

FaaS – Development of a serverless web service with AWS Lambda

Objective: Get practical experience in development and deployment of a serverless functions on Cloud FaaS with AWS Lambda.

Tasks:

1. Create a Calculator Lambda function
2. Configure permissions and trust relations
3. Test the Calculator Lambda function
4. Create the Calculator Web API
5. Create HTTP methods to call the Lambda Function
6. Deploy and testing the API
7. Un-deploy the API

Lab environment:

- SoapUI tool

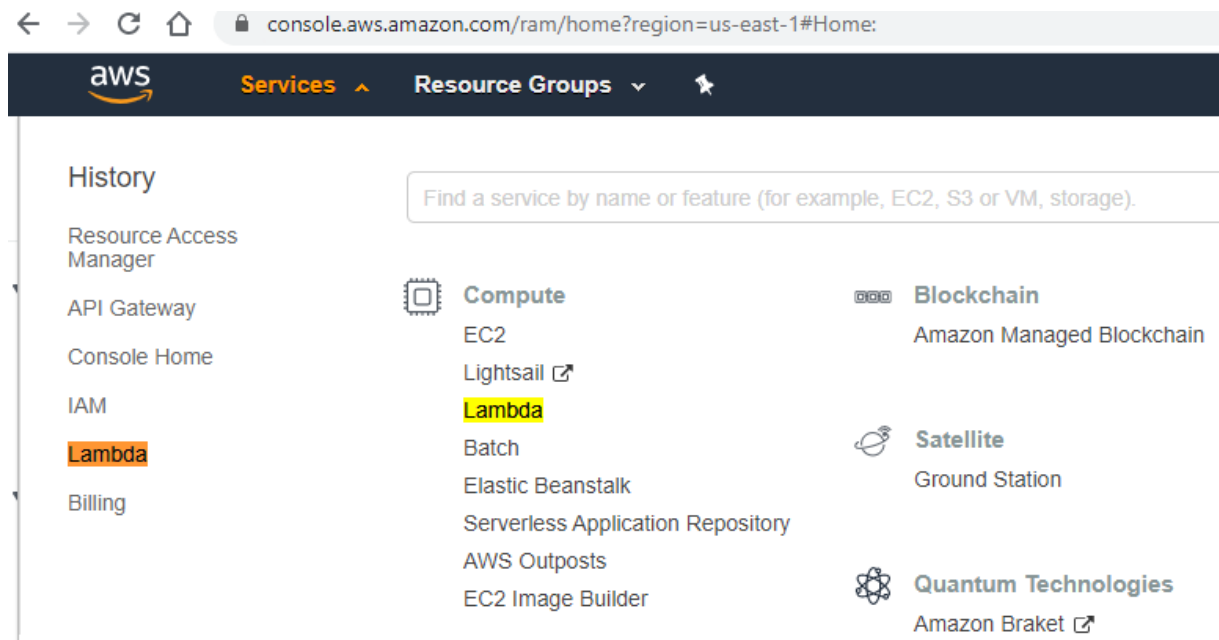
Contents

Step 0: Preparation and Logging to AWS management console	3
Step 1: Creating a Calculator Lambda function	4
Step 2: Configuring permissions and trust relations	7
Step 3: Testing the Calculator Lambda function	12
Step 4: Creating a Web API for the Calculator Lambda function	14
Step 5: Integration – Creating HTTP methods to call the Lambda Function	15
Create a GET method with query parameters to call the Lambda function	15
Create a POST method with a JSON payload to call the Lambda function	21
Create a GET method with path parameters to call the Lambda function	21
Step 6: Deploying and testing the API	22
Step 7: Un-deploying the API	23

Step 0: Preparation and Logging to AWS management console

You'll create a Lambda function using the AWS console.

1. Sign into your AWS account and open AWS Console: <https://console.aws.amazon.com/>
2. Open AWS Lambda Console



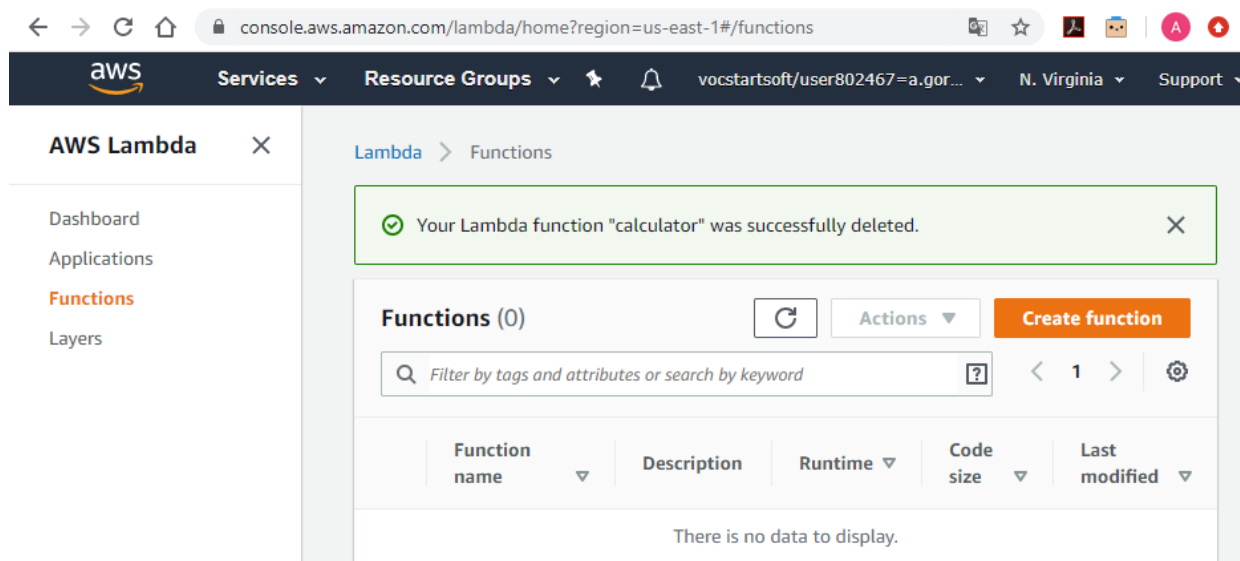
Step 1: Creating a Calculator Lambda function

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code.

The purpose of Lambda, as compared to AWS EC2, is to simplify building smaller, on-demand applications that are responsive to events and new information. AWS targets starting a Lambda instance within milliseconds of an event. AWS Lambda officially supports Node.js, Python, Java, Go,[2] Ruby,[3] and C# (through .NET Core).

Unlike Amazon EC2, which is priced by the hour but metered by the second, AWS Lambda is metered in increments of 100 milliseconds. Usage amounts below a documented threshold fall within the AWS Lambda free tier - which does not expire 12 months after account signup, unlike the free tier for other AWS services.

1. In the AWS Lambda console, choose **Create function**



2. Choose **Author from Scratch**.
3. For **Name**, enter **Calculator**.
4. Set the **Runtime** to Node.js 12.x (use 12.x version).
5. Set **Architecture** to x86_64.
6. Choose **Create function**.

Basic information

Function name

Enter a name that describes the purpose of your function.

Calculator

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor

Node.js 12.x

Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

☒ x86_64

☐ arm64

7. Paste ([use MS Word to copy/paste](#)) the following Lambda function and paste it into the code editor in the Lambda console.

```
console.log('Loading the Calc function');
exports.handler = function(event, context, callback) {
  console.log('Received event:', JSON.stringify(event, null, 2));
  if (event.a === undefined || event.b === undefined || event.op === undefined) {
    callback("400 Invalid Input");
  }
  var res = {};
  res.a = Number(event.a);
  res.b = Number(event.b);
  res.op = event.op;


  if (isNaN(event.a) || isNaN(event.b)) {
    callback("400 Invalid Operand");
  }
  switch(event.op)
  {
    case "+":
    case "add":
      res.c = res.a + res.b;
      break;
    case "-":
    case "sub":
      res.c = res.a - res.b;
      break;
    case "*":
    case "mul":
      res.c = res.a * res.b;
      break;
    case "/":
    case "div":
      res.c = res.b===0 ? NaN : Number(event.a) / Number(event.b);
      break;
    default:
      callback("400 Invalid Operator");
      break;
  }
  callback(null, res);
}
```


Lambda > Functions > Calculator

Calculator

Throttle Copy ARN Actions

▼ Function overview Info

 Calculator

 Layers (0)

+ Add trigger + Add destination

Description
-

Last modified
4 minutes ago

Function ARN
[arn:aws:lambda:us-east-1:604421567424:function:Calculator](#)

Function URL Info
-

Code Test Monitor Configuration Aliases Versions

Code source Info Upload from

File Edit Find View Go Tools Window Test Deploy

Go to Anything (Ctrl-P)

Environment

Calculator - / index.js

```
1 console.log('Loading the Calc function');
2 exports.handler = function(event, context, callback) {
3   console.log('Received event:', JSON.stringify(event, null, 2));
4   if (event.a === undefined || event.b === undefined || event.op === undefined) {
5     callback("400 Invalid Input");
6   }
7   var res = {};
8   res.a = Number(event.a);
9   res.b = Number(event.b);
10  res.op = event.op;
11
12  if (isNaN(event.a) || isNaN(event.b)) {
13    callback("400 Invalid Operand");
14  }
15  switch(event.op)
16  {
```

8. Notice the ARN (Amazon Resource Locator) of the created function on the top of the page
9. Choose **Deploy**.

Step 3: Testing the Calculator Lambda function

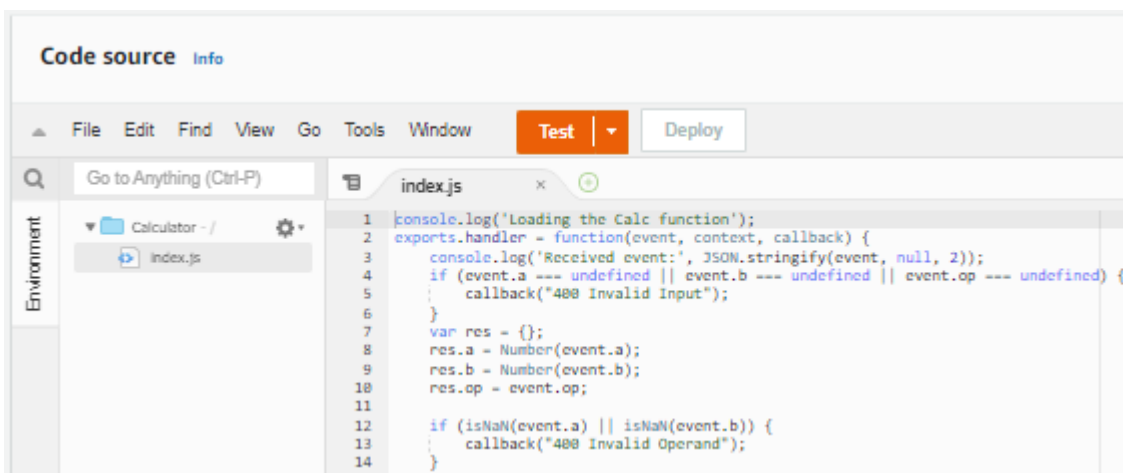
Created function requires two operands (a and b) and an operator (op) from the event input parameter. The input is a JSON object of the following format:

```
{
  "a": "Number" | "String",
  "b": "Number" | "String",
  "op": "String"
}
```

This function returns the calculated result (c) and the input. For an invalid input, the function returns either the null value or the "Invalid op" string as the result. The output is of the following JSON format:

```
{
  "a": "Number",
  "b": "Number",
  "op": "String",
  "c": "Number" | "String"
}
```

You should test the function in the Lambda console before integrating it with the API in the next step. Here's how to test your Calculator function in the Lambda console:



1. Click on **Test** and create a new test event; for the test event name enter **calc2plus5**.
2. Replace the test event definition with the following:

```
{
  "a": "2",
  "b": "5",
  "op": "+"
}
```

Configure test event



A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event

☐ Edit saved event

Event name

calc2plus5

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

[Format JSON](#)

```
1 {  
2   "a": "2",  
3   "b": "5",  
4   "op": "+",  
5 }
```

3. Choose **Save**.
4. In the dropdown menu, choose **calc2plus5** and press **Test**.
5. Expand **Execution result: succeeded**. You should see the following:

The screenshot shows the AWS Lambda console interface. At the top, there's a 'Code source' section with 'Info' and 'Test' buttons. Below this is a menu bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', and 'Test' (highlighted). The main area is divided into two panes. The left pane shows the 'Environment' tab with a file explorer view showing 'Calculator - /' and 'index.js'. The right pane shows the 'Execution result' tab, which is expanded to show 'Test Event Name: calc2plus5'. Below this, the 'Response' is displayed as a JSON object: `{ "a": 2, "b": 5, "op": "+", "c": 7 }`. The 'Function Logs' section shows the execution details, including the start and end times, request ID, and the received event: `{ "a": "2", "b": "5", "op": "+" }`. The 'Request ID' is also shown at the bottom.

Step 3: Configuring permissions and trust relations

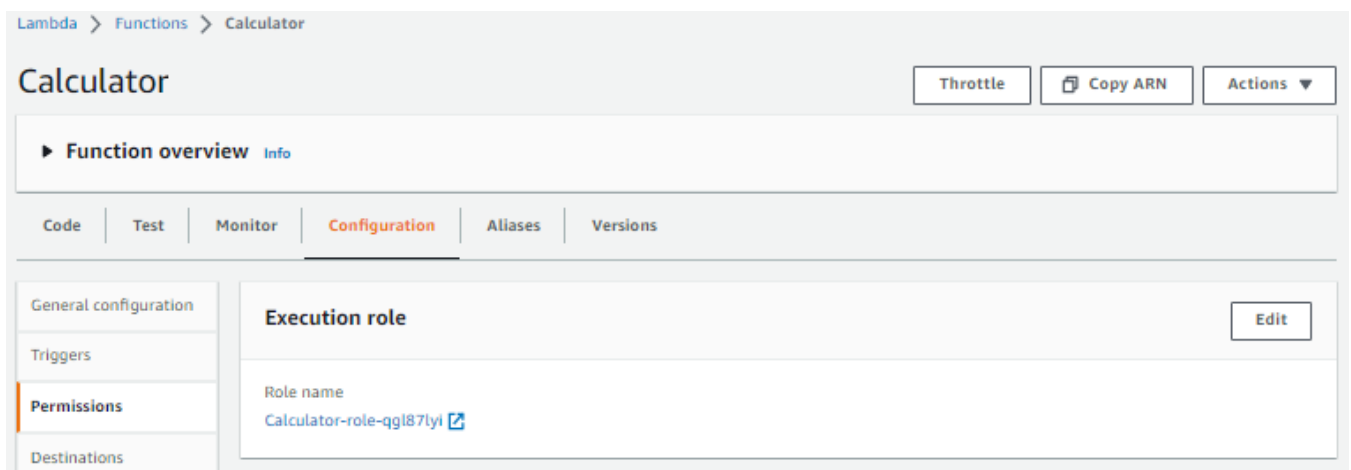
AWS Identity and Access Management (IAM) enables users to manage access to their AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

An **Execution Role** is an IAM role that Lambda has permission to assume when you invoke a function. An AWS Lambda function's **Execution Role** grants it permission to access AWS services and resources. The role is created once you create a function. You can also create an execution role in the IAM console. You can restrict the scope of a user's permissions by specifying resources and conditions in an **IAM policy** which is applied to function's **Execution Role**.

You can restrict the resources that a user can access by specifying the **Amazon Resource Name (ARN)** of a resource, or an ARN pattern that matches multiple resources. For example, you can create and apply a policy which allows the specified user (e.g. with the user account 123456789012) to invoke the specified function (e.g. named my-function) in the specified location (e.g. in US West Oregon Region) by specifying the following ARN: `"arn:aws:lambda:us-west-2:123456789012:function:my-function"`.

When the action doesn't act on a named resource, or when you grant permission to perform the action on all resources, the value of the resource in the policy is a wildcard (*). Read more about AWS Access Management at: <https://docs.aws.amazon.com/IAM/latest/UserGuide/access.html>

1. Proceed to the **Configuration** -> **Permission** tab and choose the role created by default



2. You will be redirected to the IAM console where you can specify role's permissions and trust relations.

Calculator-role-qgl87lyi

[Delete](#)

Summary

[Edit](#)

Creation date
November 23, 2022, 20:59 (UTC)

Last activity
None

ARN
[arn:aws:iam::604421567424:role/service-role/Calculator-role-qgl87lyi](#)

Maximum session duration
1 hour

[Permissions](#) | [Trust relationships](#) | [Tags](#) | [Access Advisor](#) | [Revoke sessions](#)

Permissions policies (1) [Info](#)

You can attach up to 10 managed policies.

[Simulate](#)[Remove](#)[Add permissions](#) ▼

Filter policies by property or policy name and press enter.

< 1 >

<input type="checkbox"/>	Policy name ↗	Type	Description
<input type="checkbox"/>	AWSLambdaBasicExecutionRole-502bdf5b-f3dc-4919-9245-483a72772667	Customer managed	

Summary

Policy ARN [arn:aws:iam::604421567424:policy/service-ro](#)

Description

[Permissions](#) | [Policy usage](#) | [Tags](#) | [Policy versions](#) | [Access Advisor](#)

[Policy summary](#)[{} JSON](#)[Edit policy](#)

Service ▼	Access level	Resource
Allow (1 of 340 services) Show remaining 339		
CloudWatch Logs	Limited: Write	Multiple

3. Replace the default permission policy with the new one.

By default, Lambda creates an execution role with minimal permissions when you create a function in the Lambda console. In this scenario we would like to edit role's policy to simply allow all users to invoke all functions in all locations.

Identity and Access Management (IAM)

- Dashboard
- Access management
 - Groups
 - Users
 - Roles
 - Policies**
 - Identity providers
 - Account settings
- Access reports
 - Access analyzer
 - Archive rules
 - Analyzers
 - Settings
 - Credential report
 - Organization activity
 - Service control policies (SCPs)

Search IAM

Policies > AWSLambdaBasicExecutionRole-2e00c2ce-93fe-4f4c-948a-a9e0bdae5d36

Summary

Delete policy

Policy ARN: `arn:aws:iam::713621717571:policy/service-role/AWSLambdaBasicExecutionRole-2e00c2ce-93fe-4f4c-948a-a9e0bdae5d36`

Description

Permissions | Policy usage | Policy versions | Access Advisor

Policy summary | {} JSON | Edit policy

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "logs:CreateLogGroup",
7       "Resource": "arn:aws:logs:us-east-1:713621717571:*"
8     },
9     {
10      "Effect": "Allow",
11      "Action": [
12        "logs:CreateLogStream",
13        "logs:PutLogEvents"
14      ],
15      "Resource": [
16        "arn:aws:logs:us-east-1:713621717571:log-group:/aws/lambda/calculator:*"
17      ]
18    }
19  ]
20 }

```

a. In the **JSON** tab, replace the existing policy with the following:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "*"
    }
  ]
}

```

Edit AWSLambdaBasicExecutionRole-2e00c2ce-93fe-4f4c-948a-a9e0bdae5d36

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

Visual editor | **JSON** | Import managed policy

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "lambda:InvokeFunction",
7       "Resource": "*"
8     }
9   ]
10 }

```

Character count: 105 of 6,144.

Cancel **Review policy**

b. Choose **Review policy** and **Save** changes and close this tab.

Edit AWS Lambda Basic Execution Role-2e00c2ce-93fe-4f4948a-a9e0bdae5d36

1

2

Review policy

Review this policy before you save your changes.

☒ Save as default

Summary

Filter		
Service	Access level	Resource
Allow (1 of 231 services) Show remaining 230		
Lambda	Limited: Write	All resources

* Required

[Cancel](#)

[Previous](#)

[Save changes](#)

4. In the Roles settings select role's **Trust relationships** tab

[IAM](#) > [Roles](#) > [Calculator-role-qgl87lyi](#)

Calculator-role-qgl87lyi

[Delete](#)

Summary

[Edit](#)

Creation date
November 23, 2022, 20:59 (UTC)

Last activity
None

ARN
[arn:aws:iam::604421567424:role/service-role/Calculator-role-qgl87lyi](#)
Maximum session duration
1 hour

[Permissions](#) | [Trust relationships](#) | [Tags](#) | [Access Advisor](#) | [Revoke sessions](#)

Trusted entities

Entities that can assume this role under specified conditions.

[Edit trust policy](#)

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Effect": "Allow",  
6       "Principal": {  
7         "Service": "lambda.amazonaws.com"  
8       },  
9       "Action": "sts:AssumeRole"  
10    }  
11  ]  
12 }
```

Trust policy is a JSON policy document in which you define the principals that you trust to assume the role. A role trust policy is a required resource-based policy that is attached to a role in IAM, and is one-half of the permissions. The other half is a permissions policy attached to the user in the trusted account that allows that user to switch to, or assume the role. The principals that you can specify in the trust policy include users, roles, accounts, and services.

- Choose the **Trust relationships** tab.
- Choose **Edit trust relationship**.

Identity and Access Management (IAM)

- Dashboard
- Access management
 - Groups
 - Users
 - Roles**
 - Policies
 - Identity providers
 - Account settings
- Access reports
 - Access analyzer
 - Archive rules
 - Analizers
 - Settings
 - Credential report
 - Organization activity
 - Service control policies (SCPs)

Roles > calculator-role-8n2im1h7

Summary

Role ARN	arn:aws:iam::713621717571:role/service-role/calculator-role-8n2im1h7
Role description	Edit
Instance Profile ARNs	Add
Path	/service-role/
Creation time	2020-06-05 16:23 UTC+0100
Last activity	Not accessed in the tracking period
Maximum CLI/API session duration	1 hour Edit

Permissions **Trust relationships** **Tags** **Access Advisor** **Revoke sessions**

You can view the trusted entities that can assume the role and the access conditions for the role. [Show policy document](#)

[Edit trust relationship](#)

Trusted entities

The following trusted entities can assume this role.

Trusted entities

The identity provider(s) `lambda.amazonaws.com`

Conditions

The following conditions define how a user assumes the role.

There are no conditions associated with this role.

- Replace the existing policy with the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "apigateway.amazonaws.com",
          "lambda.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

IAM > Roles > Calculator-role-qgl87lyi > Edit trust policy

Edit trust policy

1 | {
2 | "Version": "2012-10-17",
3 | "Statement": [
4 | {
5 | "Sid": "",
6 | "Effect": "Allow",
7 | "Principal": {
8 | "Service": [
9 | "apigateway.amazonaws.com",
10 | "lambda.amazonaws.com"
11 |]
12 | },
13 | "Action": "sts:AssumeRole"
14 | }
15 |]
16 | }

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

[+ Add new statement](#)

[+ Add new statement](#)

JSON Ln 16, Col 1

Security: 0 Errors: 0 Warnings: 0 Suggestions: 1

Preview external access

[Cancel](#) [Update policy](#)

- Make a note of the role ARN for the role you just created. You'll need it later.

Calculator-role-qgl87lyi

Delete

Summary

Edit


Creation date

November 23, 2022, 20:59 (UTC)

Last activity

None

ARN

 [arn:aws:iam::604421567424:role/service-role/Calculator-role-qgl87lyi](#)

Maximum session duration

1 hour

Permissions

Trust relationships

Tags

Access Advisor

Revoke sessions

6. Make sure you have saved ARNs for the created Lambda function and its IAM role, e.g.:

- **Function ARN:** `arn:aws:lambda:us-east-1: <account_id>:function:Calculator`
- **Calculator-role-qgl87lyi:** `arn:aws:iam::<account_id>:role/service-role/Calculator-role-qgl87lyi`

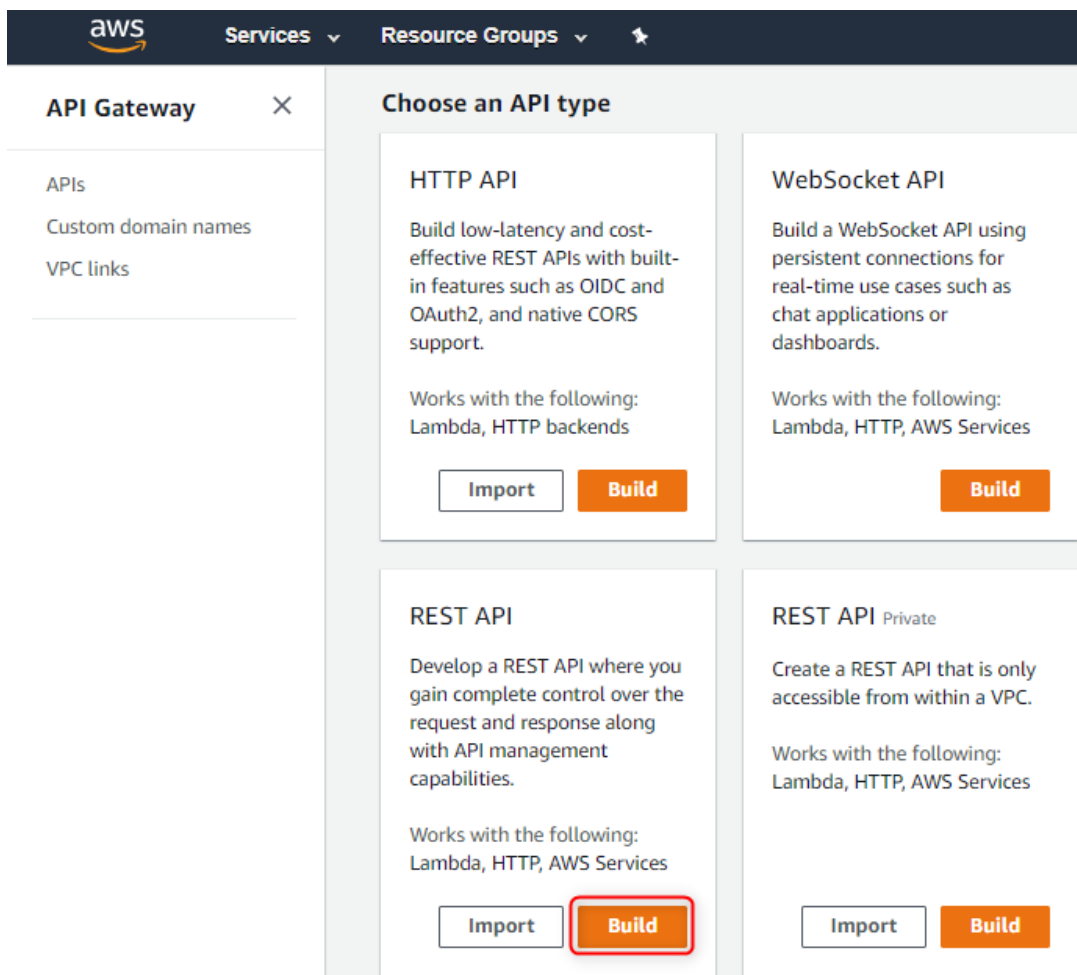
Step 4: Creating a Web API for the Calculator Lambda function

Developers can create a web API with an HTTP endpoint for their Lambda functions by using Amazon API Gateway. Web APIs route HTTP requests to Lambda functions. API Gateway provides tools for creating and documenting web APIs. Developers can also secure access to your API with authentication and authorization controls.


The following procedure shows how to create an API for the `Calculator` Lambda function you just created. In subsequent sections, you'll add resources and methods to it.

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. If this is your first time using API Gateway, you see a page that introduces you to the features of the service. Under **REST API**, choose **Build**. When the **Create Example API** popup appears, choose **OK**.

If this is not your first time using API Gateway, choose **Create API**. Under **REST API**, choose **Build**.




3. Under **Create new API**, choose **New API**.
4. For **API Name**, enter `LambdaCalc`.
5. Leave the **Description** blank, and leave the **Endpoint Type** set to **Regional**.
6. Choose **Create API**.

 Amazon API Gateway

APIs > Create

Show all hints



Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

☒ REST ☐ WebSocket


Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="LambdaCalc"/>
Description	<input type="text"/>
Endpoint Type	<div>Regional </div>

* Required

Create API

Step 5: Integration – Creating HTTP methods to call the Lambda Function

Create a GET method with query parameters to call the Lambda function

By creating a `GET` method that passes query string parameters to the Lambda function, you enable the API to be invoked from a browser. This approach can be useful, especially for APIs that allow open access. To set up the `GET` method with query string parameters do the following.

1. In the API Gateway console, under your `LambdaCalc` API's **Resources**, choose `/`.
2. In the **Actions** drop-down menu, choose **Create Resource**.
3. For **Resource Name**, enter `calculator`.
4. Choose **Create Resource**.

APIs

Custom Domain Names

VPC Links

API: **LambdaCalc**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Resources

Actions

New Child Resource

Use this page to create a new child resource for your resource.

Configure as ☒ proxy resource

Resource Name* calculator

Resource Path* / calculator

You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

Enable API Gateway CORS ☐

* Required

Cancel Create Resource

5. Choose the **/calculator** resource you just created.
6. In the **Actions** drop-down menu, choose **Create Method**.
7. In the method drop-down menu that appears, choose **GET**.
8. Choose the checkmark icon to save your choice.
9. In the **Set up** pane:
 - a. For **Integration type**, choose **AWS Service**.
 - b. For **AWS Region**, choose the region (e.g., **us-east-1**) where you created the Lambda function - check function's ARN if not sure.
 - c. For **AWS Service**, choose **Lambda**.
 - d. Leave **AWS Subdomain** blank, because our Lambda function is not hosted on any AWS subdomain.
 - e. For **HTTP method**, choose **POST** and choose the checkmark icon to save your choice.
Lambda requires that the POST request be used to invoke any Lambda function. This example shows that the HTTP method in a frontend method request can be different from the integration request in the backend.
 - f. Choose Use path override for **Action Type**. This option allows us to specify the ARN of the invoke action to execute our Calculator function.
 - g. Enter **/2015-03-31/functions/arn:aws:lambda:region:account-id:function:Calculator/invocations** in **Path override**, where **region** is the region where you created your Lambda function and **account-id** is the account number for the AWS account (you can notice region:account-id from the Function's ARN at a time it was created – see Step 1).
 - h. For **Execution role**, enter the role ARN for the **IAM role** you created earlier.
 - i. Leave **Content Handling** set to **Passthrough**, because this method will not deal with any binary data.
 - j. Leave **Use default timeout** checked.

APIs > LambdaCalc (7ruihwj1s1) > Resources > /calculator (xwo9jn) > GET

Resources Actions ▾ /calculator - GET - Setup

Choose the integration point for your new method.

Integration type ☐ Lambda Function ⓘ ☐ HTTP ⓘ ☐ Mock ⓘ ☒ AWS Service ⓘ ☐ VPC Link ⓘ

AWS Region

AWS Service

AWS Subdomain

HTTP method

Action Type ☐ Use action name ☒ Use path override

Path override (optional)

Execution role ⓘ

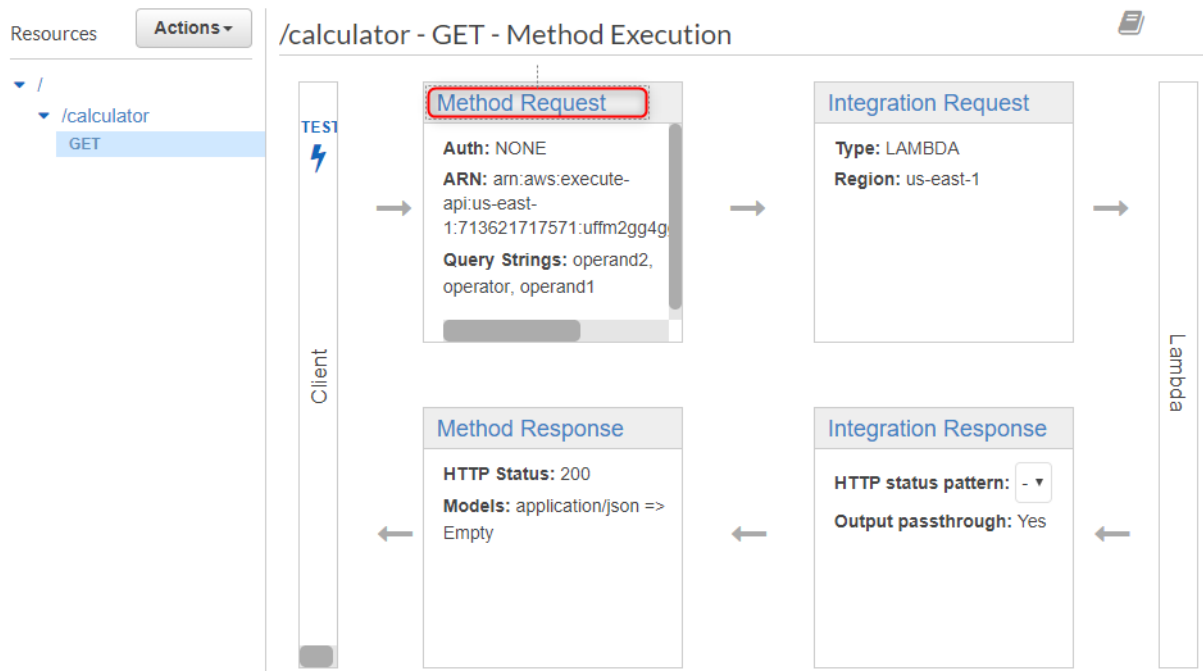
Content Handling ⓘ

Use Default Timeout ☒ ⓘ

k. Choose **Save**.

10. Choose **Method Request**.

Now you will set up query parameters for the **GET** method on **/calculator** so it can receive input on behalf of the backend Lambda function.



a. Choose the pencil icon next to **Request Validator** and choose **Validate query string parameters and headers** from the drop-down menu. This setting will cause an error message to return to

state the required parameters are missing if the client does not specify them. You will not get charged for the call to the backend.

- b. Choose the checkmark icon to save your changes.
- c. Expand the **URL Query String Parameters** section.
- d. Choose **Add query string**.
- e. For **Name**, type **operand1**.
- f. Choose the checkmark icon to save the parameter.
- g. Repeat the previous steps to create parameters named **operand2** and **operator**.
- h. Check the **Required** option for each parameter to ensure that they are validated.

Resources Actions ▾ **Method Execution** /calculator - GET - Method Request

Provide information method's authorization settings and the parameters it can receive.

Settings

Authorization NONE ⓘ

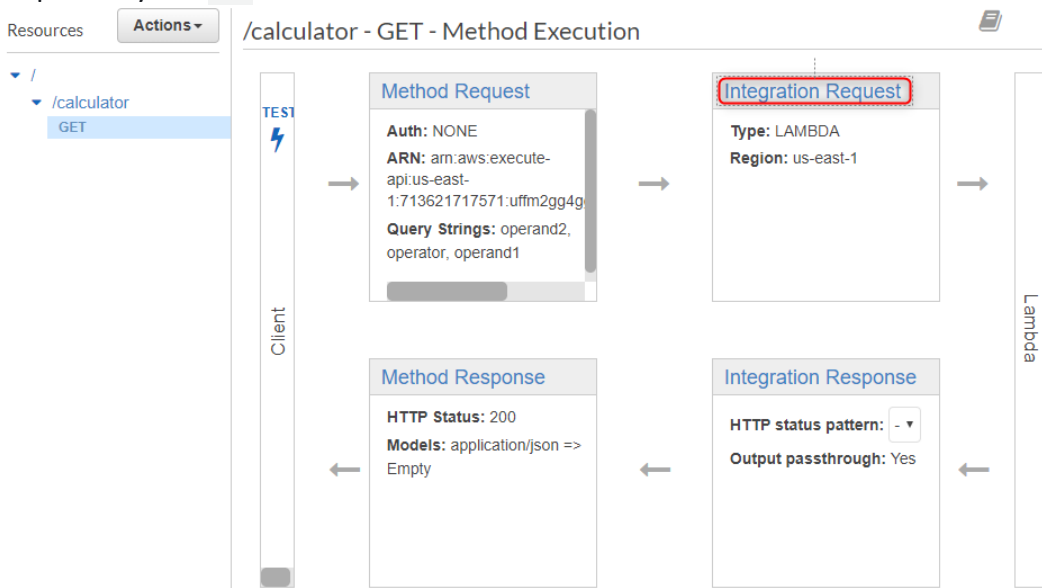
Request Validator Validate query string parameters and headers ⓘ

API Key Required false ⓘ

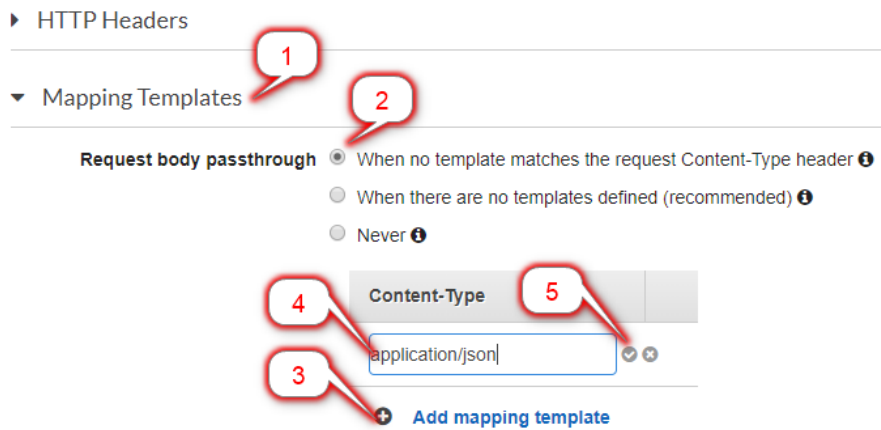
URL Query String Parameters

Name	Required	Caching	
operand1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
operand2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
operator	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
+ Add query string			

11. Choose **Method Execution** at the top of the page and then choose **Integration Request** to set up the mapping template to translate the client-supplied query strings to the integration request payload as required by the Calc function.



- Expand the **Mapping Templates** section.
- Choose **When no template matches the request Content-Type header** for **Request body passthrough**.
- Under **Content-Type**, choose **Add mapping template**.
- Type **application/json** and choose the checkmark icon to open the template editor.



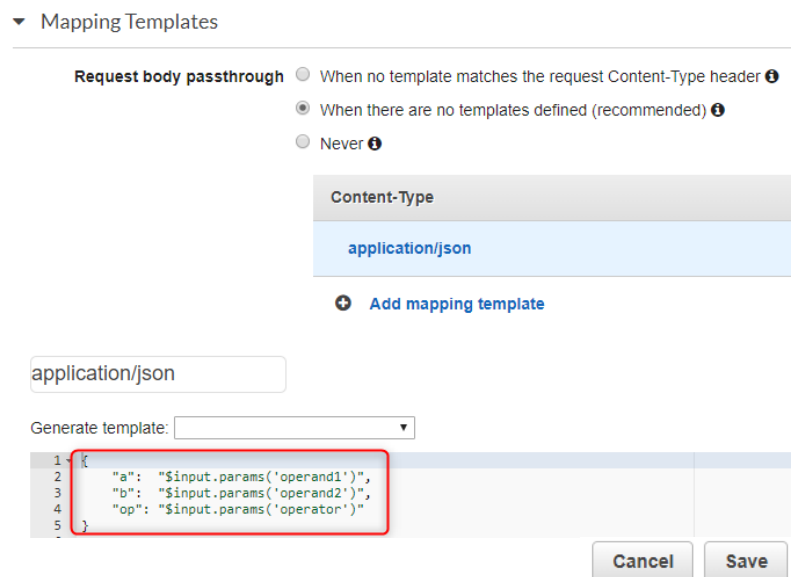
- Choose **Yes, secure this integration** to proceed.

⚠ Your current passthrough behavior will pass all request payloads directly to the endpoint without transformation, unless there is a match for the incoming Content-Type. Do you want to secure this integration to only allow requests that match one of your defined Content-Types?

[No, use current settings](#) [Yes, secure this integration](#)

- Copy the following mapping script into the mapping template editor:

```
{
  "a": "$input.params('operand1')",
  "b": "$input.params('operand2')",
  "op": "$input.params('operator')"
}
```

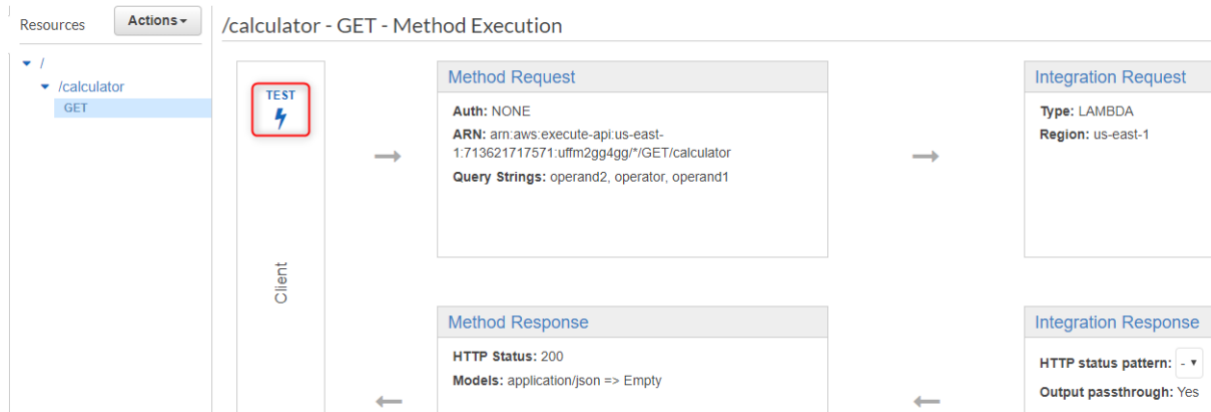


*This template maps the three query string parameters declared in **Method Request** into designated property values of the JSON object as the input to the backend Lambda function. The transformed JSON object will be included as the integration request payload.*

- Choose **Save**.

12. Choose **Method Execution**.

13. You can now test your GET method to verify that it has been properly set up to invoke the Lambda function.



- For **Query Strings**, type **operand1=2&operand2=3&operator=+**
- Choose **Test**.

The screenshot shows the 'Test' method configuration in the API Gateway console. The left pane shows the resource hierarchy: '/' > '/calculator' > 'GET'. The main area is titled '/calculator - GET - Method Test'. It contains the following sections:

- Path**: No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.
- Query Strings**: **{calculator}** operand1=2&operand2=3&operator=+
- Headers**: **{calculator}** Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.
- Stage Variables**: No [stage variables](#) exist for this method.
- Request Body**: Request Body is not supported for GET methods.

A blue 'Test' button is located at the bottom right.

The results should look similar to this:

Resources Actions ▾

Method Execution /calculator - GET - Method Test

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax **(myPathParam)** in a resource path.

Query Strings

(calculator)

operand1=2&operand2=3&operator=+

Headers

(calculator)

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept: application/json.

Stage Variables

No [stage variables](#) exist for this method.

Request Body

Request Body is not supported for GET methods.

Test

Request: /calculator?operand1=2&operand2=3&operator=+

Status: 200

Latency: 526 ms

Response Body

```
{
  "a": 2,
  "b": 3,
  "op": "+",
  "c": 5
}
```

Response Headers

```
{ "X-Amzn-Trace-Id": "Root=1-5eda3ad7-4a919b61d86d92743e224548;n/json" }
```

Logs

```
Execution log for request 6195446e-2686-4b4c-a0c9-c2920cfec2f
Fri Jun 05 12:30:15 UTC 2020 : Starting execution for request 8
Fri Jun 05 12:30:15 UTC 2020 : HTTP Method: GET, Resource Pat
Fri Jun 05 12:30:15 UTC 2020 : Method request path: {}
Fri Jun 05 12:30:15 UTC 2020 : Method request query string: {
Fri Jun 05 12:30:15 UTC 2020 : Method request headers: {}
Fri Jun 05 12:30:15 UTC 2020 : Method request body before tra
Fri Jun 05 12:30:15 UTC 2020 : Request validation succeeded f
```

Create the POST method with a JSON payload to call the Lambda function

After you complete the next step (Step 6), go back here and create the POST method. Consider this as an optional self-task. How is the POST method different from the GET method discussed above?

Refer to the guideline for more technical details:

<https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-lambda.html#api-as-lambda-proxy-expose-post-method-with-json-body-to-call-lambda-function>

Create a GET method with path parameters to call the Lambda function

After you complete the next step (Step 6), go back here and create the GET method with path parameters. Consider this as an optional self-task. How is the GET method with path parameters different from the GET method with query parameters discussed above?

Refer to the guideline for more technical details:

<https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-lambda.html#api-as-lambda-proxy-expose-get-method-with-path-parameters-to-call-lambda-function>

Step 6: Deploying and testing the API

At this point the API can be called only via **Test Invoke** in the API Gateway console. To make it available to clients, you'll need to deploy it as follows:

1. Choose **Deploy API** from the **Actions** dropdown menu.

APIs > LambdaCalc (7ruihwj1s1) > Resources > /calculator (xwo9jn) > GET

Resources Actions /calculator - GET - Method Execution

METHOD ACTIONS

- Edit Method Documentation
- Delete Method

RESOURCE ACTIONS

- Create Method
- Create Resource
- Enable CORS
- Edit Resource Documentation
- Delete Resource

API ACTIONS

- Deploy API
- Import API
- Edit API Documentation
- Delete API

Method Request

Auth: NONE

ARN: am:aws:execute-api:us-east-1:604421567424:7ruihwj1s1/*/*/GET/calculator

Query Strings: operand1, operand2, operator

Method Response

HTTP Status: 200

Models: application/json => Empty

2. Choose **[New Stage]** from the **Deployment Stage** dropdown menu.
3. For **Stage Name**, enter **test**.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage: [New Stage]

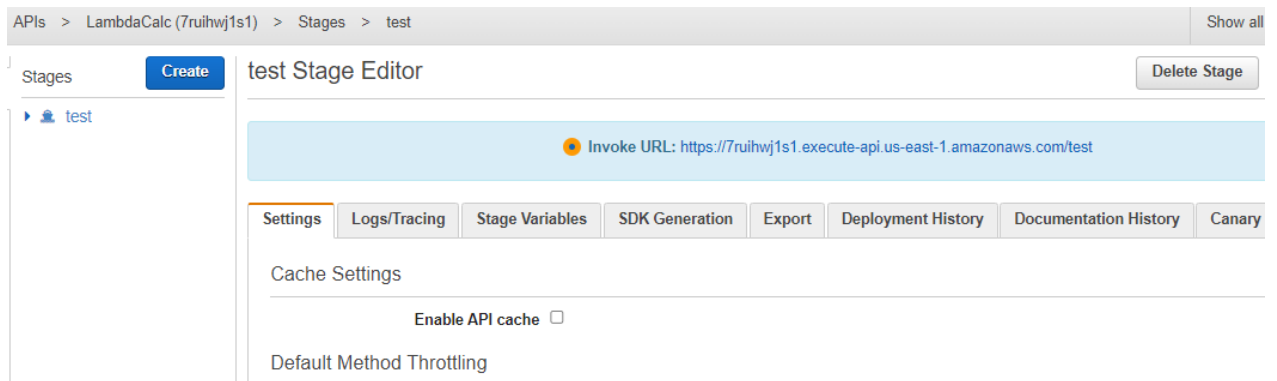
Stage name*: test

Stage description:

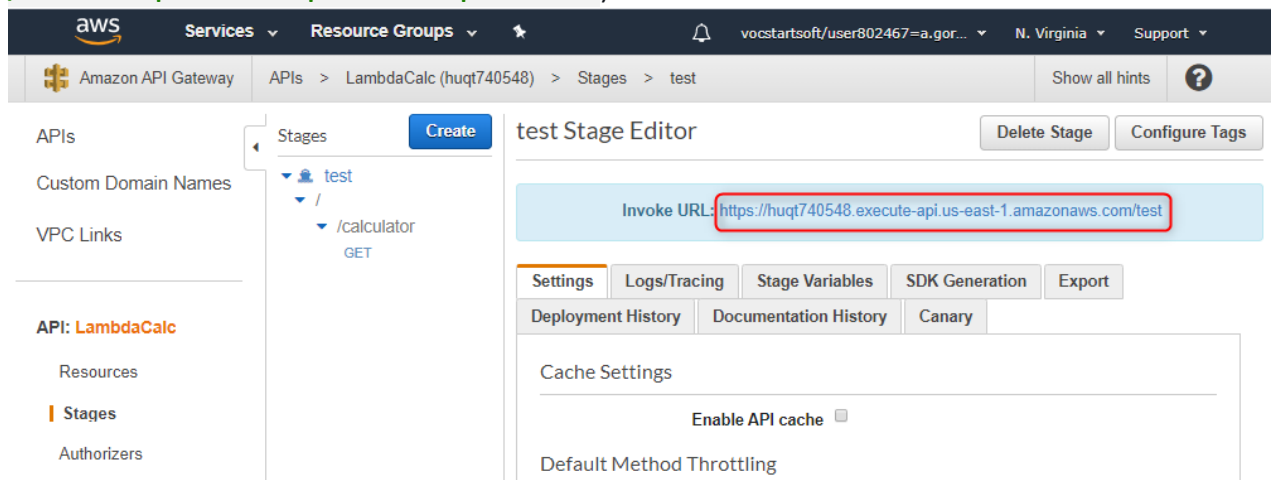
Deployment description:

Cancel Deploy

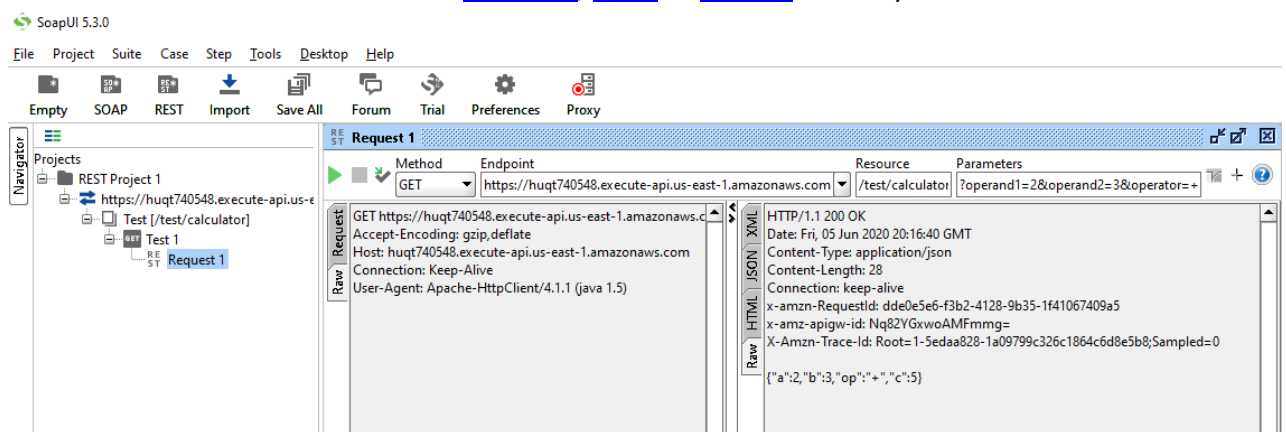
4. Choose **Deploy**.
5. Note the **Invoke URL** at the top of the console window.



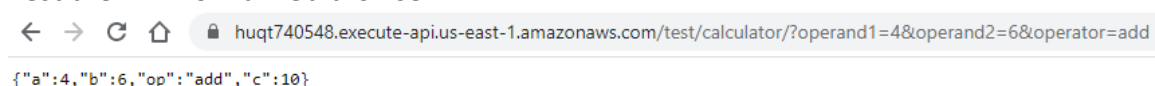
6. You need to add API name and parameter at the end of listed URL (e.g. [/calculator?operand1=2&operand2=3&operator=add](https://huqt740548.execute-api.us-east-1.amazonaws.com/test/calculator?operand1=2&operand2=3&operator=add)) to invoke API



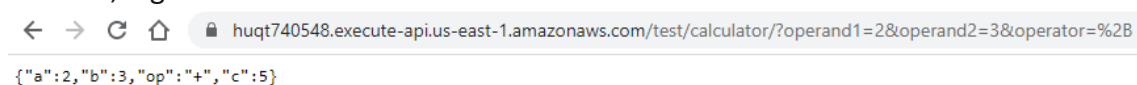
7. You can use this with tools such as [Postman](#), [cURL](#) or [SoapUI](#) to test your API.



8. Test the API from a web browser



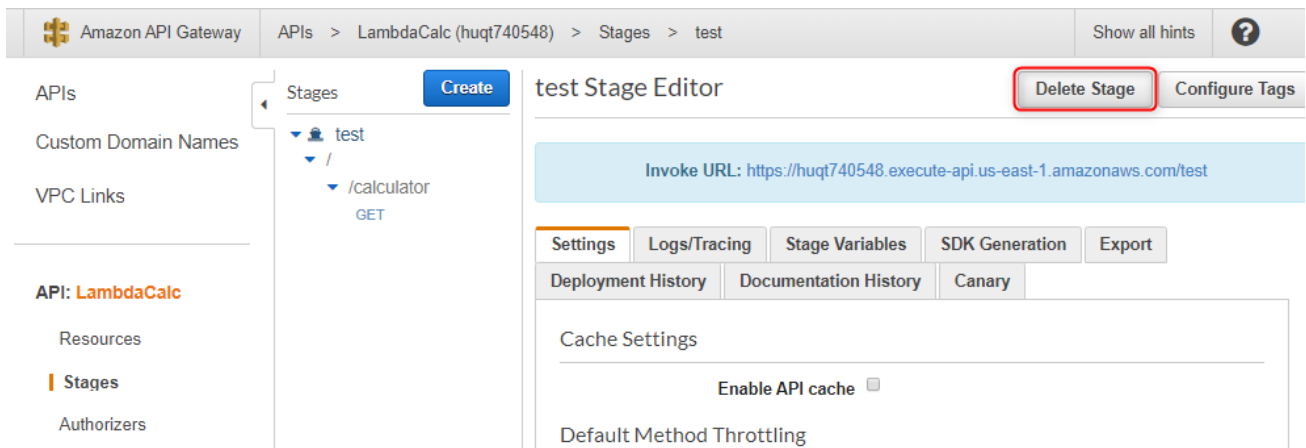
If you would like to use math symbols instead of the name of the operator, you should use code number, e.g. '+' = %2B:



Always be sure to redeploy your API whenever you add, modify, or delete a resource or method, update a data mapping, or update stage settings. Otherwise, new features or updates will not be available to clients of your API.

Step 7: Un-deploying the API and delete the Lambda function

Delete the Stage to undeploy your API after completing the exercise!



Go to Lambda functions and delete Calculator function you created.

