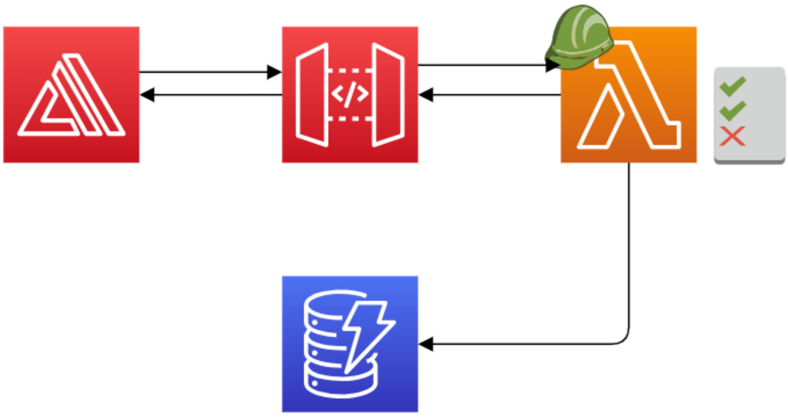
Project 1: Deploying a web application using AWS Amplify, DynamoDB, AWS Lambda and API Gateway

This aim of this project is to demonstrate awareness and proficiency in using the below services to achieve a manual app deployment using the services below.

Services deployed:

Architectural Diagram



What did I do?

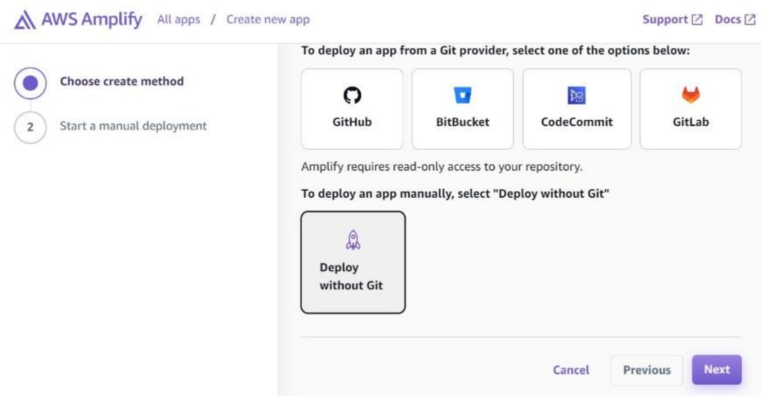
- Created and hosted a webpage – AWS Amplify.
- Invoked the math functionality using functions to do the math calculation – API Gateway & Lambda.
- Created a place to store/return the result of the calculation - DynamoDB
- Handled permissions between the AWS services - IAM.

Step 1: Create and host a webpage --> AWS Amplify

This project makes use of Amplify to host my html webpage.

This is a simple web application that takes two numbers, base and exponent, and returns another number, which is the power of the base raised to the exponent. The result is returned as an alert message to the user.

- Write and save the code in an index.html file (the code is saved in this repository)
- The file (index.html) is saved, zipped and
- Go to the console, search for amplify. Choose to 'Deploy without git'. Click Next.



AWS Amplify All apps / Create new app Support Docs

1 Choose create method

2 Start a manual deployment

To deploy an app from a Git provider, select one of the options below:

GitHub BitBucket CodeCommit GitLab

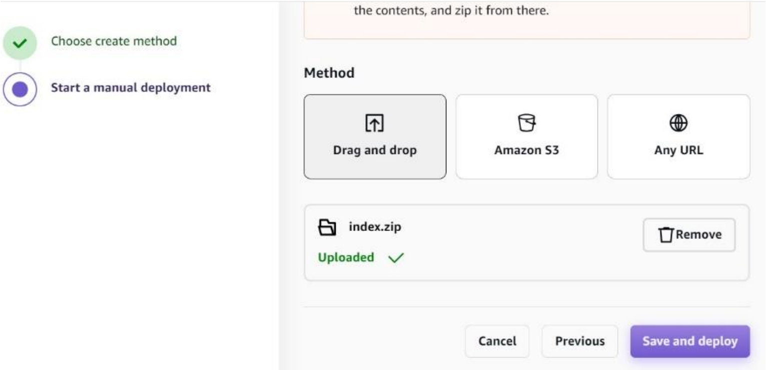
Amplify requires read-only access to your repository.

To deploy an app manually, select "Deploy without Git"

Deploy without Git

Cancel Previous Next

- I then upload the zipped html file



the contents, and zip it from there.

Method

Drag and drop
Amazon S3
Any URL

index.zip
Uploaded ✓

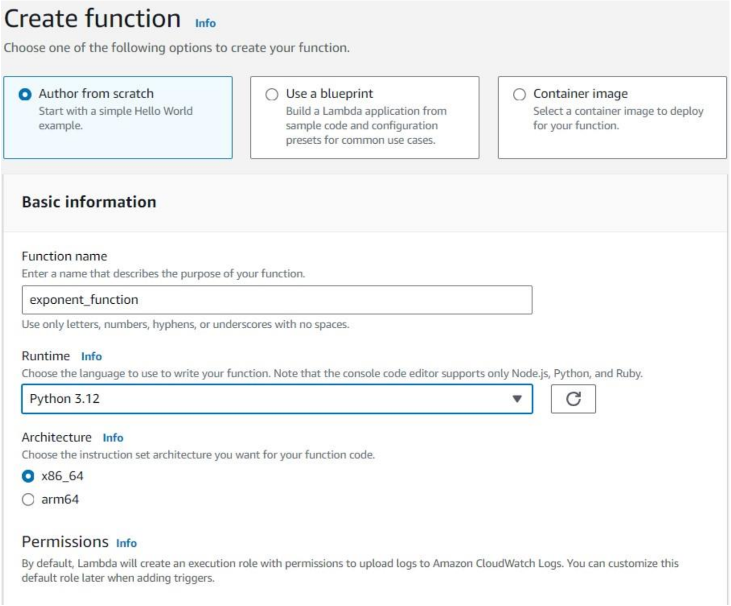
-

That's it.

Step 2: create lambda function.

The lambda function I created does 2 things: performs the math functionality and sends the result to dynamo db. Therefore, the function will require access to DynamoDB. This is achieved by editing the Lambda's role in the IAM service.

- Go to lambda on the console and create new function. The function is based on Python. Click save.



Create function Info

Choose one of the following options to create your function.

● 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

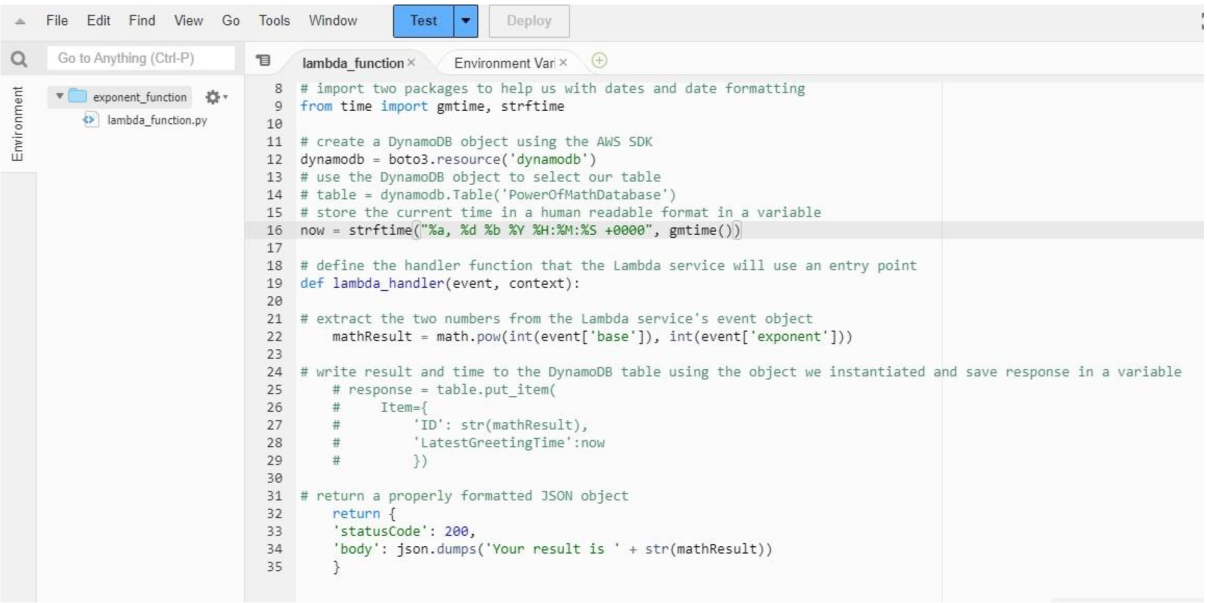
Function name
Enter a name that describes the purpose of your function.
exponent_function
Use only letters, numbers, hyphens, or underscores with no spaces.

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 3.12

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

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

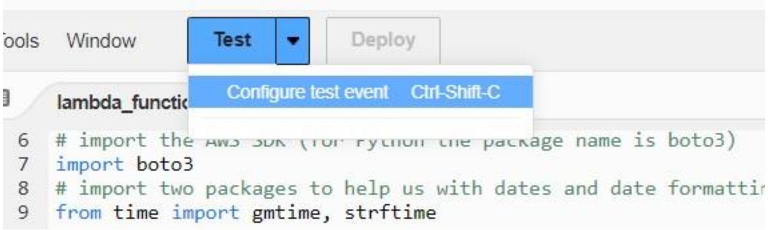
- Write the lambda code. The boto3 is the AWS SDK for python. It is imported as a library and used to gain access to DynamoDB. The DynamoDB table, math_db will be created soon. The function takes 2 numbers, parsed through the event object, and performs the math calculation. The DynamoDB 'putItem' method is then used to send the result, together with the timestamp to the math_db table.



```
8  # import two packages to help us with dates and date formatting
9  from time import gmtime, strftime
10
11 # create a DynamoDB object using the AWS SDK
12 dynamodb = boto3.resource('dynamodb')
13 # use the DynamoDB object to select our table
14 # table = dynamodb.Table('PowerOfMathDatabase')
15 # store the current time in a human readable format in a variable
16 now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())
17
18 # define the handler function that the Lambda service will use an entry point
19 def lambda_handler(event, context):
20
21 # extract the two numbers from the Lambda service's event object
22     mathResult = math.pow(int(event['base']), int(event['exponent']))
23
24 # write result and time to the DynamoDB table using the object we instantiated and save response in a variable
25     # response = table.put_item(
26     #     Item={
27     #         'ID': str(mathResult),
28     #         'LatestGreetingTime':now
29     #         })
30
31 # return a properly formatted JSON object
32     return {
33     'statusCode': 200,
34     'body': json.dumps('Your result is ' + str(mathResult))
35     }
```

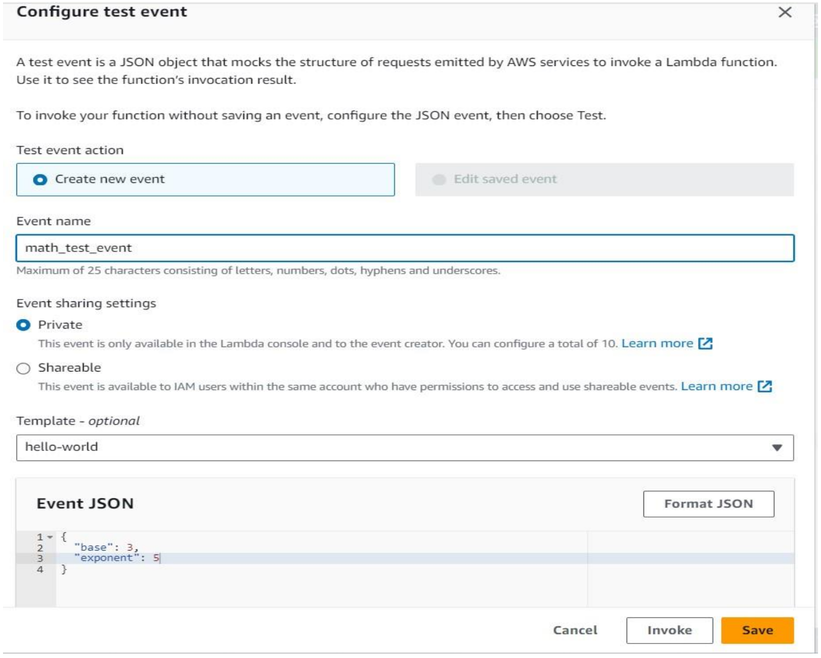
It is advised to test the function before proceeding. However, because the table hasn't been created, testing the entire code will raise errors. For this reason, I commented out the code at line 14 and 24 – 29 before testing.

- On the top of the code tab of the lambda function, I clicked the down arrow next to the Test button to create a test event with two test numbers 3 and 5.



```
6 # import the AWS SDK (for Python the package name is boto3)
7 import boto3
8 # import two packages to help us with dates and date formattir
9 from time import gmtime, strftime
```

- I



Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event ○ Edit saved event

Event name

math_test_event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

hello-world

Event JSON

```
{
  "base": 3,
  "exponent": 5
}
```

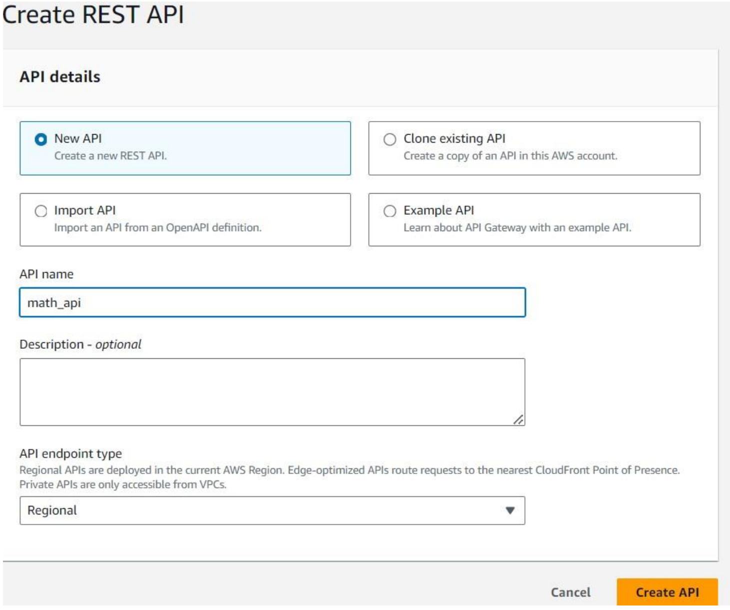
- Save the test event and click on the Test button. It should a status 200 code with the exponential of 2 to the power of 5 which equals 243. This shows that the lambda function works.

Step 3: Configure API Gateway

The reason for the use of API gateway is to enable our application invoke the lambda function to do the calculation when the base and exponents are inputted.

- Still on the AWS console, I searched for API Gateway and created a new API. I called the Rest API math_api and accepted all the other defaults.

A rest API is used as preference to the others because



Create REST API

API details

● New API
Create a new REST API.

○ Clone existing API
Create a copy of an API in this AWS account.

○ Import API
Import an API from an OpenAPI definition.

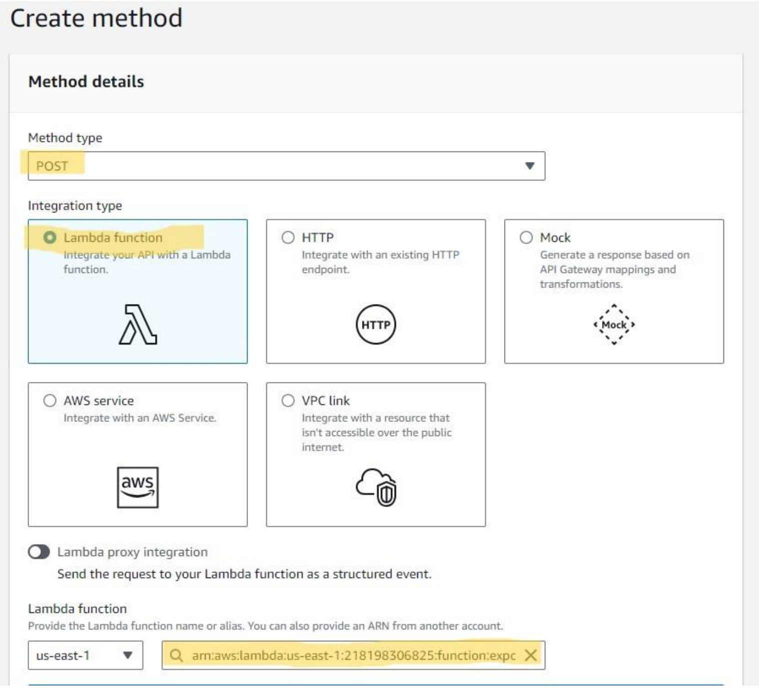
○ Example API
Learn about API Gateway with an example API.

API name
math_api

Description - *optional*

API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.
Regional

- Next, under the Resource section, I created a POST method that will make a post request to the Lambda function (its arn selected during creation) I created earlier with the 2 numbers entered by the visitor to my app. Recall that the web application will call this API function from the code.



Create method

Method details

Method type
POST

Integration type

● Lambda function
Integrate your API with a Lambda function.

○ HTTP
Integrate with an existing HTTP endpoint.

○ Mock
Generate a response based on API Gateway mappings and transformations.

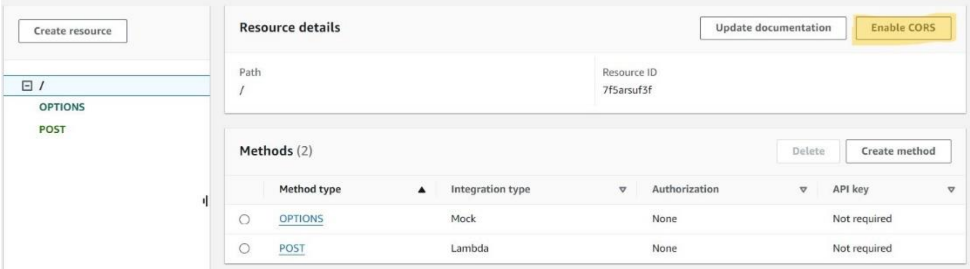
○ AWS service
Integrate with an AWS Service.

○ VPC link
Integrate with a resource that isn't accessible over the public internet.

Lambda proxy integration
Send the request to your Lambda function as a structured event.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.
us-east-1
arn:aws:lambda:us-east-1:218198306825:function:expc

- After creating the API, I enabled CORS to allow appl



Resource details

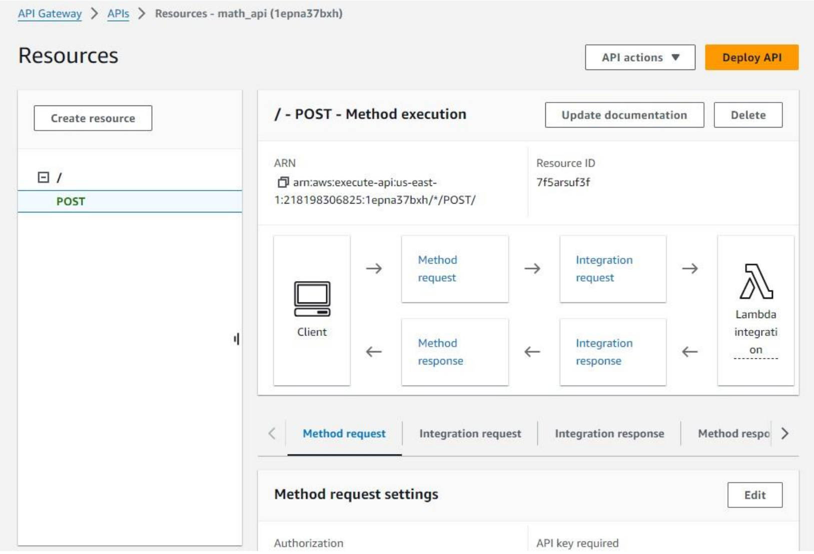
Path
/

Resource ID
7f5arsuf3f

Methods (2)

	Method type	Integration type	Authorization	API key
○	OPTIONS	Mock	None	Not required
○	POST	Lambda	None	Not required

The web UI should look like this showing the flow of traffic between the client, the API and the lambda function.



/ - POST - Method execution

ARN
arn:aws:execute-api:us-east-1:218198306825:1epna37bxh/*/POST/

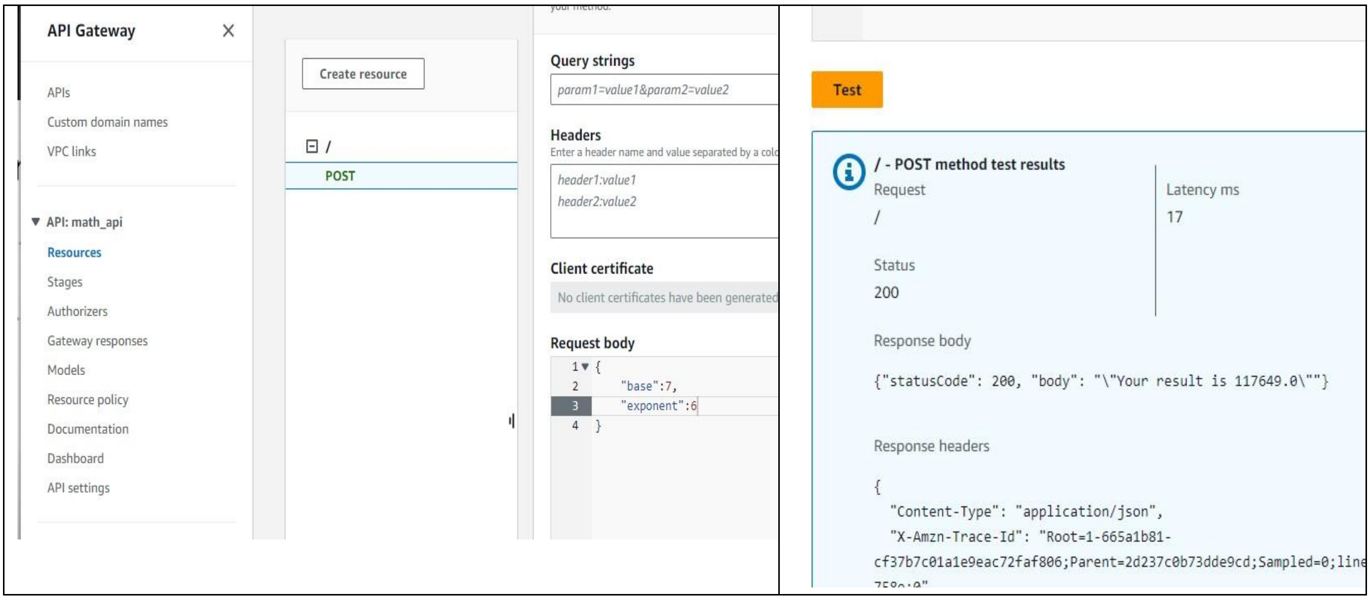
Resource ID
7f5arsuf3f

Method request settings

Authorization

API key required

- Next, I deployed the API using the button in orange at the top-right and called the deployment, 'dev'.
- To test connectivity, I used sample numbers in the 'Request body' field. The output shows that the API reaches the lambda function successfully.



Query strings

param1=value1&param2=value2

Headers

Enter a header name and value separated by a colo

header1:value1
header2:value2

Client certificate

No client certificates have been generated

Request body

```
{
    "base":7,
    "exponent":6
}
```

/ - POST method test results

Request
/

Latency ms
17

Status
200

Response body

```
{"statusCode": 200, "body": "\"Your result is 117649.0\""}
```

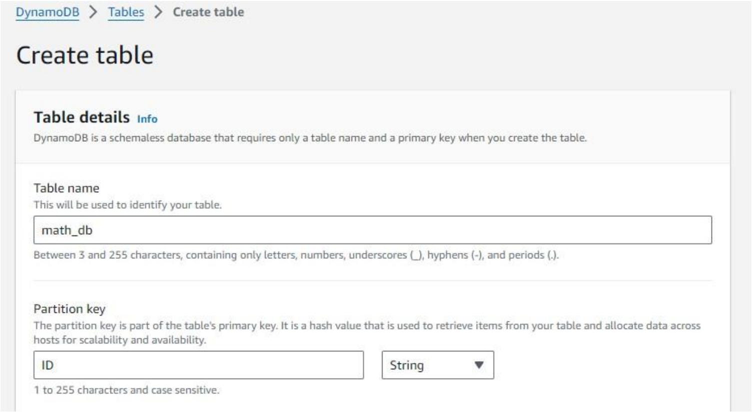
Response headers

```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-665a1b81-
cf37b7c01a1e9eac72faf806;Parent=2d237c0b73dde9cd;Sampled=0;line
7E8c:0"
```

Step 4: Configure DynamoDB

I used DynamoDB database to store the result of the calculation returned by lambda. This DB is perfect for our use case as it is a serverless key-value pair database that does not require me to specify the schema before using it.

- I created a DynamoDB table called math_db with its partition key as 'ID'



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.

math_db

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 ▼

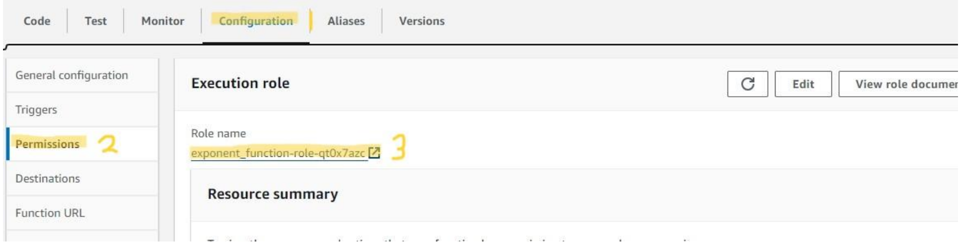
1 to 255 characters and case sensitive.

- Once created, I copied the arn of the table to a safe location, say notepad.

Step 5: Edit the IAM permissions of the lambda function

This step is necessary to grant lambda the permissions to read, write and update the DynamoDB table I just created.

- Back at the lambda function UI, under the configuration tab, under the permissions section, I clicked on the role name to access its permissions in IAM



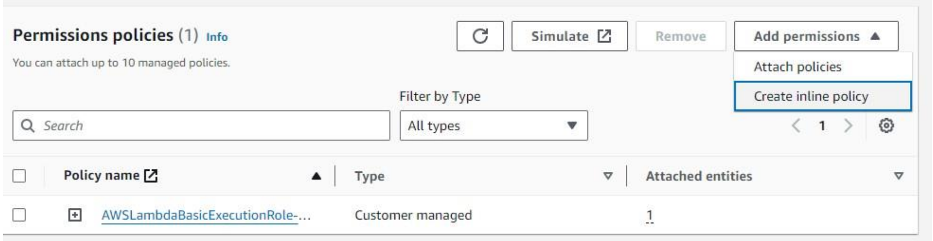
Execution role

Role name

exponent_function-role-qt0x7azc

Resource summary

- Next was I created a new policy for DynamoDB.



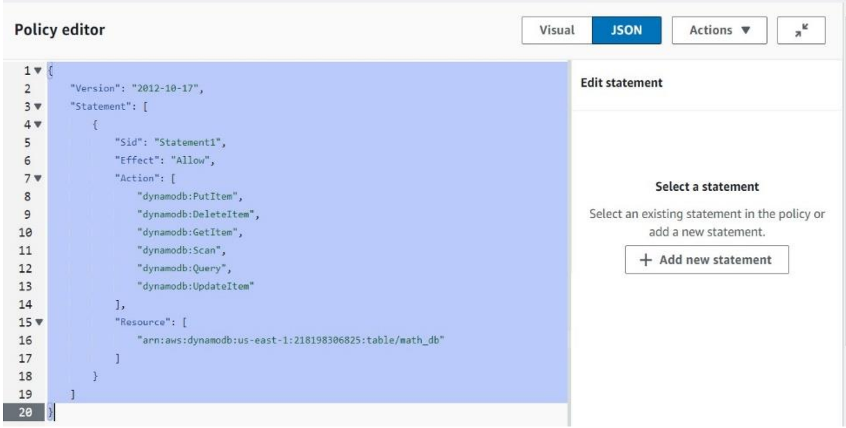
Permissions policies (1) Info

You can attach up to 10 managed policies.

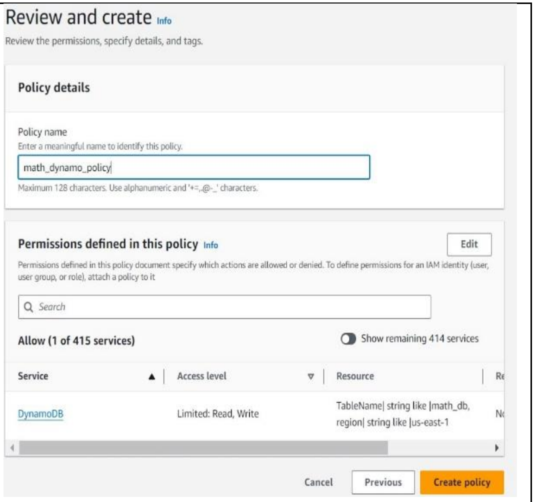
Filter by Type: All types ▼

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

- Using the JSON policy statement builder. I named the policy 'math_dynamo_policy'



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:218198306825:table/math_db"
            ]
        }
    ]
}
```



Review and create

Review the permissions, specify details, and tags.

Policy details

Policy name

Enter a meaningful name to identify this policy.

math_dynamo_policy

Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

Permissions defined in this policy

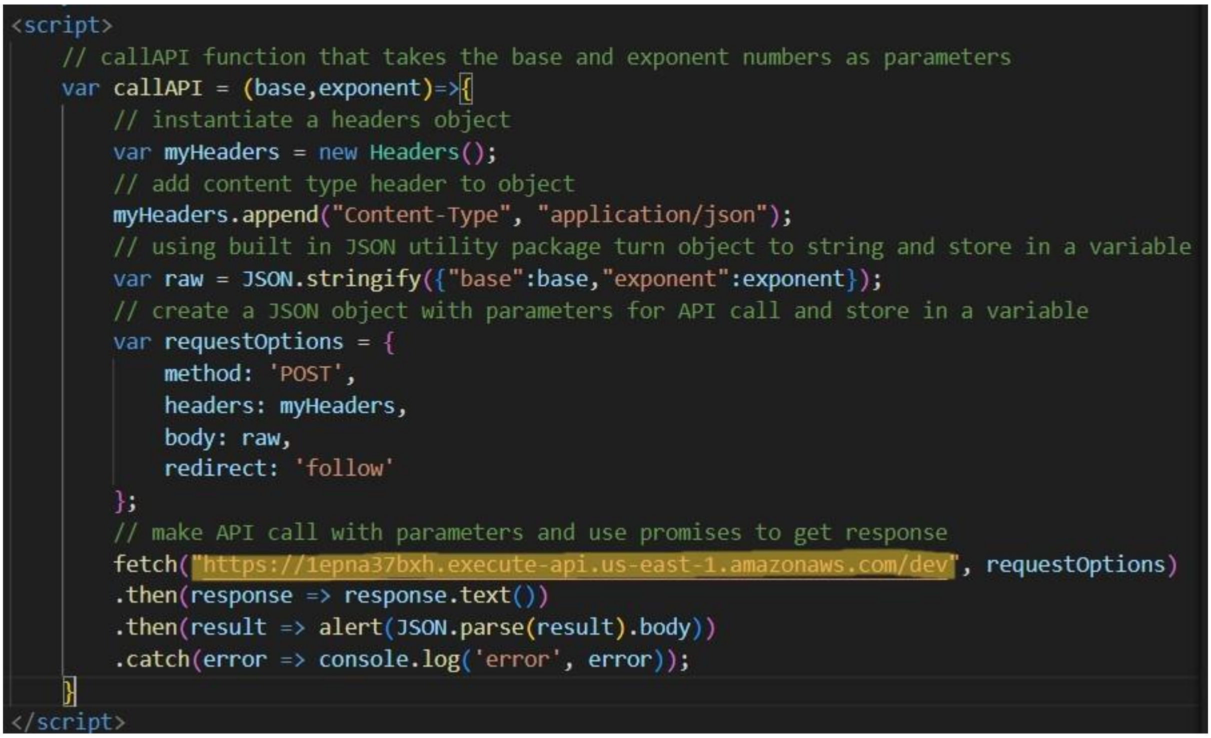
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (1 of 415 services) — Show remaining 414 services

Service	Access level	Resource	Re
DynamoDB	Limited: Read, Write	TableName| string like |math_db, region| string like |us-east-1	N

Step 6: editing the code with the API Gateway's endpoint

In this last step, I simply edited the web application with the endpoint of the API Gateway to grant it application access to the lambda function.

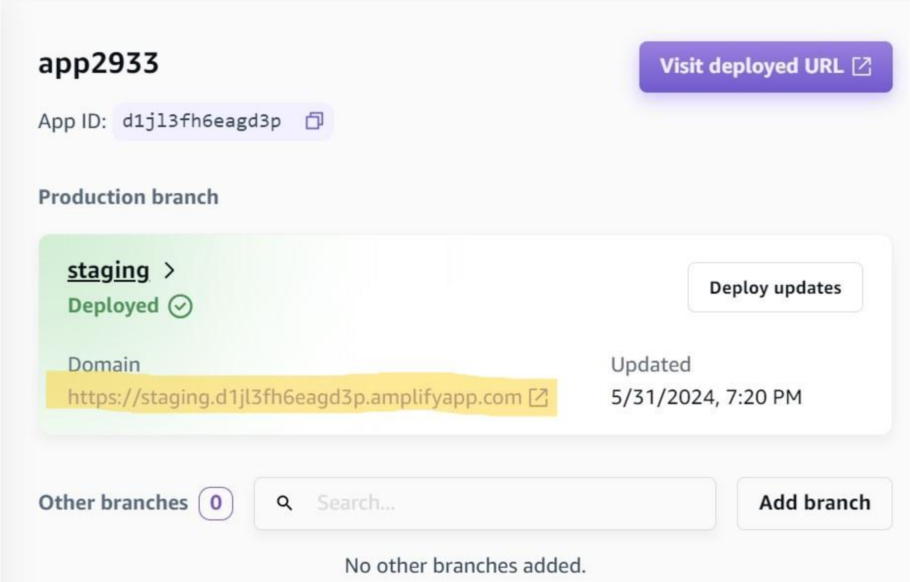


```
<script>
    // callAPI function that takes the base and exponent numbers as parameters
    var callAPI = (base,exponent)=>{
        // instantiate a headers object
        var myHeaders = new Headers();
        // add content type header to object
        myHeaders.append("Content-Type", "application/json");
        // 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://1epna37bxh.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>
```

- Next, I saved and zipped this html file again and deployed it to AWS amplify as I did previously.

Testing: moment of truth

- In AWS Amplify, clicking the domain url takes us to the deployed where we can test the application and its interaction with all the other AWS services used in this project.



app2933

App ID: d1jl3fh6eagd3p

Production branch

staging >

Deployed

Domain

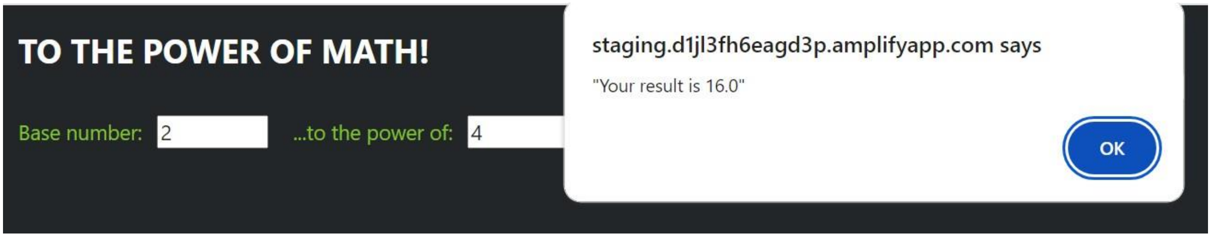
https://staging.d1jl3fh6eagd3p.amplifyapp.com

Updated

5/31/2024, 7:20 PM

Other branches 0

No other branches added.



TO THE POWER OF MATH!

Base number: 2 ...to the power of: 4

staging.d1jl3fh6eagd3p.amplifyapp.com says

"Your result is 16.0"

- Yaay!!! It worked.

Challenges:

The two challenges I had in this project were:

1. deciding which type of API to use: I had to make the decision to go with either REST, HTTP or Websockets. I eventually chose REST API since the web application was going to be accessed

publicly, would require caching, and I was not really concerned about latency issues. I would have gone with HTTP API perhaps if the application was going to be internal only, or if I cared about latency. I would have used websockets if I was building a stateful application and required real time-time updates, like in a game.

2. I did not enable CORS for the API initially, and I thought the code was broken because the application was not returning the alert message with the result of the lambda calculation. However, after careful consideration of my code and inspecting what the web-browser returned, I discovered I needed to enable this setting. Cross-Origin Resource Sharing (CORS) is a security feature implemented by web browsers to control how web applications can request resources from a different domain (origin) than the one that served the web page. Enabling this fixed the issue.

Note: Obviously, there are better ways to launch/deploy your applications, for instance, using CI/CD pipelines, but that's not the focus for this project. I will do a later project that uses the advanced/real-world application deployment strategies.