Create your first API with IaC

Use AWS SAM to create your first API Gateway REST API

The AWS Serverless Application Model (AWS SAM) is a toolkit that improves the developer experience of building and running serverless applications on AWS. AWS SAM consists of two primary parts:

- **AWS SAM template specification** – Open-source framework that you can use to define your serverless application infrastructure on AWS.
- **AWS SAM command line interface (AWS SAM CLI)** – A command line tool that you can use with AWS SAM templates and supported third-party integrations to build and run your serverless applications.

First, let's review code snippets for each file in this project. You'll copy/paste these snippets into the appropriate files. Then you will use SAM CLI to build and deploy the project. Last you will test the deployment.

Review the SAM template

Review the code snippet below. This code belongs in your SAM template file `template.yaml`. Notice that this yaml file defines a resource `AWS::Serverless::Api`. Notice references to the additional definition file: `openapi.yaml`.

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

```
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3  Description: >
4    Sample SAM Template for my first API with IaC
5
6  Globals:
7      # Enable Logs
8      Api:
9          MethodSettings:
10             - ResourcePath: "/*"
11               HttpMethod: "*"
12               DataTraceEnabled: True
13               LoggingLevel: INFO
14               MetricsEnabled: True
15     Function:
16         Timeout: 3
17         Runtime: nodejs18.x
18
19 Resources:
20
21     # REST API
22     FirstAPI:
23         Type: AWS::Serverless::Api
24         Properties:
25             StageName: dev
26             OpenApiVersion: 3.0.3
27             DefinitionBody: # an OpenApi definition
28                 "Fn::Transform":
29                     Name: "AWS::Include"
30                     Parameters:
31                         Location: "openapi.yaml"
32             EndpointConfiguration:
33                 Type: REGIONAL
34
35
36     # Lambda functions
37     CostCalculator:
38         Type: AWS::Serverless::Function
39         Properties:
40             CodeUri: handlers/
41             Handler: cost-calculator.CostCalculator
42
43     # Execution Role
44     LambdaExecutionRole:
45         Type: AWS::IAM::Role
46         Properties:
47             AssumeRolePolicyDocument:
48                 Version: "2012-10-17"
49                 Statement:
50                   - Effect: Allow
51                     Principal:
52                       Service:
53                         - apigateway.amazonaws.com
54                     Action:
55                       - 'sts:AssumeRole'
56             Policies:
57                 - PolicyName: AllowLambdaExec
58                   PolicyDocument:
59                     Version: "2012-10-17"
60                     Statement:
61                       - Effect: Allow
62                         Action: 'lambda:InvokeFunction'
63                         Resource: !GetAtt CostCalculator.Arn
64 Outputs:
65   FirstAPI:
66     Description: "API Gateway endpoint URL to call my first api"
67     Value: !Sub "https://${FirstAPI}.execute-api.${AWS::Region}.amazonaws.com/dev/"
```

On **lines 1 to 4** we have the initial declaration of the template, here we are using the AWS::Serverless transform, which is a macro hosted by CloudFormation taht takes an entire template written in the AWS SAM syntax and transforms and expands it into a compliant CloudFormation template.

On **lines 6 to 17** we have the Globals section, here we can declare common configurations to all template resources. Here we're enabling logs and tracing for all resources and methods of our first API and also the timeout and the default runtime for all lambda functions of the module.

On **lines 19 to 63** we're defining the initial resources for the first API:

- FirstAPI resource: Creates a collection of Amazon API Gateway resources and methods that can be invoked through HTTPS endpoints. Here we're defining our first API and referencing the `openapi.yaml` file as the OpenAPI specification in the `DefinitionBody` attribute.
- CostCalculator resource: Creates an AWS Lambda function, this is the backend service that will be invoked when our method is called.
- LambdaExecutionRole resource: Creates a new role for your AWS account. In order to Amazon API Gateway have permission to invoke the Lambda, we have to create a role with `lambda:InvokeFunction` permission and associate with the integration in our OpenAPI file.

On **lines 64 to 67** we're defining the Outputs for the stack, which declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console.

> 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.

3. This code belongs in your OpenAPI 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"
```

On **lines 1 to 5** we have the metadata section. Here we're defining the OpenAPI version and other infos about our first API.

On **lines 7 to 28** we have the paths section, which defines individual endpoints (paths) in your API, and the HTTP methods (operations) supported by these endpoints. Here we're defining our path `/pricepermeter` and the method **POST**.

On **lines 18 to 28** we have the x-amazon-apigateway-integration object, which specifies details of the backend integration used for this method. Here we're defining the credentials to use the IAM Role created by the SAM template and we're setting the `uri` to integrate with the Lambda function `CostCalculator` (**line 23**).

> **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 **line 28** we have `type: "aws"`, responsible for configuring the integration as **Lambda Custom** type.

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-2/first-api`, where the `template.yaml` file for the sample application is located:

```
1  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-2-first-api`
   - **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`
   - **Save arguments to configuration file**: `Y`
   - **SAM configuration file** and **SAM configuration environment** leave blank

```
Setting default arguments for 'sam deploy'
=========================================
Stack Name [sam-app]: module-2-first-api
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
Save arguments to configuration file [Y/n]: Y
SAM configuration file [samconfig.toml]:
SAM configuration environment [default]:
```

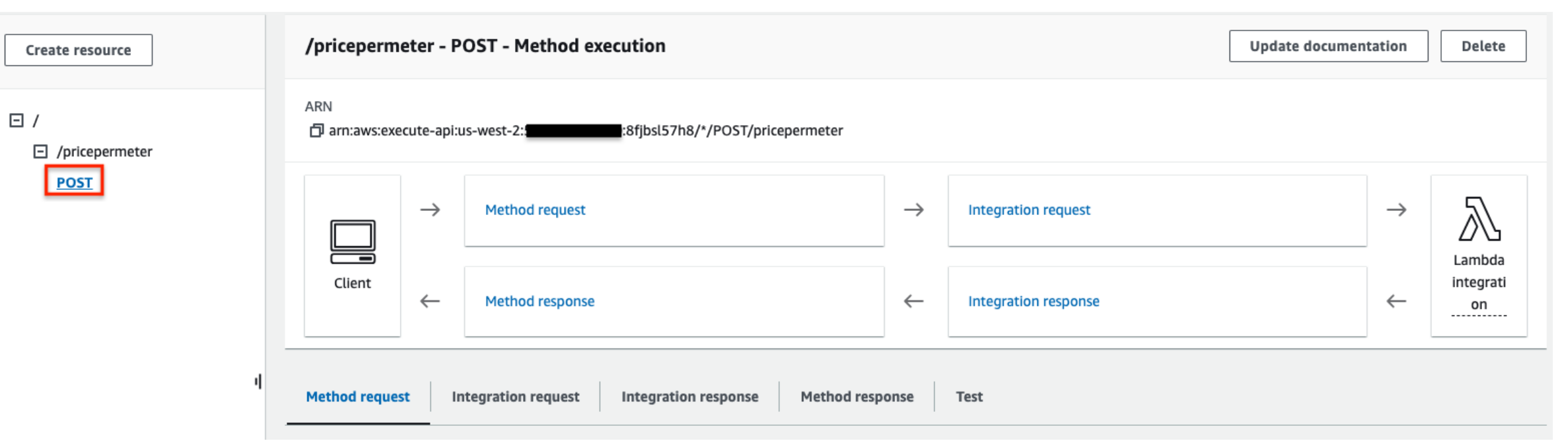
2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.

```
CloudFormation stack changeset
---------------------------------------------------------------------------------------------------------------
Operation                 LogicalResourceId                  ResourceType                      Replacement
---------------------------------------------------------------------------------------------------------------
+ Add                     FirstAPIDeployment71beb09307       AWS::ApiGateway::Deployment       N/A
+ Add                     FirstAPIdevStage                   AWS::ApiGateway::Stage            N/A
+ Add                     FirstAPI                           AWS::ApiGateway::RestApi          N/A
+ Add                     LambdaExecutionRole                AWS::IAM::Role                    N/A
+ Add                     MediaPriceCalculatorRole           AWS::IAM::Role                    N/A
+ Add                     MediaPriceCalculator               AWS::Lambda::Function             N/A
---------------------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-east-1:907236258605:changeSet/samcli-deploy1680654916/4199bb8c-9341-46f4-963e-7580c27318a5


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

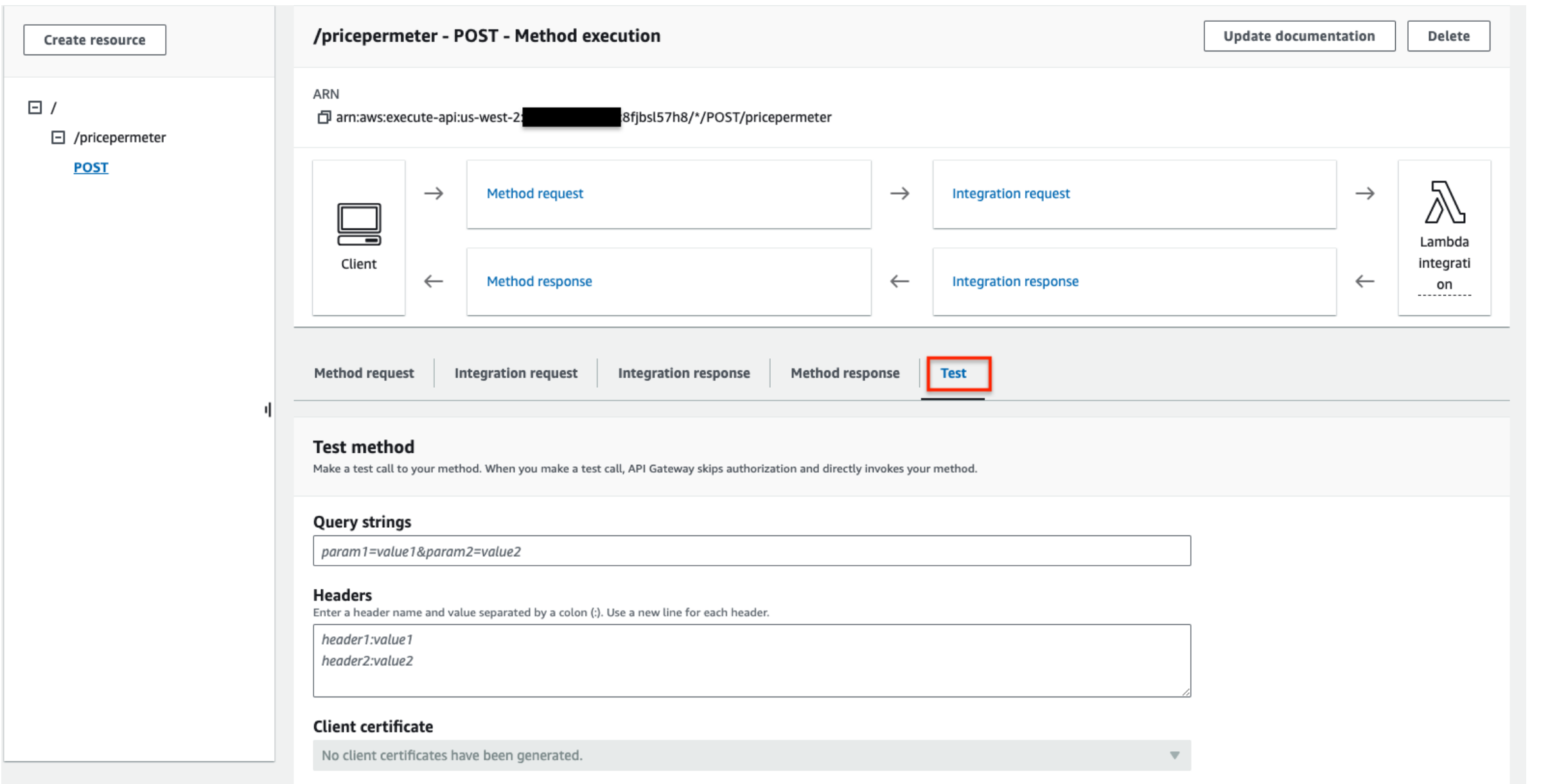
3. After it was finished, a new stack `module-2-first-api` will be successfully created. Now let's explore the resources created in the console and test the API.

Test your first API with IaC

1. Open the Amazon API Gateway console, select the right region if necessary and sign in.
2. Choose your REST API named, `module-2-my-first-api`.
3. In Resources click in /pricepermeter `POST`

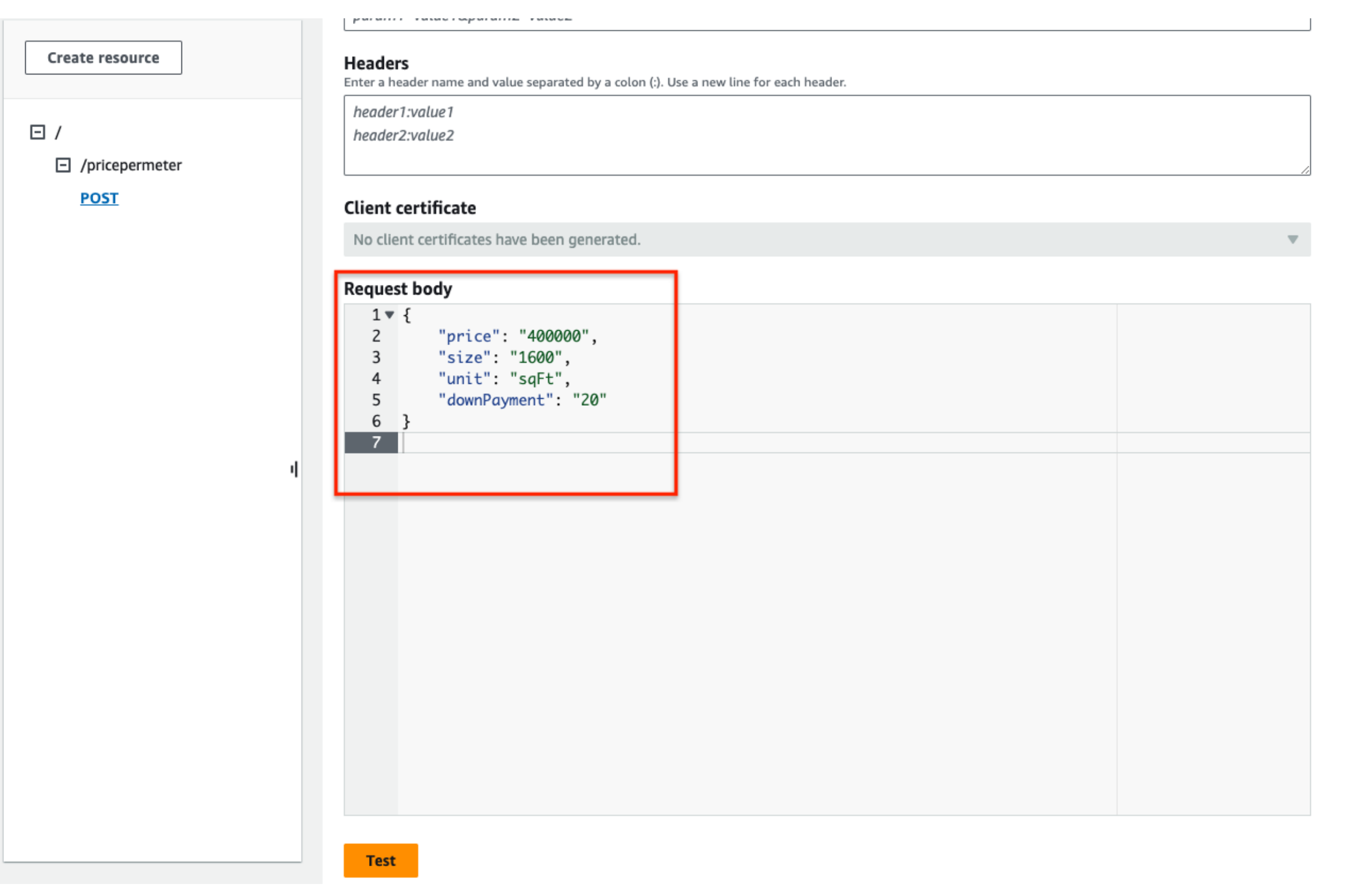


4. Click on the **Test** option to provide a sample request message.



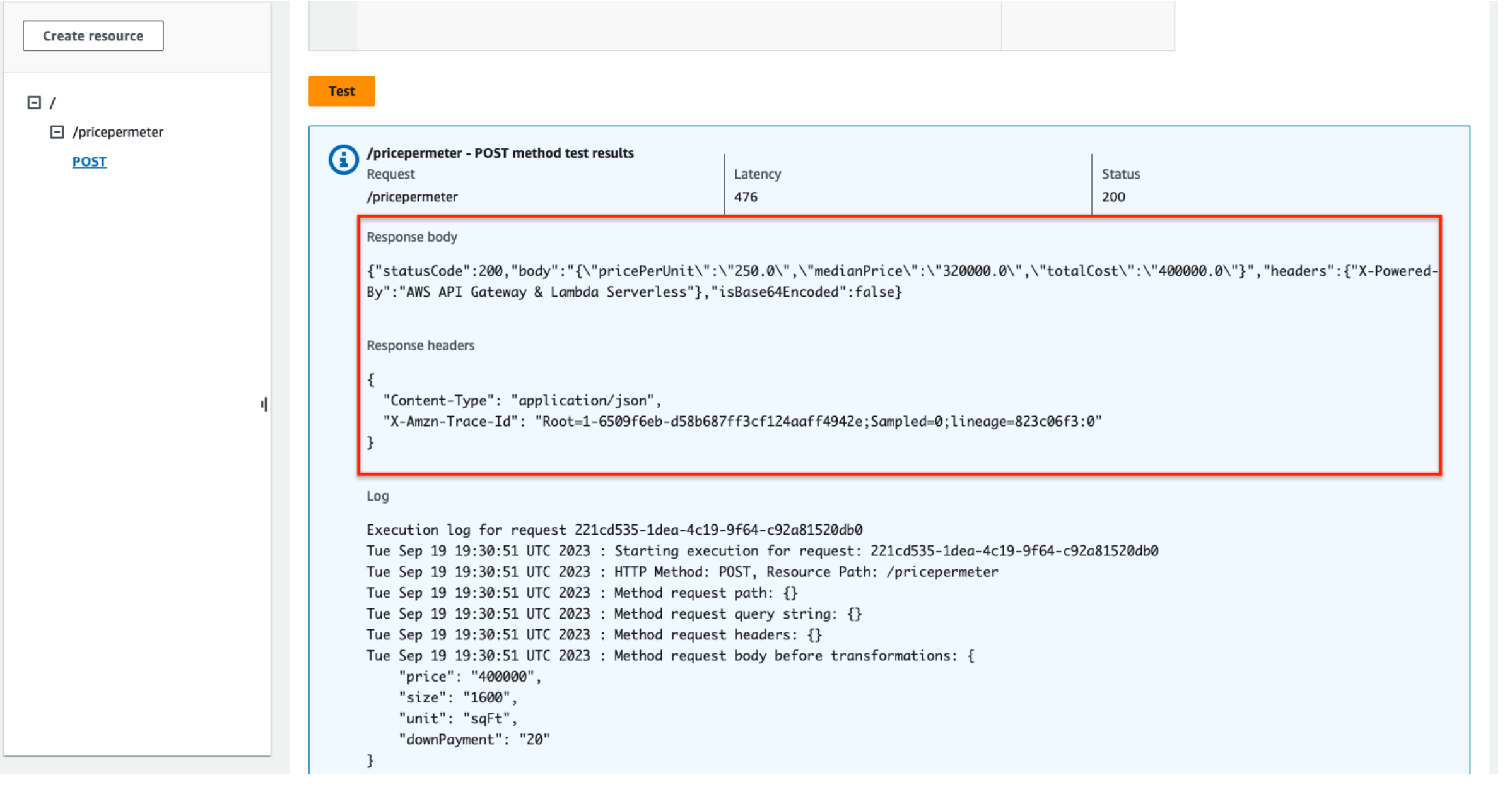
5. Copy and Paste the following JSON Sample in the *Request Body* section

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



6. Click **Test**

   - Please note the Response Body with a status code of **200**. The body of the response contains the price per unit and the total cost. The header of the response along with the message body is composed by the lambda function. The testing page also shows a complete request/response log.



   - Notice that the lambda service returned the total cost, but did not account for the down payment. In the next section, we will transform the message to pass the down payment amount to the lambda function.