😊 Single Calculator Microservice Expose HTTPS methods with Lambda Function URL

Overview:

You're going to create a **calculator microservice** using **AWS Lambda** and expose it via a **Lambda Function URL**. This service will accept POST HTTP requests containing numbers and an operator, perform the calculation, and return the result.

Steps Involved:

1. Design the Event JSON Object

- **Goal**: Define the structure of the request data that the client will send to the Lambda function.
- **What's Included**: The JSON object will contain:
  - num1: First number.
  - num2: Second number.
  - operator: The operation to perform (e.g., +, -, *, /).

2. Develop the Lambda Function

- **Goal**: Write the function that will process the input data (numbers and operator), perform the calculation, and return the result.
- **How it Works**: The Lambda function will:
  - Take the numbers and operator from the request.
  - Perform the specified calculation.
  - Return the result back to the client.

3. Expose the Lambda Function via HTTP

- **Goal**: Make your Lambda function accessible via HTTPS, so clients can send HTTP POST requests to it.
- **How it Works**: AWS provides a **Lambda Function URL**, which allows you to expose the function as a web endpoint. You will configure this to handle incoming POST requests.

4. (Optional) Create ZIP File for Lambda

- **Goal**: If your function uses external libraries or dependencies, you'll package the code into a ZIP file before uploading it to AWS Lambda.
- **When to Use**: If you need to include extra files or dependencies that aren't in the default Lambda environment, this step is required.

5. Configure Lambda (Optional)

- **Goal**: Adjust your Lambda function's settings, such as memory, execution time, and permissions, to make sure it’s optimized for your use case.

6. Test Using Postman

- **Goal**: Send test requests to your Lambda function to ensure it’s working as expected.
- **How to Test**: Use **Postman** to send a **POST** request to the Lambda function URL with the JSON payload, and check that the function performs the calculation correctly.

7. Final Testing and Validation

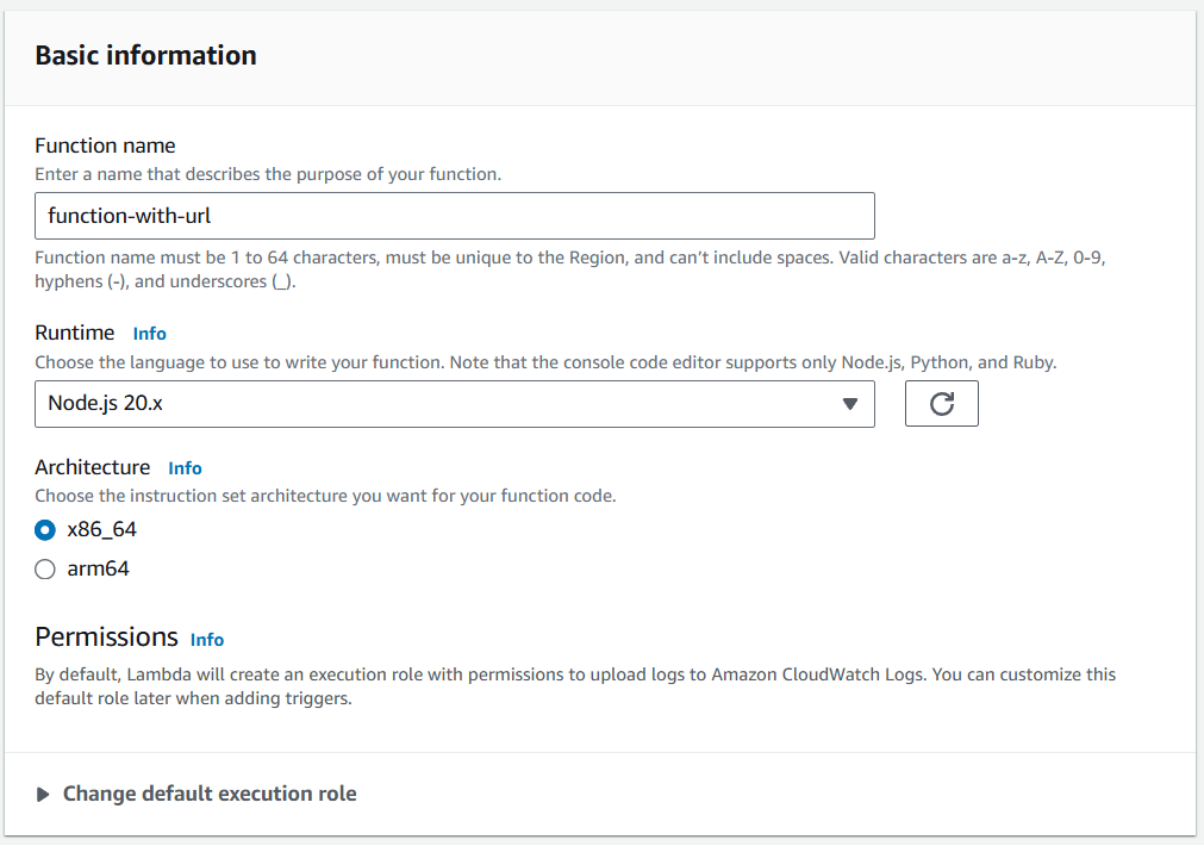
- **Goal**: Perform final tests with different inputs to ensure the Lambda function is handling various cases correctly (e.g., different operations and edge cases like division by zero).

Summary of Steps:

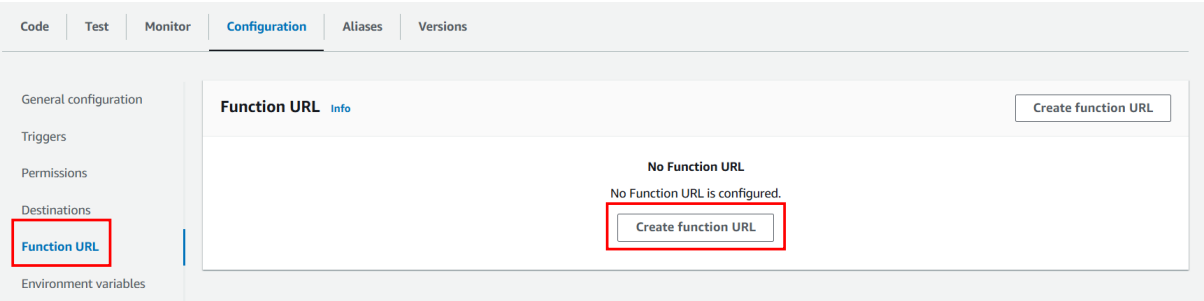
1. **Design the Input**: Decide what data (numbers and operator) will be sent to the Lambda function.
2. **Write the Lambda Function**: Create a function to process the data and calculate the result.
3. **Expose via HTTP**: Set up a Lambda Function URL to handle HTTP POST requests.
4. **Package Code (If needed)**: If required, zip and upload the function with dependencies.
5. **Configure Lambda**: Set Lambda function settings like memory and timeout.
6. **Test with Postman**: Send test requests to ensure the function works.
7. **Final Testing**: Validate the function with different inputs.

😄 To begin with the Lab

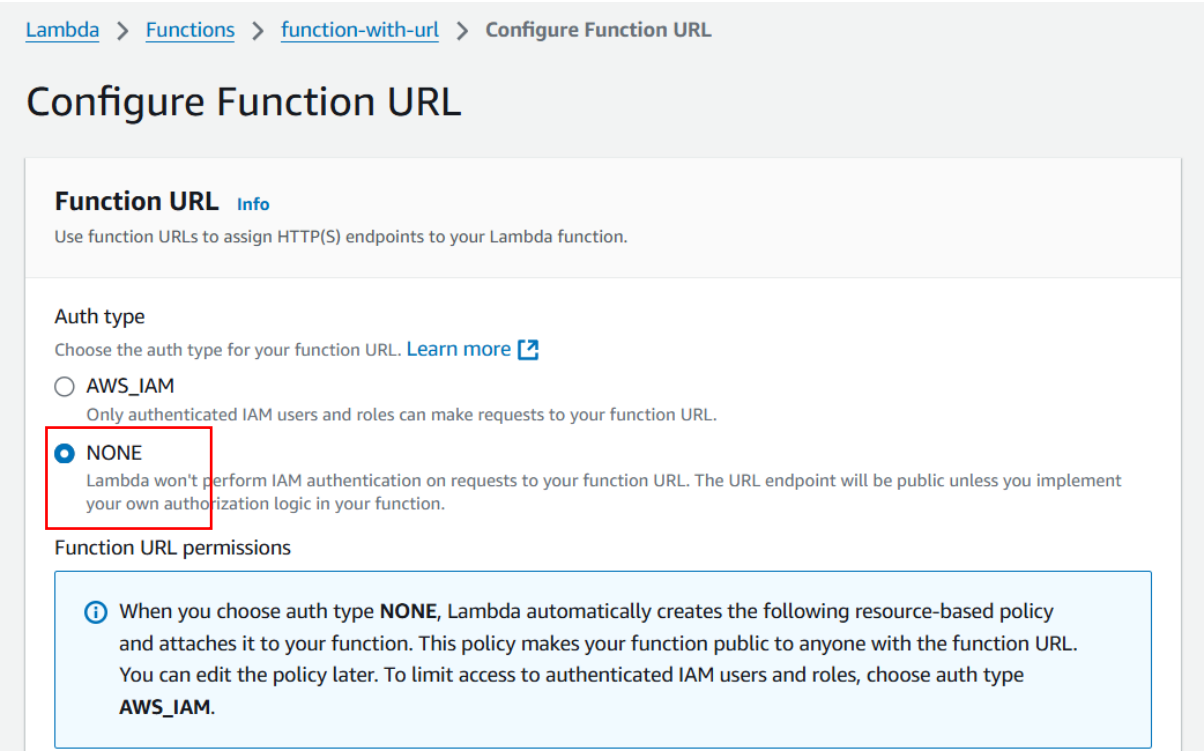
1. The first step is to create our lambda function. Open your AWS Console, search for lambda, and click on the create function from its dashboard.
2. Choose the author from scratch option, give your function a name, choose the runtime as node.js, and create your lambda function.



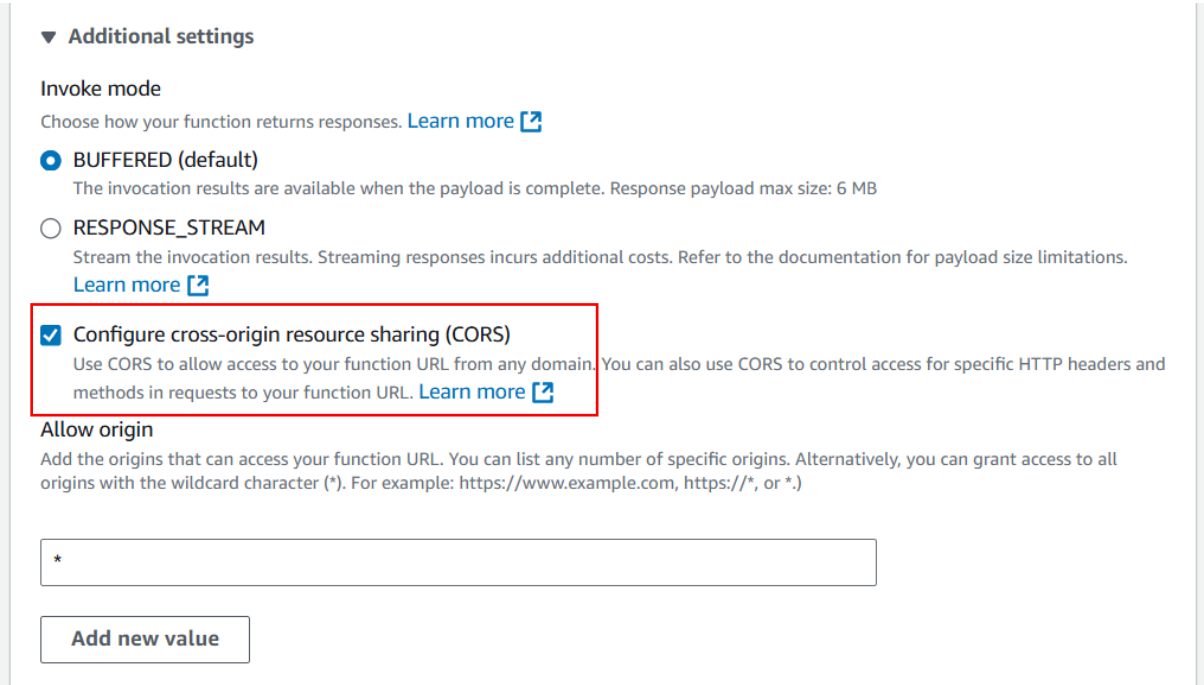
3. Once your function is created go inside, and go to the configuration tab in your function.
4. Here you need to navigate to the function URL and click on Create function URL.



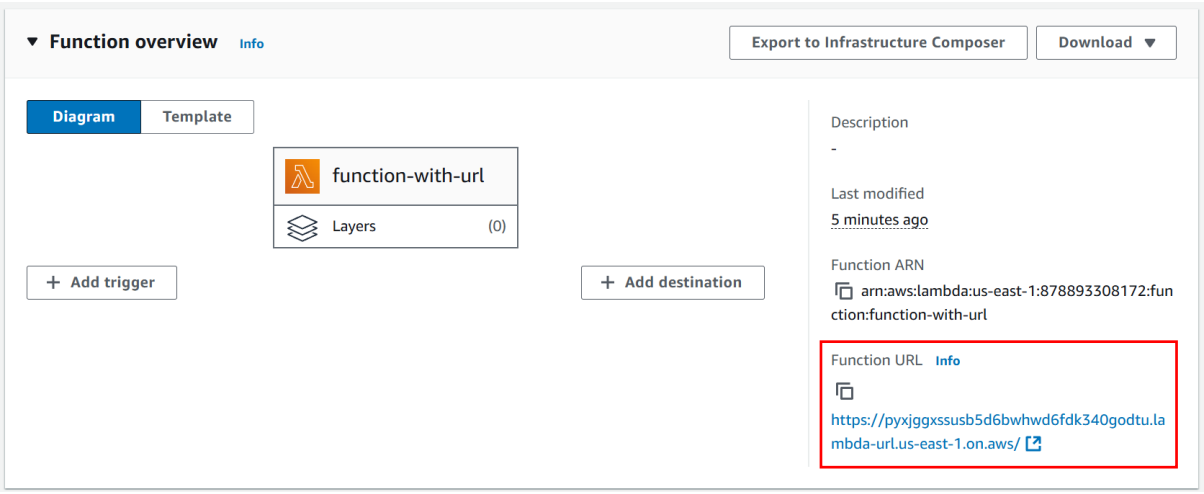
5. Now here you need to choose None for the Authentication type and scroll down to expand the additional features.



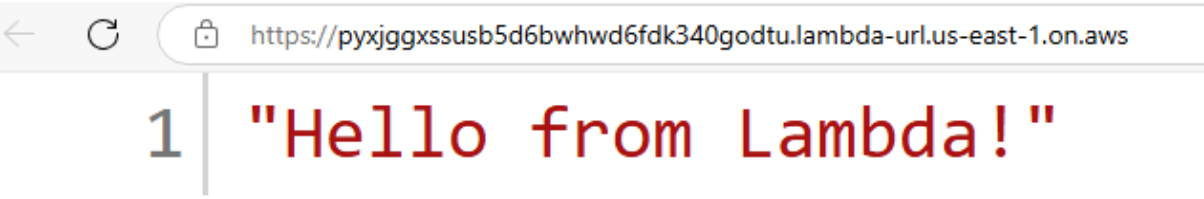
6. In the additional settings you need to enable the CORS feature and then scroll down and save your function URL.



7. In the function overview you will see that now you have a Function URL open this in a new tab.



8. So, using the function URL you have directly invoked your lambda function.

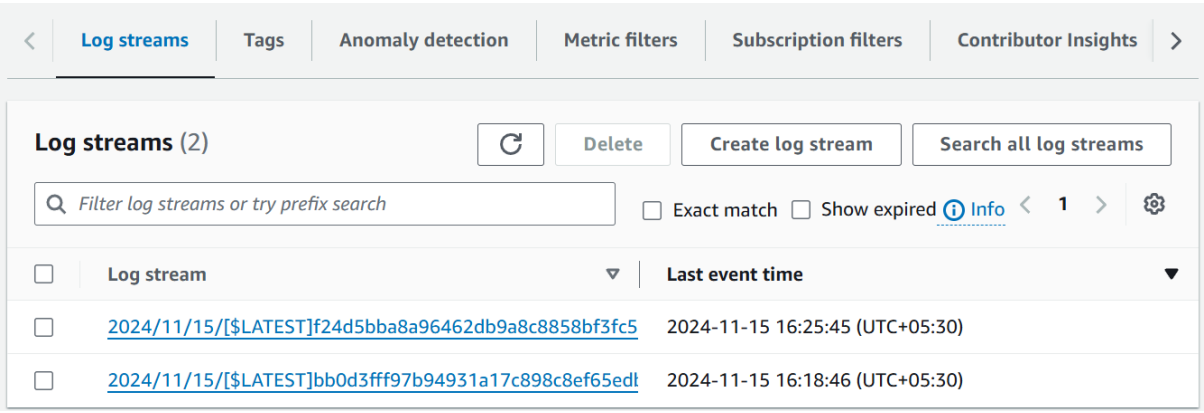


9. Now you need to open the code section of your function and add the line that you can see on number 2. This is typically used for **debugging** and **logging** to understand the structure and contents of the event object in AWS Lambda or any other JavaScript application.

   console.log("EVENT: \n" + JSON.stringify(event, null, 2));

10. Once you have made the changes deploy them and now once again open the function URL to invoke your function.

11. After invoking your function, you need to open the CloudWatch logs for you function and open the latest log stream.



12. The main purpose of adding that line of code was to get the information about the invocation of our function.

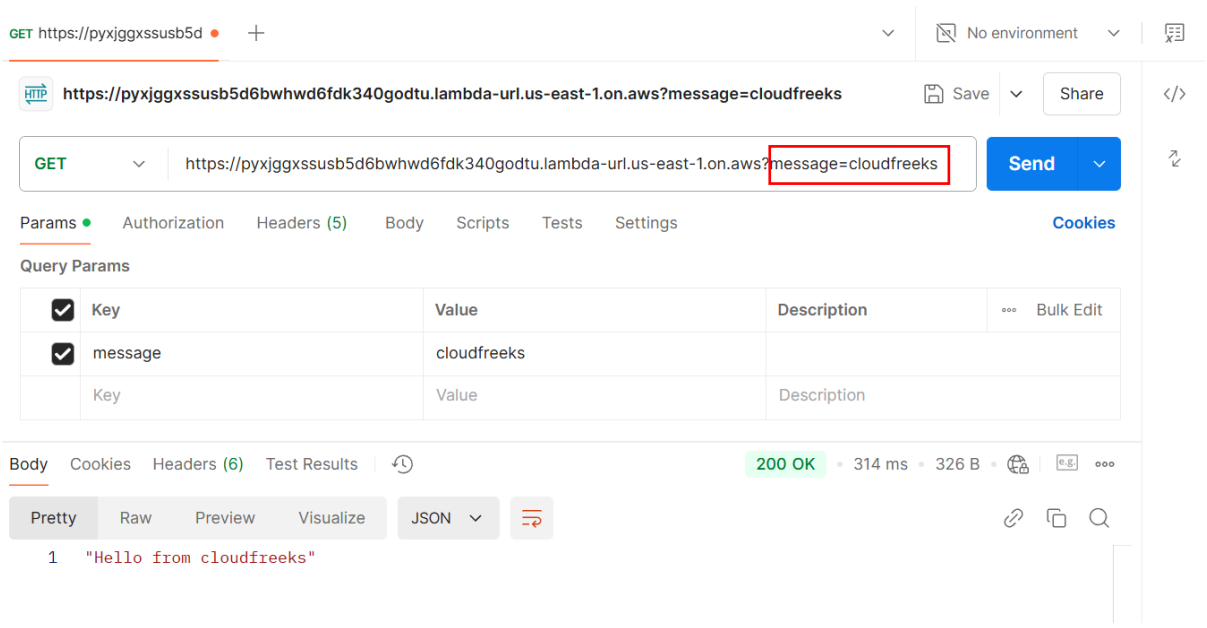
13. Now we are going to change the function code and then invoke our function. So, use the code given below and deploy it in your lambda function.

```
export const handler = async (event) => {
  console.log('Received event:', JSON.stringify(event, null, 2));

  const method = event.requestContext.http.method;
  const queryParam = event.queryStringParameters.message;
  console.log(`Received ${method} request with ${queryParam}`);

  const response = {
    statusCode: 200,
    body: JSON.stringify(`Hello from ${queryParam}`),
  };
  return response;
};
```

index.mjs

```
exports.handler = async (event) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    // fetch method and querystring
    const method = event.requestContext.http.method;
    const queryParam = event.queryStringParameters.message;
    console.log(`Received ${method} request with ${queryParam}`);

    const response = {
        statusCode: 200,
        body: JSON.stringify(`Hello from ${queryParam}`),
    };
    return response;
}
```

14. Now you need to open Postman on your laptop and copy the function URL from your lambda function. Then you need to create a new workspace in your Postman and choose the GET method. After that paste your URL and append it with a message as you can see in the snapshot. You will get a Hello message which means that our function is working fine.



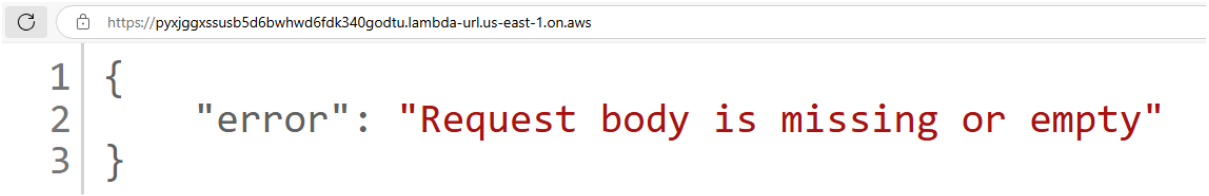
15. Go to the CloudWatch of your lambda function and view your events.
16. Now we are going to design the lambda function code. This will be our final code for the calculator microservice. Use the below code and deploy it on your lambda function.

export const handler = async (*event*) => {
console.log('Received event:', JSON.stringify(event, null, 2));

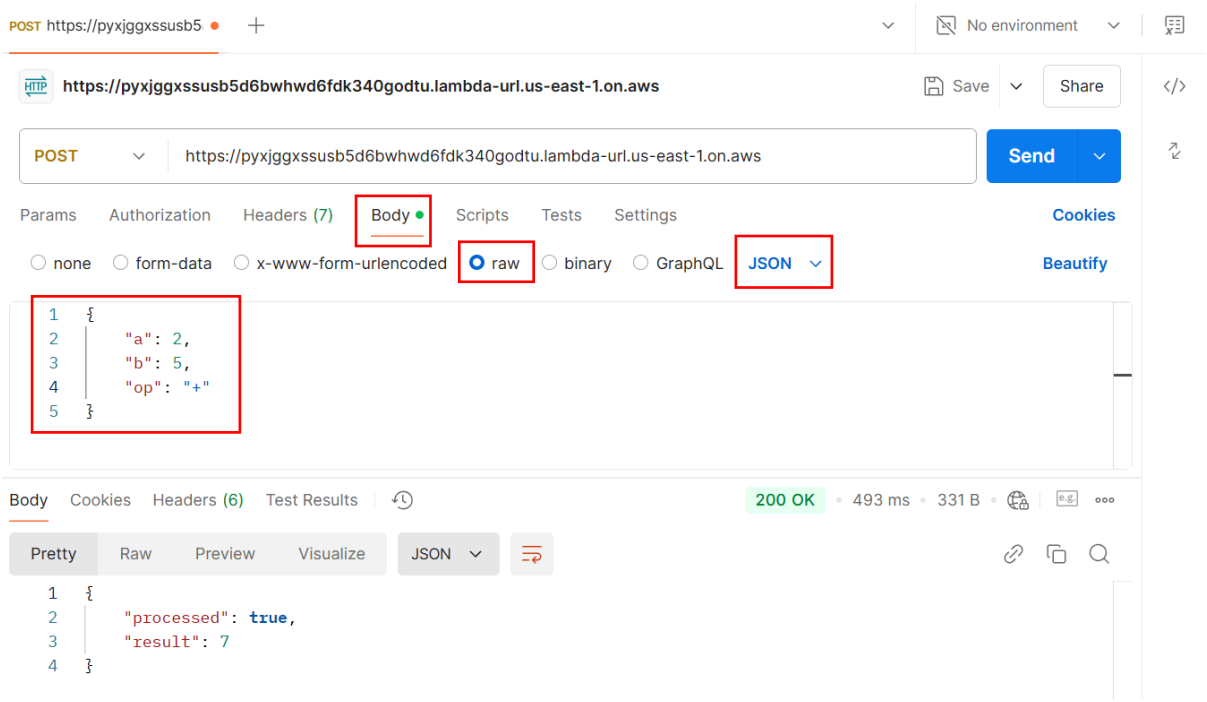
```
    // Check if event.body exists and is not empty before parsing
    if (!event.body) {
        return {
            statusCode: 400,
            body: JSON.stringify({
                error: "Request body is missing or empty"
            })
        };
    }

    let payload;
    try {
        // Try parsing the body
        payload = JSON.parse(event.body);
    } catch (error) {
        // If parsing fails, return an error response
        return {
            statusCode: 400,
            body: JSON.stringify({
                error: "Invalid JSON format in request body",
                details: error.message
            })
        };
    }

    let result = 0;
    try {
         if (payload.a === undefined || payload.b === undefined || payload.op === undefined) {
                throw new Error(`event should have properties a, b, and op: "${JSON.stringify(payload)}"`);
        }

        switch (payload.op) {
            case "+":
                result = payload.a + payload.b;
                break;
            case "-":
                result = payload.a - payload.b;
                break;
            case "*":
                result = payload.a * payload.b;
                break;
            case "/":
                result = payload.b === 0 ? NaN : payload.a / payload.b;
```

```
            break;
        default:
            throw new Error(`Unsupported operation: ${payload.op}`);
    }
    console.log('Result is : ', result);
} catch (error) {
    console.error(error);
    return {
        statusCode: 400,
        body: JSON.stringify({
            error: "Error processing the request",
            details: error.message
        }),
    };
}

return {
    statusCode: 200,
    body: JSON.stringify({
        processed: true,
        result: result
    }),
};
};
```

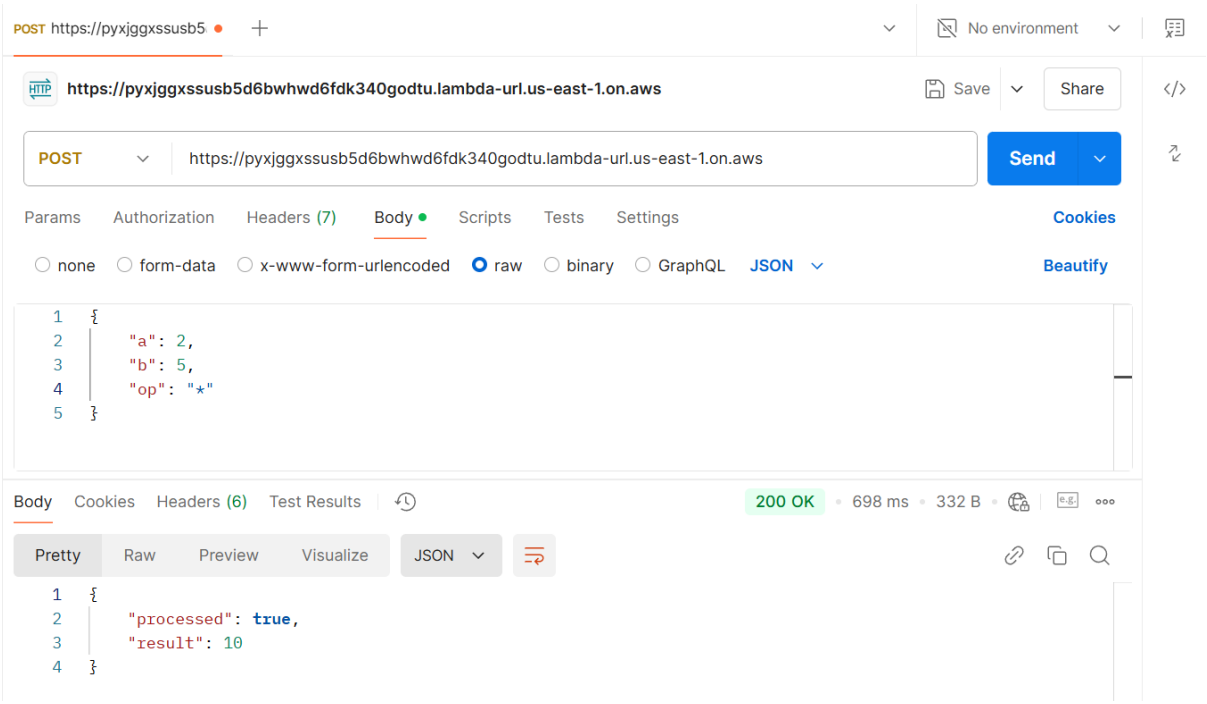
17. Once the code is deployed open the function URL once more. You should get this type of error shown below.



18. Now go to Postman and use the POST method there you need to paste the function URL and then choose **body and from body choose raw then JSON.**
19. In the end put the same JSON code as shown below and click on Send to invoke the request.
20. You will get the same output, based on the operator you choose your calculator will work.



21. As you can see below, we changed the operator from addition to multiplication and based on our operator we get the output.



22. Once you are done using the calculator delete all the resources. Start with the lambda function and then your CloudWatch logs.