Build AWS Lambda Integration with AWS SAM and OpenAPI

You can integrate an API method with a Lambda function using **Lambda proxy integration** or **Lambda non-proxy (custom) integration**.

Use AWS SAM and OpenAPI to create an API Gateway REST API with Lambda proxy and custom integration

1. Using AWS Cloud9 console, return to the root folder `module-3/lambda`
2. This code belongs in your SAM template file `template.yaml`

   Review the code and then **copy/paste** it into the `template.yaml` file.

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: >
  module3-lambda-rest-api: Sample SAM Template for module3-lambda-rest-api

Globals:

    # Enable Logs
    Api:
        MethodSettings:
            - ResourcePath: "/*"
              HttpMethod: "*"
              DataTraceEnabled: True
              LoggingLevel: INFO
              MetricsEnabled: True
    Function:
        Timeout: 3
        Runtime: nodejs18.x

Resources:

    # Example API Gateway
    LambdaIntegrationExampleAPI:
        Type: AWS::Serverless::Api
        Properties:
            StageName: dev
            OpenApiVersion: 3.0.3
            DefinitionBody: # an OpenApi definition
                "Fn::Transform":
                    Name: "AWS::Include"
                    Parameters:
                        Location: "openapi.yaml"
            EndpointConfiguration:
                Type: REGIONAL

    #Lambda Function with Proxy Integration for single method GET
    LambdaProxySingleMethodFunction:
        Type: 'AWS::Serverless::Function'
        Properties:
            CodeUri: ./handlers
            Handler: proxy-example.handler

    #Lambda Function with Proxy Integration for any method and path ANY /{proxy+}
    LambdaProxyAnyMethodFunction:
        Type: 'AWS::Serverless::Function'
        Properties:
            CodeUri: ./handlers
            Handler: proxy-example.handler

    #Lambda Function with Custom Integration for any method and path ANY /{proxy+}
    LambdaCustomFunction:
        Type: 'AWS::Serverless::Function'
        Properties:
            CodeUri: ./handlers
            Handler: proxy-example.handler

    # Execution Role for lambda functions
    LambdaExecutionRole:
        Type: AWS::IAM::Role
        Properties:
            AssumeRolePolicyDocument:
                Version: "2012-10-17"
                Statement:
                  - Effect: Allow
                    Principal:
                      Service:
                        - apigateway.amazonaws.com
                    Action:
                      - 'sts:AssumeRole'
            Policies:
                - PolicyName: AllowLambdaExec
                  PolicyDocument:
                    Version: "2012-10-17"
                    Statement:
                      - Effect: Allow
                        Action: 'lambda:InvokeFunction'
                        Resource: [
                            !GetAtt LambdaProxySingleMethodFunction.Arn,
                            !GetAtt LambdaProxyAnyMethodFunction.Arn,
                            !GetAtt LambdaCustomFunction.Arn
                        ]

Outputs:

    Endpoint:
        Description: API Gateway Endpoint
        Value:
            Fn::Sub: https://${LambdaIntegrationExampleAPI}.execute-api.${AWS::Region}.amazonaws.com/dev/resource
```

Resources in your API define one or more methods, such as GET or POST. Methods have an integration that routes requests to a Lambda function or another integration type. You can define each resource and **method individually**, or use special resource and method types to **match all requests** that fit a pattern. A **proxy** resource catches all paths beneath a resource. The **ANY** method catches all HTTP methods.

> The Path property set to `resource/{proxy+}` is to allow the function to handle any path under the resource URL. In this Lambda proxy integration through the `ANY` method, a single backend Lambda function serves as the event handler for `ANY` requests.

3. This code belongs in your OpenAPI definition file `openapi.yaml`

   Review the code and then **copy/paste** it into the `openapi.yaml` file

```yaml
1   openapi: "3.0.1"
2   info:
3     title: "module-3-lambda-integration"
4     description: "API Gateway example for AWS Lambda proxy Integration"
5     version: "1.0"
6
7   paths:
8
9     /resource:
10        get:
11            x-amazon-apigateway-integration:
12                httpMethod: "POST"
13                credentials:
14                    Fn::GetAtt: [LambdaExecutionRole, Arn]
15                uri:
16                  Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${LambdaProxySingleMethodFunction.Arn}/invocations"
17                passthroughBehavior: "when_no_match"
18                type: "aws_proxy"
19        put:
20          consumes:
21          - "application/json"
22          produces:
23          - "application/json"
24          responses:
25            "200":
26              description: "200 response"
27          x-amazon-apigateway-integration:
28            httpMethod: "POST"
29            credentials:
30                    Fn::GetAtt: [LambdaExecutionRole, Arn]
31            uri:
32               Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${LambdaCustomFunction.Arn}/invocations"
33            responses:
34              default:
35                statusCode: "200"
36                responseTemplates:
37                  application/json: "#set($inputRoot = $input.path('$'))\n{\n    \"status\"\
38                    : $inputRoot.statusCode,\n    \"message\": \"$inputRoot.body\",\n\
39                    \    \"message_details\" \"This message was inserted from the response\
40                    \ transformation.\"\n}\n"
41            requestTemplates:
42              application/json: "#set($body = {\n  \"name\": $input.json('$.name'),\n\
43                \  \"age\": $input.json('$.age'),\n  \"email\": $input.json('$.email'),\n\
44                \  \"id\": \"$input.params('id')\"\n})\n#set($jsonBody = $util.escapeJavaScript($body.toString()))\n\
45                {\n  \"body\": \"$jsonBody\"\n}\n"
46            passthroughBehavior: "when_no_templates"
47            contentHandling: "CONVERT_TO_TEXT"
48            type: "aws"
49
50    /resource/{proxy+}:
51        x-amazon-apigateway-any-method:
52            parameters:
53            - name: "proxy"
54              in: "path"
55              required: true
56              schema:
57                type: "string"
58            x-amazon-apigateway-integration:
59                httpMethod: "POST"
60                credentials:
61                    Fn::GetAtt: [LambdaExecutionRole, Arn]
62                type: "aws_proxy"
63                uri:
64                    Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${LambdaProxyAnyMethodFunction.Arn}/invocations"
65                passthroughBehavior: "when_no_match"
```

The x-amazon-apigateway-any-method object **(line 46)** Specifies the OpenAPI Operation Object for the API Gateway catch-all `ANY` method.

The x-amazon-apigateway-integration object **(lines 11, 25 and 53)** specifies details of the backend integration used for this method.

> **Important**
> For Lambda integrations, you must use the HTTP method of **POST** for the integration request, according to the specification of the Lambda service action for function invocations. In **x-amazon-apigateway-integration** all methods were defined with `httpMethod: "POST"`. The HTTP method of exposure to the client can be any one, such as GET, but for integration with AWS Lambda it must be POST.

On **lines 16** and **55** we have `type: "aws_proxy"`, responsible for configuring the integration as **Lambda Proxy** type.

On **line 43** we have `type: "aws"`, responsible for configuring the integration as **Lambda Custom** type.

With custom Lambda integration it is possible to have more control in the request and response workflow. In this example we are transforming the input payload before passing it to Lambda and transforming the payload returned by Lambda before returning it to the client using Apache Velocity Template Language (VTL).

- In **Line 36** we have the request transformation template definition.
- In **line 31** we have the response transformation template definition.

We will see more details of this transformation in the test section.

Deploy the project

1. To deploy the Amazon API Gateway and the AWS Lambda to your AWS account, run the following commands from the application root `module-3/lambda`, where the `template.yaml` file for the sample application is located:

```
sam build && sam deploy --guided
```

   The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you needed to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure as it is above.

   - **Stack Name**: `module-3-lambda-integration`
   - **AWS Region**: Put the chosen region to run the workshop. **e.g.** `us-east-1`
   - **Confirm changes before deploy**: `Y`
   - **Allow SAM CLI IAM role creation**: `Y`
   - **Disable rollback**: `N`
   - **LambdaProxySingleMethodFunction may not have authorization defined, Is this okay?**: `Y`
   - **LambdaProxyAnyMethodFunction may not have authorization defined, Is this okay?** `Y`
   - **LambdaCustomFunction may not have authorization defined, Is this okay?** `Y`
   - **Save arguments to configuration file**: `Y`
   - **SAM configuration file** and **SAM configuration environment** leave blank

```
Setting default arguments for 'sam deploy'
=========================================
Stack Name [module-3-lambda-proxy-integration]: module-3-lambda-integration
AWS Region [us-east-1]:
#Shows you resources changes to be deployed and require a 'Y' to initiate deploy
Confirm changes before deploy [Y/n]: Y
#SAM needs permission to be able to create roles to connect to the resources in your template
Allow SAM CLI IAM role creation [Y/n]: Y
#Preserves the state of previously provisioned resources when an operation fails
Disable rollback [Y/n]: n
LambdaProxySingleMethodFunction may not have authorization defined, Is this okay? [y/N]: y
LambdaProxyAnyMethodFunction may not have authorization defined, Is this okay? [y/N]: y
Save arguments to configuration file [Y/n]: y
SAM configuration file [samconfig.toml]:
SAM configuration environment [default]:
```

2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.

```
CloudFormation stack changeset
-----------------------------------------------------------------------------------------------------------------
Operation                 LogicalResourceId                               ResourceType                      Replacement
-----------------------------------------------------------------------------------------------------------------
+ Add                     LambdaIntegrationExampleAPIDeployment6f599cc7   AWS::ApiGateway::Deployment       N/A
                          52
+ Add                     LambdaIntegrationExampleAPIdevStage             AWS::ApiGateway::Stage            N/A
+ Add                     LambdaIntegrationExampleAPI                     AWS::ApiGateway::RestApi          N/A
+ Add                     LambdaProxyAnyMethodFunctionApiEventsPermissi   AWS::Lambda::Permission           N/A
                          ondev
+ Add                     LambdaProxyAnyMethodFunctionRole                AWS::IAM::Role                    N/A
+ Add                     LambdaProxyAnyMethodFunction                    AWS::Lambda::Function             N/A
+ Add                     LambdaProxySingleMethodFunctionApiEventsPermi   AWS::Lambda::Permission           N/A
                          ssiondev
+ Add                     LambdaProxySingleMethodFunctionRole             AWS::IAM::Role                    N/A
+ Add                     LambdaProxySingleMethodFunction                 AWS::Lambda::Function             N/A
-----------------------------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-east-1:[redacted]:changeSet/samcli-deploy1678911432/97e78a73-f8ca-4c70-874c-659aea1c32f4


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: Y
```

3. After it was finished, a new stack `module-3-lambda-integration` will be successfully created. Now let's explore the resources created in the console and test the API.

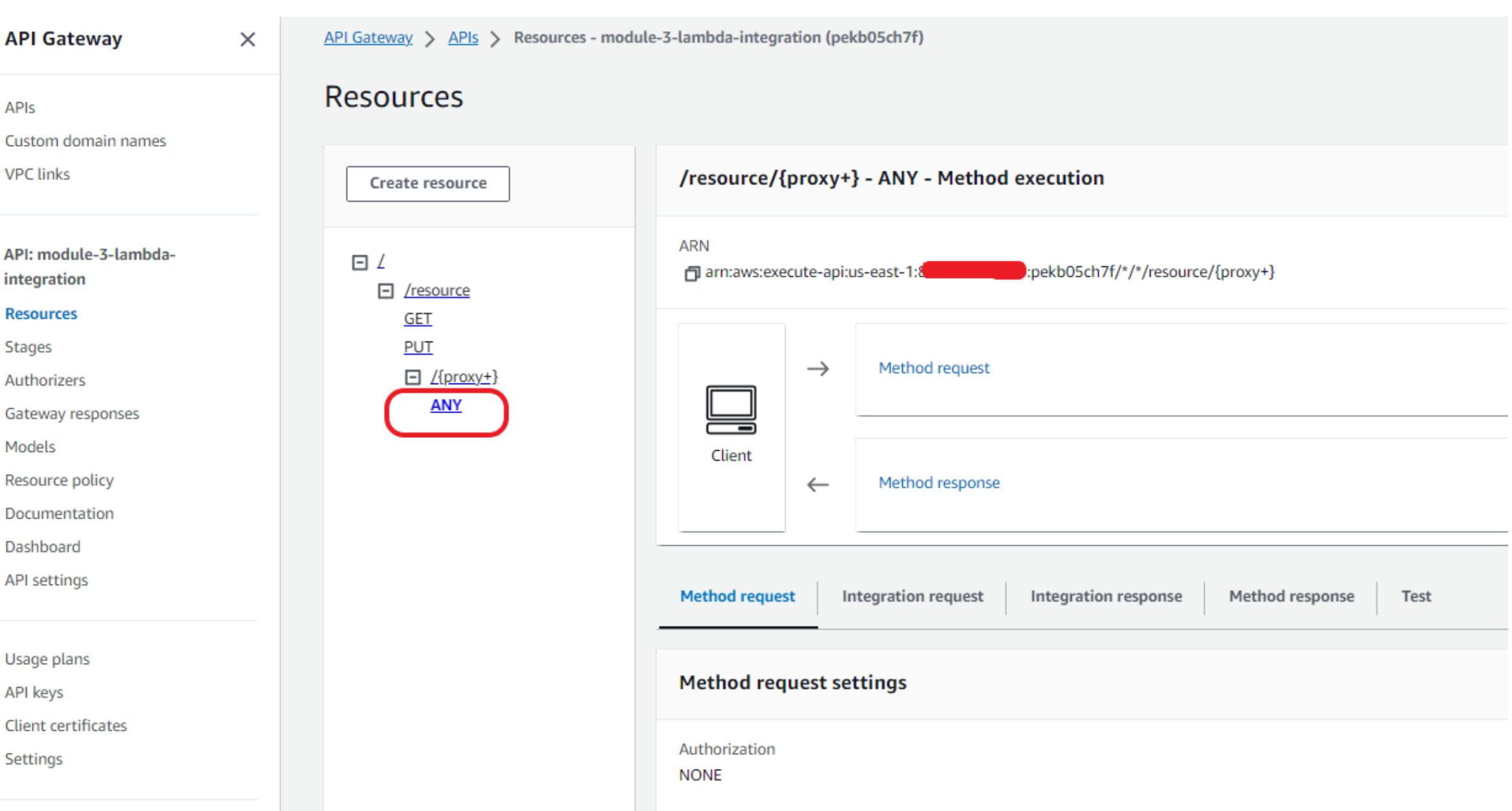
Test Integration

After you create your API Gateway REST API with Lambda integration, you can test the API.

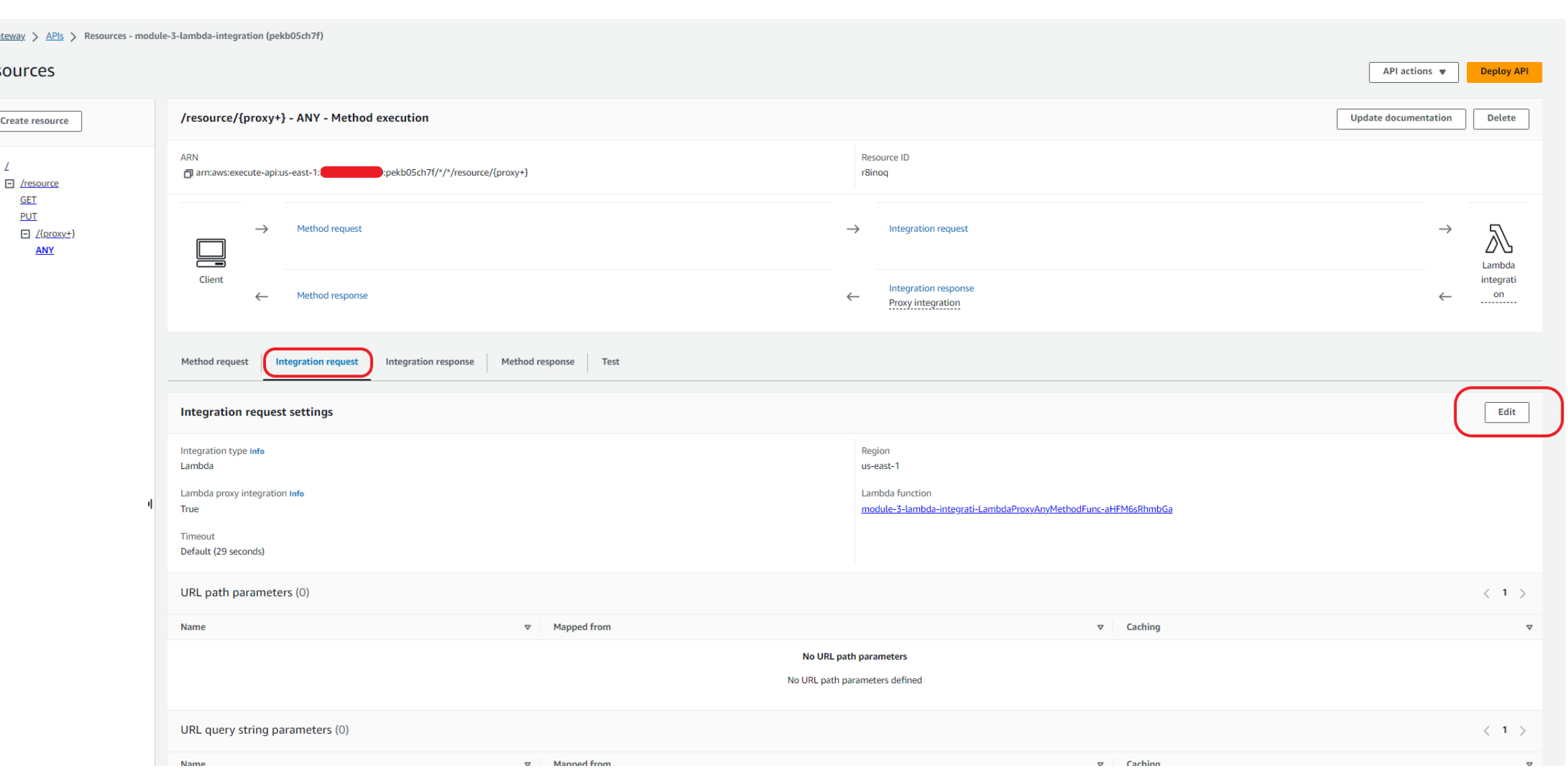
Test Lambda Proxy Integration using API Gateway console

First let's test the **proxy** type integration.

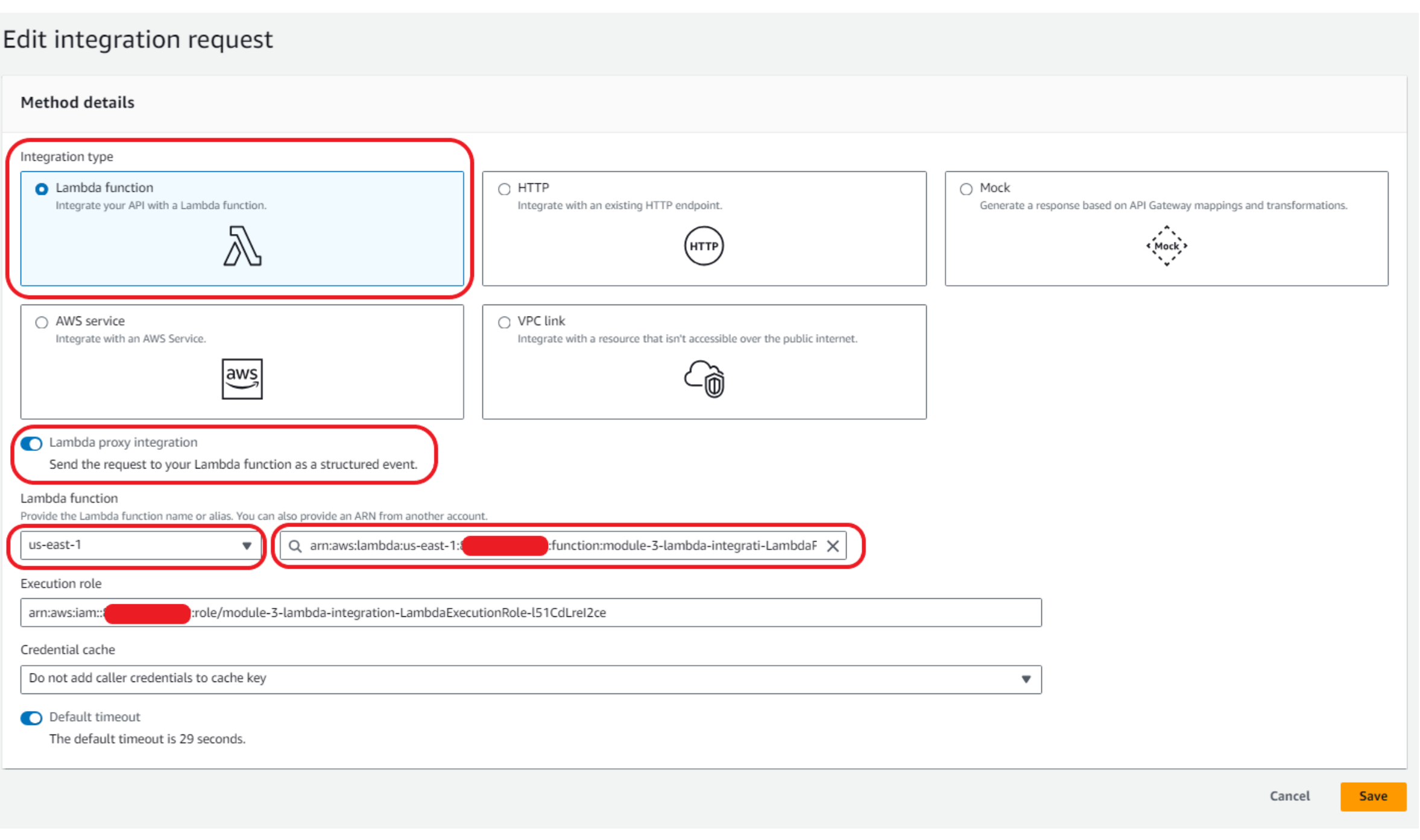
1. Open the Amazon API Gateway console and sign in.
2. Choose your REST API named, `module-3-lambda-integration`.
3. In the **Resources** pane, you can select the method you want to test. Click on the `ANY` method.



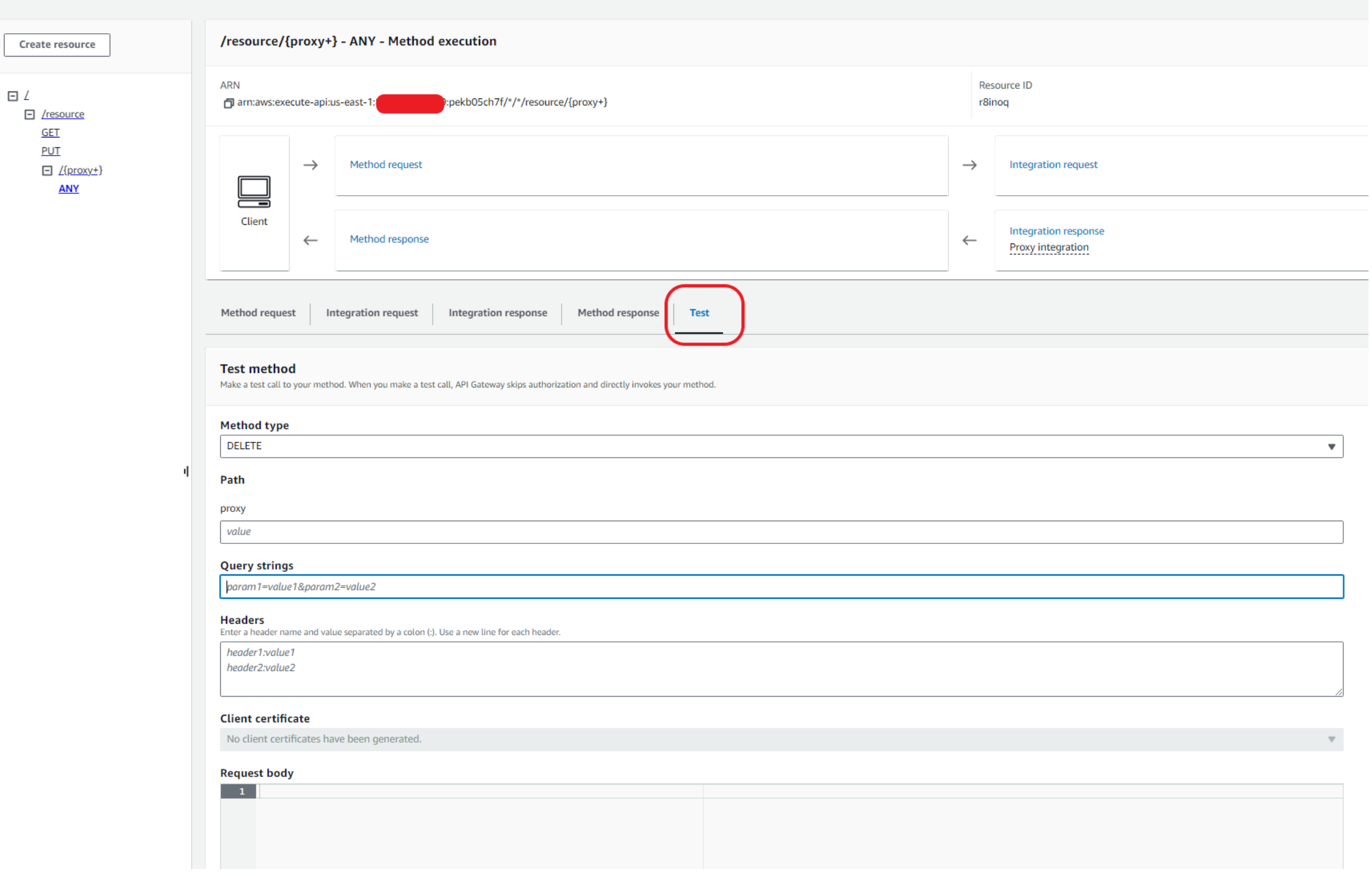
4. Click on **Integration request** tab, and then **Edit**.



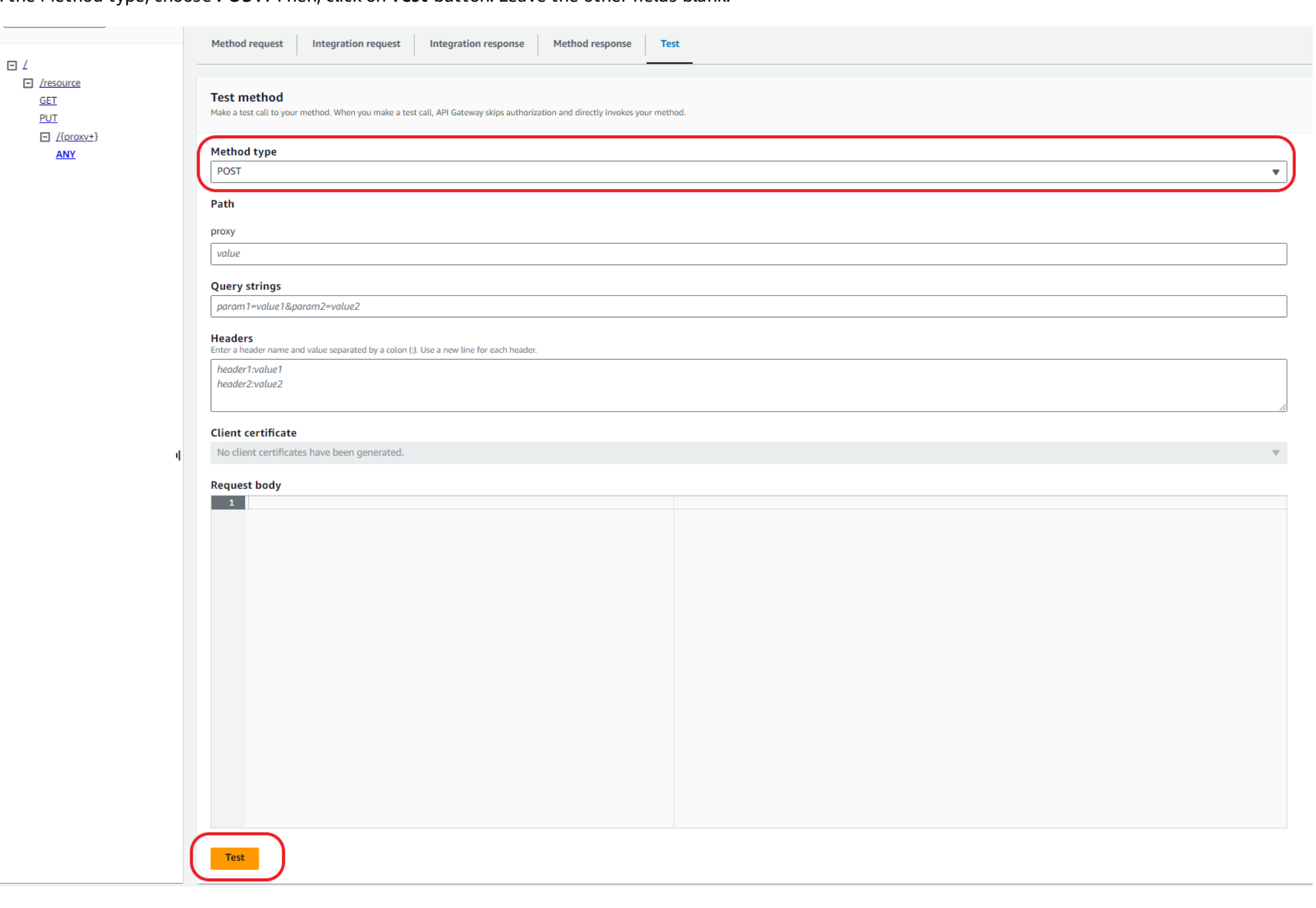
5. Review the configuration details for integration.



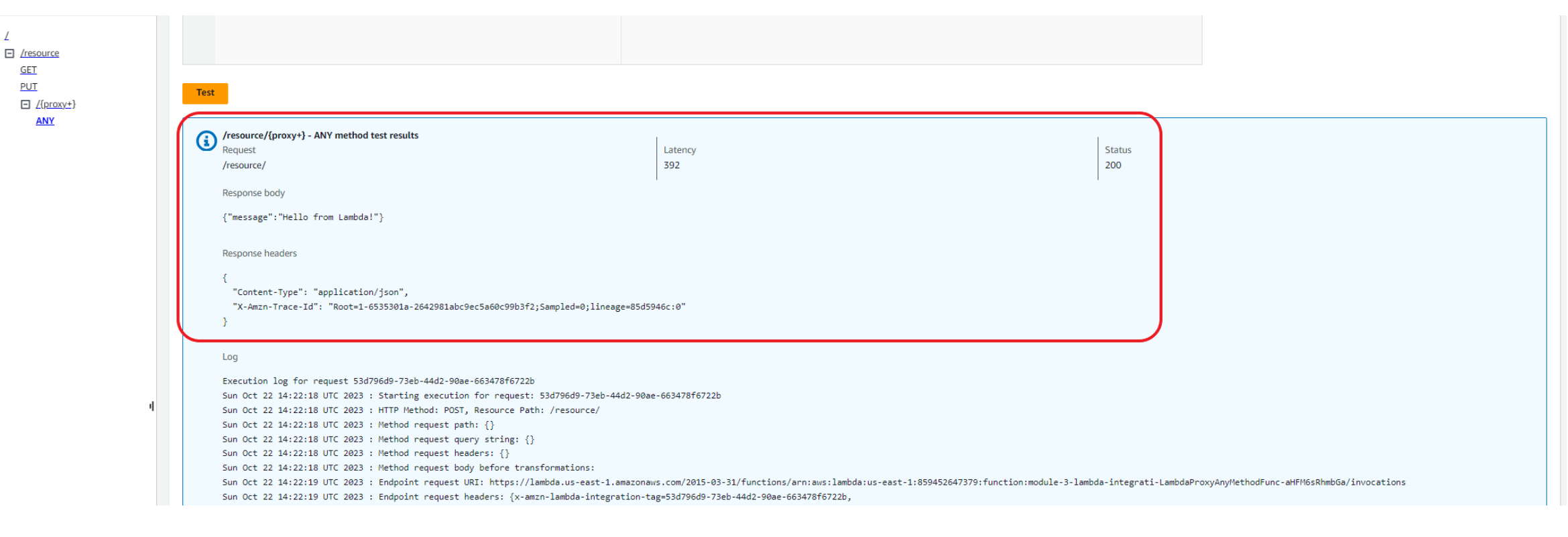
6. Click **Cancel**, Go back to the **Method execution** pane, choose the **Test** tab.



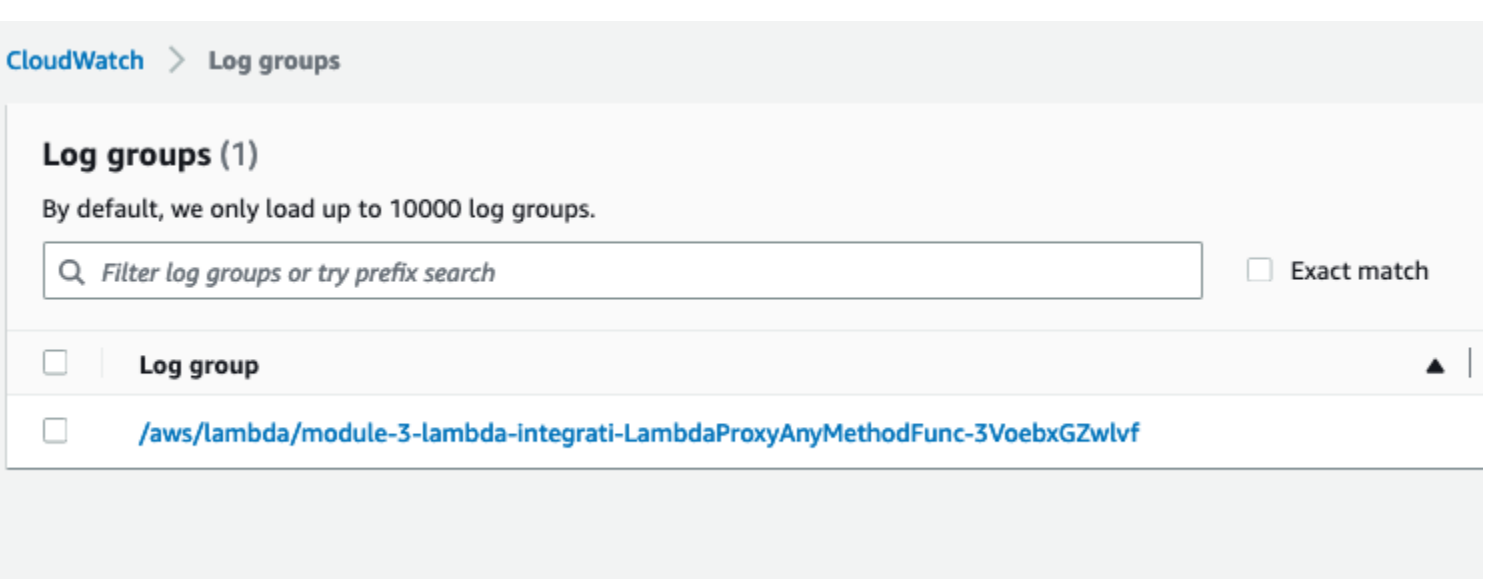
7. In the Method type, choose **POST**. Then, click on **Test** button. Leave the other fields blank.



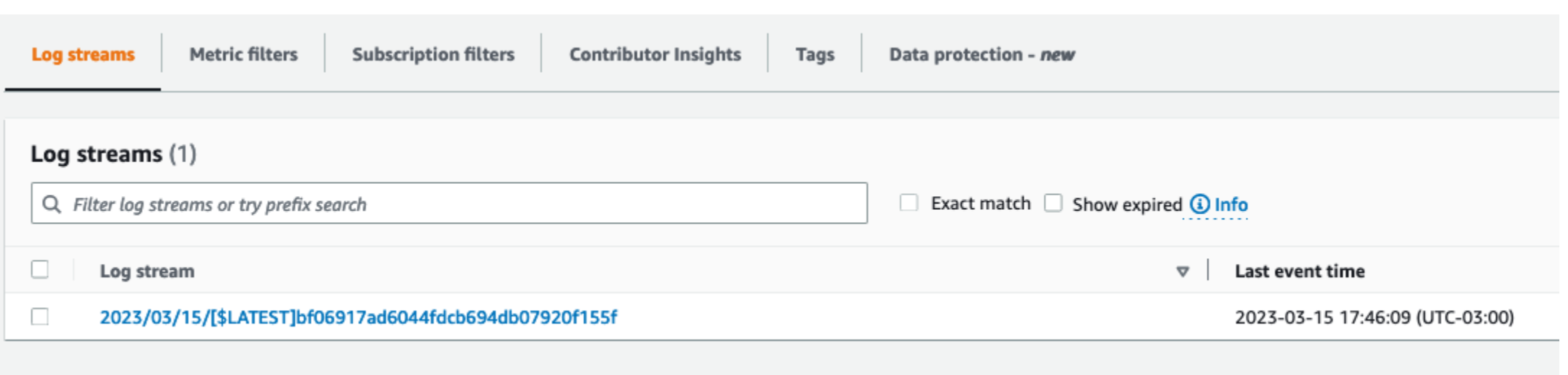
8. The test result are showed in the log. You can see the function called and the response:



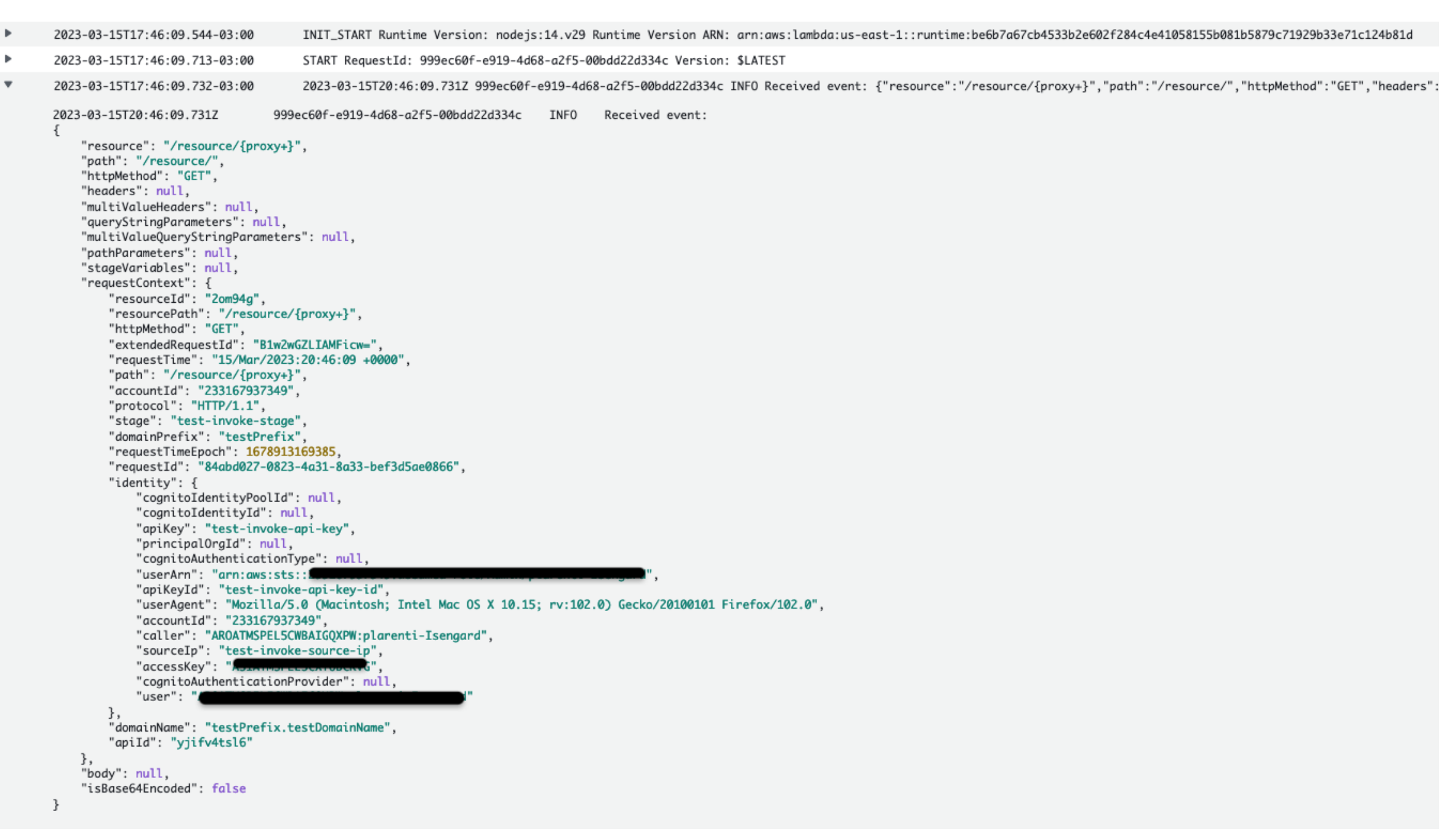
9. Now let's check an important detail of the integration. Let's understand **input and response event** format, that Lambda receives from API Gateway when the integration is of the proxy type.
10. Navigate to CloudWatch in the AWS console.
11. Click on **Log Groups**, and then click on a lambda log group with the prefix `module-3-lambda-integrati...`



12. Click on the most recent logstream



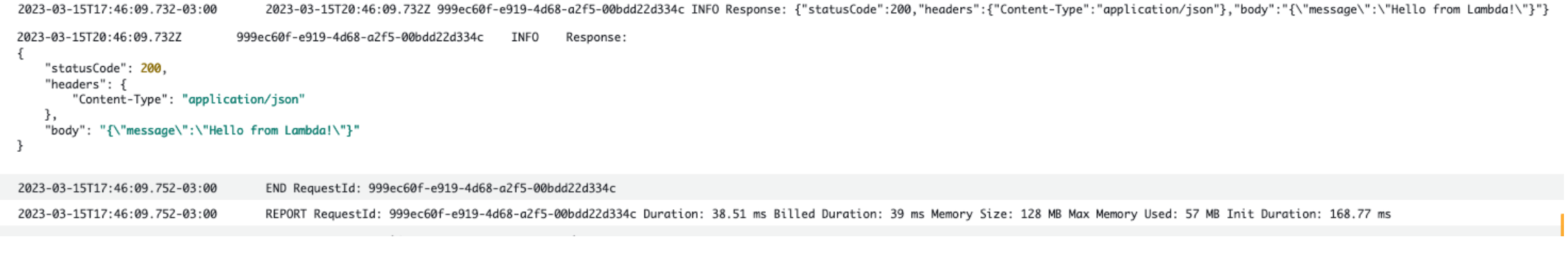
13. Look for **INFO Received** event and take a look at the input event payload:



- Example event.json API Gateway proxy **input event** (REST API)

```
{
   "resource":"/",
   "path":"/",
   "httpMethod":"GET",
   "requestContext":{
      "resourcePath":"/",
      "httpMethod":"GET",
      "path":"/Prod/",
      "..."
   },
   "headers":{
      "accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
      "accept-encoding":"gzip, deflate, br",
      "Host":"70ixmpl4fl.execute-api.us-east-2.amazonaws.com",
      "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36",
      "X-Amzn-Trace-Id":"Root=1-5e66d96f-7491f09xmpl79d18acf3d050",
      "..."
   },
   "multiValueHeaders":{
      "accept":[
         "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
      ],"accept-encoding":[
         "gzip, deflate, br"
      ],
      "..."
   },
   "queryStringParameters":null,
   "multiValueQueryStringParameters":null,
   "pathParameters":null,
   "stageVariables":null,
   "body":null,
   "isBase64Encoded":false
}
```

14. Now Look for **INFO Response** event and take a look at the Response event payload:

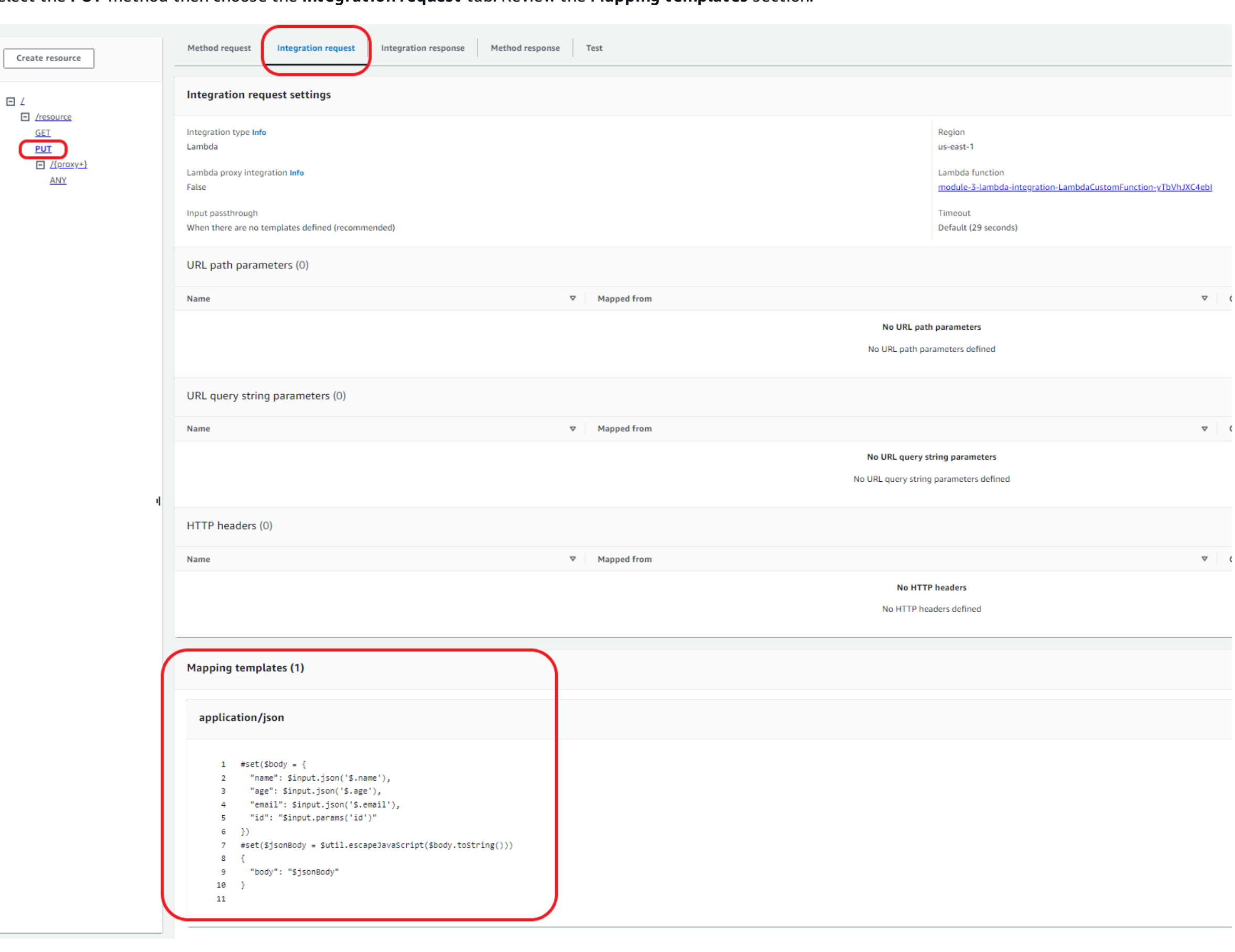


- Example Proxy integration **response object**

```
var response = {
   "statusCode": 200,
   "headers": {
   "Content-Type": "application/json"
},
"isBase64Encoded": false,
"multiValueHeaders": {
   "X-Custom-Header": ["My value", "My other value"],
},
"body": "{\n  \"TotalCodeSize\": 104330022,\n  \"FunctionCount\": 26\n}"
}
```

Test Lambda Custom Integration using API Gateway console

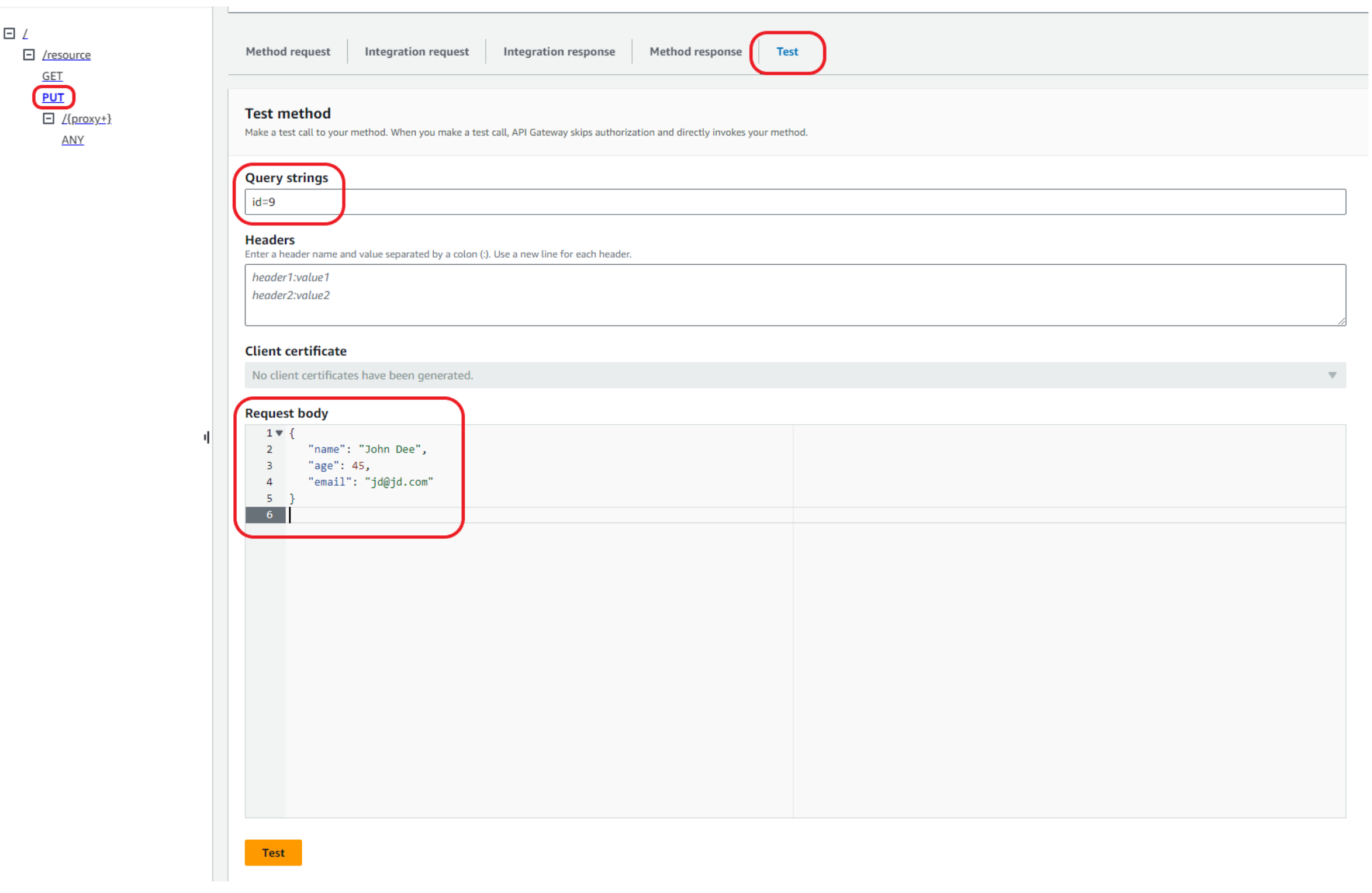
1. Select the **PUT** method then choose the **Integration request** tab. Review the **Mapping templates** section.



This template converts the payload **before sending it to Lambda**. We add a parameter received via **query string to the JSON body**.

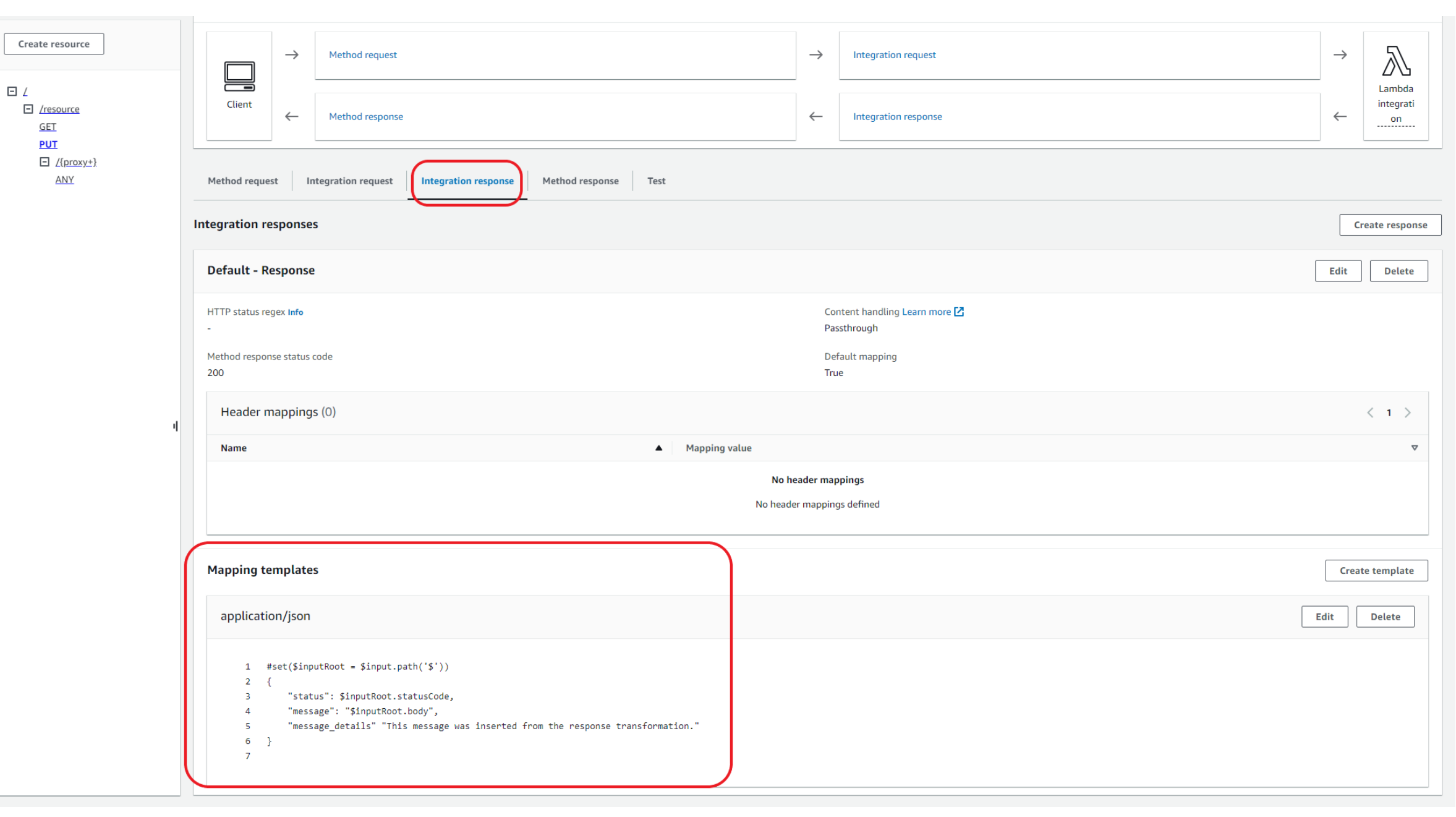
2. Go back to **Method Execution** and **click Test**. Fill in the fields with the values below then **click** on **Test** button.
   - For **Query Strings**: `id=9`
   - For **Request Body**:

```
{
   "name": "John Dee",
   "age": 45,
   "email": "jd@jd.com"
}
```



3. Click **Test** and check the execution result:

4. Go back and choose the **Integration response** tab.
5. Review the **Mapping templates**. See that a transformation is being performed on the response payload, inserting a static message: *This message was inserted from the response transformation.*



Test the deployed API using cURL

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the following cURL command and paste it into the terminal window, replacing `<api-id>` with your API's API ID and `<region>` with the region where your API is deployed. You may have copied this URL in from the CloudFormation output in the last step. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**.

```
curl -X GET \
 'https://<api-id>.execute-api.<region>.amazonaws.com/dev/resource/{proxy+}'
```

3. The **Response Body** output should be:

```
"Hello from Lambda!"
```

4. Do the same to test the **custom integration** with the PUT method

```
curl -X PUT https://<api-id>.execute-api.<region>.amazonaws.com/dev/resource \
 -H 'Content-Type: application/json' \
 -d '{"name": "John Dee", "age": 45,"email": "jd@jd.com"}'
```

5. The **Response Body** output should be

```
{
    "status": 200,
    "message": "{"message":"Hello from Lambda!"}",
    "message_details" "This message was inserted from the response transformation."
}
```

Tip
When testing a method using the API Gateway console, logs are not sent to CloudWatch Logs. When calling externally, for example via cURL, the logs are sent normally.

Challenge
Find the payload logs **before** and **after** the **request** and **response** transformations in **CloudWatch**.