The Amazon API Gateway Workshop
- Introduction
- Getting Started
- Module 1 - Introduction to Amazon API Gateway
- Module 2 - Deploy your first API with IaC
  - Module Goals
  - Set up your AWS SAM Project
  - Create your first API with IaC
  - Message Transformation
  - Request Validation
  - Authentication and Authorization
  - Message Caching
  - Usage Plans and Message Throttling
  - Export OpenAPI via Console
  - Clean up
- Module 3 - API Gateway REST Integrations
- 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 2 - Deploy your first API with IaC > Message Transformation

Message Transformation

In API Gateway, a mapping template is used to transform data from one format to another. JSON path expressions can be used to map and transform the integration payload to any desired format. In addition, a model (schema) can be created to define the structure of a message payload. Having a model also enables you to generate an SDK that can be used by the client application to send properly formatted messages.

In this section, you will first create a model to represent the schema of the incoming request and response messages. Then you will validate and transform the incoming request message to match the downstream service (lambda) specification. The returned response message will also be formatted to properly capture the unit metrics.

Building a Model

1. First, let's add a new **Model** called **costCalculatorRequest** to the definition file `openapi.yaml`.

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

```
1  openapi: "3.0.1"
2  info:
3    title: "module-2-my-first-api"
4    description: "First API with IaC example"
5    version: "1.0"
6
7  paths:
8
9      /pricepermeter:
10         post:
11             consumes:
12             - "application/json"
13             produces:
14             - "application/json"
15             responses:
16                 "200":
17                     description: "200 response"
18             x-amazon-apigateway-integration:
19                 httpMethod: "POST"
20                 credentials:
21                     Fn::GetAtt: [LambdaExecutionRole, Arn]
22                 uri:
23                   Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${CostCalculator.Arn}/invocations"
24                 responses:
25                     default:
26                         statusCode: "200"
27                 passthroughBehavior: "when_no_match"
28                 type: "aws"
29
30 components:
31     schemas:
32         costCalculatorRequest:
33             type: object
34             properties:
35                 price:
36                     type: number
37                 size:
38                     type: number
39                 unit:
40                     type: string
41                 downPayment:
42                     type: number
```

On **line 30** we have components Object, which holds a set of reusable objects for different aspects of the OAS.

> All objects defined within the components object will have no effect on the API unless they are explicitly referenced from properties outside the components object.

On **line 31** we have schemas Object, which allows the definition of input and output data types. These types can be objects, but also primitives and arrays.

On **lines 32 to 42** we have the **costCalculatorRequest** schema, responsible for defining the model.

2. Deploy the project again using SAM to verify that the model was added in your first API.

```
sam build && sam deploy
```

> Notice that now you don't need to use the argument --guided to the **sam deploy** command again, because the deploy settings is already saved locally into the sam configuration file `samconfig.toml`

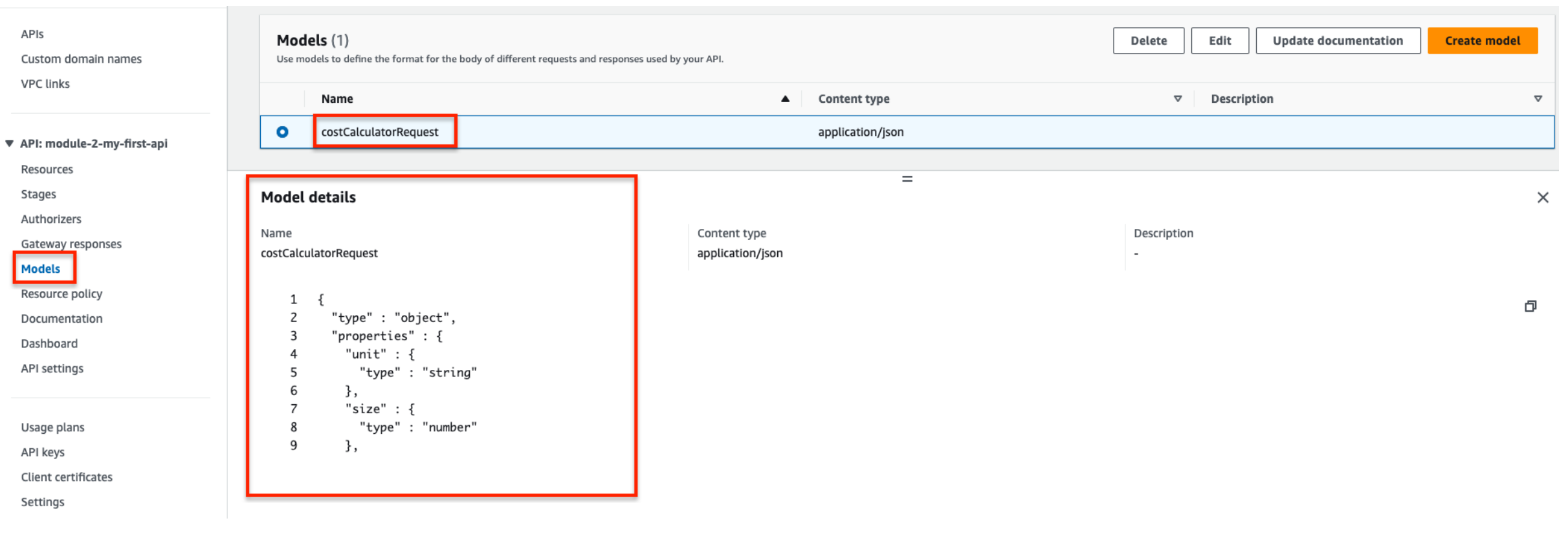
3. After the new changeset is created, review the resources and type `y` to confirm the new deployment.

```
CloudFormation stack changeset
-------------------------------------------------------------------------------------------------
Operation            LogicalResourceId                  ResourceType                  Replacement
-------------------------------------------------------------------------------------------------
+ Add                FirstAPIDeployment12c42beaef       AWS::ApiGateway::Deployment   N/A
* Modify             FirstAPIdevStage                   AWS::ApiGateway::Stage        False
* Modify             FirstAPI                           AWS::ApiGateway::RestApi      False
- Delete             FirstAPIDeployment9802b7b189       AWS::ApiGateway::Deployment   N/A
-------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-east-1:907236258605:changeSet/samcli-deploy1680665929/a52594df-bb06-4eef-8bc1-29fe2450ed7c

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

4. Check if the stack was successfully updated and open the Amazon API Gateway console.
5. Choose your REST API named, `module-2-my-first-api`.
6. Click in **Models** and then **costCalculatorRequest** to see the model that you just created.



Now that the models are created, the next step is to transform the incoming request and response messages.

Transform Request Payload

1. Go back to the AWS Cloud9 console.
2. Add a new **requestTemplates object** to the integration in your file `openapi.yaml`.

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

```
1  openapi: "3.0.1"
2  info:
3    title: "module-2-my-first-api"
4    description: "First API with IaC example"
5    version: "1.0"
6
7  paths:
8
9      /pricepermeter:
10         post:
11             consumes:
12             - "application/json"
13             produces:
14             - "application/json"
15             responses:
16                 "200":
17                     description: "200 response"
18             x-amazon-apigateway-integration:
19                 httpMethod: "POST"
20                 credentials:
21                     Fn::GetAtt: [LambdaExecutionRole, Arn]
22                 uri:
23                   Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${CostCalculator.Arn}/invocations"
24                 responses:
25                     default:
26                         statusCode: "200"
27                 requestTemplates:
28                     application/json: |
29                         #set($inputRoot = $input.path("$"))
30                         {
31                             "price": "$inputRoot.price",
32                             "size": "$inputRoot.size",
33                             "unit": "$inputRoot.unit",
34                             "downPaymentAmount": $inputRoot.downPayment
35                         }
36                 passthroughBehavior: "when_no_match"
37                 type: "aws"
38
39 components:
40     schemas:
41         costCalculatorRequest:
42             type: object
43             properties:
44                 price:
45                     type: number
46                 size:
47                     type: number
48                 unit:
49                     type: string
50                 downPayment:
51                     type: number
```

On **line 27** we have **requestTemplates object**, which especifies mapping templates for a request payload of the specified MIME types.

On **lines 28 to 35**, the first statement #set($inputRoot = $input.path("$")) uses a JSONPath expression and returns an object representation of the result. This allows you to access and manipulate elements of the payload natively in Apache Velocity Template Language (VTL).

After the inputRoot variable is assigned to the root of the request, we can map the value of price, size and unit into the appropriate fields.

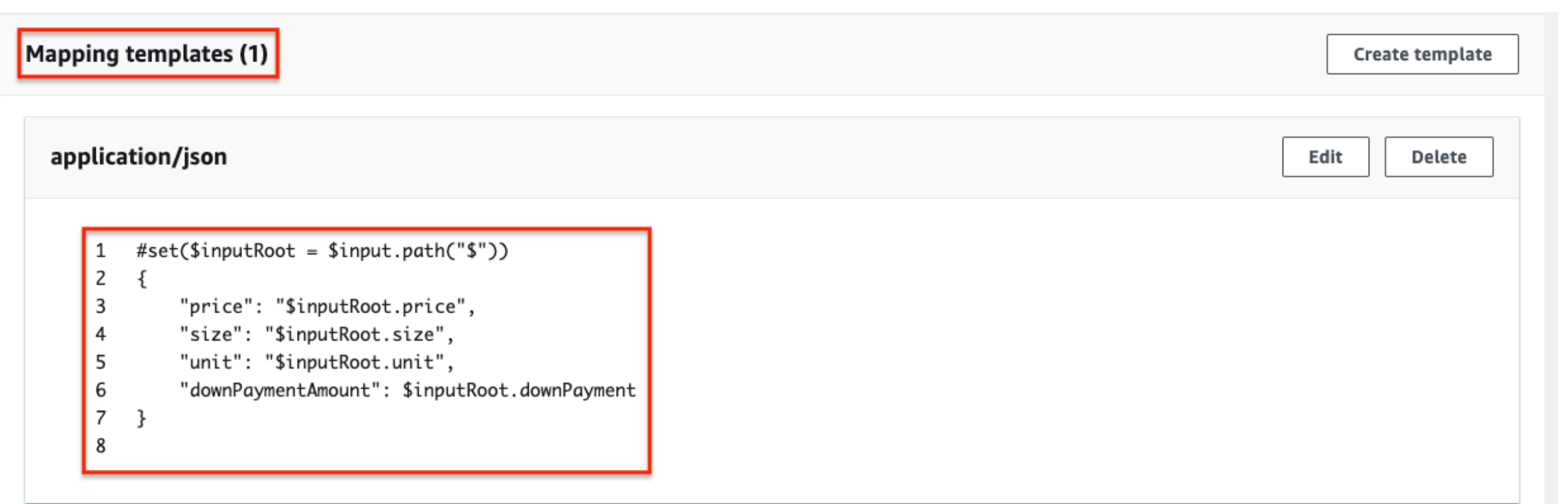
> **Notice**
> The last statement is mapping the value of downPayment to a new attribute called downPaymentAmount.

The downstream lambda function uses the downPaymentAmount to calculate the total price. (i.e. `totalPrice = price + downPaymentAmount`).

3. Deploy the project again using SAM to verify that the requestTemplate was added to your API Integration Request.

```
sam build && sam deploy
```

4. Check if the stack was successfully updated and open the Amazon API Gateway console.
5. Choose your REST API named, `module-2-my-first-api`.
6. In Resources click in /pricepermeter `POST`
7. Click in **Integration Request** and then scroll down to the **Mapping Templates** section to see that the mapping template was added to your integration resquest.



The next step is to map the integration response. An API Gateway response is identified by a response type defined by API Gateway. The response consists of an HTTP status code, a set of additional headers that are specified by parameter mappings, and a payload that is generated by a VTL mapping template

Transform Response Payload

Similar to the request mapping, create the mapping template for the response payload. Perform the following:

1. Go back to the AWS Cloud9 console.
2. Add a new **requestTemplates object** to the integration in your file `openapi.yaml`.

```
1  openapi: "3.0.1"
2  info:
3    title: "module-2-my-first-api"
4    description: "First API with IaC example"
5    version: "1.0"
6
7  paths:
8
9      /pricepermeter:
10         post:
11             consumes:
12             - "application/json"
13             produces:
14             - "application/json"
15             responses:
16                 "200":
17                     description: "200 response"
18             x-amazon-apigateway-integration:
19                 httpMethod: "POST"
20                 credentials:
21                     Fn::GetAtt: [LambdaExecutionRole, Arn]
22                 uri:
23                   Fn::Sub: "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${CostCalculator.Arn}/invocations"
24                 responses:
25                     default:
26                         statusCode: "200"
27                         responseTemplates:
28                             application/json: |
29                                 #set($allParams = $input.params())
30                                 {
31                                 "body-json" : $input.json('$'),
32                                 "params" : {
33                                 #foreach($type in $allParams.keySet())
34                                     #set($params = $allParams.get($type))
35                                 "$type" : {
36                                     #foreach($paramName in $params.keySet())
37                                     "$paramName" : "$util.escapeJavaScript($params.get($paramName))"
38                                         #if($foreach.hasNext),#end
39                                     #end
40                                 }
41                                     #if($foreach.hasNext),#end
42                                 #end
43                                 },
44                                 "stage-variables" : {
45                                 #foreach($key in $stageVariables.keySet())
46                                 "$key" : "$util.escapeJavaScript($stageVariables.get($key))"
47                                     #if($foreach.hasNext),#end
48                                 #end
49                                 },
50                                 "context" : {
51                                     "account-id" : "$context.identity.accountId",
52                                     "api-id" : "$context.apiId",
53                                     "api-key" : "$context.identity.apiKey",
54                                     "authorizer-principal-id" : "$context.authorizer.principalId",
55                                     "caller" : "$context.identity.caller",
56                                     "cognito-authentication-provider" : "$context.identity.cognitoAuthenticationProvider",
57                                     "cognito-authentication-type" : "$context.identity.cognitoAuthenticationType",
58                                     "cognito-identity-id" : "$context.identity.cognitoIdentityId",
59                                     "cognito-identity-pool-id" : "$context.identity.cognitoIdentityPoolId",
60                                     "http-method" : "$context.httpMethod",
61                                     "stage" : "$context.stage",
62                                     "source-ip" : "$context.identity.sourceIp",
63                                     "user" : "$context.identity.user",
64                                     "user-agent" : "$context.identity.userAgent",
65                                     "user-arn" : "$context.identity.userArn",
66                                     "request-id" : "$context.requestId",
67                                     "resource-id" : "$context.resourceId",
68                                     "resource-path" : "$context.resourcePath"
69                                     }
70                                 }
71                 requestTemplates:
72                     application/json: |
73                         #set($inputRoot = $input.path("$"))
74                         {
75                             "price": "$inputRoot.price",
76                             "size": "$inputRoot.size",
77                             "unit": "$inputRoot.unit",
78                             "downPaymentAmount": $inputRoot.downPayment
79                         }
80                 passthroughBehavior: "when_no_match"
81                 type: "aws"
82
83 components:
84     schemas:
85         costCalculatorRequest:
86             type: object
87             properties:
88                 price:
89                     type: number
90                 size:
91                     type: number
92                 unit:
93                     type: string
94                 downPayment:
95                     type: number
```

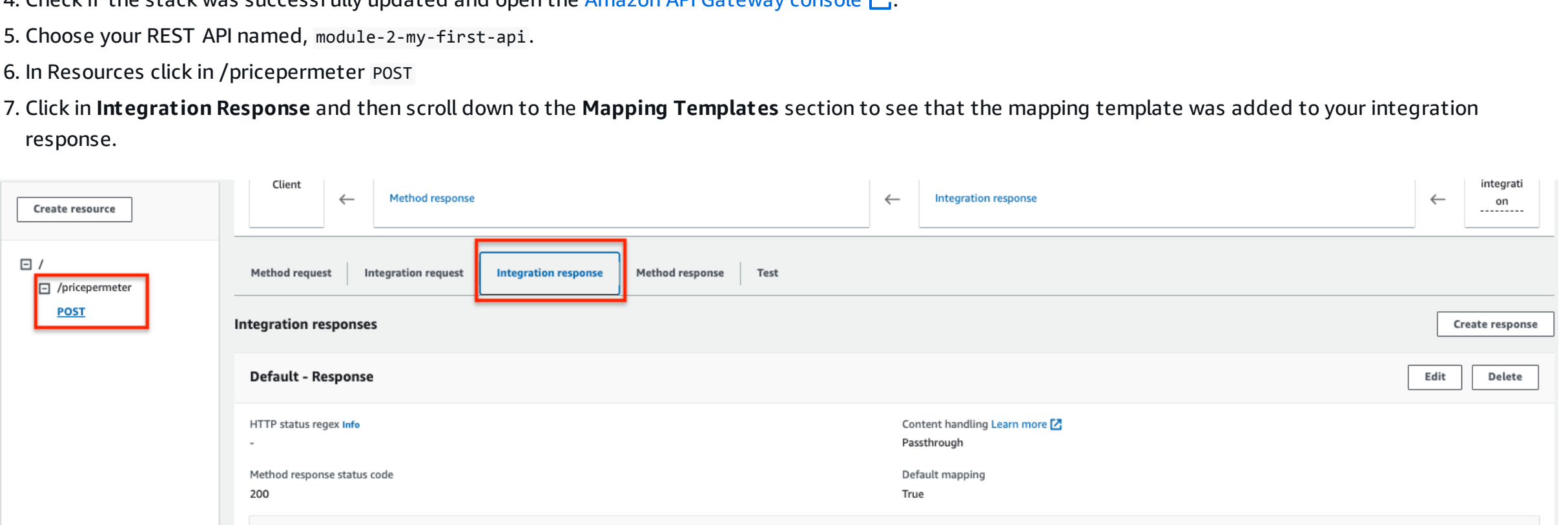
Now on **line 27** we have **responseTemplates object**, which defines GatewayResponse mapping templates, as a string-to-string map of key-value pairs, for a given gateway response.

On **lines 29 to 70**, we are defining the template that will be applied to the integration response, this template can be customized to add transformations according to your needs.

3. Deploy the project again using SAM to verify that the requestTemplate was added to your API Integration Request.

```
sam build && sam deploy
```

4. Check if the stack was successfully updated and open the Amazon API Gateway console.
5. Choose your REST API named, `module-2-my-first-api`.
6. In Resources click in /pricepermeter `POST`
7. Click in **Integration Response** and then scroll down to the **Mapping Templates** section to see that the mapping template was added to your integration response.

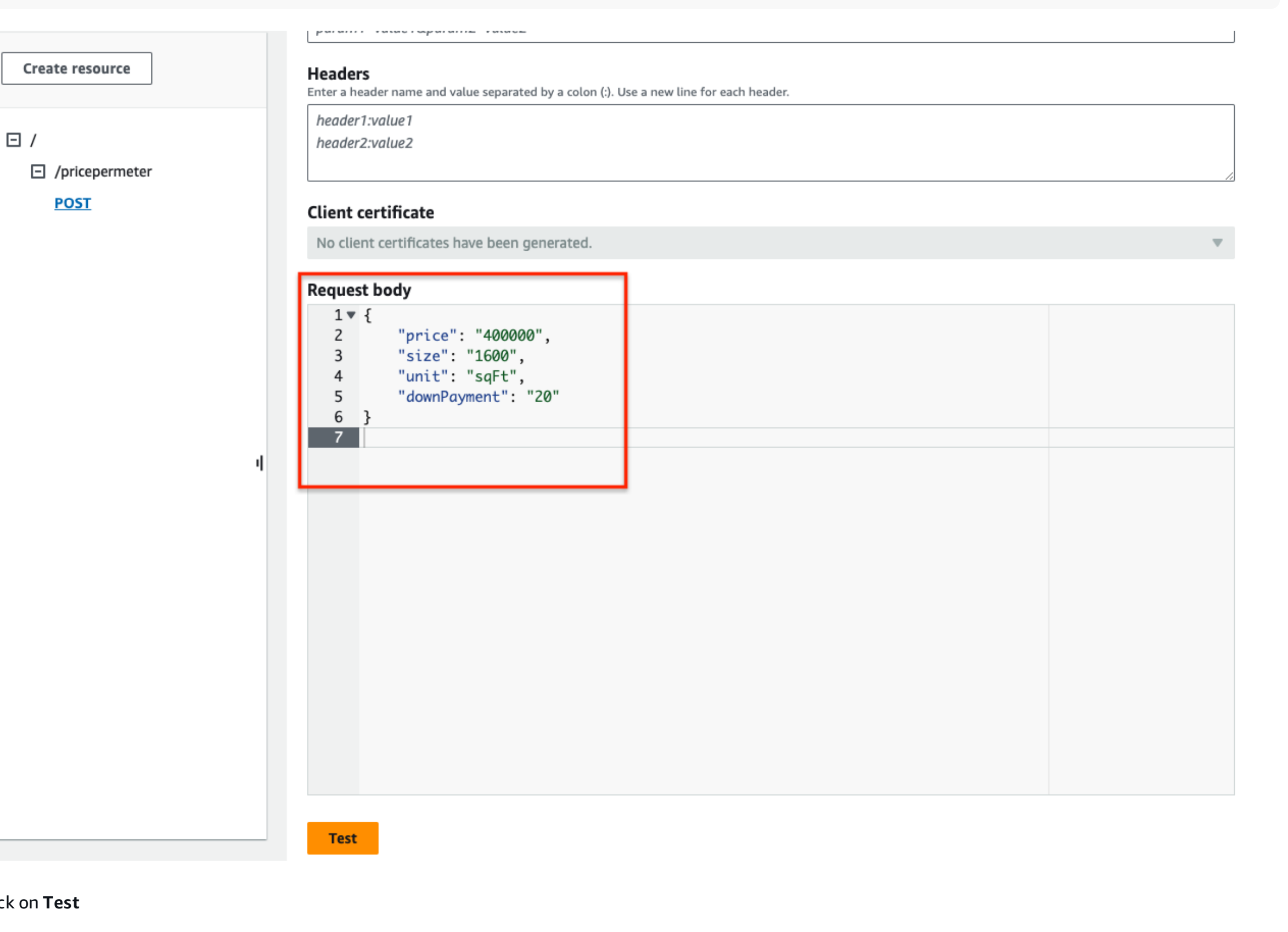


Test the API

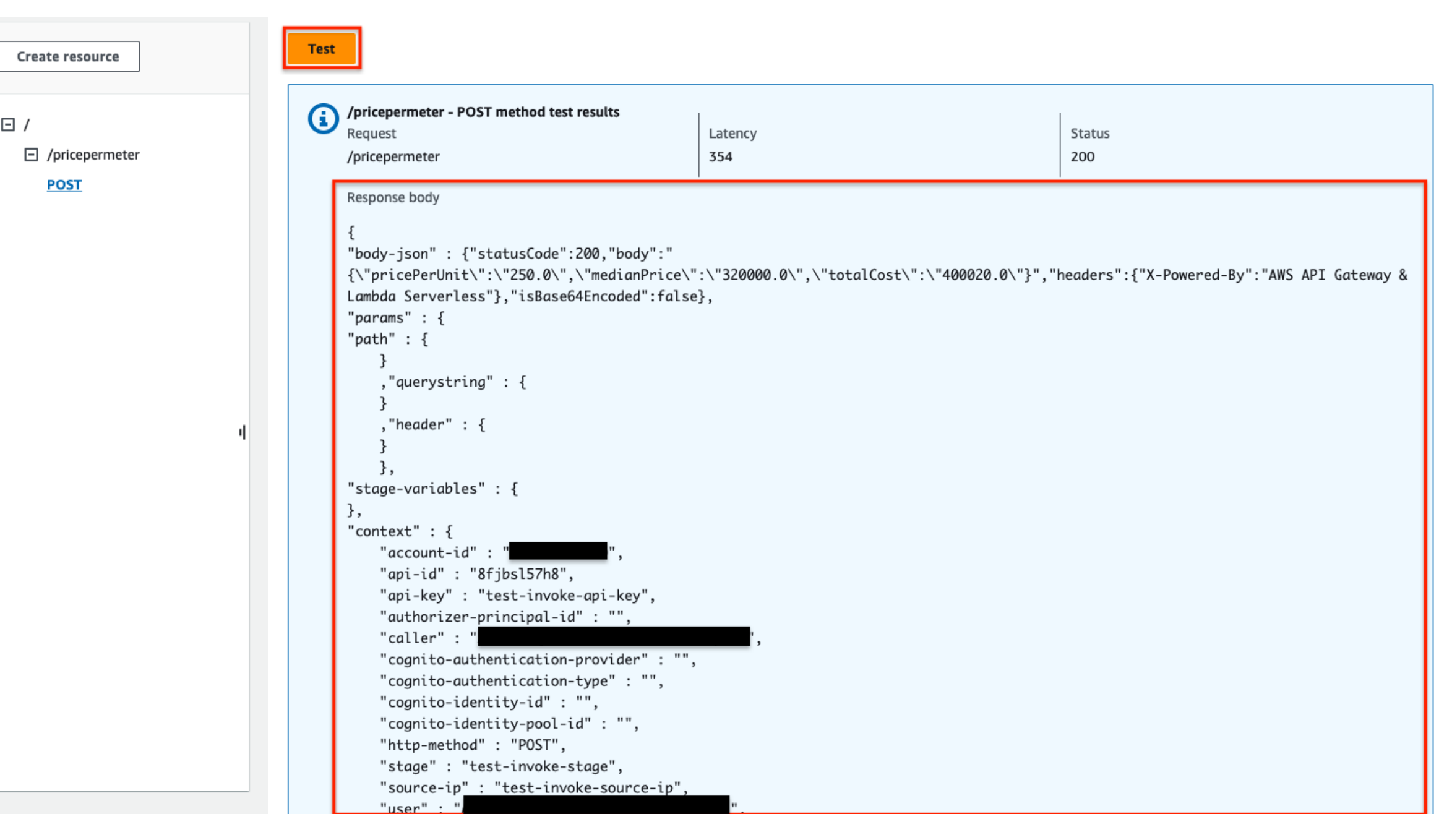
In the response mapping, we have chosen to pass through the response content from lambda without making any further modifications. The method request passthrough is a pre-defined template that maps the response headers, body, parameters and context. The `$context` variable holds all the contextual information of your API call.

1. Go back to your *module-2-my-first-api* API and click on the /pricepermeter `POST` method to test your API.
2. Click on the **Test** option to provide a sample request message.
3. Copy and Paste the following JSON Sample in the *Request Body* section

```
{
  "price": "400000",
  "size": "1600",
  "unit": "sqFt",
  "downPayment": "20"
}
```



4. Click on **Test**



> **Notice**
> The `downPayment` amount was added to the price to calculate the `totalCost` output. The response also contains the header and context attributes.