Lab: AWS Lambda, Python based

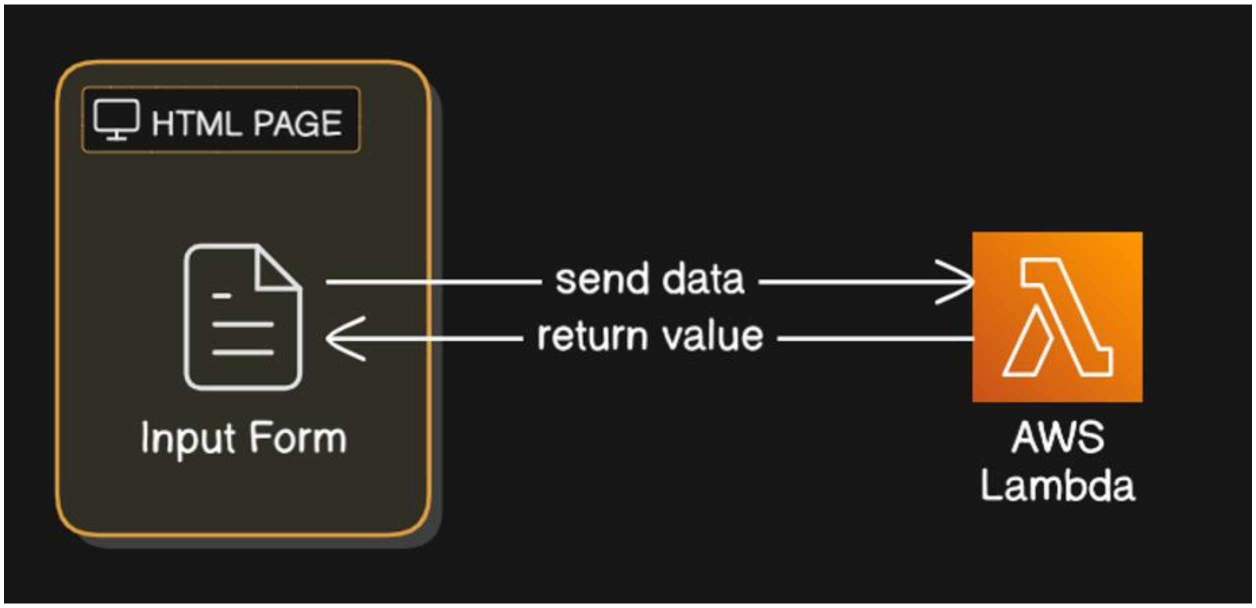
Lab overview and objectives

In this lab, you will create an AWS Lambda function using Python.

You will also create a Function URL that will trigger the AWS Lambda function from a web browser and an HTML page that triggers the AWS Lambda function.

After completing this lab, you should be able to:

- Create a Python based **AWS Lambda** function from the **AWS Management Console**.
- Test the functions from **AWS Lambda Test** tab.
- Execute the functions from a Web Browser using **Function URL**.
- Execute the functions from an HTML page using **Function URL**.



Scenario

You will create a Lambda function that will get 3 parameters, based on those a calculation will be performed, and a value will be returned.

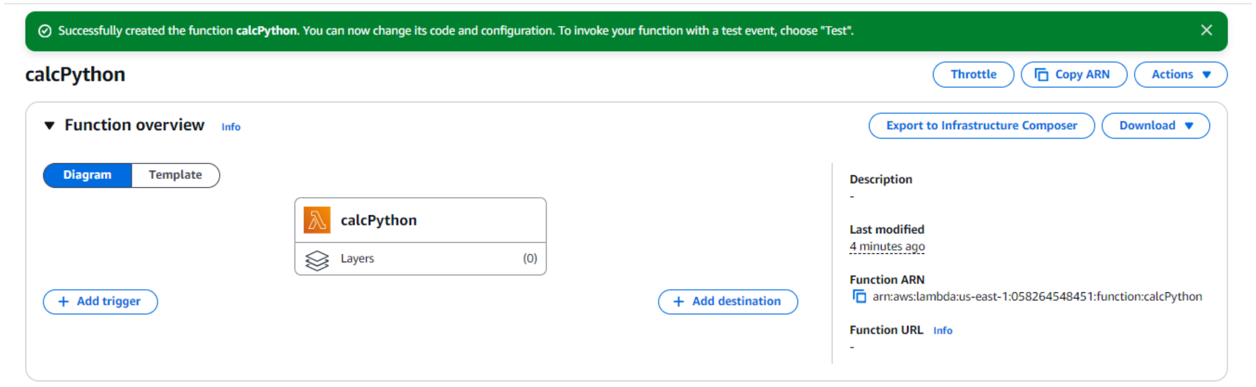
Accessing the AWS Management Console

For this lab we will use the **AWS Academy learner lab** and will be using the **AWS Management Console**.

Task 1: Create a Python Lambda function

1. In the search box to the right of **Services**, search for and click **Lambda** to open the AWS Lambda console.
2. Click **Create a function**.
3. On the **Create function** screen, configure these settings:
   - Click **Author from scratch.**
   - Function name: `calc-python`
   - Runtime: **Python 3.12**
   - Click **Change default execution role.**
   - Execution role: **Use an existing role.**
   - Existing role: From the dropdown list, choose **LabRole**
4. Click **Create function**.

5. The following screen should appear.


Successfully created the function **calcPython**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

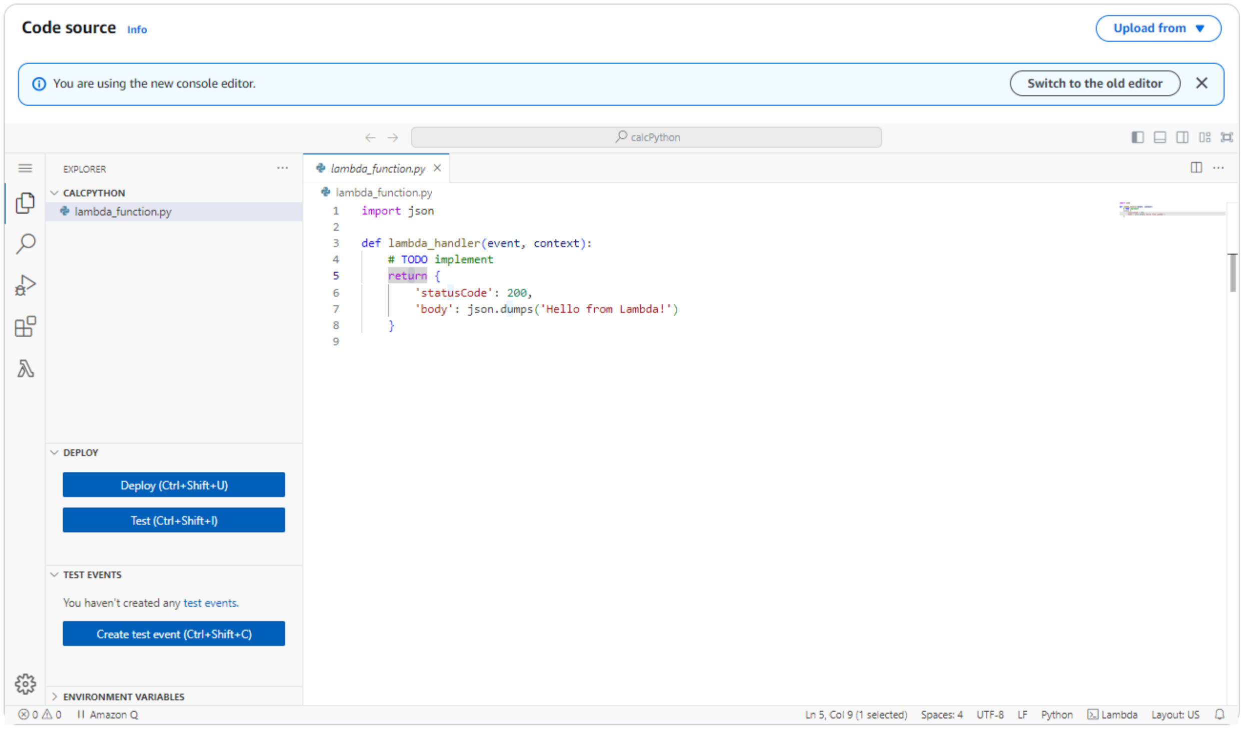
calcPython

Description
-

Last modified
4 minutes ago

Function ARN
arn:aws:lambda:us-east-1:058264548451:function:calcPython

Function URL Info
-


Code source

You are using the new console editor.

You haven't created any test events.

lambda_function.py

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Task 2: Configure the Lambda function

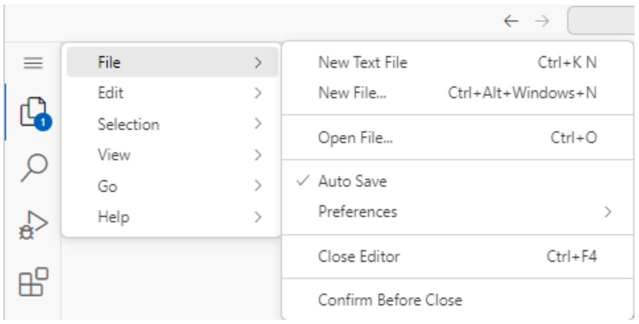
In this task, you will paste a few lines of python code to retrieve 3 parameters, calculate a result and return it. Writing code to complete this task is optional.

1. Below the **Function overview** pane, click **Code**, and then click *lambda_function.py* to display and edit the Lambda function code.
2. In the **Code source** pane, delete the existing code, and *write an AWS Lambda function in Python that performs basic calculator operations. It should accept HTTP GET requests via a Lambda Function URL. The function should extract three query string parameters from the event: num1, num2, and operation. Supported operations are +, -, *, and /. The function should return a JSON response with the result. It should handle invalid inputs (missing parameters, non-numeric values, division by zero, and unsupported operations) and return appropriate error messages and HTTP status codes. Use the standard Lambda function signature: lambda_handler(event, context).*

   Alternatively, copy the following code, and paste it in the box:

```
import json

def lambda_handler(event, context):

    print("event = ",event)

    try:
        # Retrieve parameters
        num1 = float(event['queryStringParameters']['num1'])
        num2 = float(event['queryStringParameters']['num2'])
        operation = event['queryStringParameters']['operation']

        print('num1 = ', num1 , ", num2 = " , num2 , ", operation = " , operation)

        # Perform the operation
        if operation == '+':
            result = num1 + num2
        elif operation == '-':
            result = num1 - num2
        elif operation == '*':
            result = num1 * num2
        elif operation == '/':
            if num2 == 0:
                return {
                    'statusCode': 400,
                    'body': json.dumps({'error': 'Division by zero is not allowed'})
                }
            result = num1 / num2
        else:
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'Invalid operation'})
            }

        # Return the result
        return {
            'statusCode': 200,
            'body': json.dumps({'result': result})
        }

    except (KeyError, ValueError) as e:
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Invalid input parameters'})
        }
```

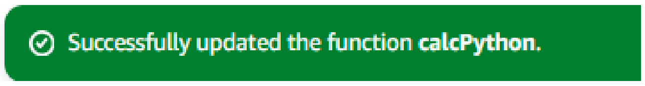
14. By default, file saving is marked as **Auto Save**. To verify this, click the **File** menu and look for the **V** next to the **Auto Save** option.



Alternatively, **Save** the changes.

15. In the **Code source** box, click **Deploy**.

Your Lambda function is now fully configured.

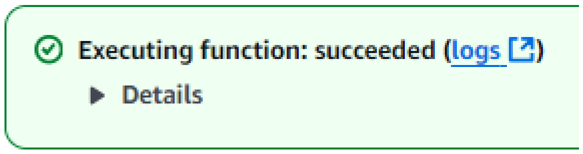

Successfully updated the function **calcPython**.

Task 3: Verify that the Lambda function works

1. Click **Test** tab, and update the following JSON in the **Event JSON** panel:

```
{
  "queryStringParameters": {
    "num1": "4",
    "num2": "2",
    "operation": "+"
  }
}
```

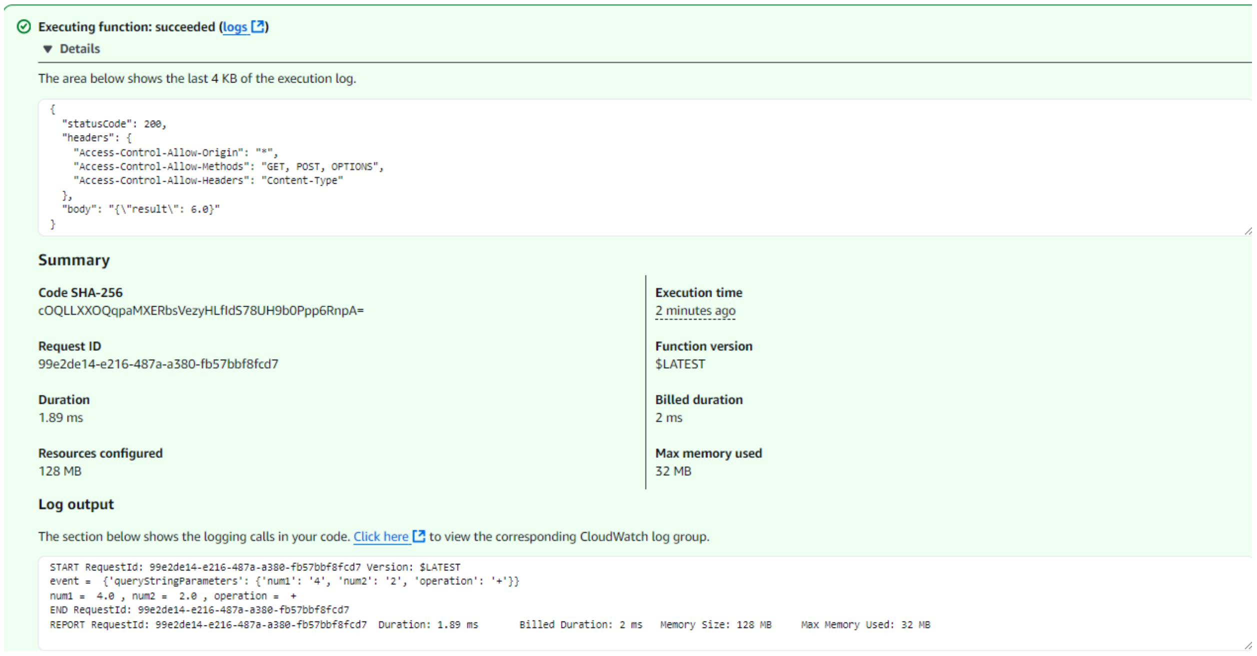
2. Click **Test**. The function will execute and should finish successfully:



Executing function: succeeded (logs)

Details

3. Click the **Details** drop down and inspect the execution. You should see the response, Summary and Log output.



Executing function: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "headers": {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET, POST, OPTIONS",
    "Access-Control-Allow-Headers": "Content-Type"
  },
  "body": "{\"result\": 6.0}"
}
```

Summary

Code SHA-256 cOQLLXXOQqpaMXERbsVezyHLfIdS78UH9b0Ppp6RnpA=	**Execution time** 2 minutes ago
Request ID 99e2de14-e216-487a-a380-fb57bbf8fcd7	**Function version** $LATEST
Duration 1.89 ms	**Billed duration** 2 ms
Resources configured 128 MB	**Max memory used** 32 MB

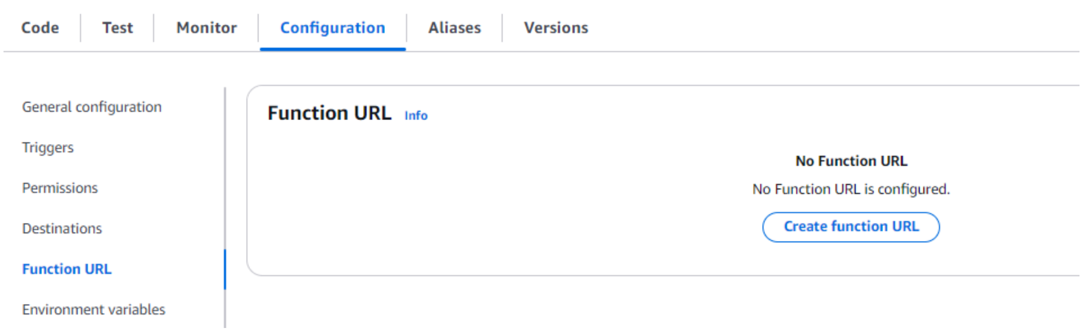
Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 99e2de14-e216-487a-a380-fb57bbf8fcd7 Version: $LATEST
event =  {'queryStringParameters': {'num1': '4', 'num2': '2', 'operation': '+'}}
num1 =  4.0 , num2 =  2.0 , operation =  +
END RequestId: 99e2de14-e216-487a-a380-fb57bbf8fcd7
REPORT RequestId: 99e2de14-e216-487a-a380-fb57bbf8fcd7  Duration: 1.89 ms      Billed Duration: 2 ms   Memory Size: 128 MB     Max Memory Used: 32 MB
```

Task 4: Execute the Lambda function from a Web Browser

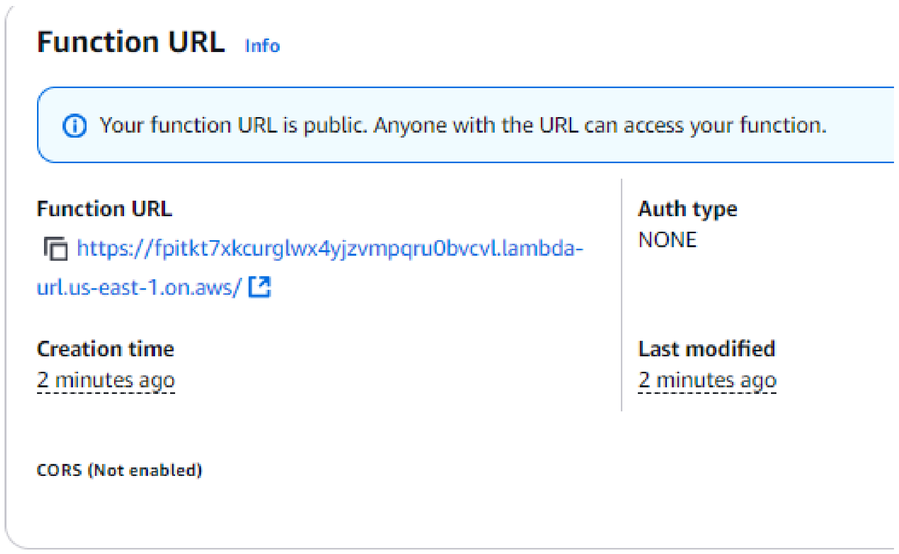
1. Click **Configuration** tab, **Function URL** (a dedicated HTTP(S) endpoint) option and click **Create Function URL**.



No Function URL

No Function URL is configured.

2. In the **Configure Function URL** section, select **None** and press **Save**. Your Function URL should now be created.



Your function URL is public. Anyone with the URL can access your function.

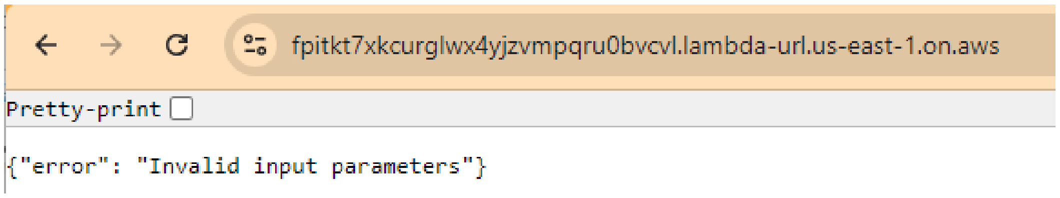
Function URL: https://fpitkt7xkcurglwx4yjzvmpqru0bvcvl.lambda-url.us-east-1.on.aws/

Auth type: NONE

Creation time: 2 minutes ago

Last modified: 2 minutes ago

3. Copy your Function URL, and from your preferred web browser, execute it. You should get an error of missing parameters.



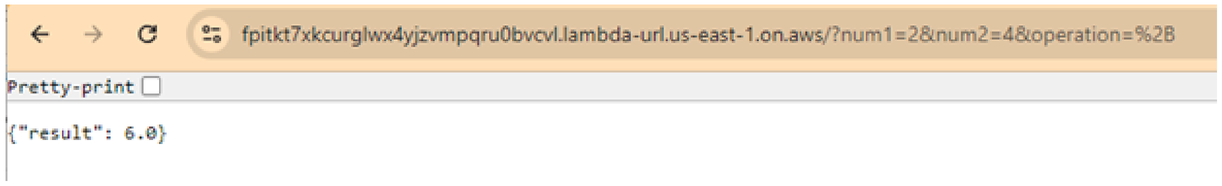
fpitkt7xkcurglwx4yjzvmpqru0bvcvl.lambda-url.us-east-1.on.aws

Pretty-print

```
{"error": "Invalid input parameters"}
```

4. Add the parameters the Lambda function is expecting and re-execute it. You should get a valid return value. Below is an example of such URL.
   Pay attention to the value **%2B**, which is a URL-encoded representation of the **+** character.

   https://fpitkt7xkcurglwx4yjzvmpqru0bvcvl.lambda-url.us-east-1.on.aws/?num1=2&num2=4&operation=%2B


fpitkt7xkcurglwx4yjzvmpqru0bvcvl.lambda-url.us-east-1.on.aws/?num1=2&num2=4&operation=%2B
Pretty-print

```
{"result": 6.0}
```

Task 5: Execute the Lambda function from an HTML page

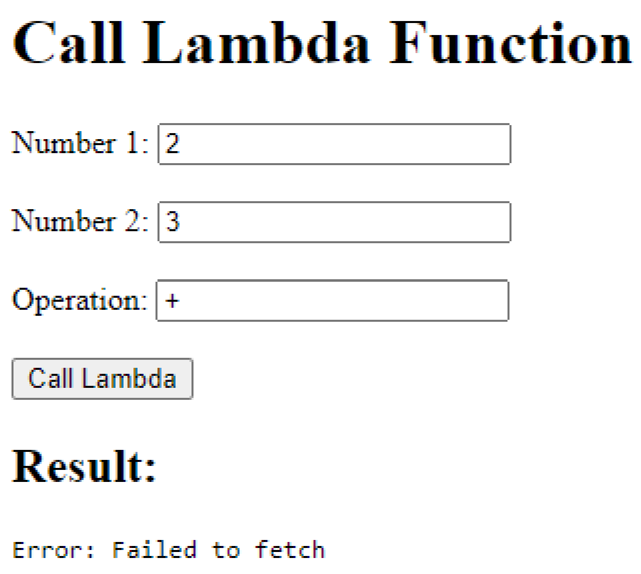
1. *Create a simple HTML page, that will execute the Lambda function, with a form to collect two numbers and an operation from the user. When the user clicks a button, use JavaScript to send a GET request to an AWS Lambda Function URL, passing the inputs as query string parameters (num1, num2, and operation). Display the result returned from the Lambda function on the page. Handle errors with a try/catch block and display them to the user. The function URL should be used directly in the fetch() call, and operation can be any of +, -, *, or /.*

   Alternatively, use the following code.

```
<!DOCTYPE  html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Lambda Function Caller</title>
   <script>
      async function callLambda() {
         const num1 = document.getElementById('num1').value;
         const num2 = document.getElementById('num2').value;
         const operation = document.getElementById('operation').value;
         const url = `https://cps6at76ppmf5yddphw72pebmq0uieib.lambda-url.us-east-
1.on.aws/?num1=${num1}&num2=${num2}&operation=${encodeURIComponent(operation)}`;

         try {
            const response = await fetch(url);
            const data = await response.json();
            document.getElementById('result').innerText = JSON.stringify(data);
         } catch (error) {
            document.getElementById('result').innerText = 'Error:  ' + error.message;
         }
      }
   </script>
</head>
<body>
   <h1>Call Lambda Function</h1>
   <label for="num1">Number 1:</label>
   <input type="text" id="num1" name="num1"><br><br>
   <label for="num2">Number 2:</label>
   <input type="text" id="num2" name="num2"><br><br>
   <label for="operation">Operation:</label>
   <input type="text" id="operation" name="operation"><br><br>
```

```
    <button onclick="callLambda()">Call Lambda</button>
    <h2>Result:</h2>
    <pre id="result"></pre>
</body>
</html>
```

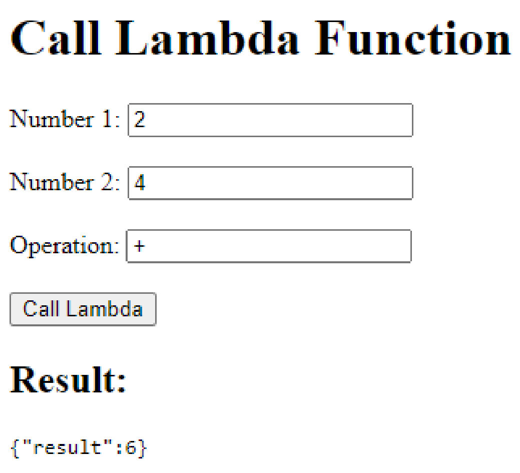
1. Open the HTML page, add the parameters values and execute it. You should get an error: **Error: Failed to fetch**.



```
Error: Failed to fetch
```

This is due to CORS Policy. Our Lambda function is not configured to allow cross-origin requests. Let’s fix this.

3. Click **Configuration** tab, **Function URL** (a dedicated HTTP(S) endpoint) option and click **Edit**.
4. In the **Configure Function URL** section, open the **Additional settings** section and select the **Configure cross-origin resource sharing (CORS),** press **Save**.
5. Re-execute the Lambda function from the HTML page. You should now see a valid response.


Call Lambda Function

Number 1: 2

Number 2: 4

Operation: +

Result:

```
{"result":6}
```

6. An alternate solution to **Configure cross-origin resource sharing (CORS)** in the **Function URL**, is to add CORS header in the Lambda Function:

   Create headers dictionary:

```
# CORS headers
headers = {
   'Access-Control-Allow-Origin':  '*',
   'Access-Control-Allow-Methods':  'GET, POST, OPTIONS',
   'Access-Control-Allow-Headers':  'Content-Type'
}
```

   And send it in the response:

```
return {
   'statusCode': 200,
   'headers': headers,
   'body': json.dumps({'result': result})
}
```

7. **Challenge**: Why do we need CORS setting when calling the Lambda function from HTML and not for Web Browser?
   Answer: When you call a Lambda function directly from a web browser by entering the URL, it works because the browser does not enforce Cross-Origin Resource Sharing (CORS) policies for direct URL access. However, when you call the Lambda function from an HTML page using JavaScript (e.g., fetch), the browser enforces CORS policies to prevent security issues.

Activity complete

Congratulations! You have successfully completed the activity.