The Amazon API Gateway Workshop

Introduction
- Getting Started
- Module 1 - Introduction to Amazon API Gateway
- Module 2 - Deploy your first API with IaC
- Module 3 - API Gateway REST Integrations
  - Module Goals
  - Mock
  - HTTP Integration
    - Set up your AWS SAM Project
    - Build HTTP Integration with AWS SAM and OpenAPI
    - Test Integration
  - AWS Lambda
  - AWS Step Functions
  - Amazon SQS
  - Amazon SNS
  - Amazon Kinesis
  - Amazon DynamoDB
  - Amazon EventBridge
  - Private Integration
  - Amazon S3
  - Clean up
- Module 4 - Observability in API Gateway
- Module 5 - WebSocket APIs
- Module 6 - Enable fine-grained access control for your APIs
- Clean up
- Resources

The Amazon API Gateway Workshop > Module 3 - API Gateway REST Integrations > HTTP Integration > Build HTTP Integration with AWS SAM and OpenAPI

Build HTTP Integration with AWS SAM and OpenAPI

You can integrate an API method with a HTTP endpoint using the **HTTP proxy integration** or the **HTTP custom integration**.

Use AWS SAM and OpenAPI to create an API Gateway REST API with HTTP proxy and Custom integration

1. Using AWS Cloud9 console, return to the root folder `module-3/http`
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-http-rest-api: Sample SAM Template for module3-http-rest-api

Globals:

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

Resources:

    # HTTP Integration Example API
    HttpIntegrationExampleAPI:
        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

    # Target API to emulate an exposed HTTP endpoint
    TargetApi:
        Type: AWS::Serverless::Api
        Properties:
            Name: TargetApi
            StageName: dev

    # Function as Back-End to Target API
    ItemsFunction:
        Type: AWS::Serverless::Function
        Properties:
            CodeUri: ./handlers
            Handler: item.handler
            Policies:
                - DynamoDBCrudPolicy:
                    TableName: !Ref SampleTable
            Environment:
                Variables:
                    SAMPLE_TABLE: !Ref SampleTable
                    REGION: !Sub "${AWS::Region}"
            Events:
                GetItems:
                    Type: Api
                    Properties:
                        Path: /items
                        Method: GET
                        RestApiId: !Ref TargetApi
                PostItem:
                    Type: Api
                    Properties:
                        Path: /items
                        Method: POST
                        RestApiId: !Ref TargetApi

    SampleTable:
        Type: AWS::Serverless::SimpleTable
        Properties:
            PrimaryKey:
                Name: id
                Type: String
            ProvisionedThroughput:
                ReadCapacityUnits: 2
                WriteCapacityUnits: 2

Outputs:

    TargetApiEndpoint:
        Description: Target API endpoint URL for Dev stage
        Value: !Sub "https://${TargetApi}.execute-api.${AWS::Region}.amazonaws.com/${TargetApi.Stage}/"

    RestApiEndpoint:
        Description: Rest API endpoint URL for Dev stage (Http Integration Example API)
        Value: !Sub "https://${HttpIntegrationExampleAPI}.execute-api.${AWS::Region}.amazonaws.com/${HttpIntegrationExampleAPI.Stage}/"
```

The `HttpIntegrationExampleAPI` resource is our **Rest API** that will integrate with a **HTTP endpoint** via Proxy and Custom.

Resource `ItemsFunction` defines a Lambda that **simulates a Back End application** that lists and saves items in the database.

Resource `TargetApi` is a Rest API to **simulate a HTTP endpoint** of an application.

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-http-integration"
4       description: "API Gateway example for HTTP Integration"
5       version: "1.0"
6
7   paths:
8
9       /{proxy+}:
10          x-amazon-apigateway-any-method:
11              parameters:
12              - name: "proxy"
13                in: "path"
14                required: true
15                schema:
16                  type: "string"
17              x-amazon-apigateway-integration:
18                  requestParameters:
19                      integration.request.path.proxy: method.request.path.proxy
20                  httpMethod: "ANY"
21                  type: "http_proxy"
22                  uri:
23                      Fn::Sub: "https://${TargetApi}.execute-api.${AWS::Region}.amazonaws.com/${TargetApi.Stage}/{proxy}"
24                  passthroughBehavior: "when_no_match"
25
26      /store:
27          get:
28              produces:
29              - "application/json"
30              responses:
31                  "200":
32                      description: "200 OK"
33                  "404":
34                      description: "404 No Content Found"
35              x-amazon-apigateway-integration:
36                  httpMethod: "GET"
37                  uri:
38                      Fn::Sub: "https://${TargetApi}.execute-api.${AWS::Region}.amazonaws.com/${TargetApi.Stage}/items"
39                  responses:
40                      "404":
41                          statusCode: "404"
42                          responseTemplates:
43                              application/json: "#set($inputRoot = $input.path('$'))\n{\n \"message_store\"\
44                                : \"There is no Items\" \n}"
45                      default:
46                          statusCode: "200"
47                          responseTemplates:
48                              application/json: "#set($inputRoot = $input.path('$'))\n{\n \"items\"\
49                                : $input.json('$'),\n \"message_store\": \"There is something around\
50                                \ the corner\"\n}"
51                  passthroughBehavior: "when_no_match"
52                  type: "http"
```

In **HTTP proxy integration**, the `/{proxy+}` resource, API Gateway passes the **client-submitted method request to the backend**.

The request data that is **passed through includes the request headers, query string parameters, URL path variables, and payload**.

The **backend HTTP endpoint** (`TargetAPI`) or the web server parses the incoming request data to determine the response that it returns.

> **Tip**
> With the all-encompassing proxy resource `{proxy+}`, and the catch-all `ANY` verb for the HTTP method, you can use an **HTTP proxy integration** to create an API of a **single API method**. The method exposes the entire set of **publicly accessible HTTP resources** from a HTTP Endpoint which can be either an **API** or a **Website**. For example:
>
> - In this API the proxy resource path of `{proxy+}` becomes the placeholder of the backend endpoints under **e.g** `https://targetapidemo.execute-api.us-east-1.amazonaws.com/dev/`. It can be `items`, `items/sports`, and `items/sports/{itemId}`. The ANY method serves as a placeholder for any of the supported HTTP verbs at run time.

The x-amazon-apigateway-any-method object **(line 10)** Specifies the OpenAPI Operation Object for the API Gateway catch-all `ANY` method.

The x-amazon-apigateway-integration object **(lines 17 and 35)** specifies details of the backend integration used for this method.

On **line 19** we have type: `"http_proxy"`, responsible for configuring the integration as **HTTP Proxy type**.

> **Note**
> HTTP proxy integration supports multi-valued headers and query strings. HTTP proxy integration makes the client and backend interact directly with no intervention from API Gateway after the API method is set up, except for known issues such as unsupported characters, which are listed in Amazon API Gateway important notes.

In **HTTP custom integration**, the `/store` resource, you need to specify how the incoming request data is mapped **to the integration request** and how the resulting integration response data is mapped to the **method response**.

On **line 52** we have type: `"http"`, responsible for configuring the integration as **HTTP Custom type**.

Note that lines "responses" **30** and **39** respectively define what are the **HTTP codes** of the method and how the response is **transformed** using Apache Velocity Template Language (VTL)

> The uri line **38** is pointing to **/items** and no /proxy as proxy integration does

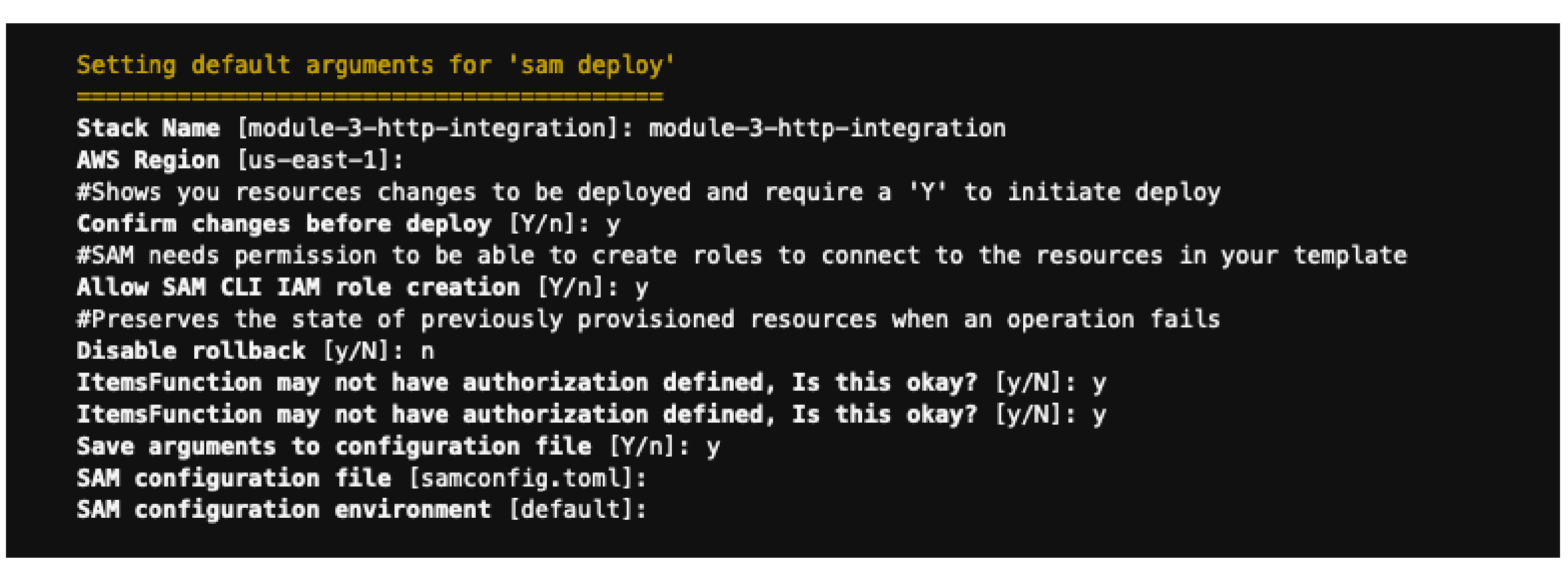
Deploy the project

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/http`, where the `template.yaml` file for the sample application is located:

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

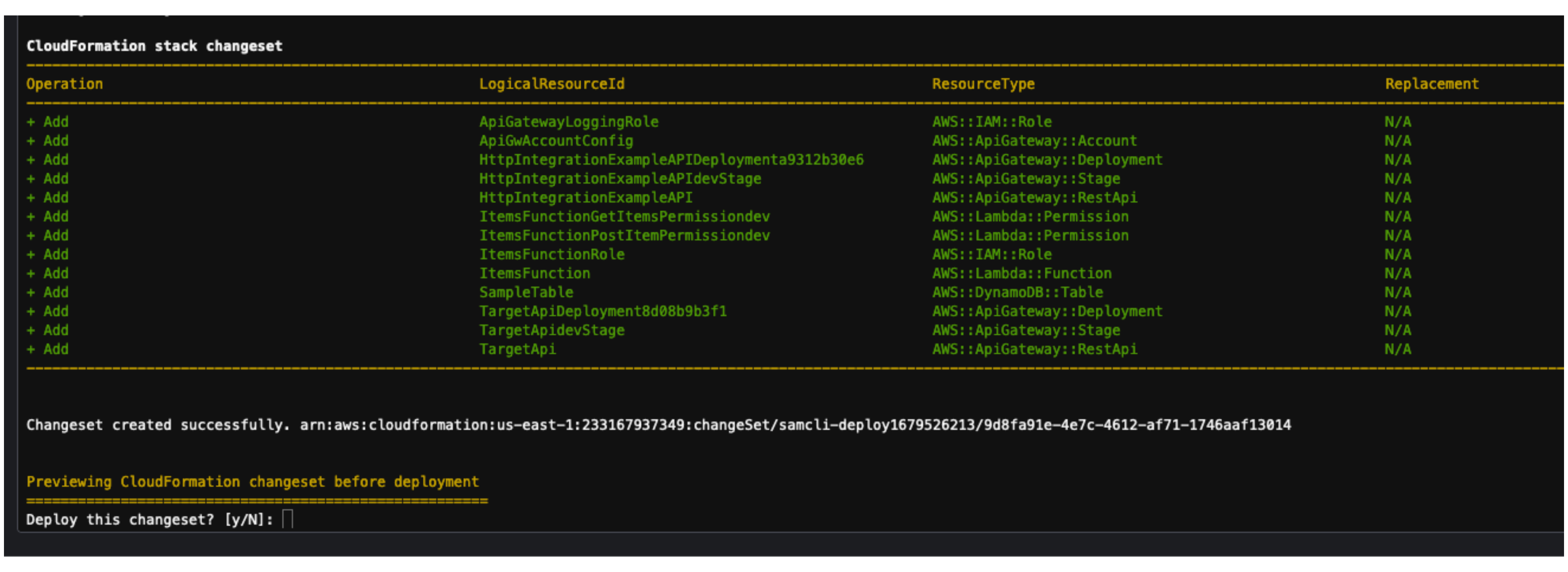
   The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you needed to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure as it is above.

   - **Stack Name**: `module-3-http-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`
   - **ItemsFunction 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-http-integration]: module-3-http-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
ItemsFunction may not have authorization defined, Is this okay? [y/N]: y
ItemsFunction 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                              ApiGatewayLoggingRole                         AWS::IAM::Role                        N/A
+ Add                              ApiGwAccountConfig                            AWS::ApiGateway::Account              N/A
+ Add                              HttpIntegrationExampleAPIDeploymenta9312b30e6 AWS::ApiGateway::Deployment           N/A
+ Add                              HttpIntegrationExampleAPIdevStage             AWS::ApiGateway::Stage                N/A
+ Add                              HttpIntegrationExampleAPI                     AWS::ApiGateway::RestApi              N/A
+ Add                              ItemsFunctionGetItemsPermissiondev            AWS::Lambda::Permission               N/A
+ Add                              ItemsFunctionPostItemPermissiondev            AWS::Lambda::Permission               N/A
+ Add                              ItemsFunctionRole                             AWS::IAM::Role                        N/A
+ Add                              ItemsFunction                                 AWS::Lambda::Function                 N/A
+ Add                              SampleTable                                   AWS::DynamoDB::Table                  N/A
+ Add                              TargetApiDeployment8d08b9b3f1                 AWS::ApiGateway::Deployment           N/A
+ Add                              TargetApidevStage                             AWS::ApiGateway::Stage                N/A
+ Add                              TargetApi                                     AWS::ApiGateway::RestApi              N/A
-------------------------------------------------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-east-1:233167937349:changeSet/samcli-deploy1679526213/9d8fa91e-4e7c-4612-af71-1746aaf13014


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

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

The Amazon API Gateway Workshop
Introduction
Getting Started
Module 1 - Introduction to Amazon API Gateway
Module 2 - Deploy your first API with IaC
Module 3 - API Gateway REST Integrations
  Module Goals
  Mock
  HTTP Integration
    Set up your AWS SAM Project
    Build HTTP Integration with AWS SAM and OpenAPI
    Test Integration
  AWS Lambda
  AWS Step Functions
  Amazon SQS
  Amazon SNS
  Amazon Kinesis
  Amazon DynamoDB
  Amazon EventBridge
  Private Integration
  Amazon S3
  Clean up
Module 4 - Observability in API Gateway
Module 5 - WebSocket APIs
Module 6 - Enable fine-grained access control for your APIs
Clean up
Resources

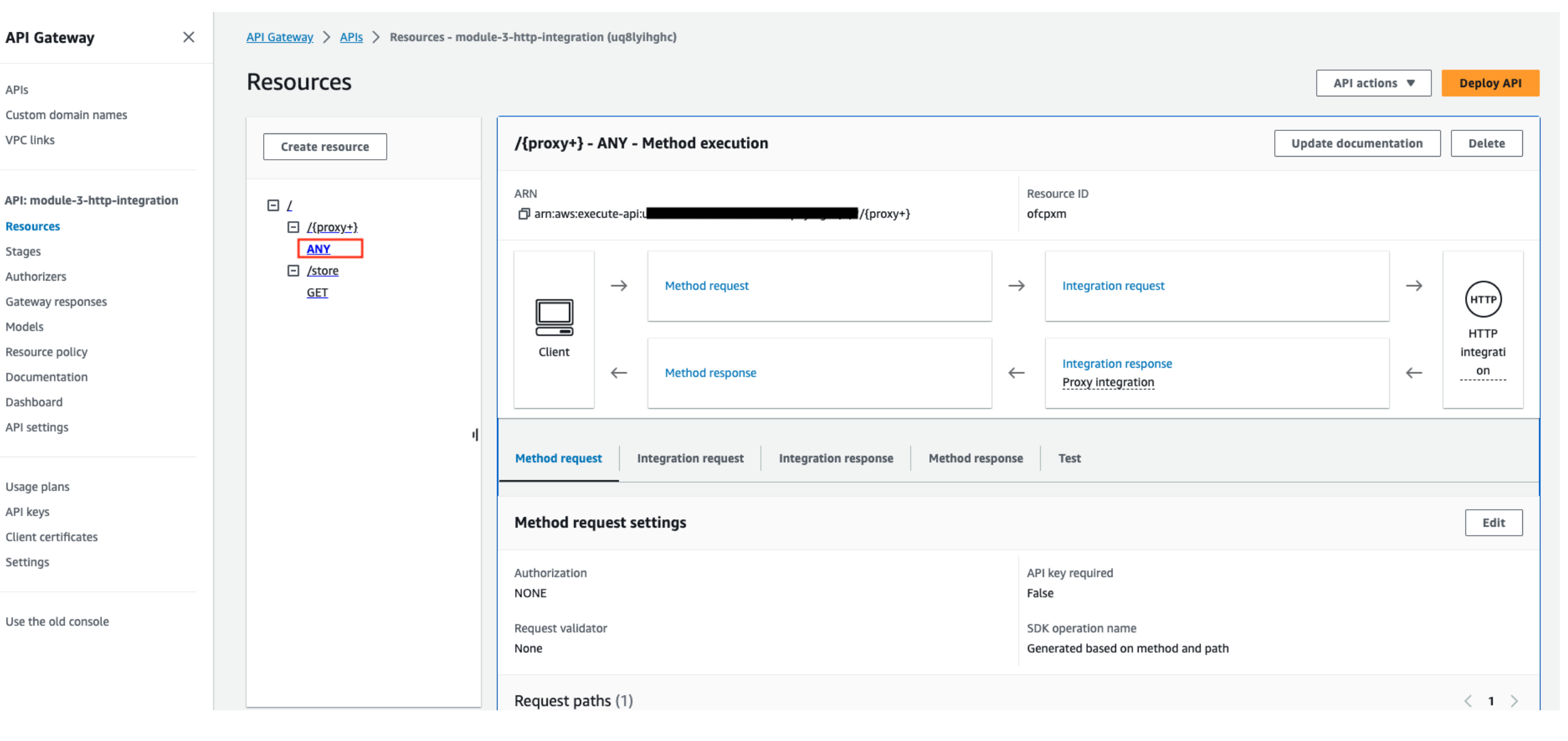
Test Integration

After you create your API Gateway REST API with HTTP integration, you can test it .

Test HTTP 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-http-integration`.
3. In the **Resources** pane, you can select the method you want to test. Click on the `ANY` method.



/{proxy+} - ANY - Method execution

ARN: arn:aws:execute-api:[redacted]/{proxy+}

Resource ID: ofcpxm

Method request settings

Authorization: NONE

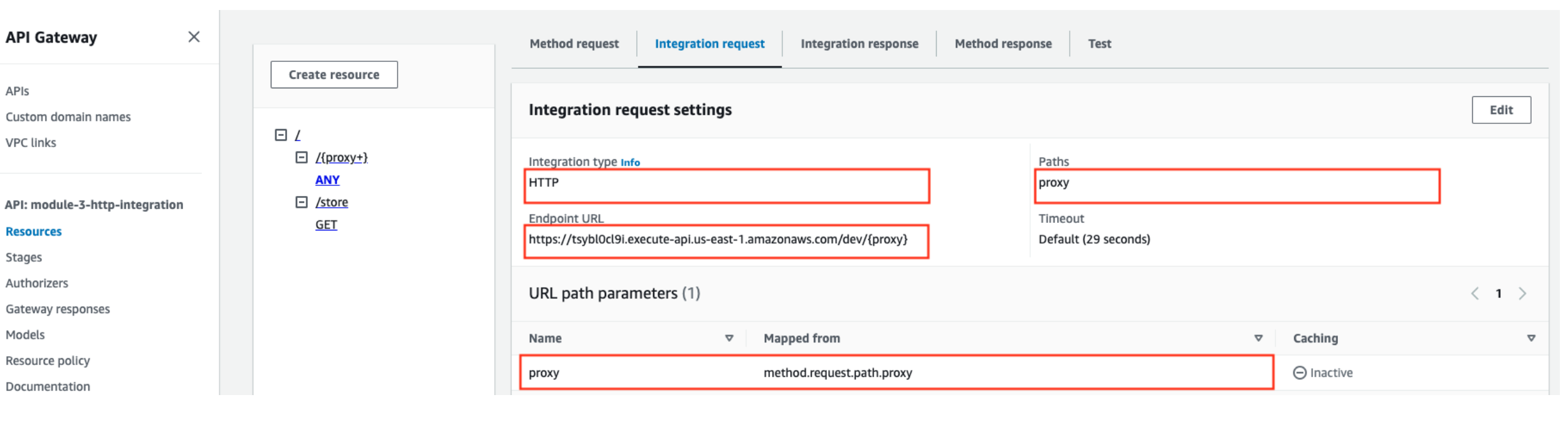
API key required: False

Request validator: None

SDK operation name: Generated based on method and path

Request paths (1)

4. Click in **Integration Request**, review the configuration details for integration.



Integration request settings

Integration type: HTTP

Paths: proxy

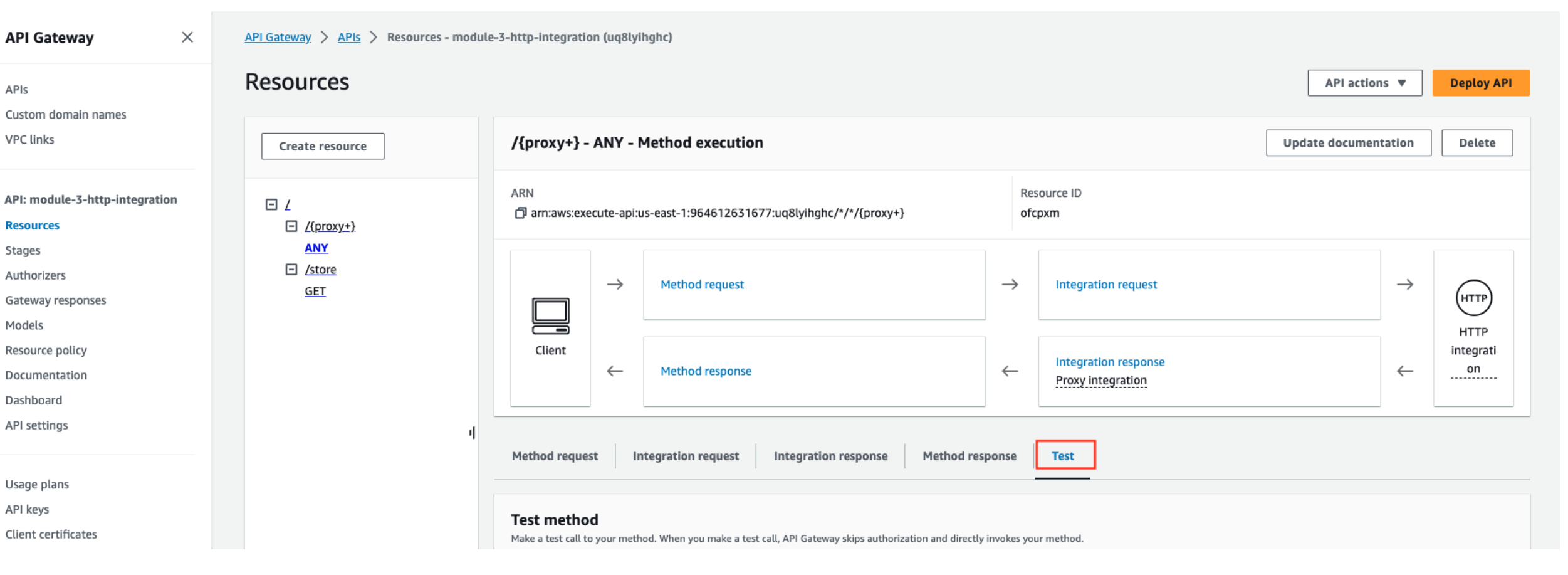
Endpoint URL: https://tsybl0cl9i.execute-api.us-east-1.amazonaws.com/dev/{proxy}

Timeout: Default (29 seconds)

URL path parameters (1)

Name	Mapped from	Caching
proxy	method.request.path.proxy	Inactive

5. Click on the **TEST** tab.



/{proxy+} - ANY - Method execution

ARN: arn:aws:execute-api:us-east-1:964612631677:uq8lyihghc/*/*/{proxy+}

Resource ID: ofcpxm

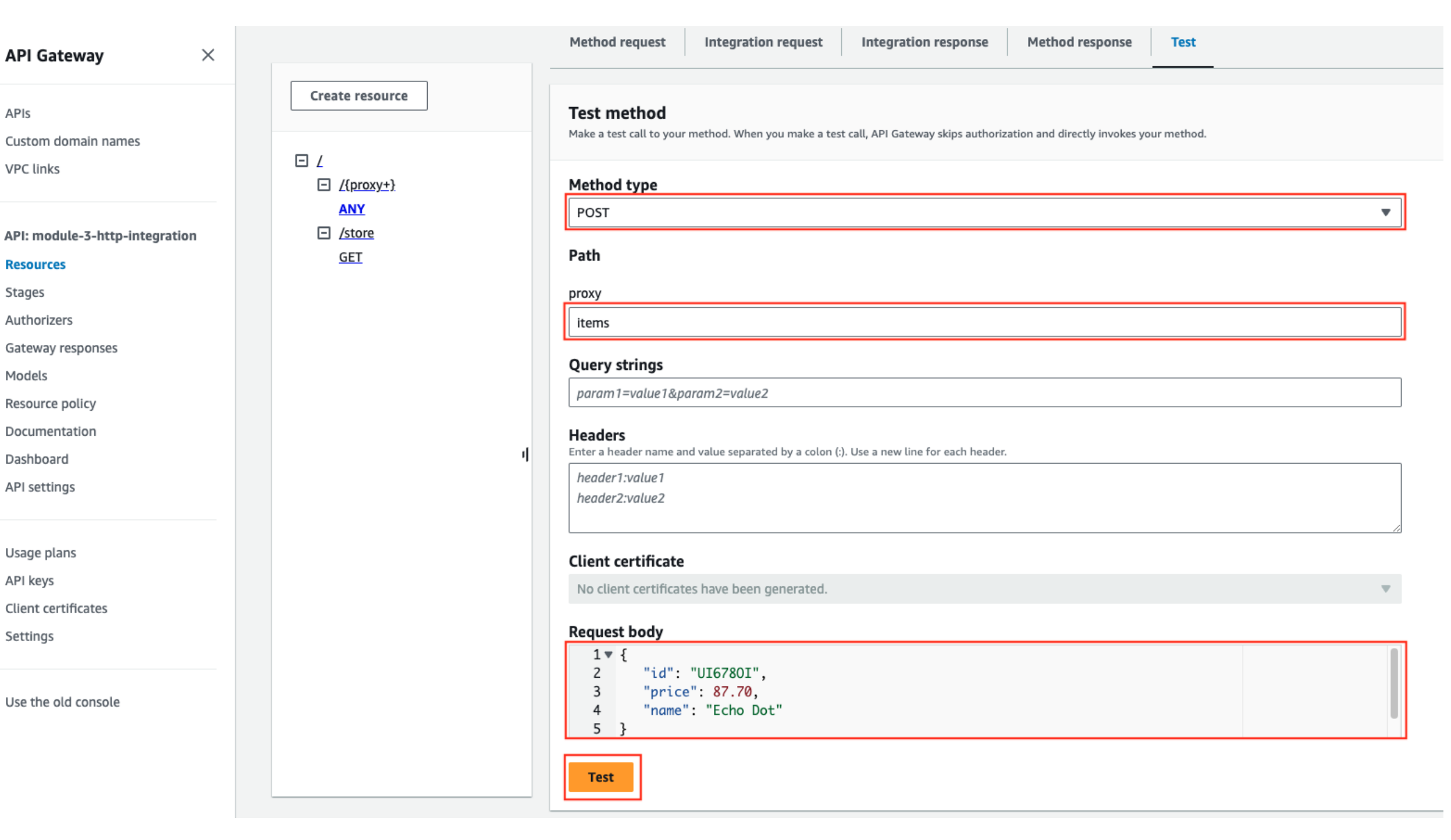
Test method

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

6. In the Method, choose **POST**. Then:
   - For **Path {proxy}**: `items`
   - For **Request Body**:

```
{
   "id": "UI678OI",
   "price": 87.70,
   "name": "Echo Dot"
}
```

   - Click on **Test** button. Leave the other fields blank.



Test method

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

Method type: POST

Path

proxy: items

Query strings: param1=value1&param2=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:

```
{
   "id": "UI678OI",
   "price": 87.70,
   "name": "Echo Dot"
}
```

   - The **test result** shows

/{proxy+} - ANY method test results

Request	Latency	Status
/items	1526	200

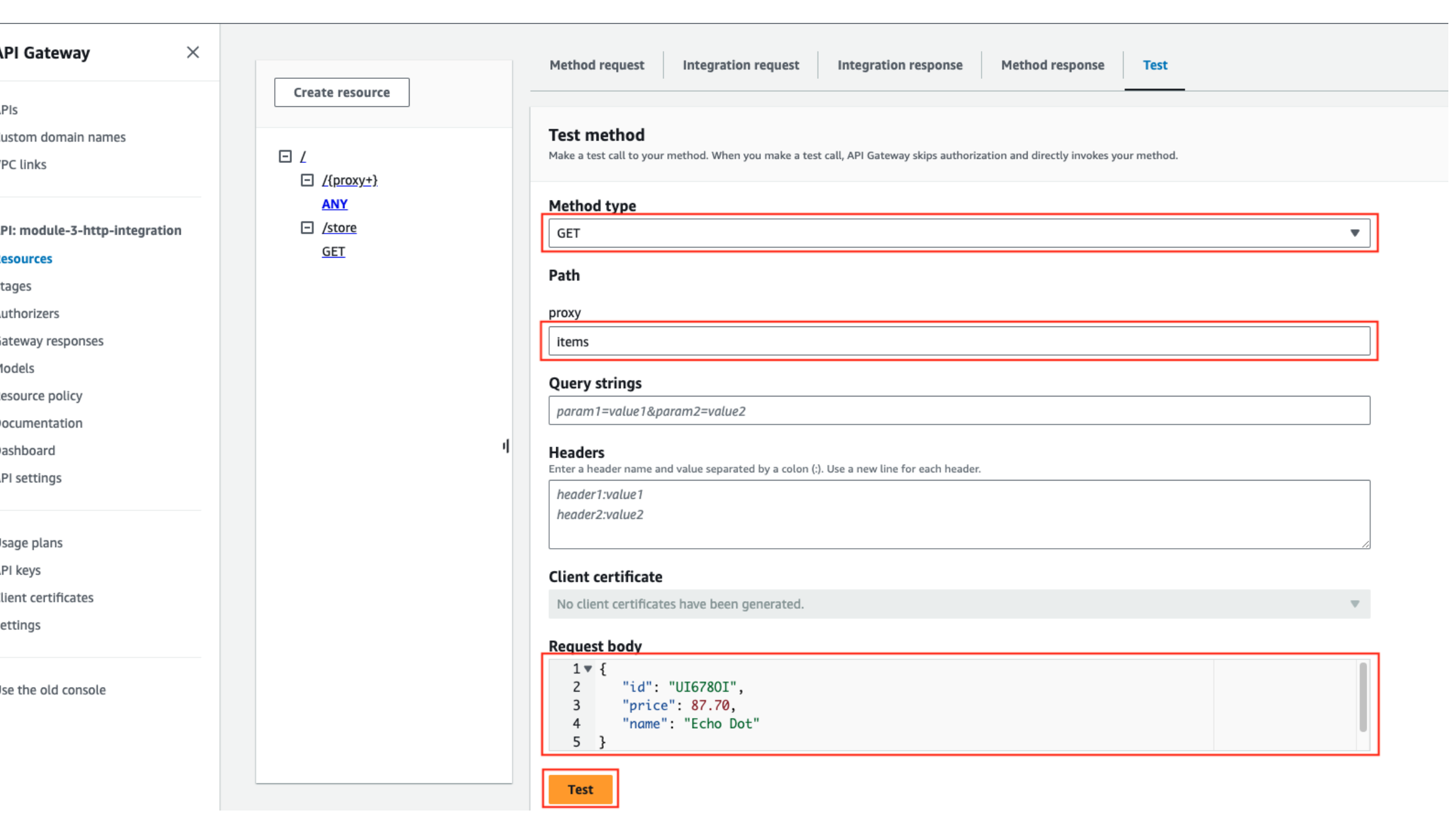
Response body

```
"Put item UI678OI"
```

Response headers

```
{
  "Connection": "keep-alive",
  "Content-Length": "18",
  "Content-Type": "application/json",
  "Date": "Thu, 19 Oct 2023 13:38:42 GMT",
  "Via": "1.1 3f2e448716e86a35bb027a469c98be3c.cloudfront.net (CloudFront)",
  "X-Amz-Cf-Id": "Exp8eWUlYiVZKzk96MQ_wQPNTxGnp_V49VTiAVSfoszE8Ug8w87y8A==",
  "X-Amz-Cf-Pop": "IAD61-P2",
  "X-Amzn-Trace-Id": "Root=1-65313160-3d418e1b032242bb211ed66e;Sampled=0;lineage=902e3a1d:0",
  "X-Cache": "Miss from cloudfront",
  "x-amz-apigw-id": "NDSnJG-_oAMFlzA=",
  "x-amzn-RequestId": "aafd2b1d-3cad-4ffe-b82b-29614be625ec"
}
```

7. Now change the **Method** to **GET** and click on **Test** button. Leave the other fields blank.



Method type: GET

Path

proxy: items

Query strings: param1=value1&param2=value2

Headers:

header1:value1
header2:value2

Client certificate: No client certificates have been generated.

Request body:

```
{
   "id": "UI678OI",
   "price": 87.70,
   "name": "Echo Dot"
}
```

8. The **test result** shows

/{proxy+} - ANY method test results

Request	Latency	Status
/items	798	200

Response body

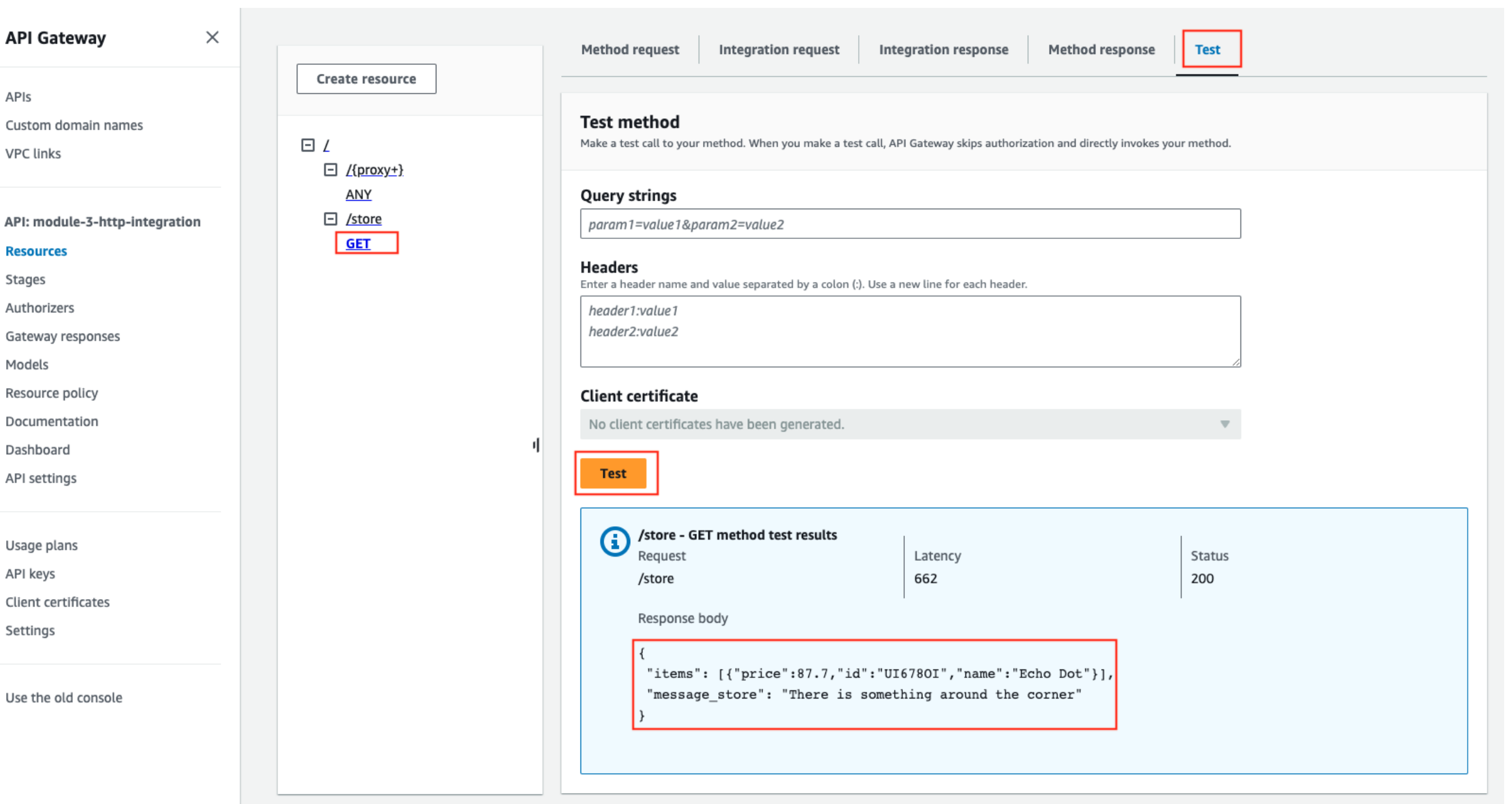
```
[{"price":87.7,"id":"UI678OI","name":"Echo Dot"}]
```

Response headers

```
{
  "Connection": "keep-alive",
  "Content-Length": "49",
  "Content-Type": "application/json",
  "Date": "Thu, 19 Oct 2023 13:43:22 GMT",
  "Via": "1.1 9cd85e528eb96b937681f7f81aea46c8.cloudfront.net (CloudFront)",
  "X-Amz-Cf-Id": "n6PKsjnSRx9gTlHN4DNUGmrLlX6_xaFffeDf8btnYPBqYeJ8IAT7KA==",
  "X-Amz-Cf-Pop": "IAD61-P2",
  "X-Amzn-Trace-Id": "Root=1-65313279-16df48d40098c53e17590647;Sampled=0;lineage=902e3a1d:0",
  "X-Cache": "Miss from cloudfront",
  "x-amz-apigw-id": "NDTTCHojIAMFbDw=",
  "x-amzn-RequestId": "d8001281-cb32-4d60-ad57-5cdf4a27c8b8"
}
```

Test HTTP Custom Integration using API Gateway Console

1. In **Resources**, select the method **GET** of `store`. Click in the `TEST` button. After that, leave all fields blank and click **Test**:



Test method

Query strings: param1=value1&param2=value2

Headers:

header1:value1
header2:value2

Client certificate: No client certificates have been generated.

/store - GET method test results

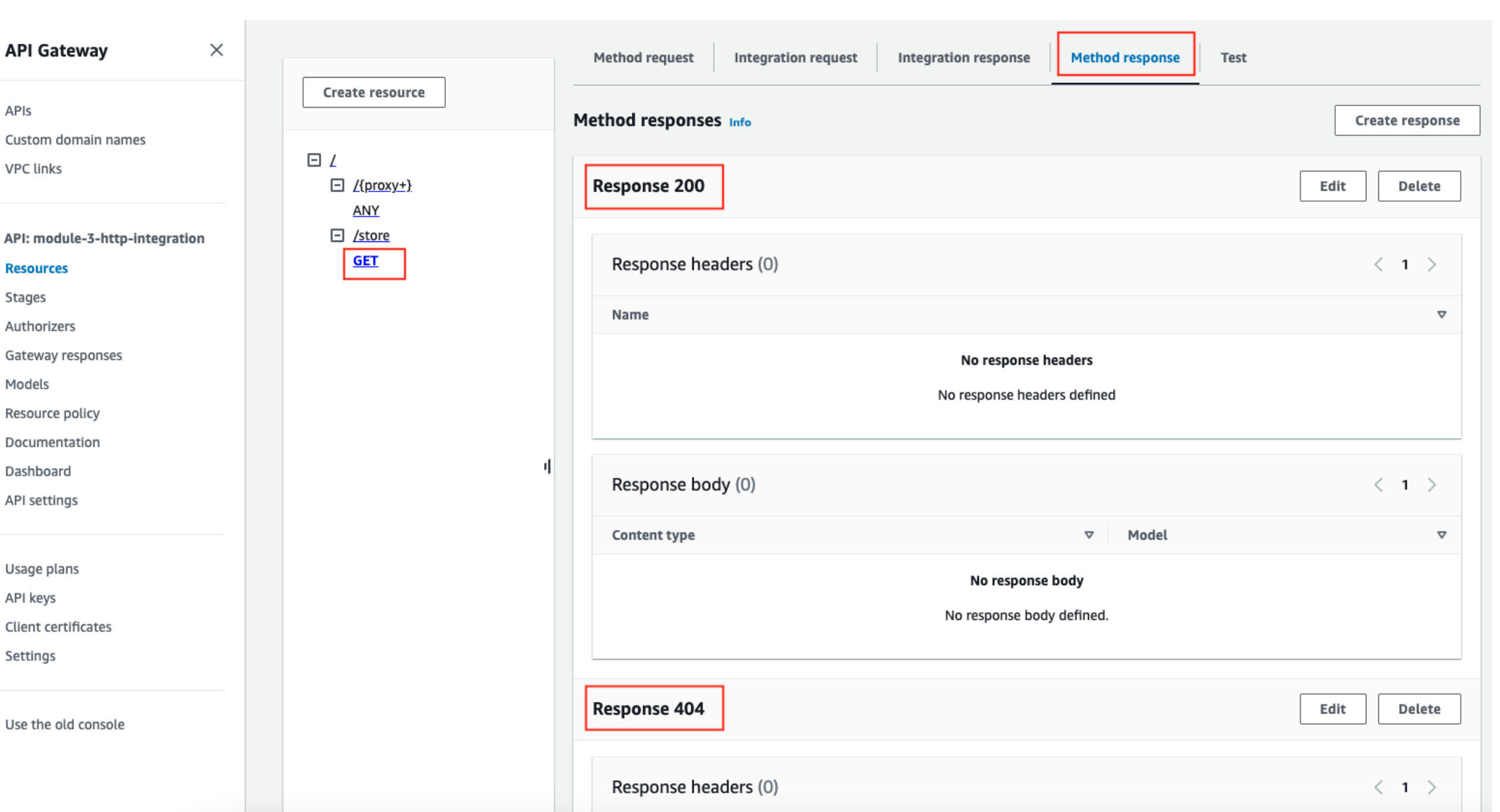
Request	Latency	Status
/store	662	200

Response body

```
{
 "items": [{"price":87.7,"id":"UI678OI","name":"Echo Dot"}],
 "message_store": "There is something around the corner"
}
```

Note that the response payload has been transformed.

2. Click on **Method Response** tab



Method responses

Response 200

Response headers (0)

Name

No response headers

No response headers defined

Response body (0)

Content type	Model

No response body

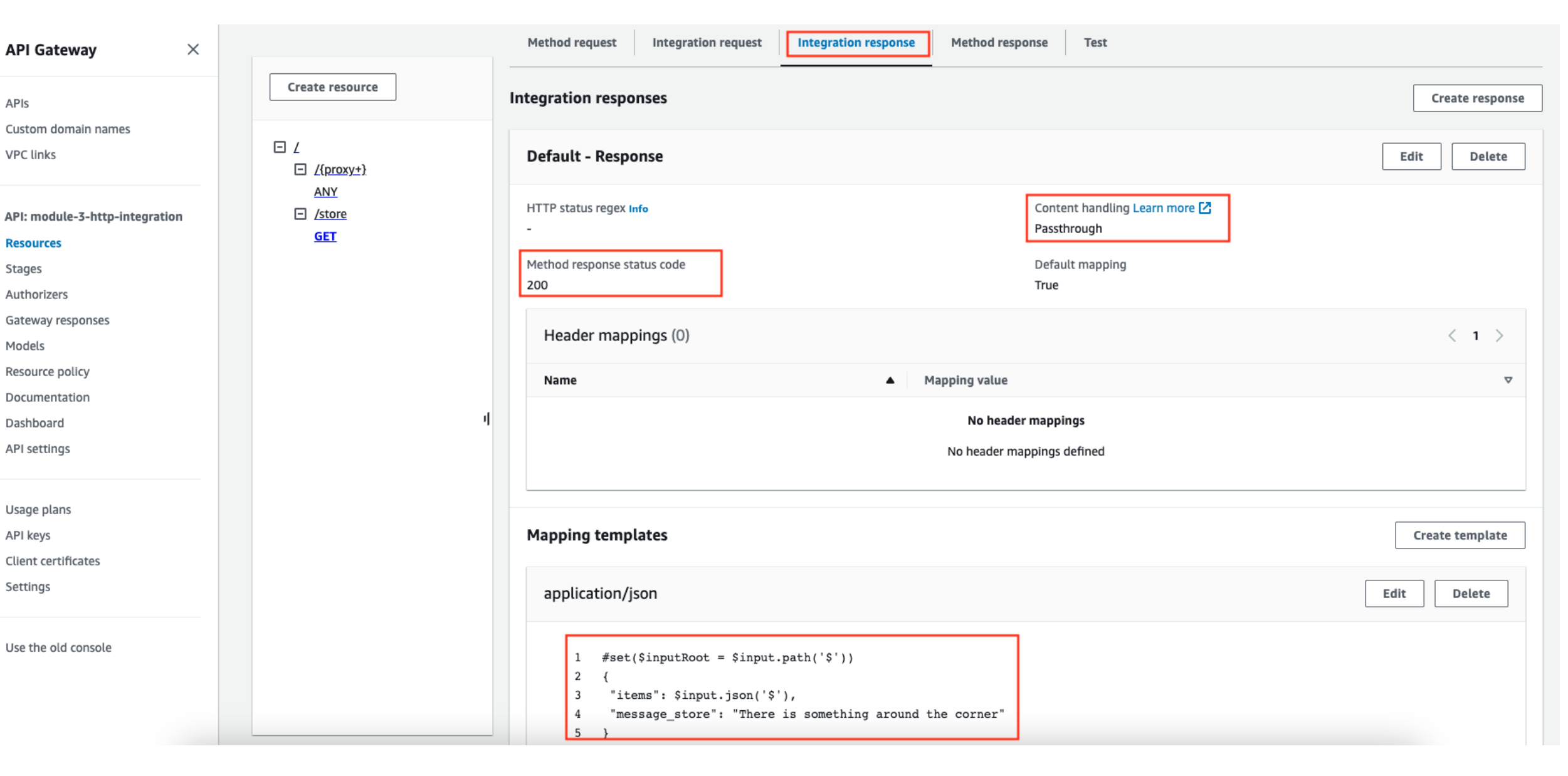
No response body defined.

Response 404

Response headers (0)

An API method response encapsulates the **output** of an API **method request** that the client will receive, like a contract. The output data includes a **HTTP status code**, **some headers**, and **possibly a body**. With non-proxy integrations, the specified response parameters and body can be mapped from the associated integration response data or can be assigned a static values according to the mapping. These mappings are specified in the **integration response** it can be an identical transformation that passes through the **integration response**.

3. Now let's explore **Integration Response**.
4. Note that a transformation is being performed on the response payload.



Integration responses

Default - Response

HTTP status regex: -

Content handling: Passthrough

Method response status code: 200

Default mapping: True

Header mappings (0)

Name	Mapping value

No header mappings

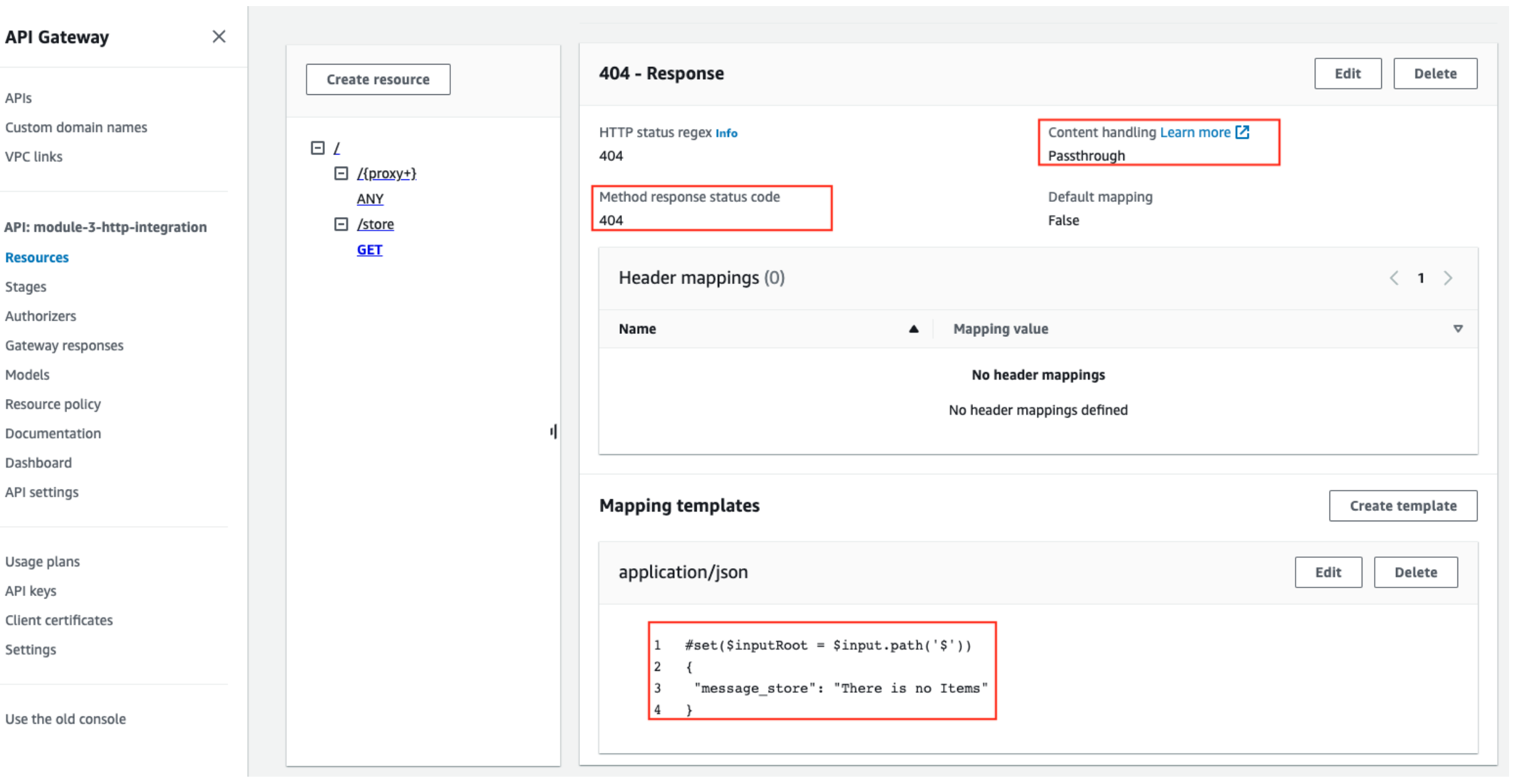
No header mappings defined

Mapping templates

application/json

```
#set($inputRoot = $input.path('$'))
{
 "items": $input.json('$'),
 "message_store": "There is something around the corner"
}
```

5. For **HTTP Code 404** it is inserting a static message: `This is no Items`.



404 - Response

HTTP status regex: 404

Content handling: Passthrough

Method response status code: 404

Default mapping: False

Header mappings (0)

Name	Mapping value

No header mappings

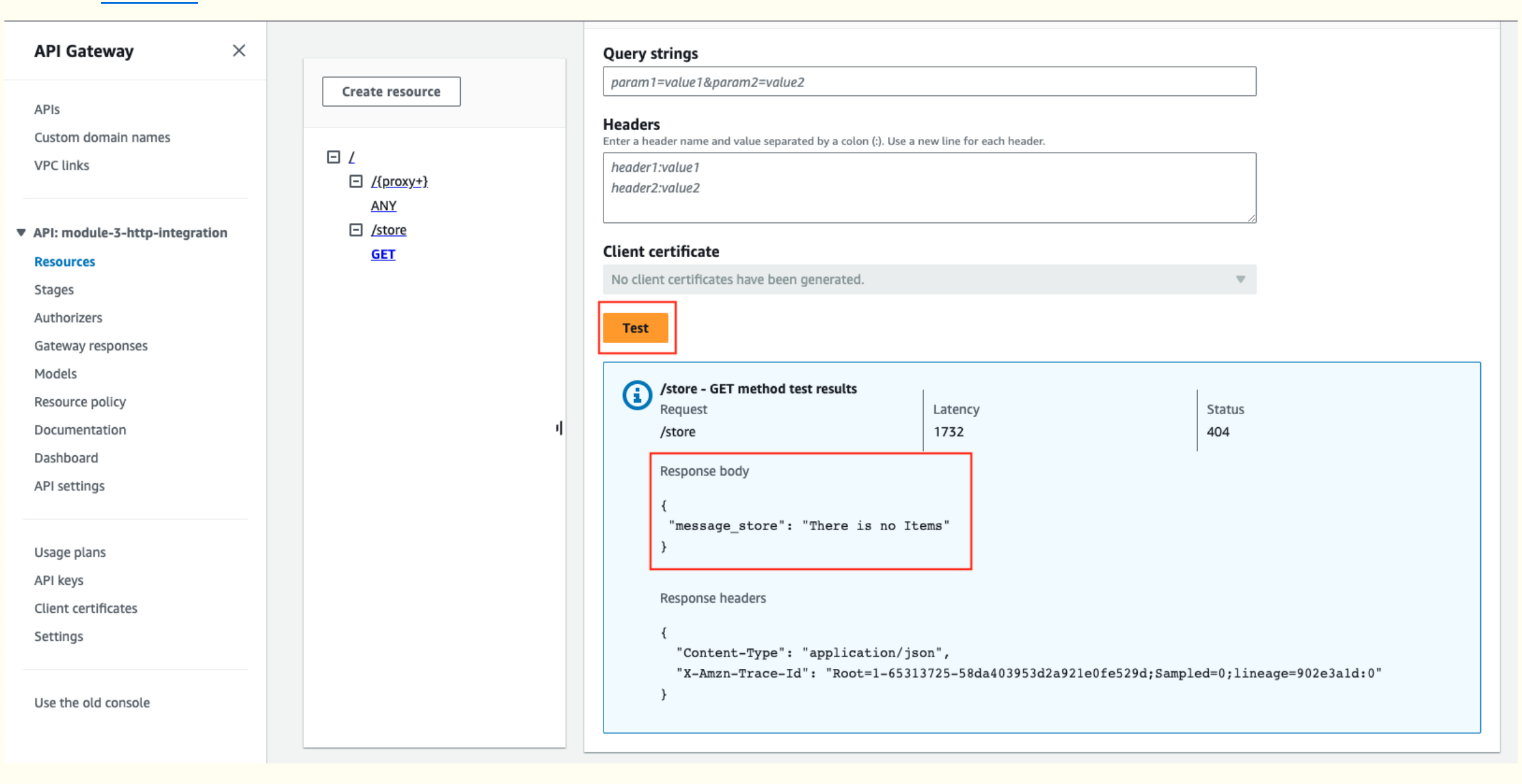
No header mappings defined

Mapping templates

application/json

```
#set($inputRoot = $input.path('$'))
{
 "message_store": "There is no Items"
}
```

> **Challenge**
> Access the DynamoDB console, find the Back End table and delete the item. This way you will be able to validate the custom response for **HTTP 404**.



Query strings: param1=value1&param2=value2

Headers:

header1:value1
header2:value2

Client certificate: No client certificates have been generated.

/store - GET method test results

Request	Latency	Status
/store	1732	404

Response body

```
{
 "message_store": "There is no Items"
}
```

Response headers

```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-65313725-58da403953d2a921e0fe529d;Sampled=0;lineage=902e3a1d:0"
}
```

Test the deployed API using cURL

Let's test the **proxy integration** first.

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 APIs `RestApiEndpoint` id that was the output of your `SAM deploy` and `<region>` with the region where your API is deployed. 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/items'
```

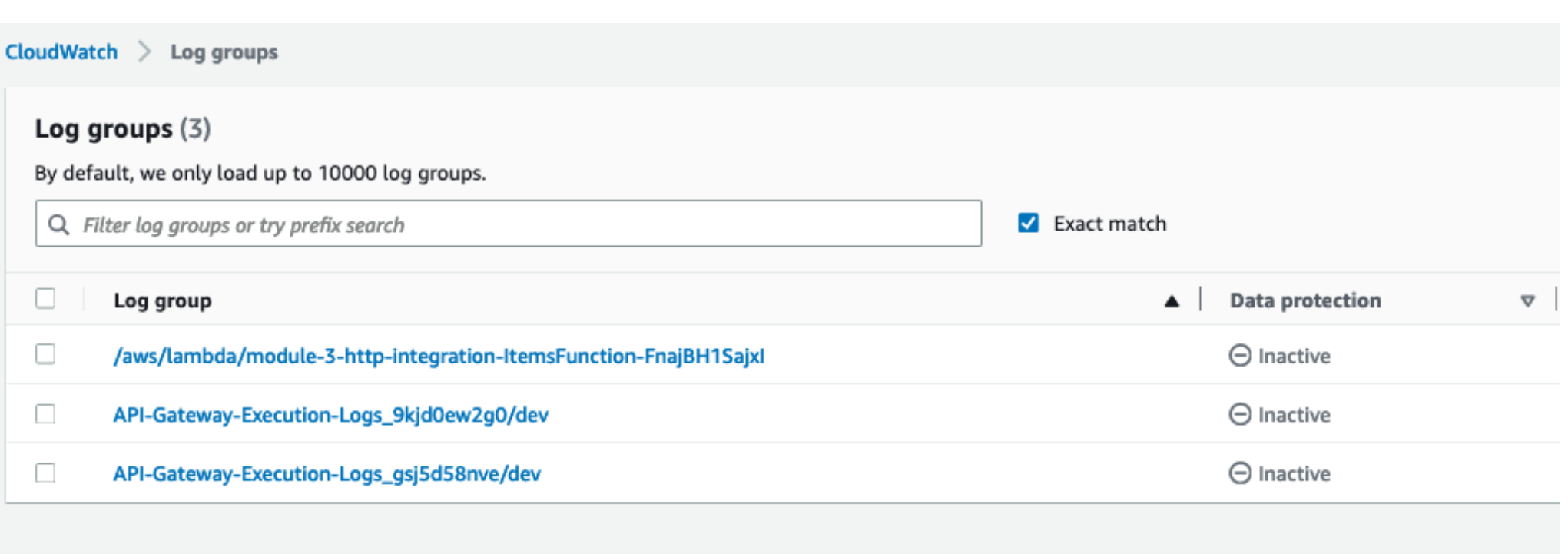
3. The **Response Body** output should be:

```
[{"price":87.7,"id":"UI678OI","name":"Echo Dot"}] or []
```

> **Note**
> - Setting **ANY** as **GET** and **{proxy+}** as `items` Method request initiated from the frontend:
>
> ```
> GET https://<RESTAPI>.execute-api.us-west-2.amazonaws.com/dev/items HTTP/1.1
> ```
>
> Integration request sent to the **backend**:
>
> ```
> GET https://<TARGETAPI>.execute-api.com/items HTTP/1.1
> ```
>
> The run-time instances of the **ANY** method and **proxy** resource are both **valid**. The call returns a **200** OK response with the payload containing items, as returned from the backend.
>
> - Setting **ANY** as **GET** and **{proxy+}** as `null` Method request initiated from the frontend:
>
> ```
> GET https://<RESTAPI>.execute-api.us-west-2.amazonaws.com/dev
> ```
>
> Integration request sent to the backend:
>
> ```
> GET https://<TARGETAPI>.com/
> ```
>
> The targeted resource is the parent of the **proxy** resource, but the run-time instance of the **ANY** method is not defined in the API on that resource. As a result, the **GET** request returns a **403 Forbidden** response with the **Missing Authentication Token** error message as returned by API Gateway.

Now let's check an important detail of the integration. Let's understand how the format of the **input and response event** of the **Rest API**, when the integration is of the proxy type.

1. Navigate to CloudWatch in the AWS console.
2. Click on **Log Groups**:

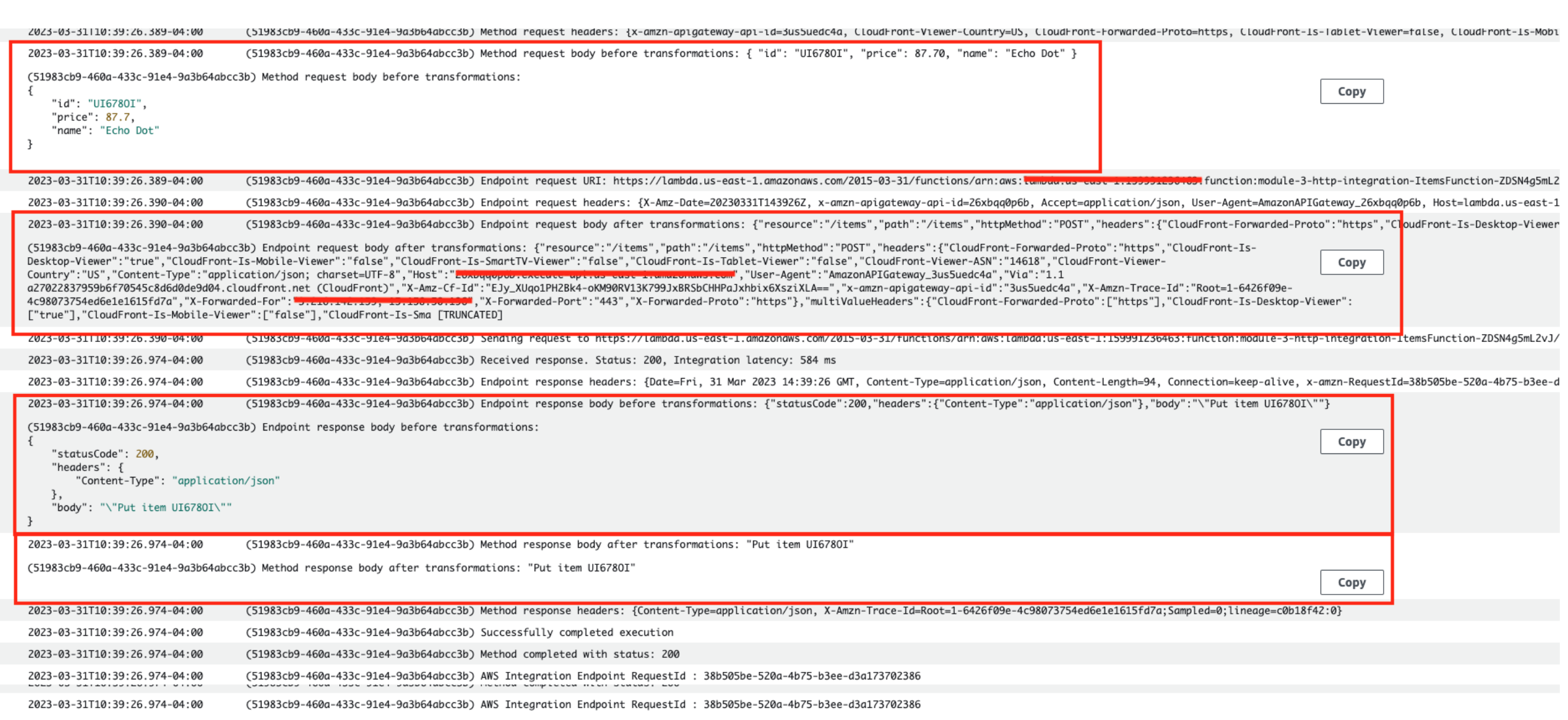


Log groups (3)

By default, we only load up to 10000 log groups.

Log group	Data protection
/aws/lambda/module-3-http-integration-ItemsFunction-FnajBH1SajxI	Inactive
API-Gateway-Execution-Logs_9kjd0ew2g0/dev	Inactive
API-Gateway-Execution-Logs_gsj5d58nve/dev	Inactive

3. In the screenshot above you can see the **Rest API**, **Target API** and **Lambda** log groups that serve as the Back End for the Target API. We want to analyze the **Rest API logs**, the focus of this session, as it is integrating with Target.
4. To know which is which you need to know what the **API ID** is. You can find the **Rest API** ID in the API Gateway console by navigating to, `module-3-http-integration` and **Stages > dev**
5. Enter the log, and select a log stream. Below, we have an example of `POST` call showed in a log stream highlighting the transformations before/after the Lambda call.



```
2023-03-31T10:39:26.389-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Method request headers: {x-amzn-apigateway-api-id=3us5uedc4a, CloudFront-Viewer-Country=US, CloudFront-Forwarded-Proto=https, CloudFront-Is-Tablet-Viewer=false, CloudFront-Is-Mobi
2023-03-31T10:39:26.389-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Method request body before transformations: { "id": "UI678OI", "price": 87.70, "name": "Echo Dot" }
(51983cb9-460a-433c-91e4-9a3b64abcc3b) Method request body before transformations:
{
    "id": "UI678OI",
    "price": 87.7,
    "name": "Echo Dot"
}
2023-03-31T10:39:26.389-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint request URI: https://lambda.us-east-1.amazonaws.com/2015-03-31/functions/arn:aws:[redacted]function:module-3-http-integration-ItemsFunction-ZDSN4g5mL2
2023-03-31T10:39:26.390-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint request headers: {X-Amz-Date=20230331T143926Z, x-amzn-apigateway-api-id=26xbqq0p6b, Accept=application/json, User-Agent=AmazonAPIGateway_26xbqq0p6b, Host=lambda.us-east-1
2023-03-31T10:39:26.390-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint request body after transformations: {"resource":"/items","path":"/items","httpMethod":"POST","headers":{"CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-Viewer
(51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint request body after transformations: {"resource":"/items","path":"/items","httpMethod":"POST","headers":{"CloudFront-Forwarded-Proto":"https","CloudFront-Is-
Desktop-Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-ASN":"14618","CloudFront-Viewer-
Country":"US","Content-Type":"application/json; charset=UTF-8","Host":"[redacted]","User-Agent":"AmazonAPIGateway_3us5uedc4a","Via":"1.1
a27022837959b6f70545c8d6d0de9d04.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"EJy_XUqo1PH2Bk4-oKM90RV13K799JxBRSbCHHPaJxhbix6XsziXLA==","x-amzn-apigateway-api-id":"3us5uedc4a","X-Amzn-Trace-Id":"Root=1-6426f09e-
4c98073754ed6e1e1615fd7a","X-Forwarded-For":"[redacted]","X-Forwarded-Port":"443","X-Forwarded-Proto":"https"},"multiValueHeaders":{"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-Viewer":
["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-Sma [TRUNCATED]
2023-03-31T10:39:26.390-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Sending request to https://lambda.us-east-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-east-1:159991236463:function:module-3-http-integration-ItemsFunction-ZDSN4g5mL2vJ/
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Received response. Status: 200, Integration latency: 584 ms
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint response headers: {Date=Fri, 31 Mar 2023 14:39:26 GMT, Content-Type=application/json, Content-Length=94, Connection=keep-alive, x-amzn-RequestId=38b505be-520a-4b75-b3ee-d
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint response body before transformations: {"statusCode":200,"headers":{"Content-Type":"application/json"},"body":"\"Put item UI678OI\""}
(51983cb9-460a-433c-91e4-9a3b64abcc3b) Endpoint response body before transformations:
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": "\"Put item UI678OI\""
}
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Method response body after transformations: "Put item UI678OI"
(51983cb9-460a-433c-91e4-9a3b64abcc3b) Method response body after transformations: "Put item UI678OI"
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Method response headers: {Content-Type=application/json, X-Amzn-Trace-Id=Root=1-6426f09e-4c98073754ed6e1e1615fd7a;Sampled=0;lineage=c0b18f42:0}
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Successfully completed execution
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) Method completed with status: 200
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) AWS Integration Endpoint RequestId : 38b505be-520a-4b75-b3ee-d3a173702386
2023-03-31T10:39:26.974-04:00    (51983cb9-460a-433c-91e4-9a3b64abcc3b) AWS Integration Endpoint RequestId : 38b505be-520a-4b75-b3ee-d3a173702386
```

For **Custom Integration**:

1. Copy the following cURL command and paste it into the terminal window, replacing `<api-id>` with your APIs `RestApiEndpoint` id that was the output of your `SAM deploy` and `<region>` with the region where your API is deployed. 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/store'
```

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

```
{
   "message_store": "There is no Items"
}
```

```
{
   "items": [{"price":87.7,"id":"UI678OI","name":"Echo Dot"}],
   "message_store": "There is something around the corner"
}
```

> **Challenge**
> Perform tests with the POST and GET for proxy and custom examples methods, and try to find the payload logs **before** and **after** the **request** and **response** transformations in **CloudWatch**.