

AWS Project: Architect and Build an End-to-End AWS Web Application from Scratch

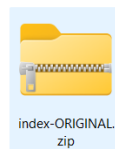
For this AWS project, I will create a simple calculator to compute powers using a given base and exponent. The purpose of this project is to piece together 5 different AWS Services to run and deploy a simple math application. This application needs a way to create and host a webpage (AWS Amplify), invoke the math functionality (API Gateway), do the calculation (Lambda), store and return the results (DynamoDB), and handle permissions (IAM).

Create and Host a Webpage

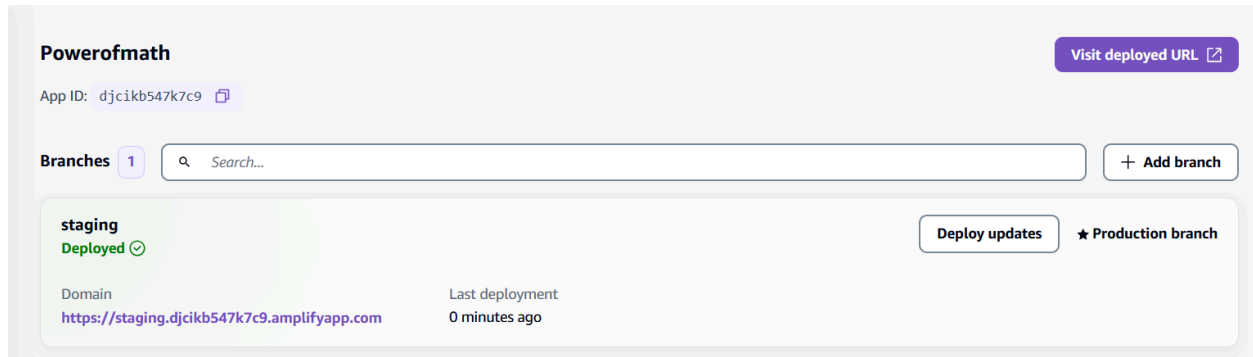
To create and host a webpage for the application, AWS Amplify will be used. AWS Amplify is a managed service used to build and host websites easily. I initially created a basic, bare-bones html file:

```
index-ORIGINAL.html X
C: > Users > Miguel Huerto > Downloads > index-ORIGINAL.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Exponents Calculator</title>
7 </head>
8 <body>
9   Raise to the Power of Infinity!
10 </body>
11 </html>
12
```

Then compressed the html file into a ZIP.

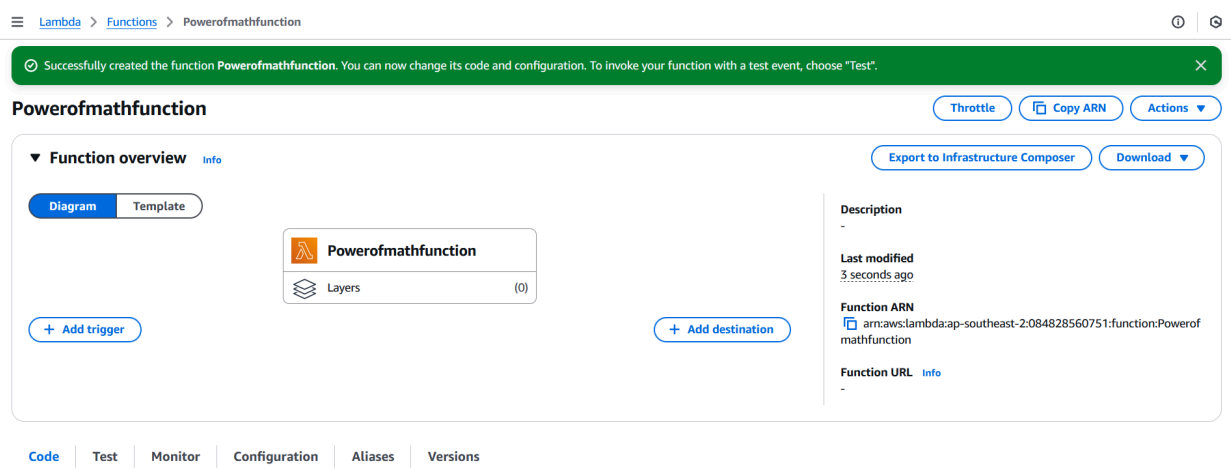


On AWS Amplify, I uploaded the ZIP file, named it Powerofmath, and deployed the application.



Calculate the Result

AWS Lambda will be used to execute the underlying calculations of the application. I first created the “Powerofmath” Lambda function to handle this.



For the logic of the lambda function, I specified the following:

```
lambda_function.py X
lambda_function.py
1  # import the JSON utility package
2  import json
3  # import the Python math library
4  import math
5
6  # define the handler function that the Lambda service will use as an entry point
7  def lambda_handler(event, context):
8
9      # extract the two numbers from the Lambda service's event object
10     mathResult = math.pow(int(event['base']), int(event['exponent']))
11
12     # return a properly formatted JSON object
13     return {
14         'statusCode': 200,
15         'body': json.dumps('Your result is ' + str(mathResult))
16     }
```

Once that was complete, I tested out the function by passing in test values and validated if the output was correct.

The screenshot shows the AWS Lambda console interface. On the left, the Python code for the lambda handler is displayed:

```
2 import json
3 # import the Python math library
4 import math
5
6 # define the handler function that the Lambda service will use
7 def lambda_handler(event, context):
8
9     # extract the two numbers from the Lambda service's event
10     mathResult = math.pow(int(event['base']), int(event['exponent']))
11
12     # return a properly formatted JSON object
13     return {
14         'statusCode': 200,
15         'body': json.dumps('Your result is ' + str(mathResult))
16     }
```

On the right, the 'Event JSON' is shown as:

```
1 {
2   "base": 2,
3   "exponent": 2
4 }
```

Below the code, the 'Execution Results' section shows the status 'Succeeded' and the response:

```
{
  "statusCode": 200,
  "body": "\"Your result is 4.0\""
}
```

Invoke the Lambda Function

A public endpoint or URL is needed for the users to invoke and trigger the lambda function. API Gateway is used to build API's, which can be used to invoke Lambda functions. I created a "PowerofMath" API and enabled CORS (Cross Origin Resource Sharing) which allows for different domains to access resources from another origin or domain. In this case Amplify which has its own domain will access resources from Lambda, which has its own domain.

Resources

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with 'Create resource' and a tree view showing the resource path '/'. The main area is titled 'Resource details' and shows the path '/' and resource ID 'jn3mt6ylk3'. There are buttons for 'API actions', 'Deploy API', 'Update documentation', and 'Enable CORS'. Below this, the 'Methods (2)' section shows a table with two methods: 'OPTIONS' and 'POST'. Both methods are of type 'Mock' and have 'None' for authorization and 'Not required' for API key.

Method type	Integration type	Authorization	API key
<input type="radio"/> OPTIONS	Mock	None	Not required
<input type="radio"/> POST	Lambda	None	Not required

Testing the API and Validating its results:

Test method

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

Query strings

param1=value1¶m2=value2

Headers

Enter a header name and value separated by a colon (:). Use a new line for each header.

header1:value1
header2:value2

Client certificate

No client certificates have been generated.

Request body

```
1 {  
2   "base": 2,  
3   "exponent": 4  
4 }
```

POST method test results

Request	Latency ms	Status
/	192	200
Response body <pre>{"statusCode": 200, "body": "\"Your result is 16.0\""}</pre>		
Response headers <pre>{ "Content-Type": "application/json", "X-Amzn-Trace-Id": "Root=1-67b6b7aa-e24c4a0b4dd1fac1ff0f0ae6;Parent=179ff184d8ac9e07;Sampled=0;Lineage=1:575effba:0" }</pre>		

Store and Return the Results

DynamoDB, a lightweight, Key-Value, NoSQL database was used for to store and return results.

PowerofMathDB ☆ Actions Explore table items

< **Overview** Indexes Monitor Global tables Backups Exports and streams Permissions >

Protect your DynamoDB table from accidental writes and deletes Edit PITR ×

When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. [Learn more](#)

General information Info

Partition key ID (String)	Sort key -	Capacity mode On-demand	Table status ✔ Active
Alarms ✔ No active alarms	Point-in-time recovery (PITR) Info ☹ Off	Resource-based policy Info ☹ Not active	

Back at the Lambda Function, an inline policy was created to allow it to write to the DynamoDB as seen below.

Policy editor

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "VisualEditor0",
6       "Effect": "Allow",
7       "Action": [
8         "dynamodb:PutItem",
9         "dynamodb>DeleteItem",
10        "dynamodb:GetItem",
11        "dynamodb:Scan",
12        "dynamodb:Query",
13        "dynamodb:UpdateItem"
14      ],
15       "Resource": "arn:aws:dynamodb:ap-southeast-2:084828560751:table/PowerofMathDB"
16     }
17   ]
18 }
```

The Code was then edited again to incorporate access to the Database into the code itself, and also some styling choices were also added.

```
lambda_function.py X
lambda_function.py
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         })
```

Final Result:

dev.d1mtnh3mxbk6kz.amplifyapp.com

☆

M

TO THE POWER OF MATH!

Base number: ...to the power of:

CALCULATE