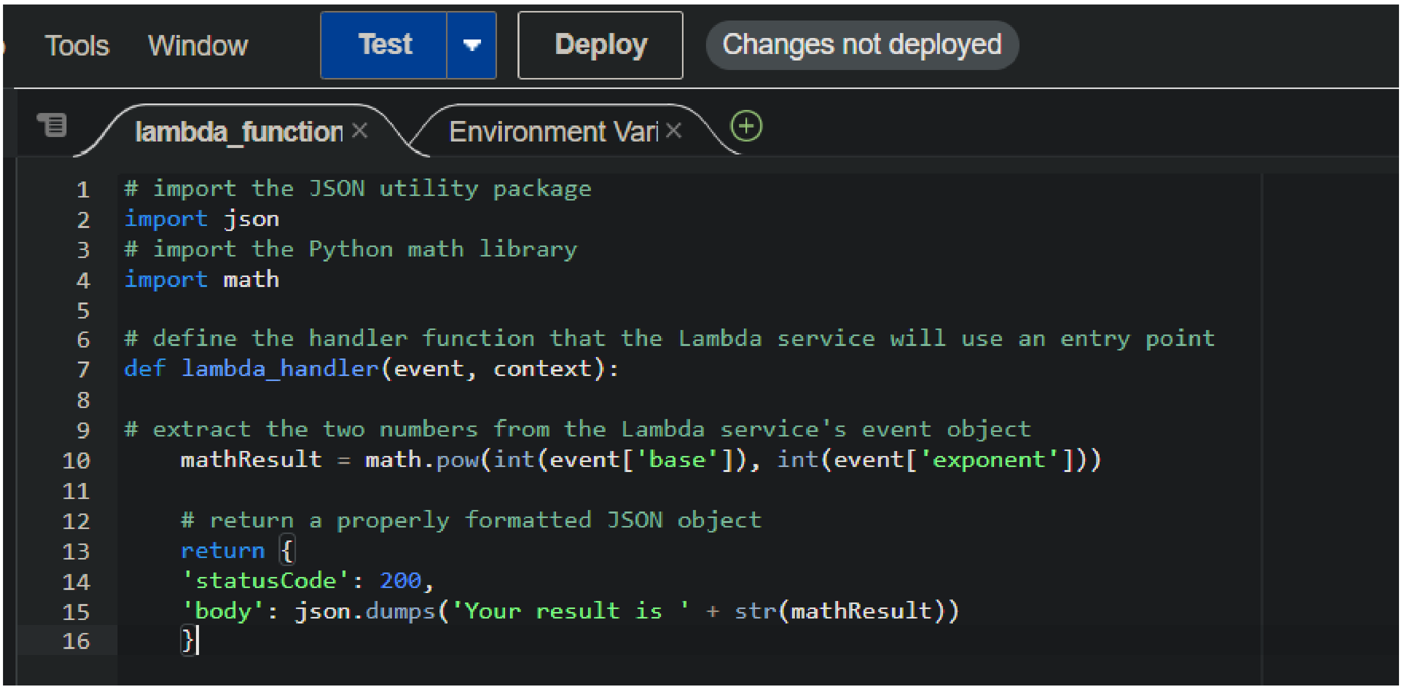
Basic information

Function name
Enter a name that describes the purpose of your function.
Power_of_Math
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.11

Architecture Info
Choose the instruction set architecture you want for your function code.
- ● x86_64
- ○ arm64

Everything else can be left to default. Then create the function. (Refer “PowerOfMathFunction - Lambda-ORIGINAL.py” from code folder).

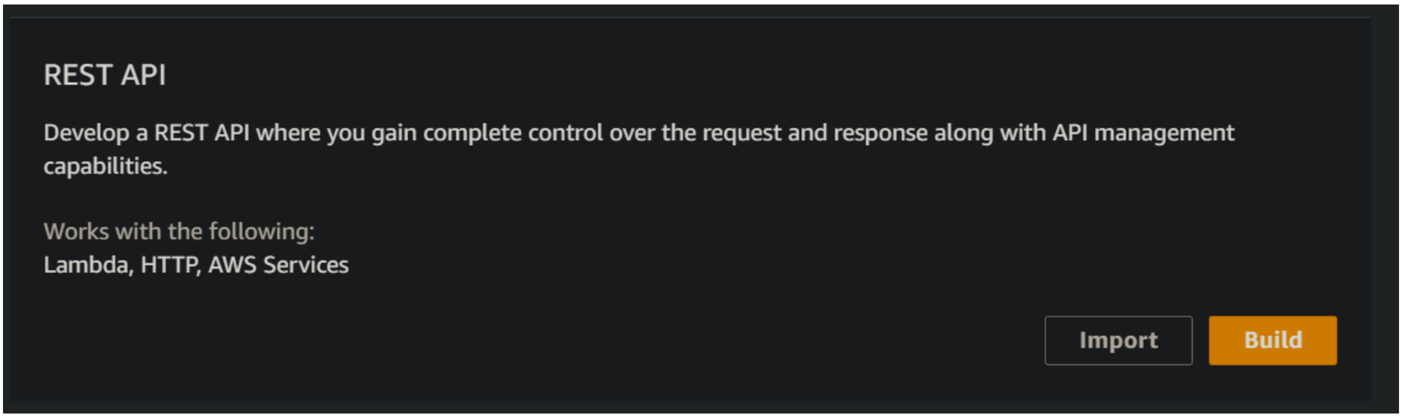


```python
# import the JSON utility package
import json
# import the Python math library
import math

# 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']))

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

Now use this code & deploy it. Now the Lambda function part is done.

Step 3: Now we need a public endpoint/URL to trigger this lambda function when interacting with the website. For this we use AWS API Gateway service.

Select REST API option

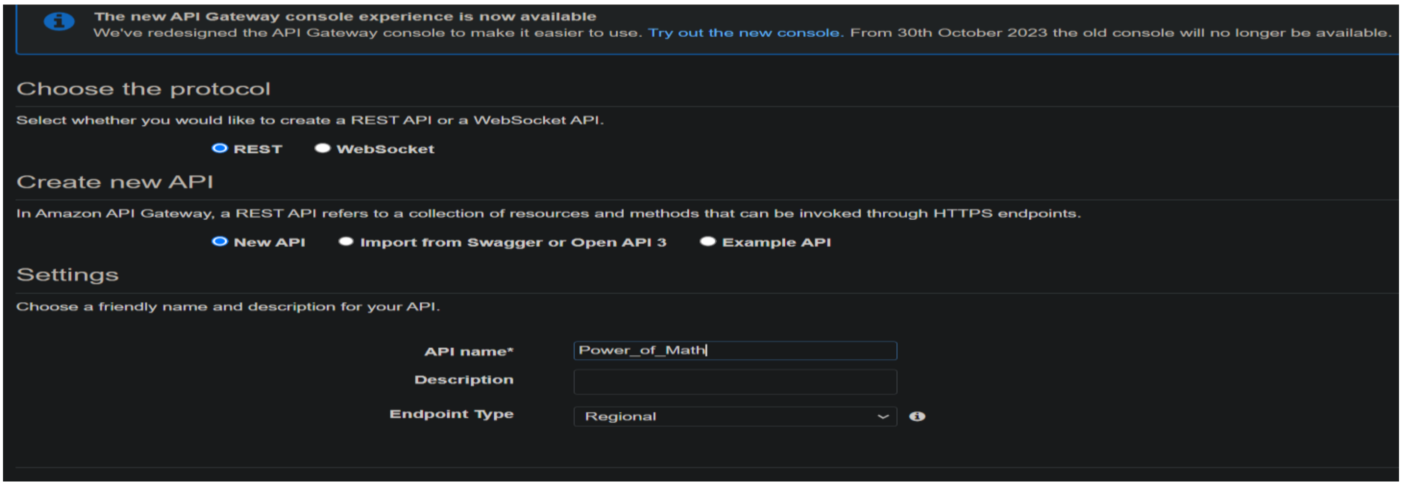


REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

Chose following options



The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available.

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

Settings

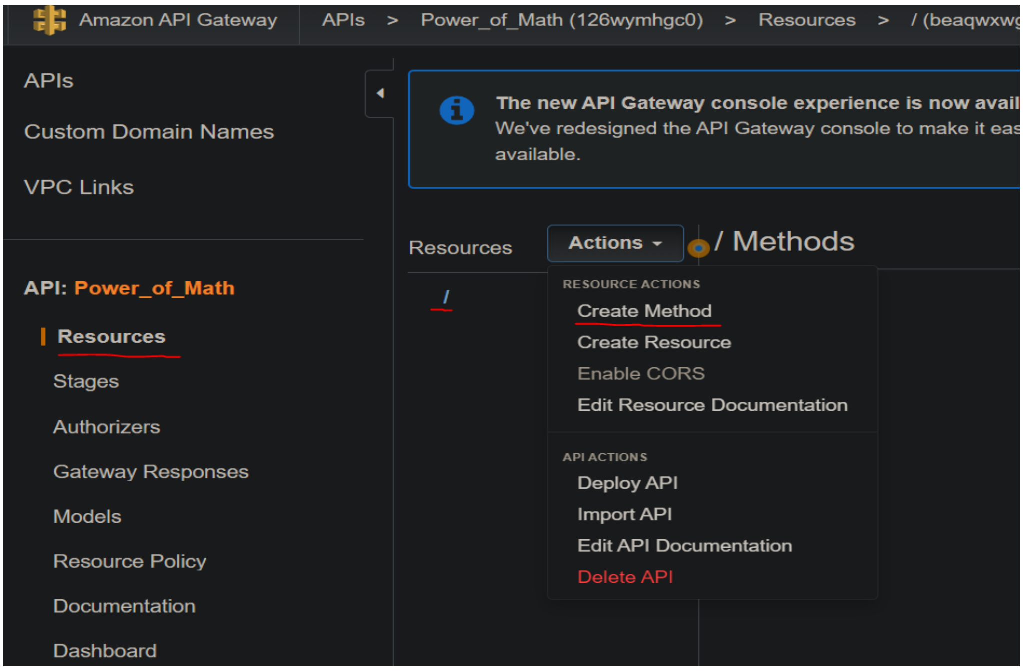
Choose a friendly name and description for your API.

API name* Power_of_Math

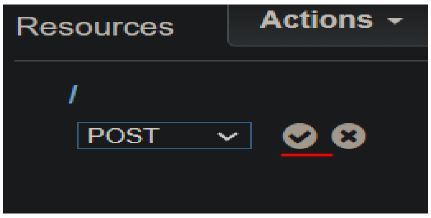
Description

Endpoint Type Regional

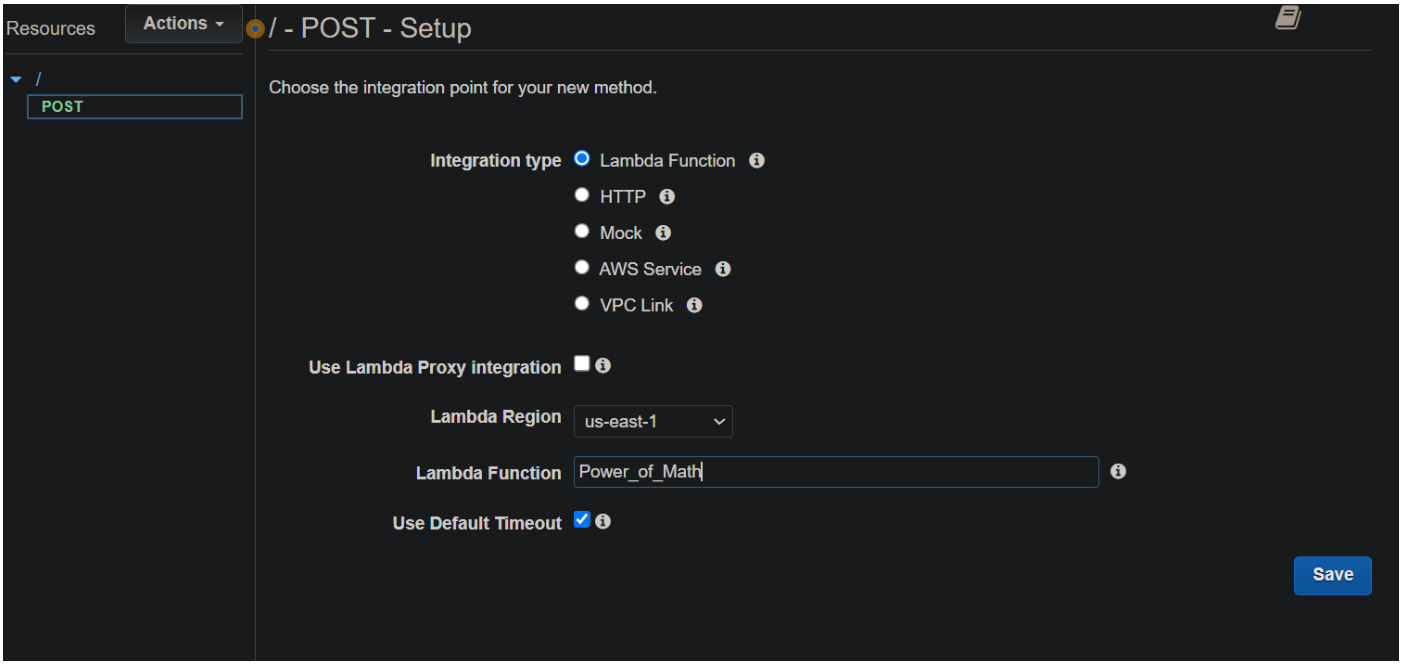
Create method



Type of Method – POST . Then select tick mark



Then,



Choose the integration point for your new method.

Integration type ● Lambda Function ○ HTTP ○ Mock ○ AWS Service ○ VPC Link

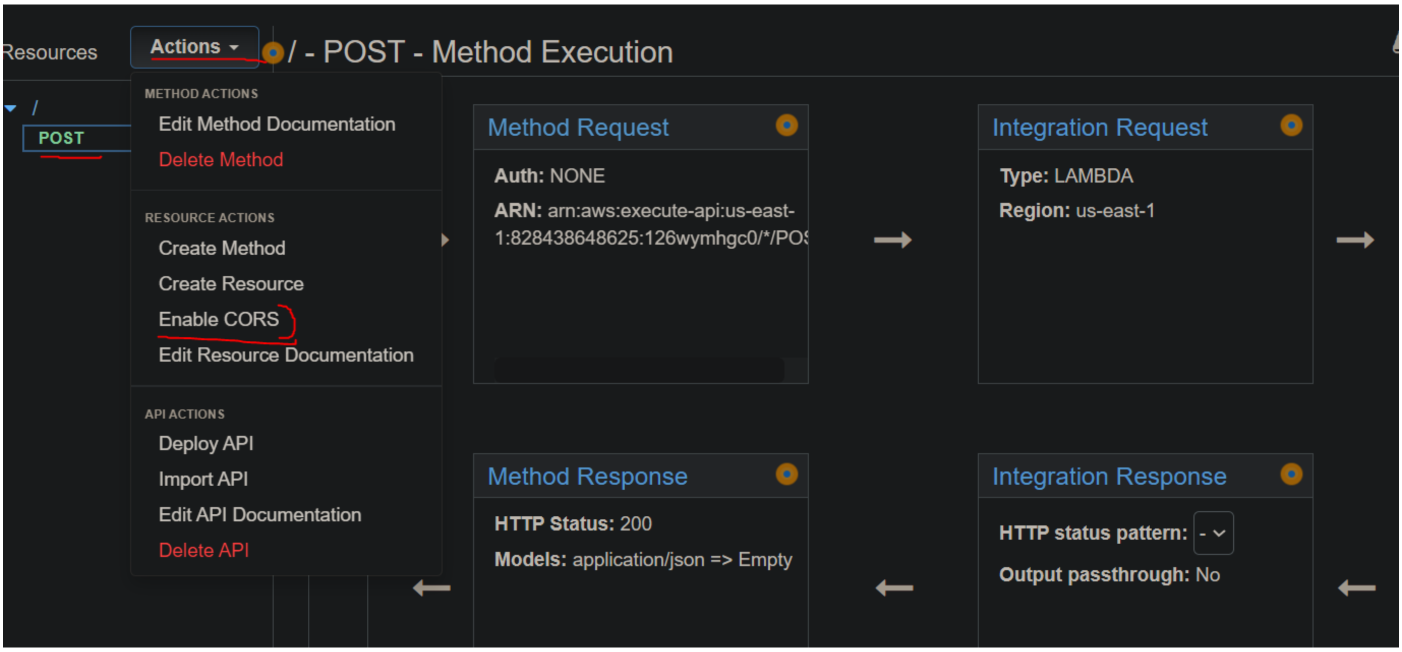
Use Lambda Proxy integration □

Lambda Region us-east-1

Lambda Function Power_of_Math

Use Default Timeout ☑

Now we have to make sure we enable **CORS** (Cross Origin Region Sharing). This is to make sure services across domain can work with one another.



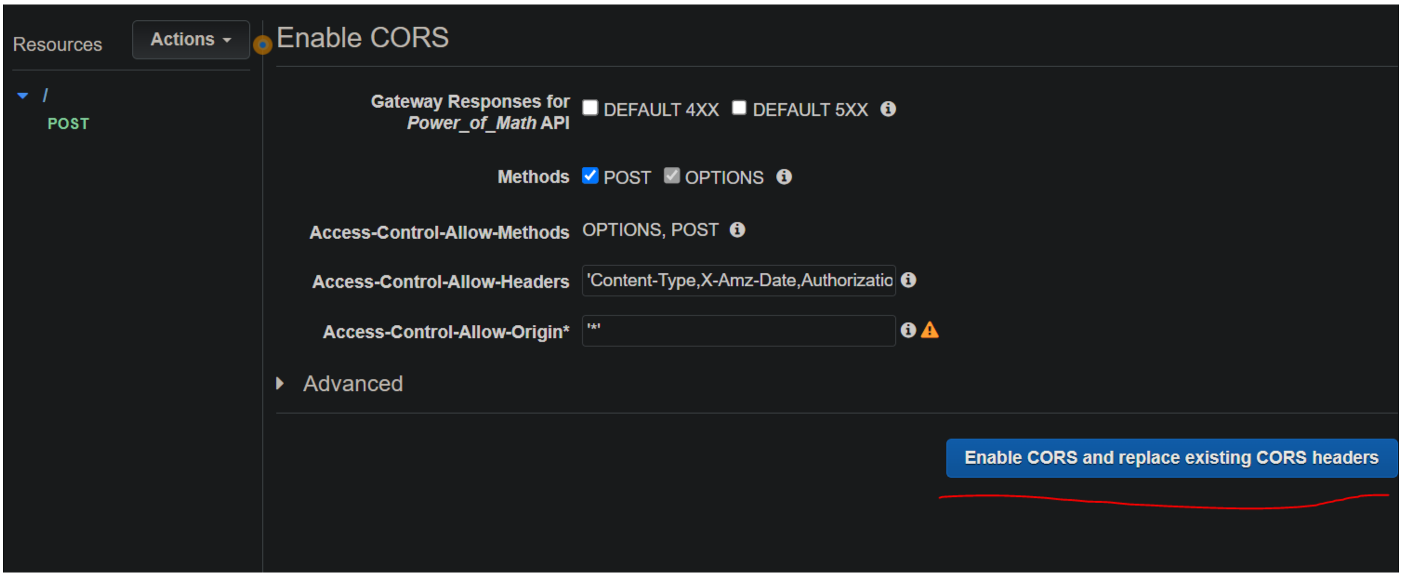
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:828438648625:126wymhgc0/*/POS

Type: LAMBDA
Region: us-east-1

HTTP Status: 200
Models: application/json => Empty

HTTP status pattern: -
Output passthrough: No

Now,



Gateway Responses for *Power_of_Math* API □ DEFAULT 4XX □ DEFAULT 5XX

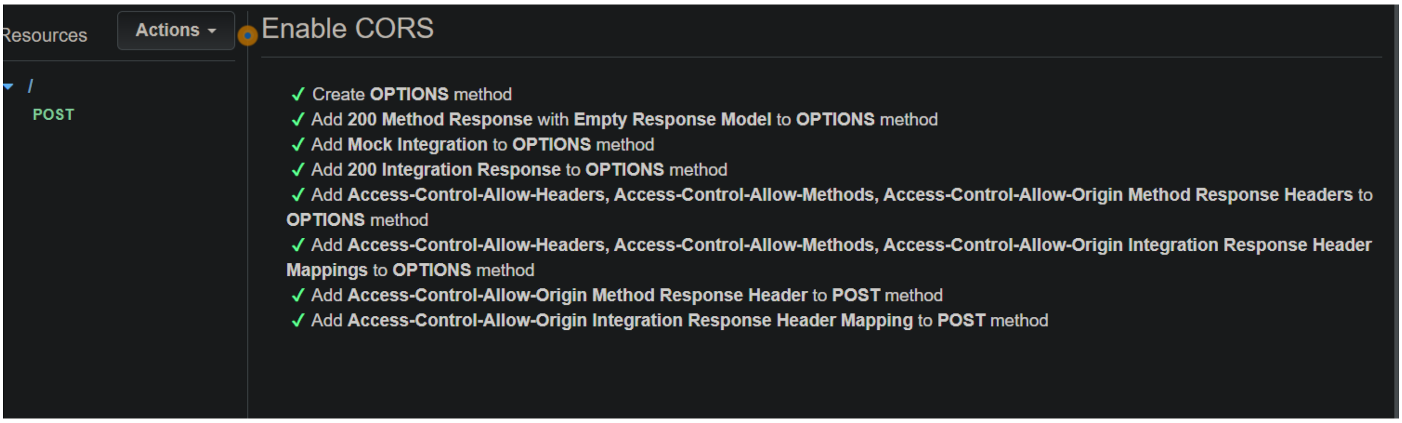
Methods ☑ POST ☑ OPTIONS

Access-Control-Allow-Methods OPTIONS, POST

Access-Control-Allow-Headers 'Content-Type,X-Amz-Date,Authorizatio

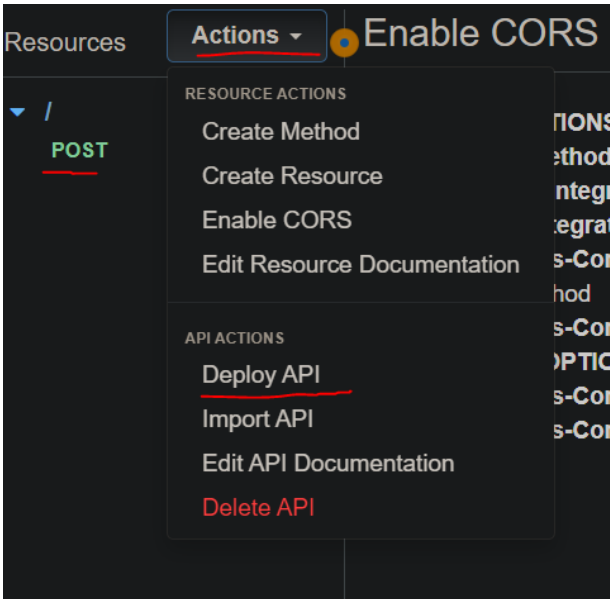
Access-Control-Allow-Origin* '*'

This is the outcome.

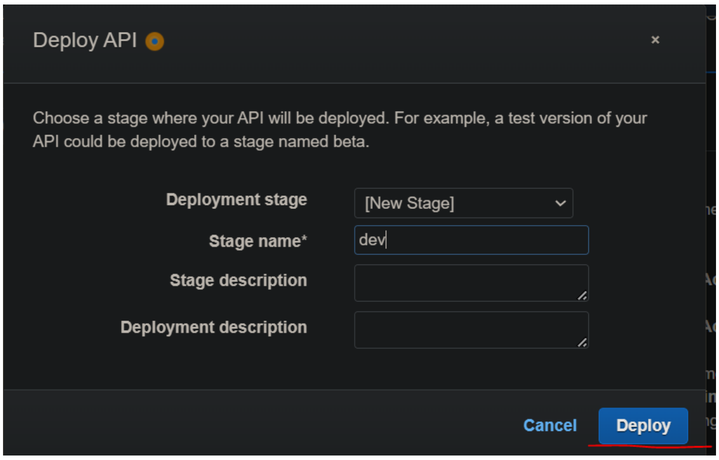


✓ Create **OPTIONS** method
✓ Add **200 Method Response** with **Empty Response Model** to **OPTIONS** method
✓ Add **Mock Integration** to **OPTIONS** method
✓ Add **200 Integration Response** to **OPTIONS** method
✓ Add **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin 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-Origin Method Response Header** to **POST** method
✓ Add **Access-Control-Allow-Origin Integration Response Header Mapping** to **POST** method

Now we have to deploy our API.



Then,


Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

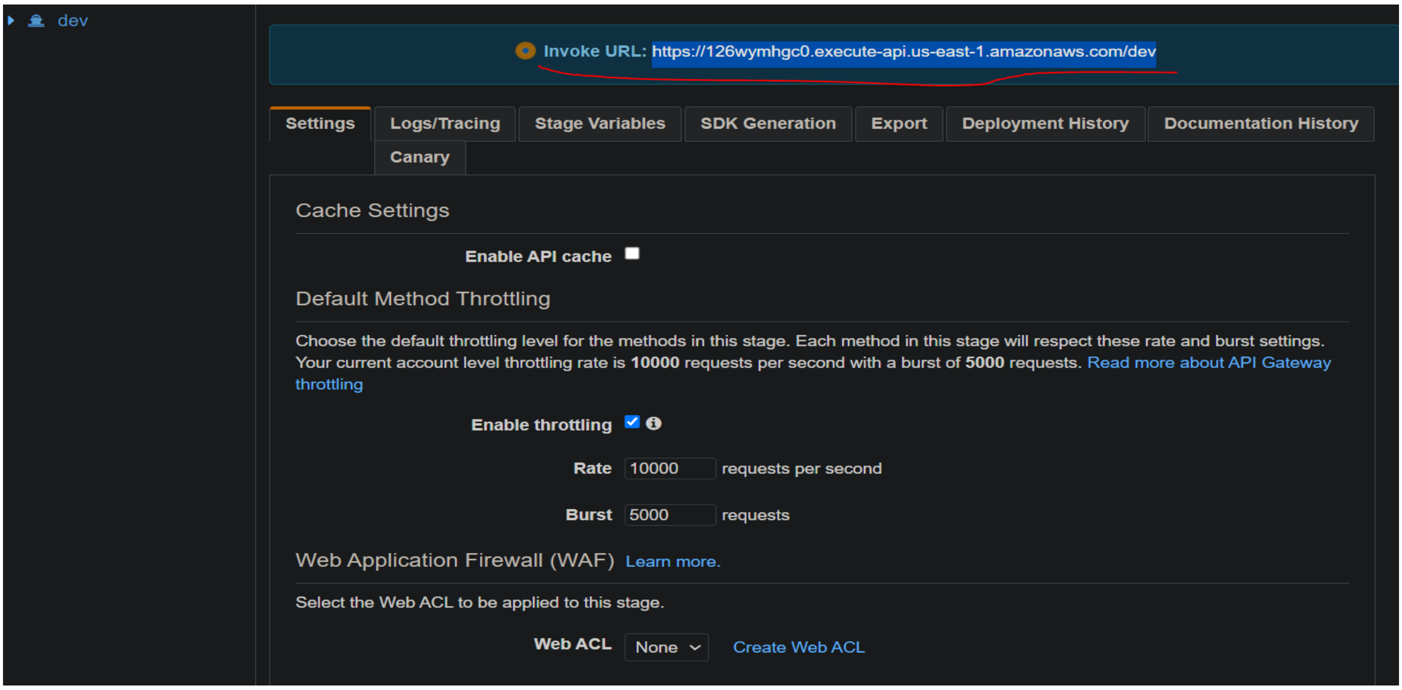
Deployment stage [New Stage]

Stage name* dev

Stage description

Deployment description

Now copy & keep the URL generated for you somewhere safe:



Invoke URL: https://126wymhgc0.execute-api.us-east-1.amazonaws.com/dev

Cache Settings

Enable API cache □

Default Method Throttling

Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** requests per second with a burst of **5000** requests. Read more about API Gateway throttling

Enable throttling ☑

Rate 10000 requests per second

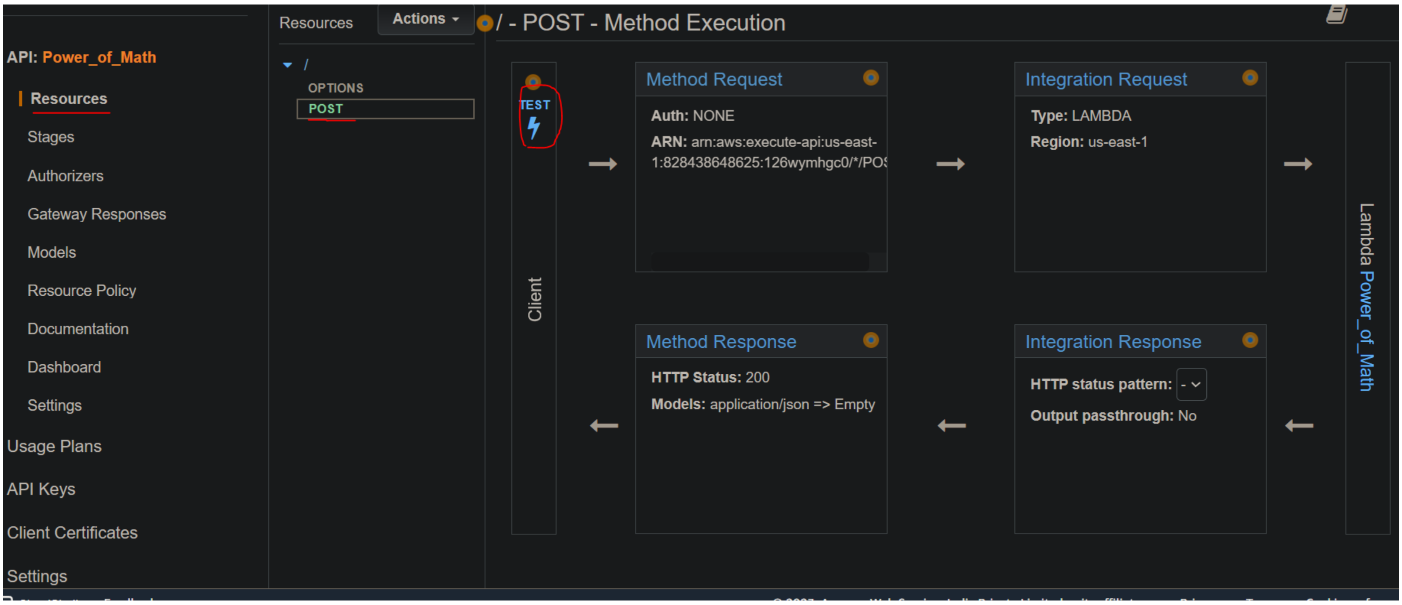
Burst 5000 requests

Web Application Firewall (WAF) Learn more.

Select the Web ACL to be applied to this stage.

Web ACL None

Now lets validate this API. Come to this path. Then click on Test Icon.



/ - POST - Method Execution

Auth: NONE
ARN: arn:aws:execute-api:us-east-1:828438648625:126wymhgc0/*/POS

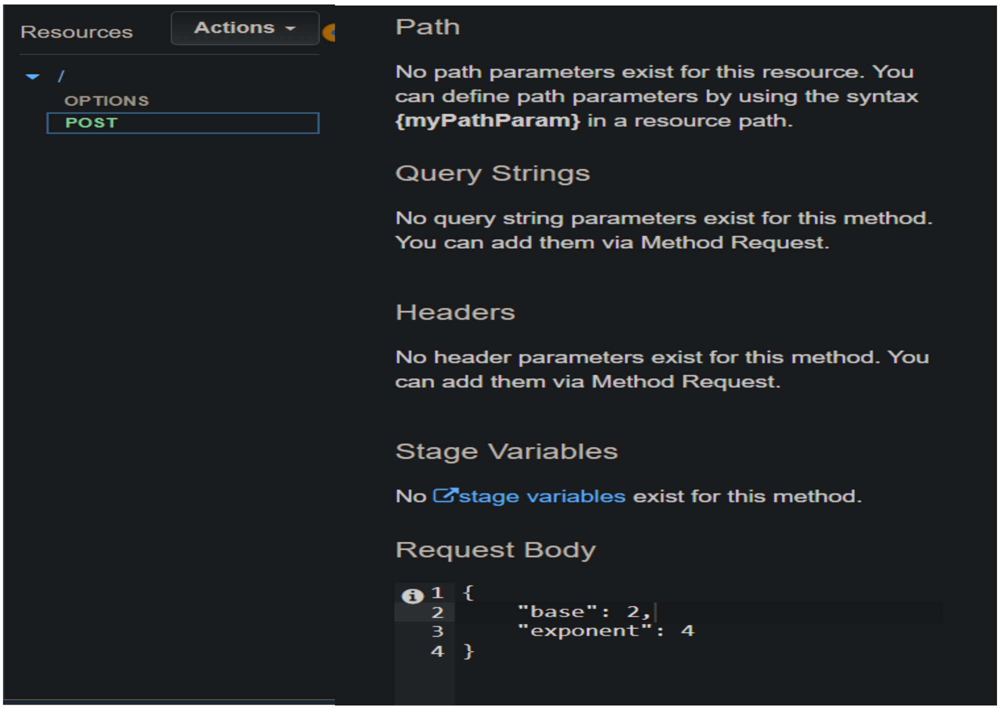
Type: LAMBDA
Region: us-east-1

HTTP Status: 200
Models: application/json => Empty

HTTP status pattern: -
Output passthrough: No

Now what we are doing here is to test if the API is working by passing the parameters that we have set in Lamda function & giving them test values. If the API & Lambda is working as intended, they will give the desired Output.

We write the test values in the request body (use the lambda parameters name) . Then click on test at the bottom.



Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

No query string parameters exist for this method. You can add them via Method Request.

Headers

No header parameters exist for this method. You can add them via Method Request.

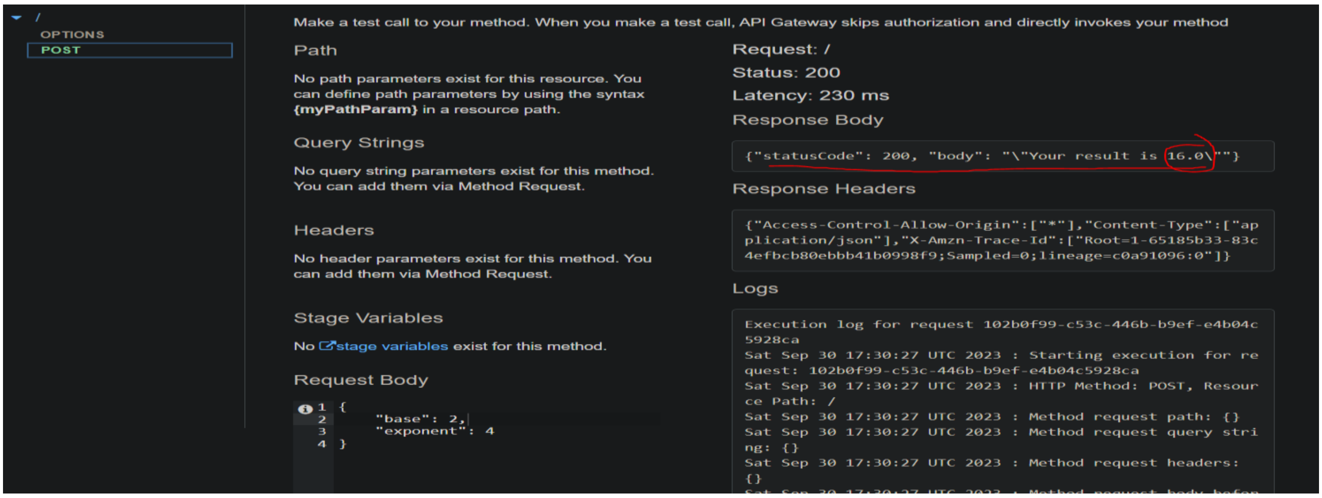
Stage Variables

No stage variables exist for this method.

Request Body

```
{
    "base": 2,
    "exponent": 4
}
```

Result is successful. Got the desired output.



Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Query Strings

No query string parameters exist for this method. You can add them via Method Request.

Headers

No header parameters exist for this method. You can add them via Method Request.

Stage Variables

No stage variables exist for this method.

Request Body

```
{
    "base": 2,
    "exponent": 4
}
```

Request: /
Status: 200
Latency: 230 ms

Response Body

```
{"statusCode": 200, "body": "\"Your result is 16.0\""}
```

Response Headers

```
{"Access-Control-Allow-Origin":["*"],"Content-Type":["ap
plication/json"],"X-Amzn-Trace-Id":["Root=1-65185b33-83c
4efbcb80ebbb41b0998f9;Sampled=0;lineage=c0a91096:0"]}
```

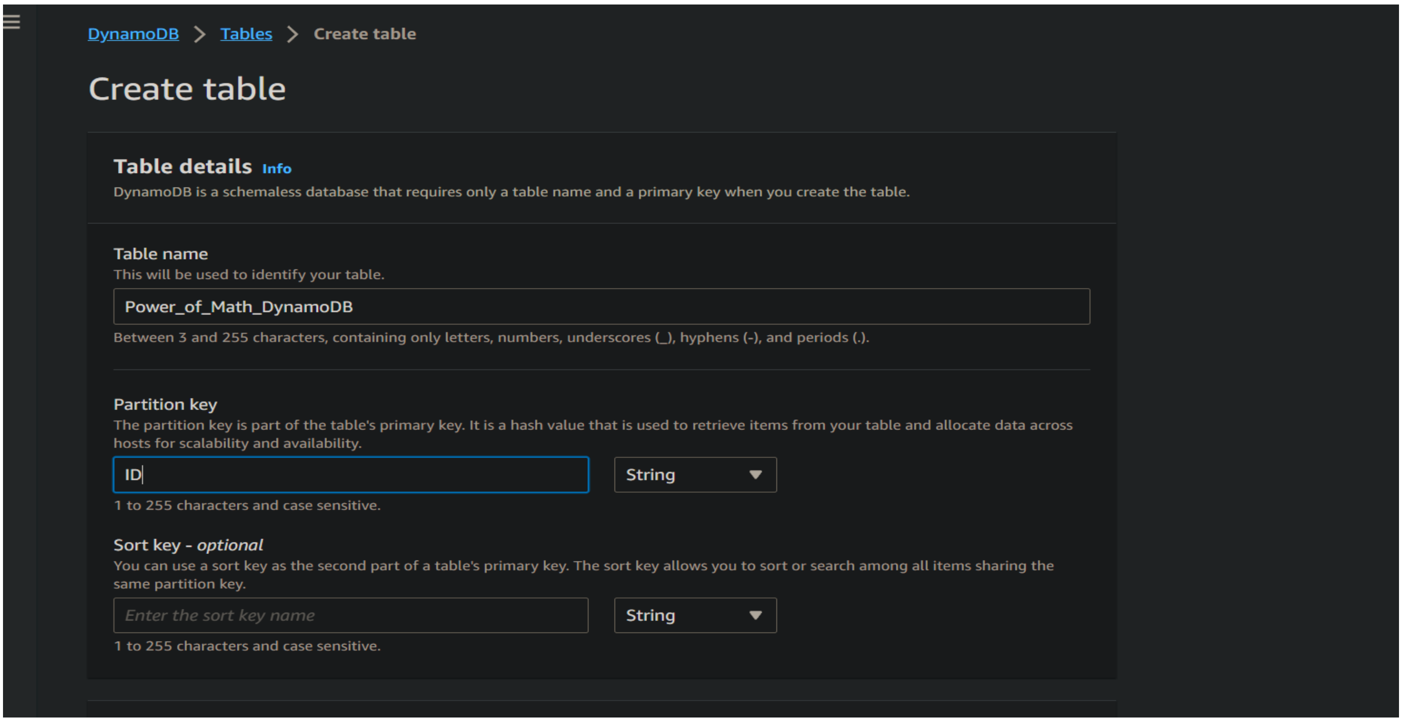
Logs

```
Execution log for request 102b0f99-c53c-446b-b9ef-e4b04c
5928ca
Sat Sep 30 17:30:27 UTC 2023 : Starting execution for re
quest: 102b0f99-c53c-446b-b9ef-e4b04c5928ca
Sat Sep 30 17:30:27 UTC 2023 : HTTP Method: POST, Resour
ce Path: /
Sat Sep 30 17:30:27 UTC 2023 : Method request path: {}
Sat Sep 30 17:30:27 UTC 2023 : Method request query stri
ng: {}
Sat Sep 30 17:30:27 UTC 2023 : Method request headers:
{}
```

Now the AWS API Gateway part is completed.

Step 4:- Now we have to store the returned values somewhere. For this we will use DynamoDB (NoSQL) to store our values.

Leave all the other values to Default. Create the 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.

Power_of_Math_DynamoDB

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.

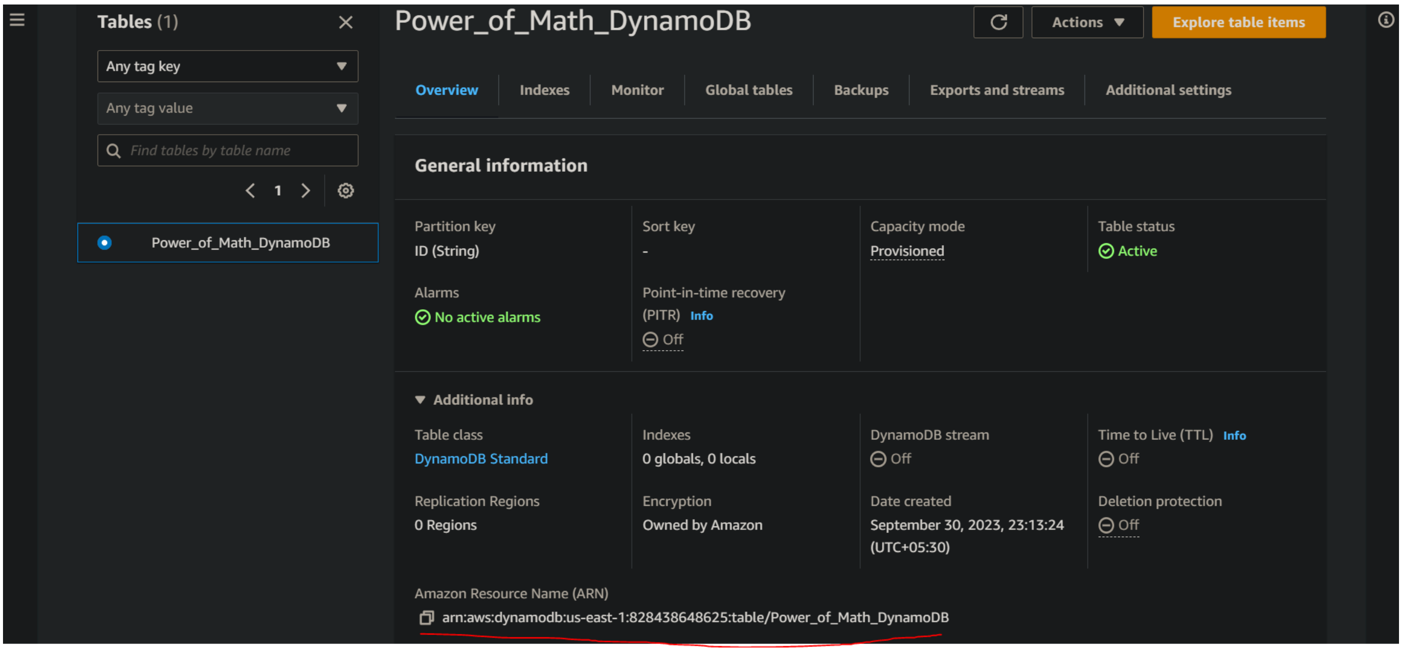
Sort key - *optional*

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.

Copy the DynamoDB arn & store it somewhere



Power_of_Math_DynamoDB

General information

Partition key ID (String)	Sort key -	Capacity mode Provisioned	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Info Off		

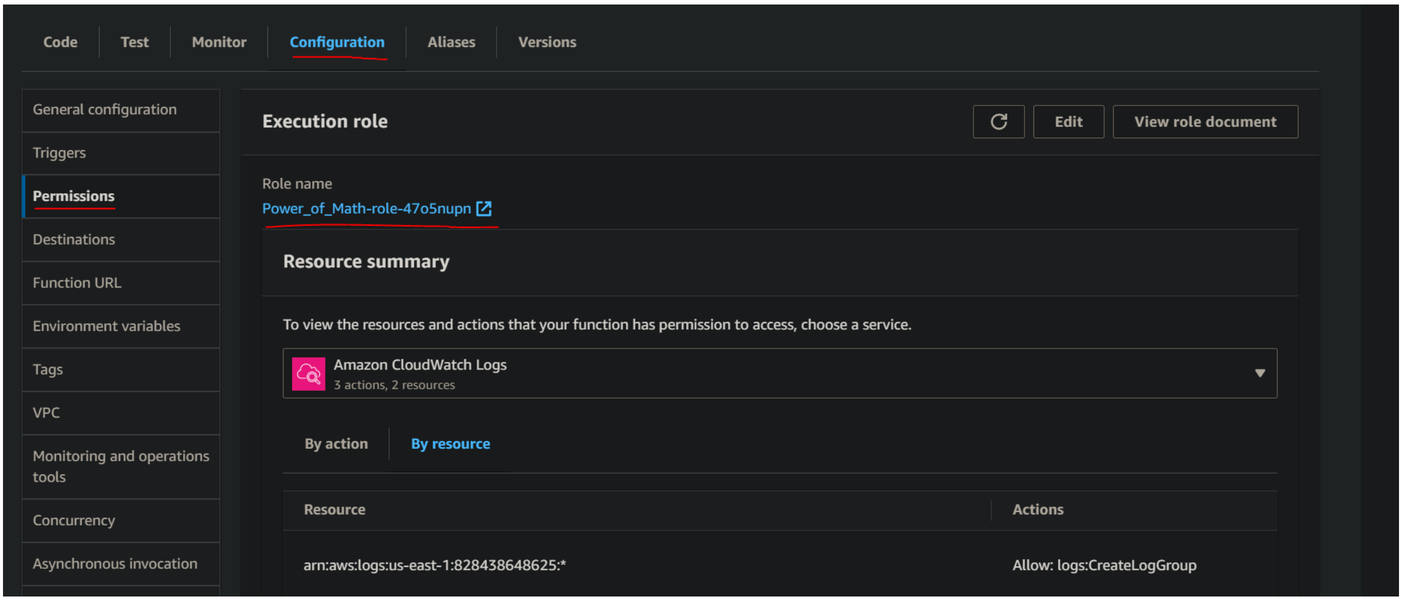
Additional info

Table class DynamoDB Standard	Indexes 0 globals, 0 locals	DynamoDB stream Off	Time to Live (TTL) Info Off
Replication Regions 0 Regions	Encryption Owned by Amazon	Date created September 30, 2023, 23:13:24 (UTC+05:30)	Deletion protection Off

Amazon Resource Name (ARN)

arn:aws:dynamodb:us-east-1:828438648625:table/Power_of_Math_DynamoDB

Now we need to have the lambda that we have created to have the permission to write values to this DynamoDB. Go to the lambda section & do this. Click on the Rolename.



Execution role

Role name
Power_of_Math-role-47o5nupn

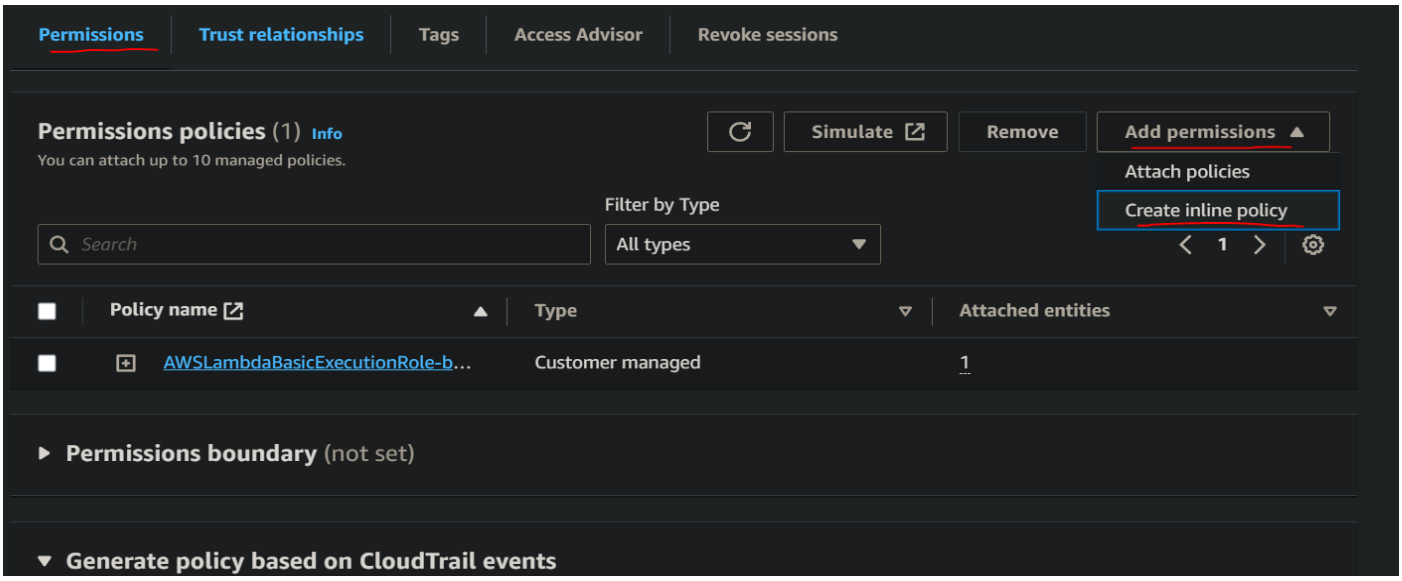
Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs
3 actions, 2 resources

Resource	Actions
arn:aws:logs:us-east-1:828438648625:*	Allow: logs:CreateLogGroup

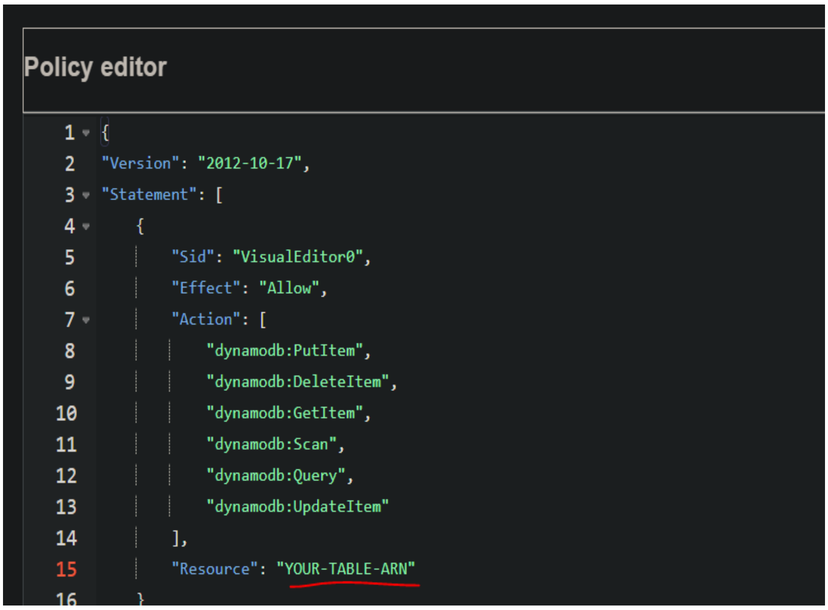
Now goto the create inline policy & copy the following JSON code to create DynamoDB Permissions.



Permissions policies (1)
You can attach up to 10 managed policies.

☐	Policy name	Type	Attached entities
☐	AWSLambdaBasicExecutionRole-b...	Customer managed	1

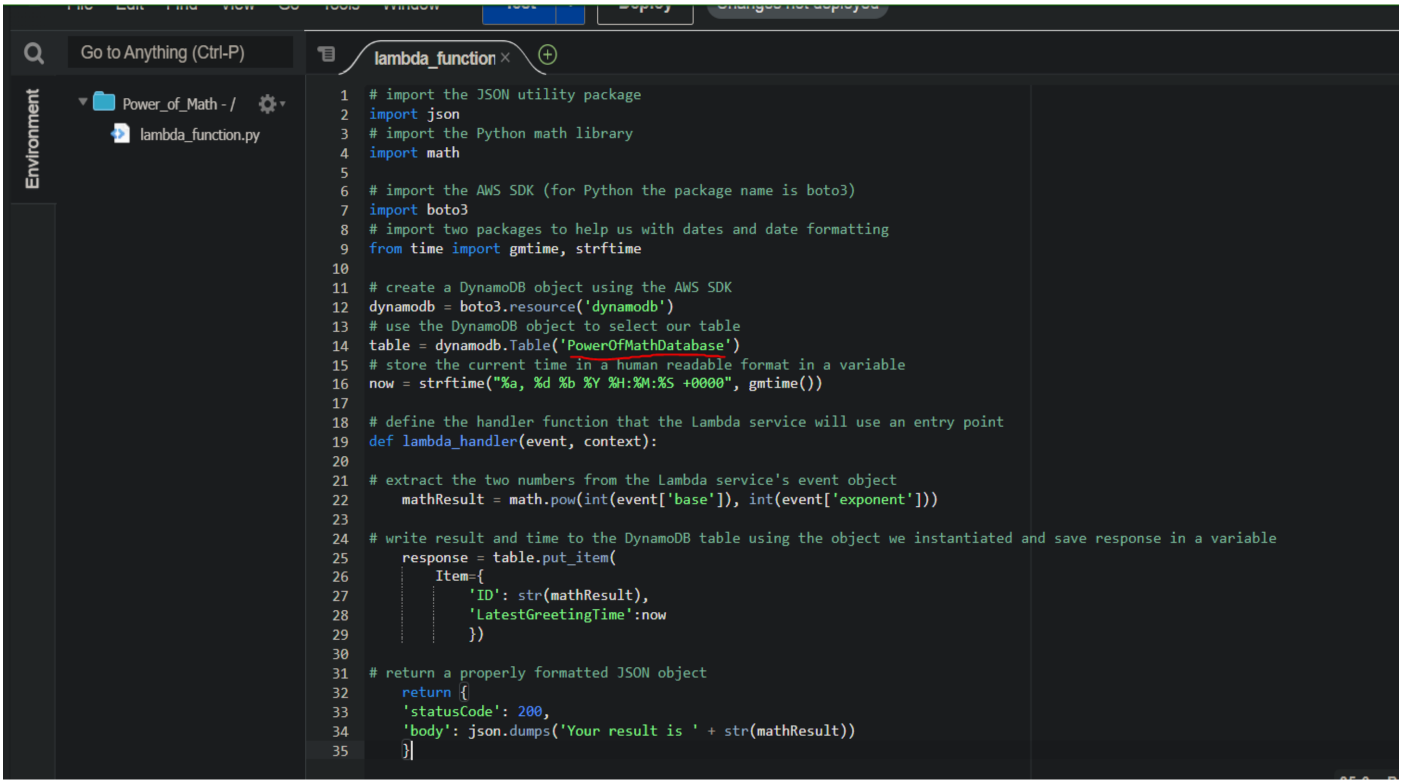
This is the code. Replace your-table-arn with the actual table arn. Basically this policy allows actions of the following things – Put, delete,get,scan,query,update. (Refer “Execution Role Policy JSON.json” from the code folder).


Policy editor

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:Scan",
            "dynamodb:Query",
            "dynamodb:UpdateItem"
        ],
        "Resource": "YOUR-TABLE-ARN"
    }
```

Permissions thing done. Now we have to update the lambda function to actually go write to the database.

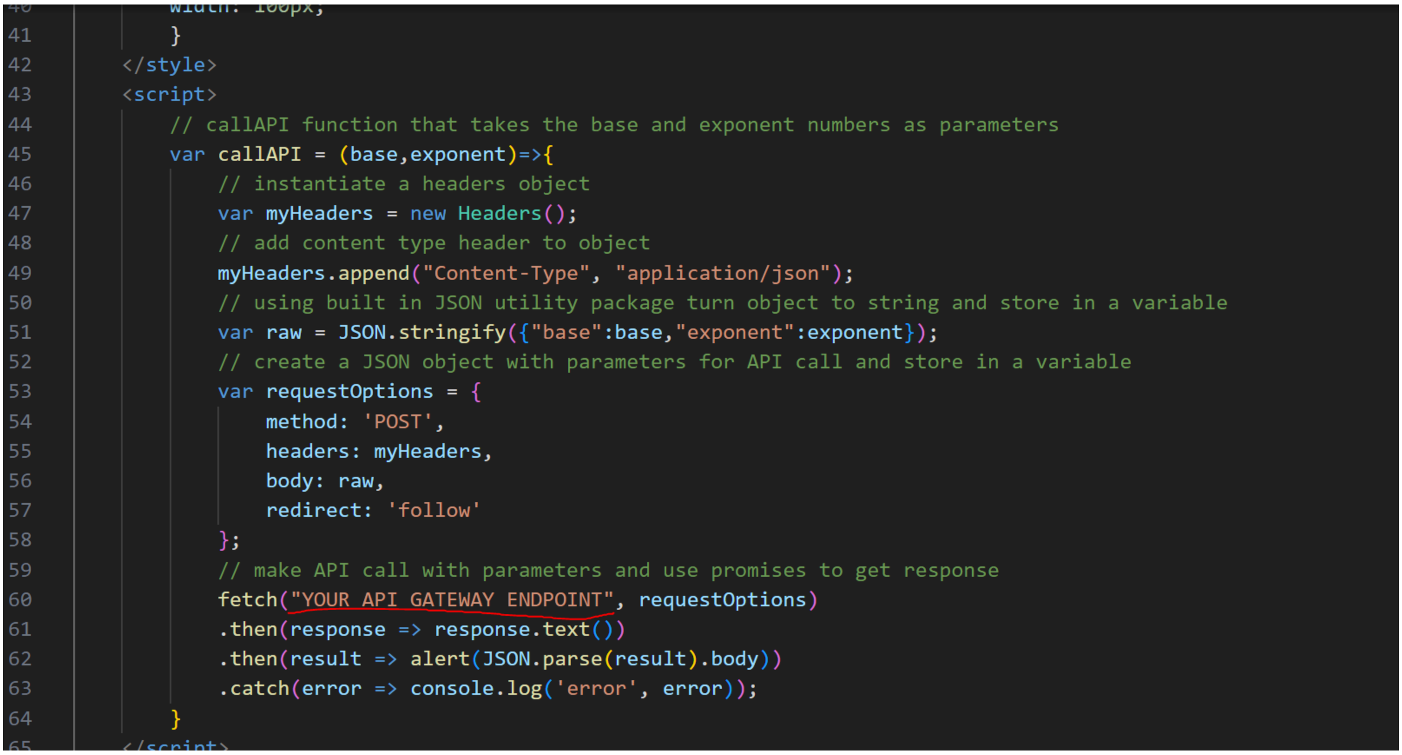
Step 5: - Updating the lambda function. Change the table variable to your table name. Next save the code & deploy. (Refer “PowerOfMathFunction - Lambda-FINAL.py” from code folder).



```
# 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))
    }
```

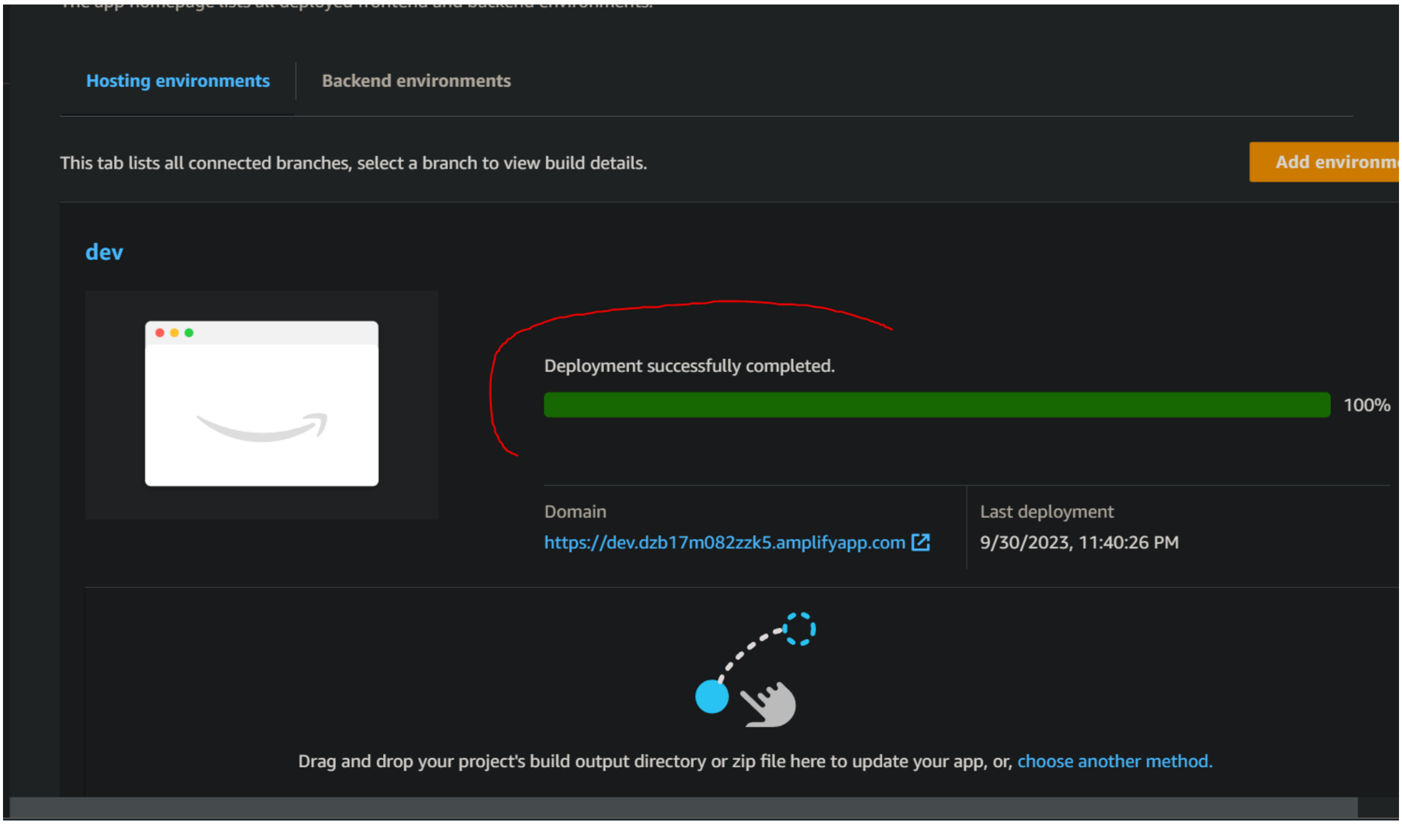
Step 6:- Now the final connector. We have to connect our index.html (UI) to the backend process that we have created.

In the final html code, replace this with your actual API arn that we have saved earlier. (Refer “index-original.html” from code folder).



```
        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>
```

After making the changes, once again create a zip file out of your new index file & redeploy it to Amplify just like before.



This tab lists all connected branches, select a branch to view build details.

Deployment successfully completed.

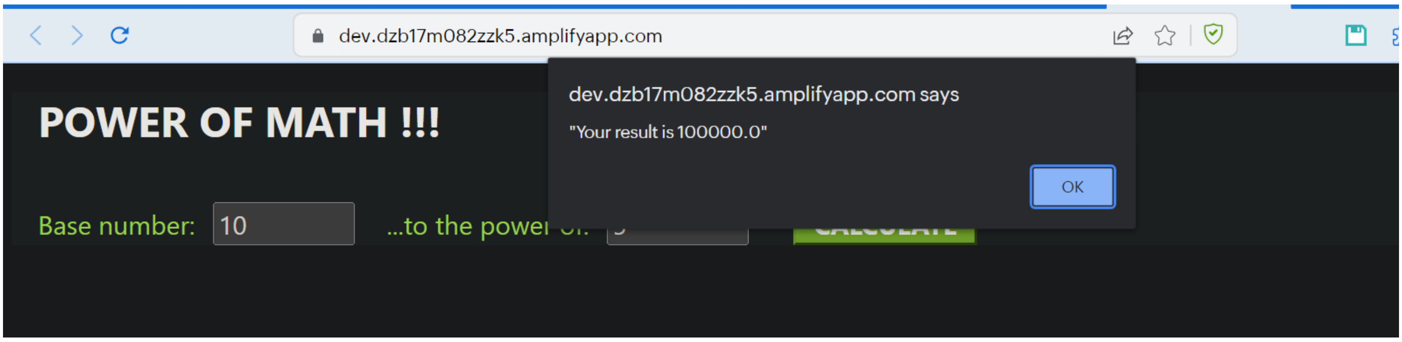
100%

Domain: https://dev.dzb17m082zzk5.amplifyapp.com

Last deployment: 9/30/2023, 11:40:26 PM

Drag and drop your project's build output directory or zip file here to update your app, or, choose another method.

Click on the domain & see the results.



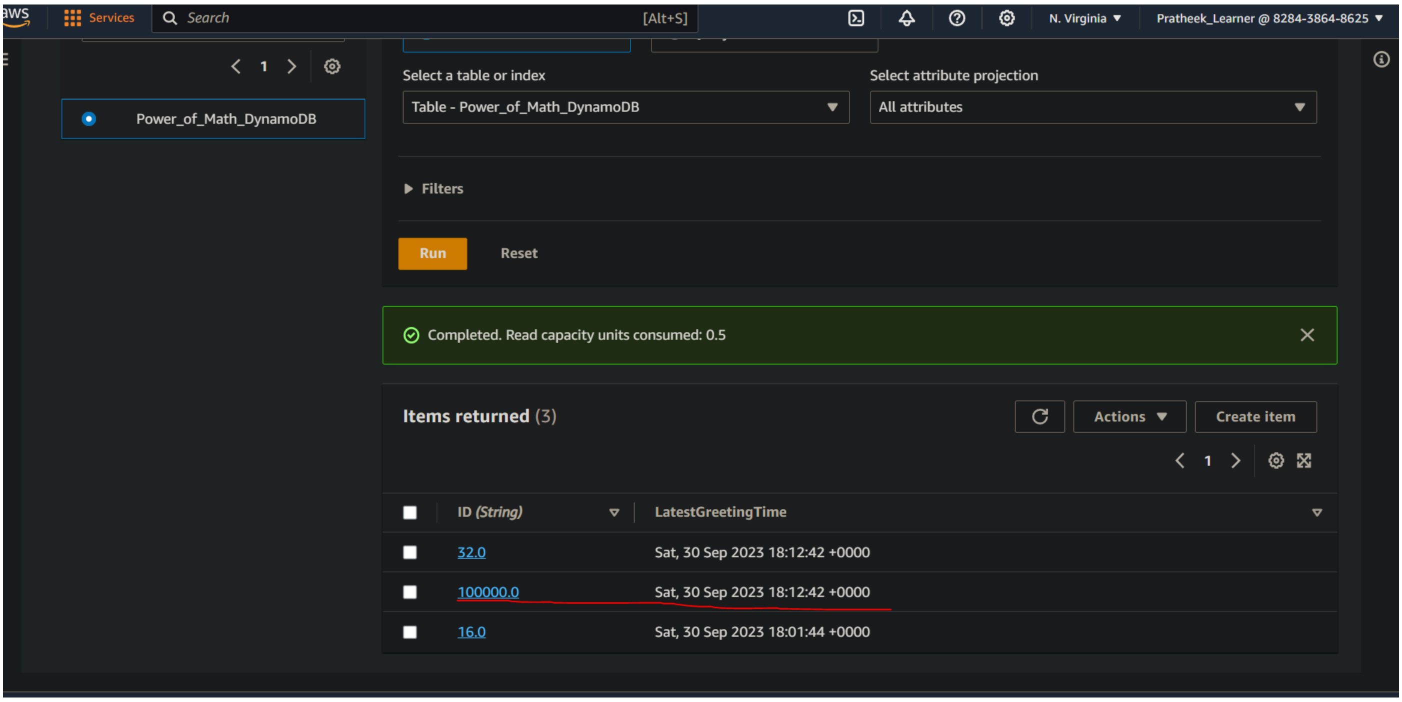
dev.dzb17m082zzk5.amplifyapp.com says

"Your result is 100000.0"

Base number: 10

...to the power of:

Lets check the DynamoDB



Select a table or index: Table - Power_of_Math_DynamoDB

Select attribute projection: All attributes

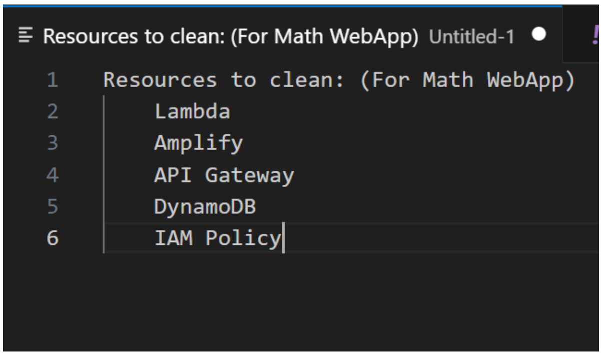
Completed. Read capacity units consumed: 0.5

Items returned (3)

ID (String)	LatestGreetingTime
32.0	Sat, 30 Sep 2023 18:12:42 +0000
100000.0	Sat, 30 Sep 2023 18:12:42 +0000
16.0	Sat, 30 Sep 2023 18:01:44 +0000

Its working. The project has been successfully completed & working as intended.

Resources to clean:



```
Resources to clean: (For Math WebApp)
    Lambda
    Amplify
    API Gateway
    DynamoDB
    IAM Policy
```