# Setting up Amazon Web Services

This document will walk you through the steps to set up a basic Lambda + API Gateway system.

## Creating an Account

To begin, set up an Amazon account. For initial development purposes, one can set up a Free Tier personal account. As the project grows, it will need to be migrated to a business account. Due note that it will ask for credit card information, though you will not be charged unless you use charged services.

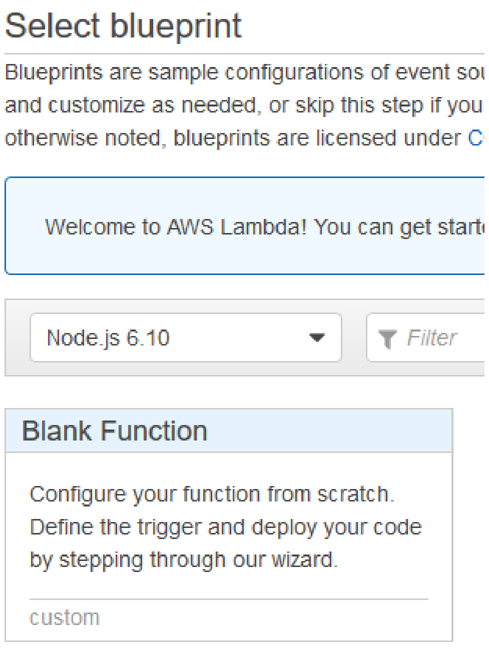
https://aws.amazon.com/s/dm/optimization/server-side-test/free-tier/free_np/

Once an account has been created, use it to sign in to the AWS Management Console: https://aws.amazon.com/console/

## Setting up Lambda

Once at the console, search for “Lambda” to go to the Lambda Management Console and choose ***Get Started*** to set up a server.

Under ***Select blueprint***, select a runtime of ***Node.js*** (we’ve developed against 6.10) and use a ***Blank Function***.



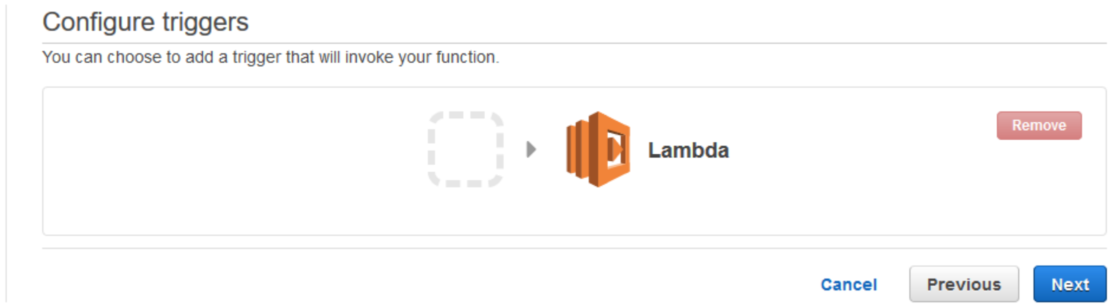
Select blueprint

Blueprints are sample configurations of event sou
and customize as needed, or skip this step if you
otherwise noted, blueprints are licensed under C

Welcome to AWS Lambda! You can get start

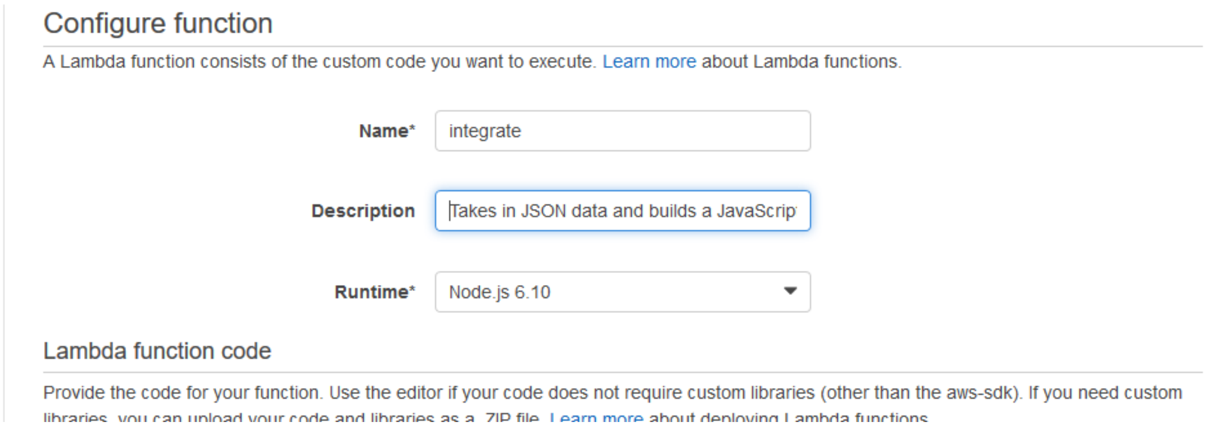
Configure your function from scratch. Define the trigger and deploy your code by stepping through our wizard.

We won't need to configure any triggers right now, so skip this step.


Configure triggers

You can choose to add a trigger that will invoke your function.

When configuring the function, give it a name that clearly represents what that function will do so you can recall it later. Also give a basic description of what that function does.


Configure function

A Lambda function consists of the custom code you want to execute. Learn more about Lambda functions.

Name* integrate

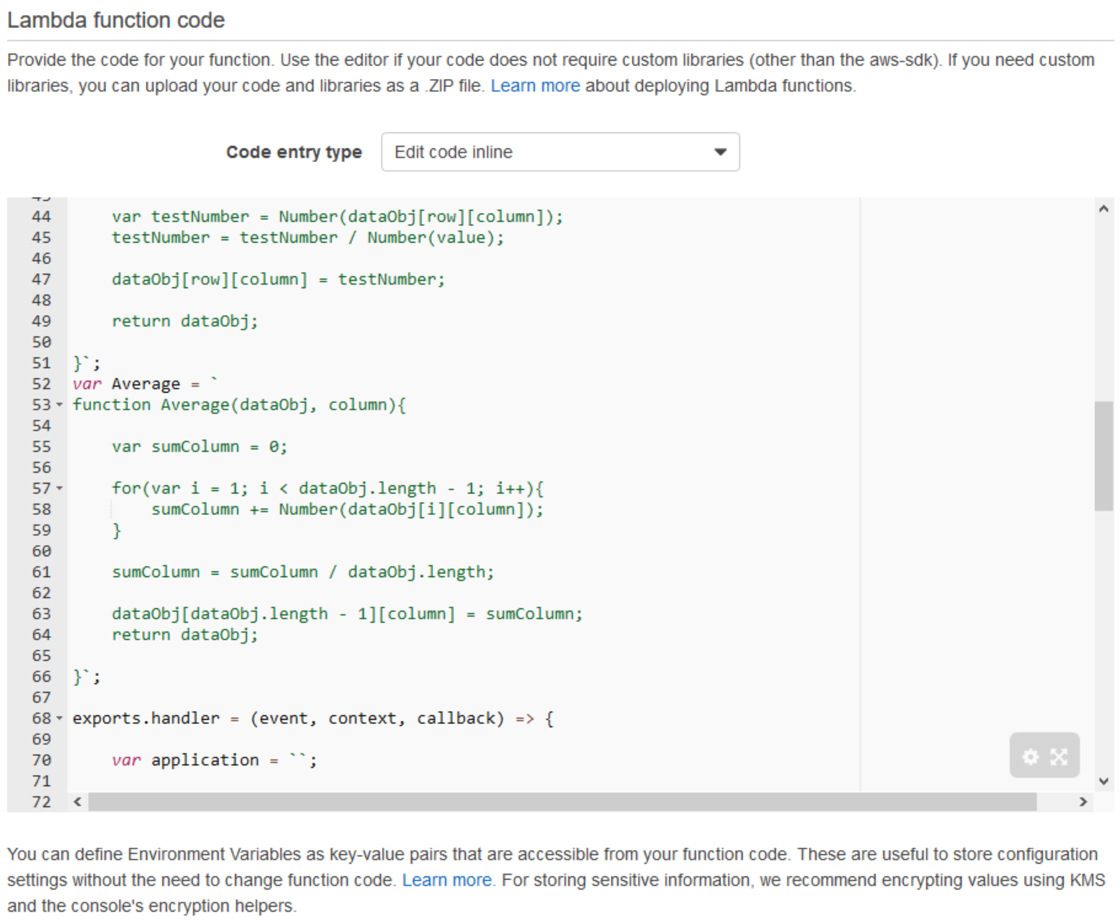
Description Takes in JSON data and builds a JavaScrip

Runtime* Node.js 6.10

Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a .ZIP file. Learn more about deploying Lambda functions.

Just below, paste in the lambda code (you can find it under \Lambda\Lambda.js in the project folder). You can paste it with the clipboard, or upload it as a ZIP archive.



Lambda function code

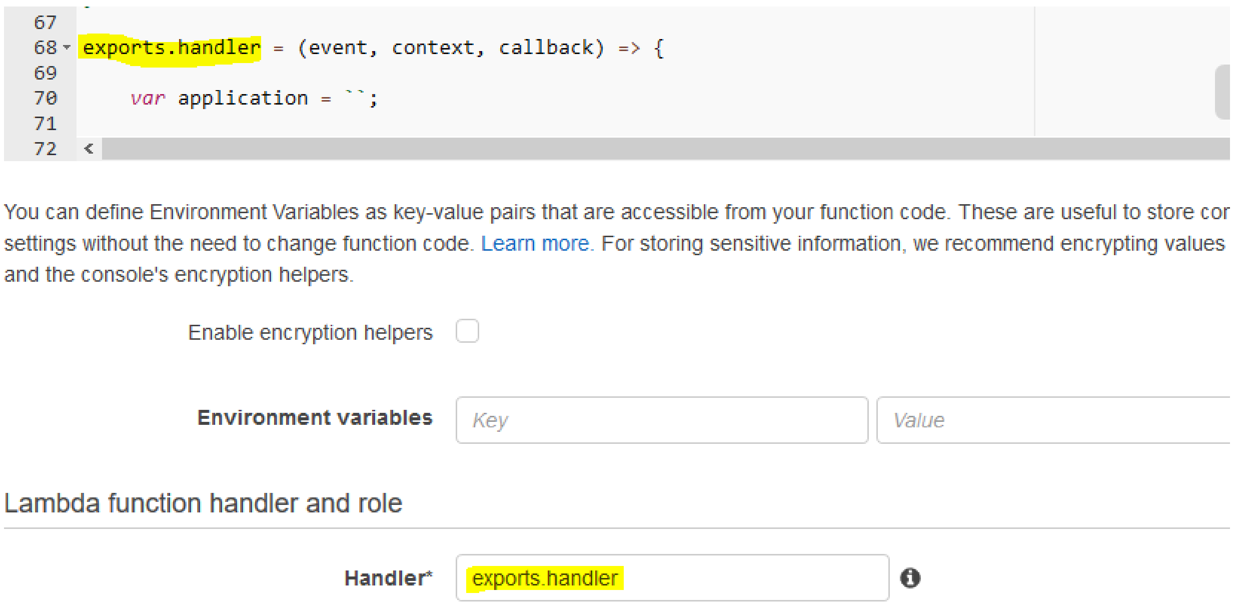
Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a .ZIP file. Learn more about deploying Lambda functions.

**Code entry type** Edit code inline

```
    var testNumber = Number(dataObj[row][column]);
    testNumber = testNumber / Number(value);

    dataObj[row][column] = testNumber;

    return dataObj;

}`;
var Average = `
function Average(dataObj, column){

    var sumColumn = 0;

    for(var i = 1; i < dataObj.length - 1; i++){
        sumColumn += Number(dataObj[i][column]);
    }

    sumColumn = sumColumn / dataObj.length;

    dataObj[dataObj.length - 1][column] = sumColumn;
    return dataObj;

}`;

exports.handler = (event, context, callback) => {

    var application = ``;

```

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more. For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.

Ensure that the name of the handler in the next box is the same as the name of the handler function in the code.



```
exports.handler = (event, context, callback) => {

    var application = ``;

<
```

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store cor settings without the need to change function code. Learn more. For storing sensitive information, we recommend encrypting values and the console's encryption helpers.

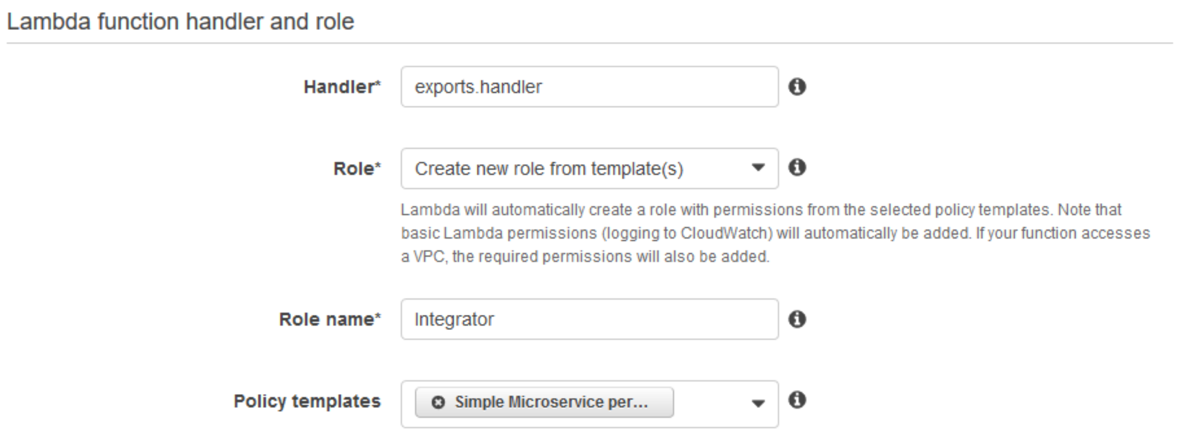
Enable encryption helpers ☐

**Environment variables** Key Value

Lambda function handler and role

**Handler*** exports.handler

Give the function a Role. A Role is Amazon's way of bundling permissions – various functions and even users can be given Roles as a reusable common package of permissions. Our function doesn't need that many permissions right now as it is, so select ***Create new role from template(s)*** and give it ***Simple Microservice permissions***. This can change later as the app is expanded.



Lambda function handler and role

**Handler*** exports.handler

**Role*** Create new role from template(s)

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, the required permissions will also be added.

**Role name*** Integrator

**Policy templates** Simple Microservice per...

Review the information and click ***Create Function***.

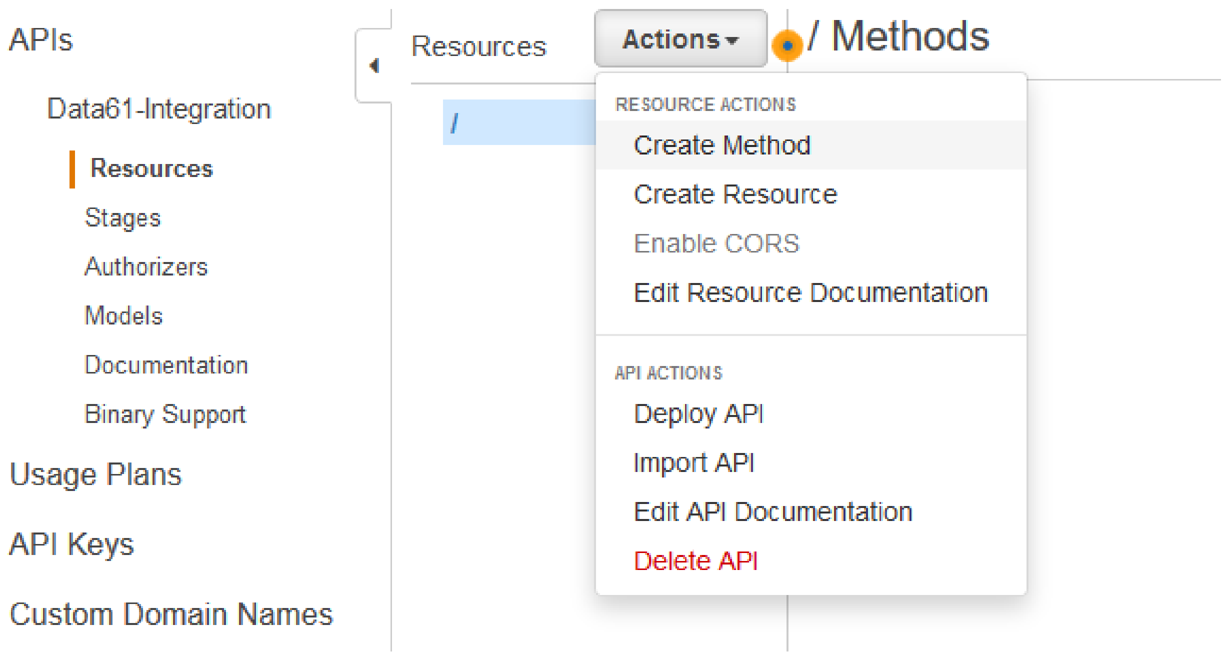
Once the function is created, use the ***Test function*** feature and paste in some test JSON data (the supplied *Integration.json* is useful for this purpose).

## SETTING UP GATEWAY

From the management console, search for "API Gateway". Navigate to that page and click ***Get Started***.

On the Create new API screen, check ***New API*** and give it a name and description. Then, continue by clicking ***Create API***.

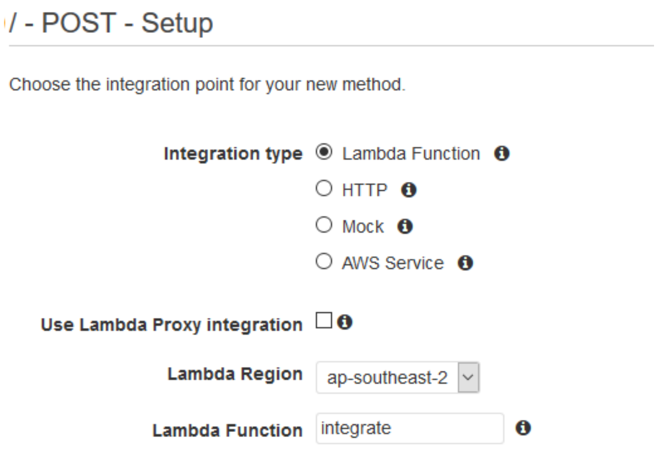
In the empty console, go up to the Actions dropdown and select ***Create Method***:



Make the method a ***POST*** method in the drop-down that appears and click the check mark to create it.

If the page doesn't auto forward you, click on the new function that you created to edit it.

For this function, we want a ***Lambda*** function. Select a region (***ap-southeast-2*** is Sydney) and type in the name of the Lambda function you created earlier (it should give you auto-populate results when you start typing). Click ***Save*** then ***OK***.

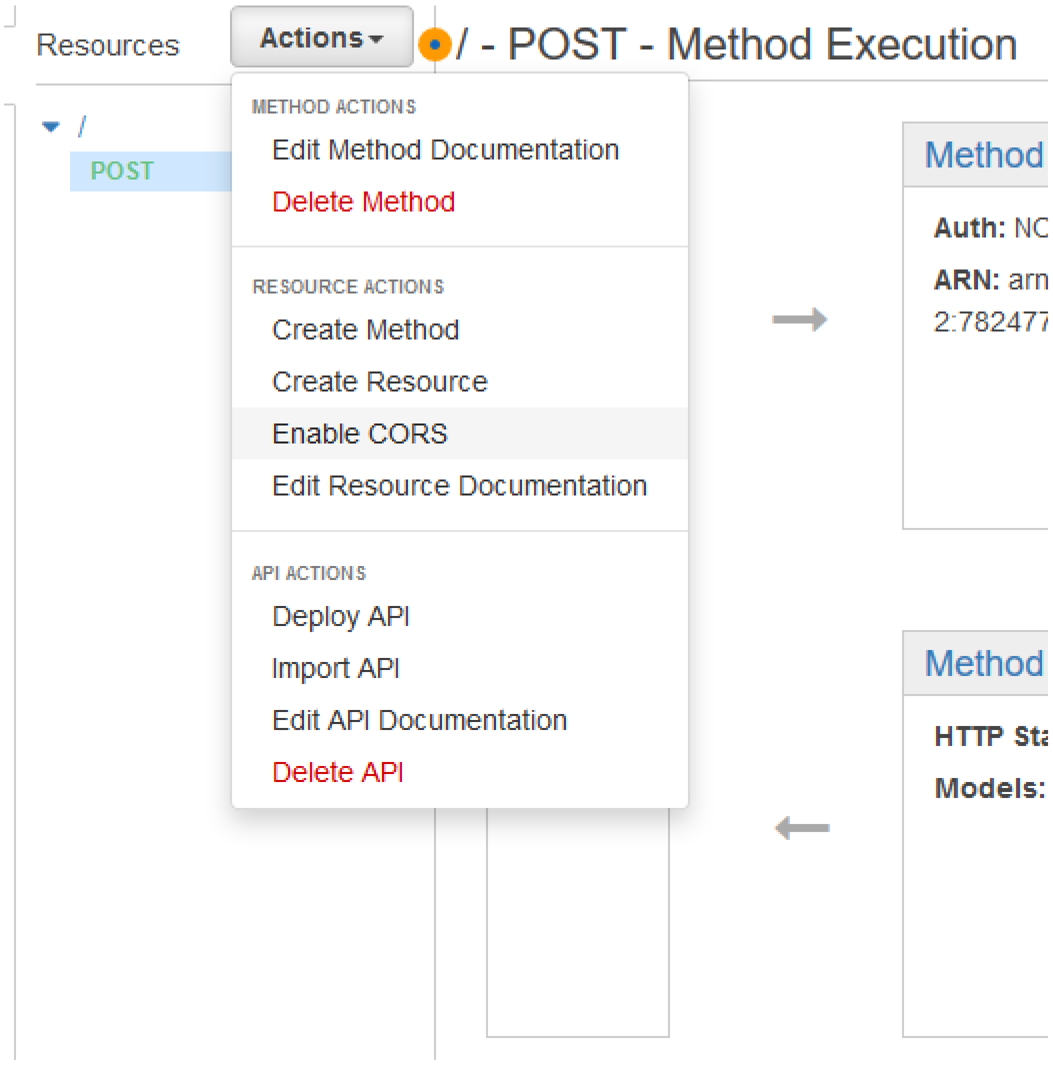


/ - POST - Setup

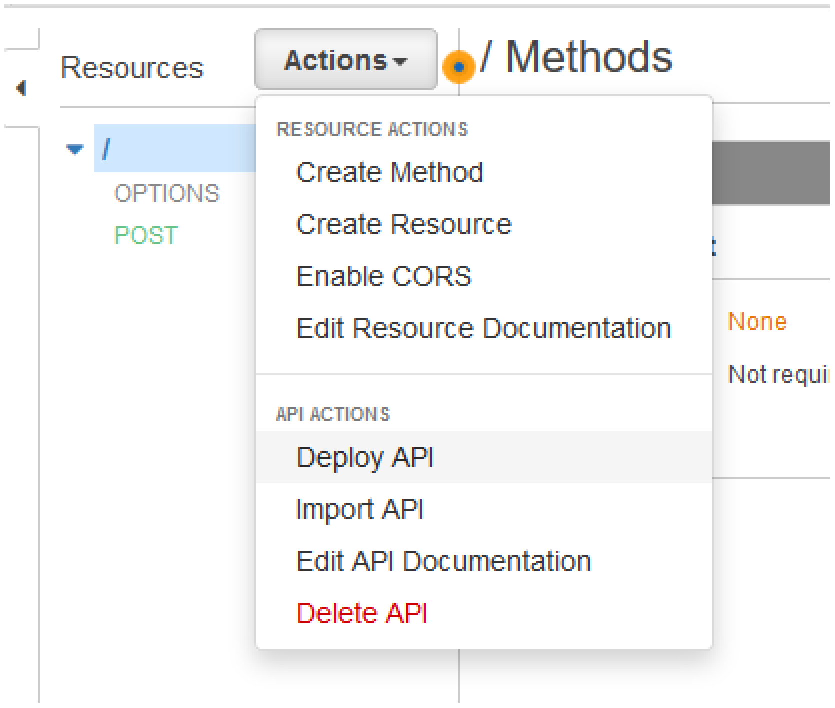
Choose the integration point for your new method.

| | |
|---|---|
| Integration type | [✓ Lambda Function] HTTP Mock AWS Service |
| Use Lambda Proxy integration | ☐ |
| Lambda Region | ap-southeast-2 |
| Lambda Function | integrate |

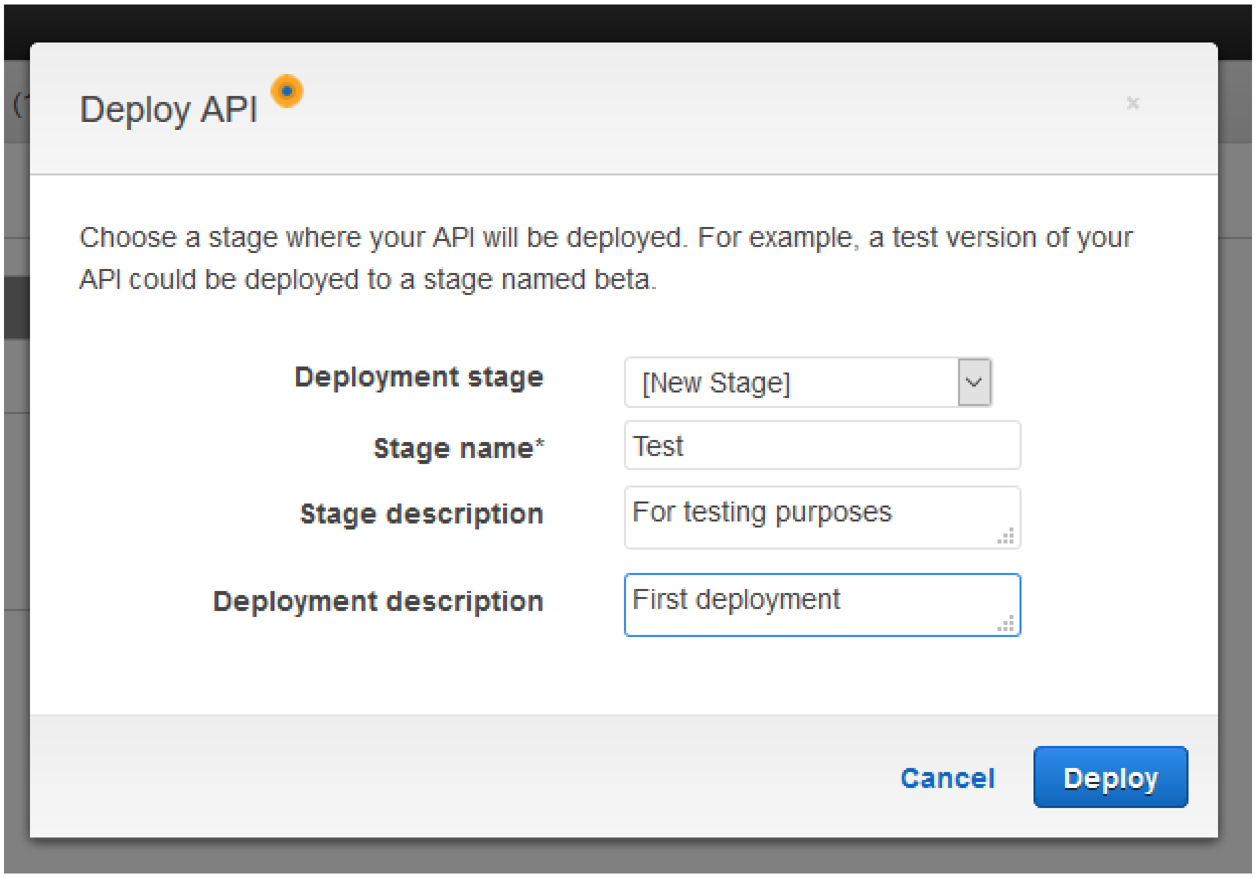
If you want this to be accessible from a browser, you'll have to enable CORS. You can add some basic CORS support by going back up to actions and selecting ***Enable CORS*** and following through the wizard. This will create an OPTIONS method along with your POST method that will be used for preflight handshakes.



Once you've created your function(s), we must deploy the API. Go up to Actions and select ***Deploy API*** from the drop-down.



In the resulting box, we will specify a Stage. By “stage”, they are referring to a reusable moniker for a development stage, such as “Alpha”, “Beta”, or “Test”. Create a new stage for your deployment and give it a description. Then, give a description of this specific snapshot of your API that you are deploying. You’ll be able to swap out the API’s deployment to any snapshot of any stage at any time, so give it a meaningful description like you would a version control commit message. Click ***Deploy*** to deploy the API to the web.



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 | For testing purposes |
| Deployment description | First deployment |

The resulting screen will give you an ***Invoke URL***. This is the base URL to which your application will send its requests. Paste this URL into the provided *server.js* and run it under Node to receive the file!