

[\(https://ko-fi.com/C0C674BW\)](https://ko-fi.com/C0C674BW)[◀ home \(/\)](#)

Creating Serverless Plotly Chart Exports

This is a step by step tutorial on creating a chart URL API on AWS. At the end of this tutorial, you will have an API endpoint that renders plot.ly charts and outputs them to the browser.

Plot.ly is a javascript charting library for the frontend and won't work on the server without some help. To render charts on the backend, we are going to use PlotlyChartExport (<https://github.com/mbejda/plotlychartexport>).

PlotlyChartExport (<https://github.com/mbejda/plotlychartexport>) is a chart rendering module that renders static plot.ly charts on the server. It's a lightweight module designed for serverless environments. There are other modules out there that can generate charts on the server, but they have dependencies that rely on graphics libraries such as Cairo.

PlotlyChartExport (<https://github.com/mbejda/plotlychartexport>) doesn't rely on those dependencies nor does it rely on those libraries. It uses phantom.js (scriptable browser) to render a Plot.ly chart which gets cropped, base64 encoded and returned in a resolving promise.

Getting Started

To create serverless chart exports we need to :

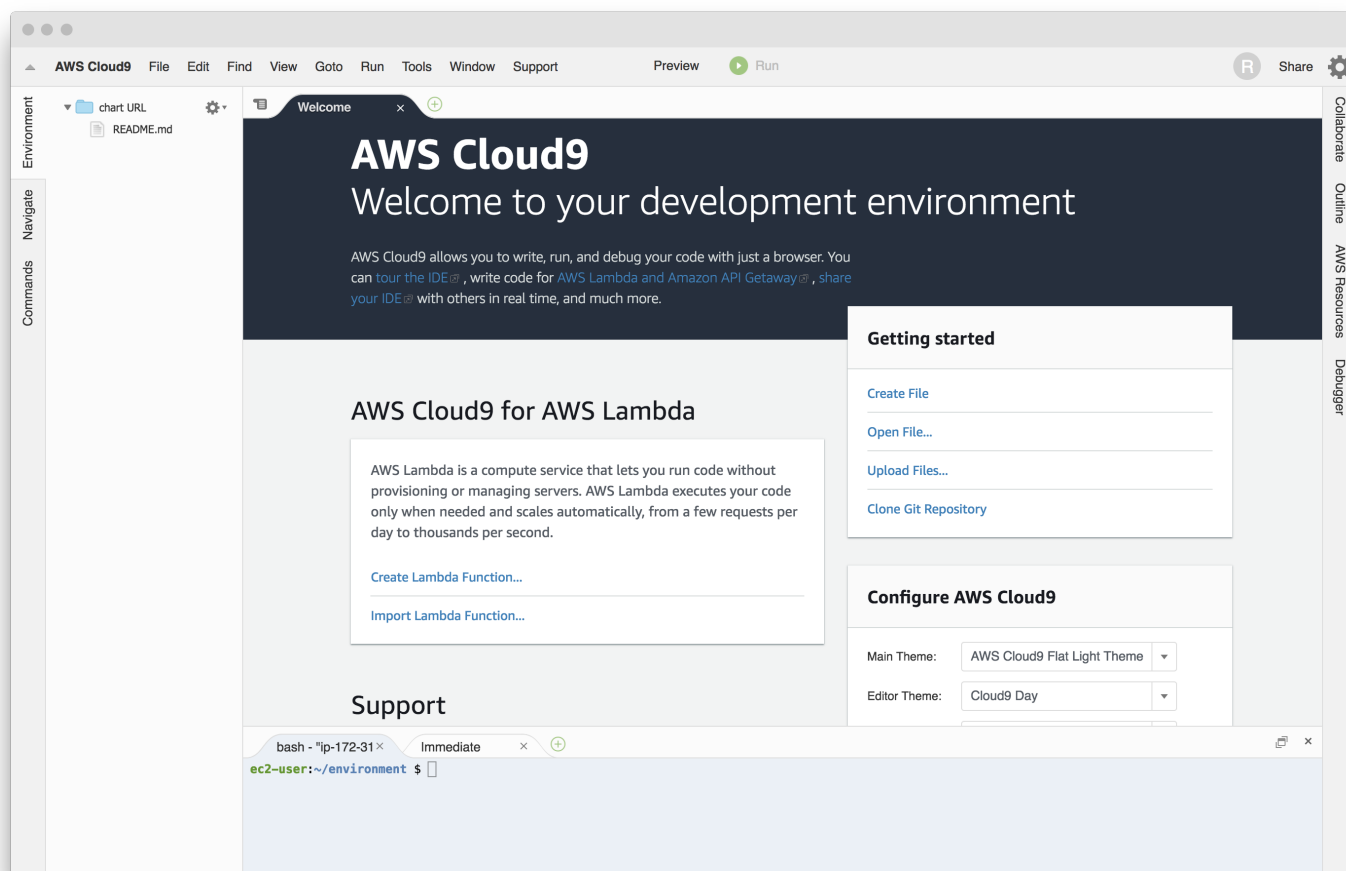
- Create a lambda function that renders a chart

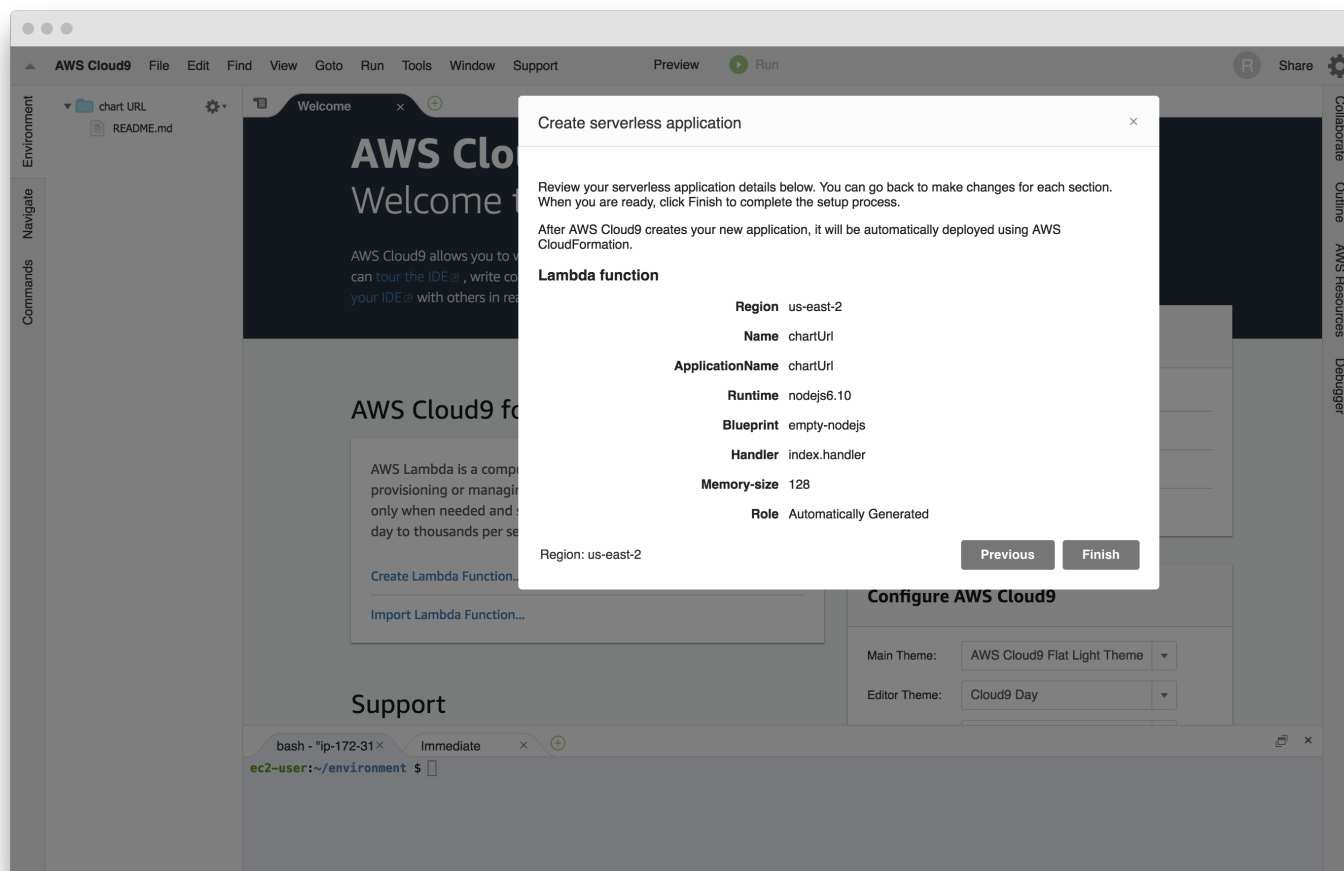
- Create an API Endpoint that invokes that lambda function and outputs that chart

Create a Node.js Lambda Function

The first thing we need to do is create lambda function that will render our chart. In the AWS console, navigate to the Cloud9 service and create a new environment. Cloud9 is an online IDE that lets you write code on a server as well

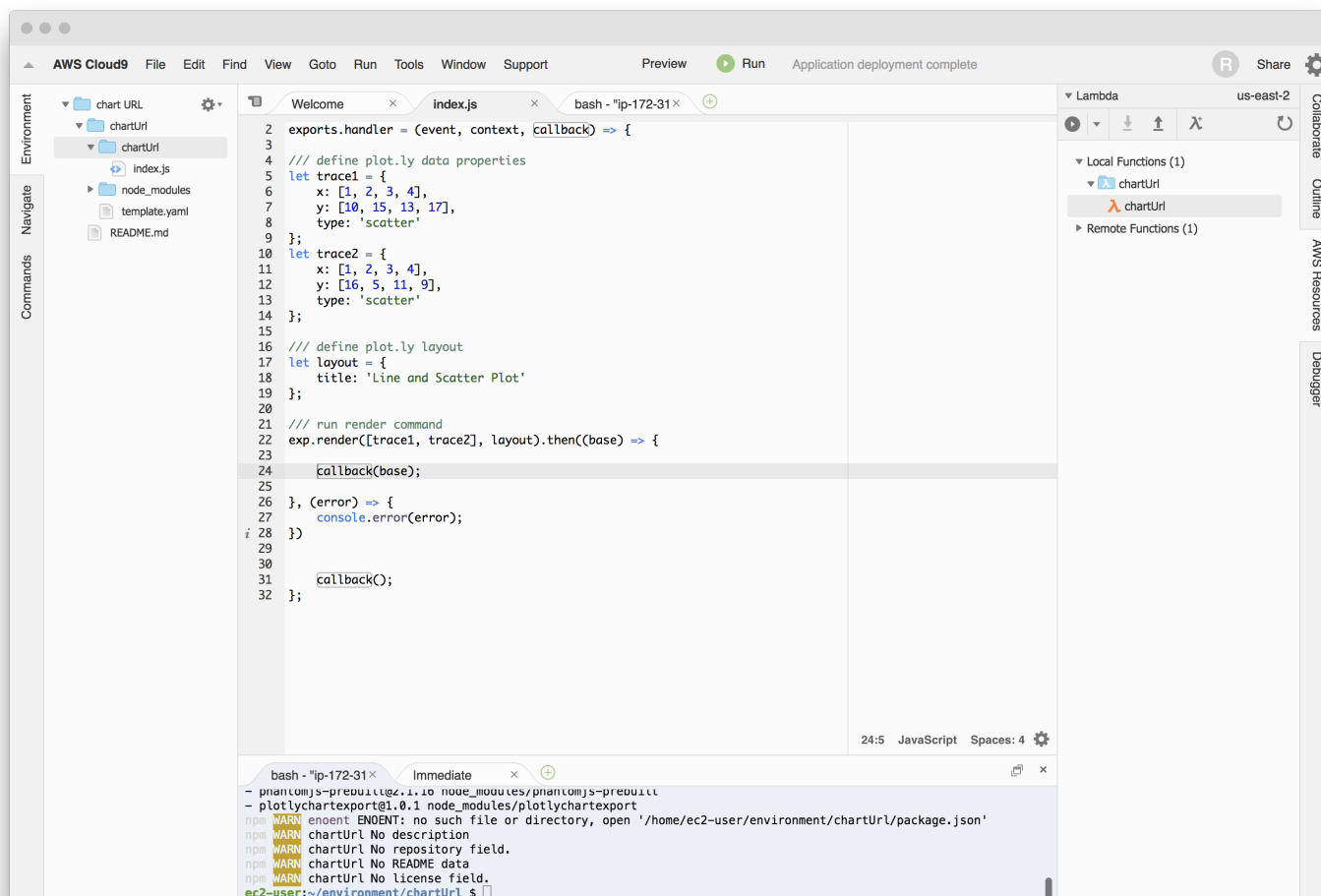
as create lambda functions, test them, and deploy them.





Inside the environment, open a shell terminal and install `plotlychartexport` .
<https://github.com/mbejda/plotlychartexport>
(<https://github.com/mbejda/plotlychartexport>)

```
npm install plotlychartexport --save
```



Add the following code to the lambda function. The code defines 2 scatter plots and sends them along with layout parameters to PlotlyChartExport (<https://github.com/mbejda/plotlychartexport>) for rendering.

```
const exp = require('plotlychartexport');
exports.handler = (event, context, callback) => {

  /// define plot.ly data properties
  let trace1 = {
    x: [1, 2, 3, 4],
    y: [10, 15, 13, 17],
    type: 'scatter'
  };

  let trace2 = {
    x: [1, 2, 3, 4],
    y: [16, 5, 11, 9],
    type: 'scatter'
  };

  /// define plot.ly layout
  let layout = {
    title: 'Line and Scatter Plot'
  };

  /// run render command
  exp.render([trace1, trace2], layout).then((base) => {

    callback(null, base);

  }, (error) => {
    console.error(error);
    callback(error);
  })

};
```

Awesome, we have our Lambda function prepared to render charts. Deploy it and let's get our API endpoint working.

AWS API Gateway

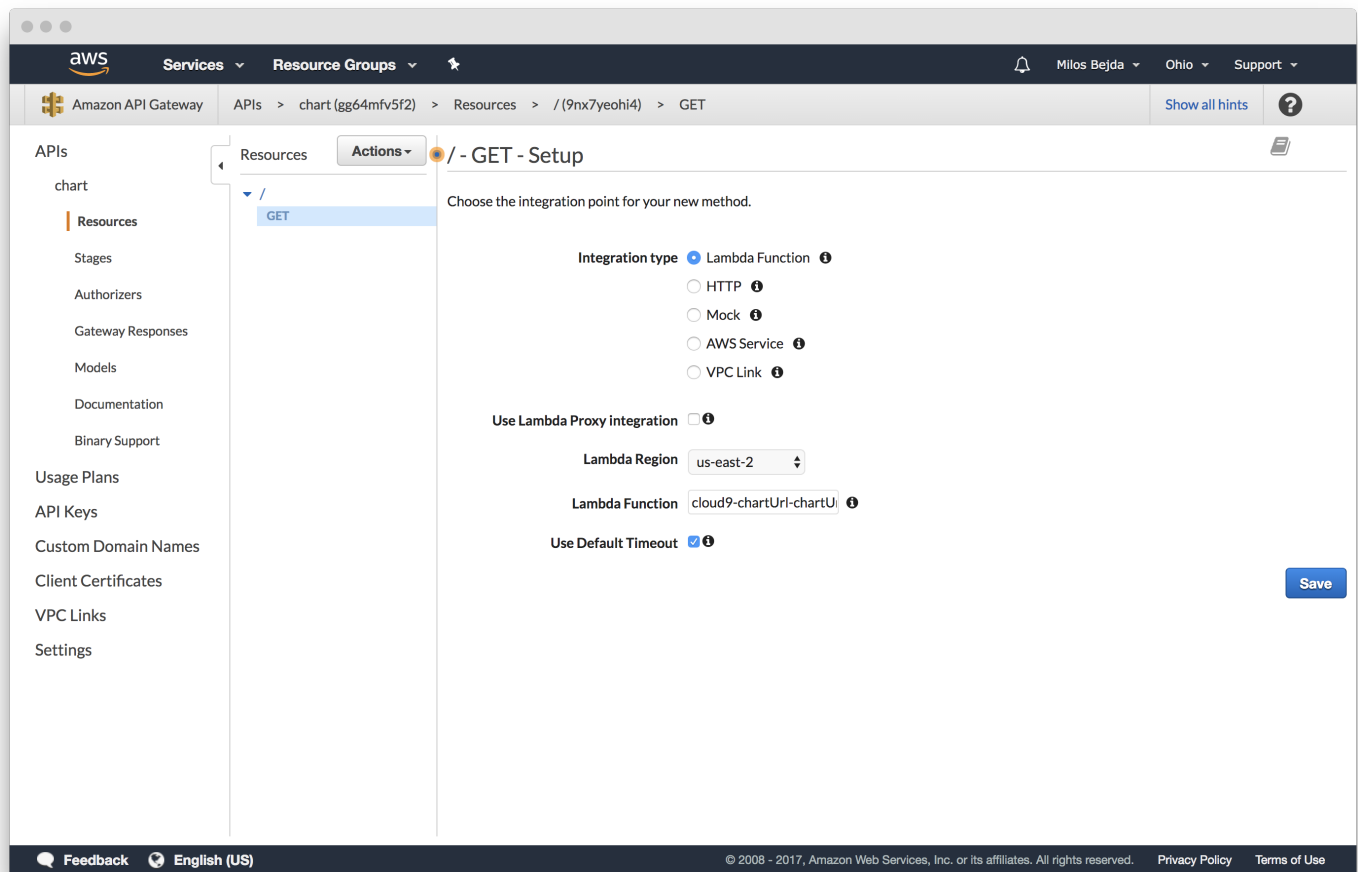
To get the API endpoint up and communicating with our lambda function we need to :

- Create a new API endpoint

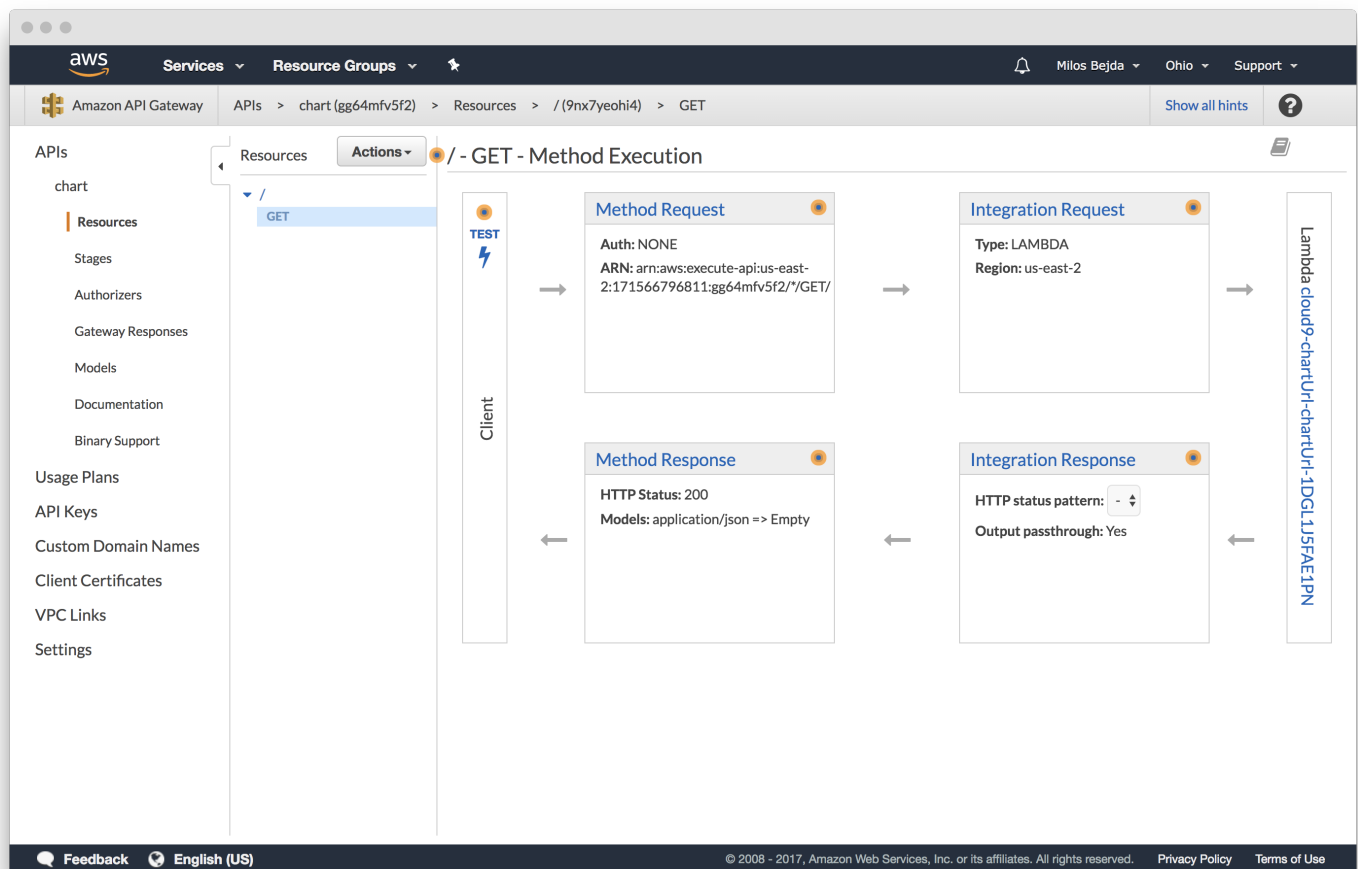
- Connect the Lambda
- Define API Content Type
- Add Binary Support for `image/png`.

Lets Get Started

Navigate to the AWS API Gateway service. Create an API Gateway endpoint by clicking on the *actions* dropdown and selecting *GET* method. Then in the setup section, for the integration type select *Lambda Function* and set our lambda function.



Navigate to method response page



Add header Content-Type and response model image/png .

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation at the top indicates the path: **APIs** > **chart (gg64mfv5f2)** > **Resources** > **/ (9nx7yeohi4)** > **GET**. The left-hand navigation menu lists various API Gateway components, with **Resources** currently selected. The main panel displays the **Method Execution / - GET - Method Response** configuration. It prompts the user to 'Provide information about this method's response types, their headers and content types.' The **HTTP Status** is configured as **200**. Below this, there are two sections: **Response Headers for 200** and **Response Body for 200**. In the headers section, a table lists a header with **Name** 'Content-Type' and a value of 'image/png'. In the response body section, the **Content type** is 'image/png' and the **Models** field is 'Empty'. Both sections have **Add** buttons. At the bottom of the main panel, there is an **Add Response** button. The footer of the console includes a **Feedback** link, the language set to **English (US)**, and copyright information for 2008-2017.

Navigate to integration response page and edit the header mappings to contain image/png . Then in the content handling drop down, select Convert to binary (if needed) .

The screenshot shows the AWS API Gateway console. The breadcrumb trail is: Amazon API Gateway > APIs > chart (gg64mfv5f2) > Resources > / (9nx7yeohi4) > GET. The left sidebar shows the 'Resources' section selected. The main panel is titled 'Method Execution / - GET - Integration Response'. It contains instructions: 'First, declare response types using Method Response. Then, map the possible responses from the backend to this method's response types.' Below this is a table with columns: Lambda Error Regex, Method response status, Output model, and Default mapping. The first row shows a status of 200 and a default mapping of 'Yes'. Below the table, there are fields for 'Lambda Error Regex' (set to 'default') and 'Content handling' (set to 'Convert to binary (if needed)'). There is a 'Header Mappings' section with a table showing 'Content-Type' mapped to 'image/png'. At the bottom, there is a 'Body Mapping Templates' section and an 'Add integration response' button.

Lambda Error Regex	Method response status	Output model	Default mapping
-	200		Yes

Map the output from your Lambda function to the headers and output model of the 200 method response.

Lambda Error Regex:

Content handling:

Header Mappings:

Response header	Mapping value
Content-Type	image/png

Body Mapping Templates

[Add integration response](#)

Navigate to binary support and add image/png

The screenshot shows the AWS API Gateway console. The breadcrumb trail is: Amazon API Gateway > APIs > chart (gg64mfv5f2) > Binary Media Types. The left sidebar shows the 'Binary Support' section selected. The main panel is titled 'Binary Support'. It contains instructions: 'You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the Content-Type and Accept HTTP headers to decide how to handle the body.' Below this is a section 'Binary media types' with a list containing 'image/png'. There is an 'Edit' button at the bottom right.

Binary Support

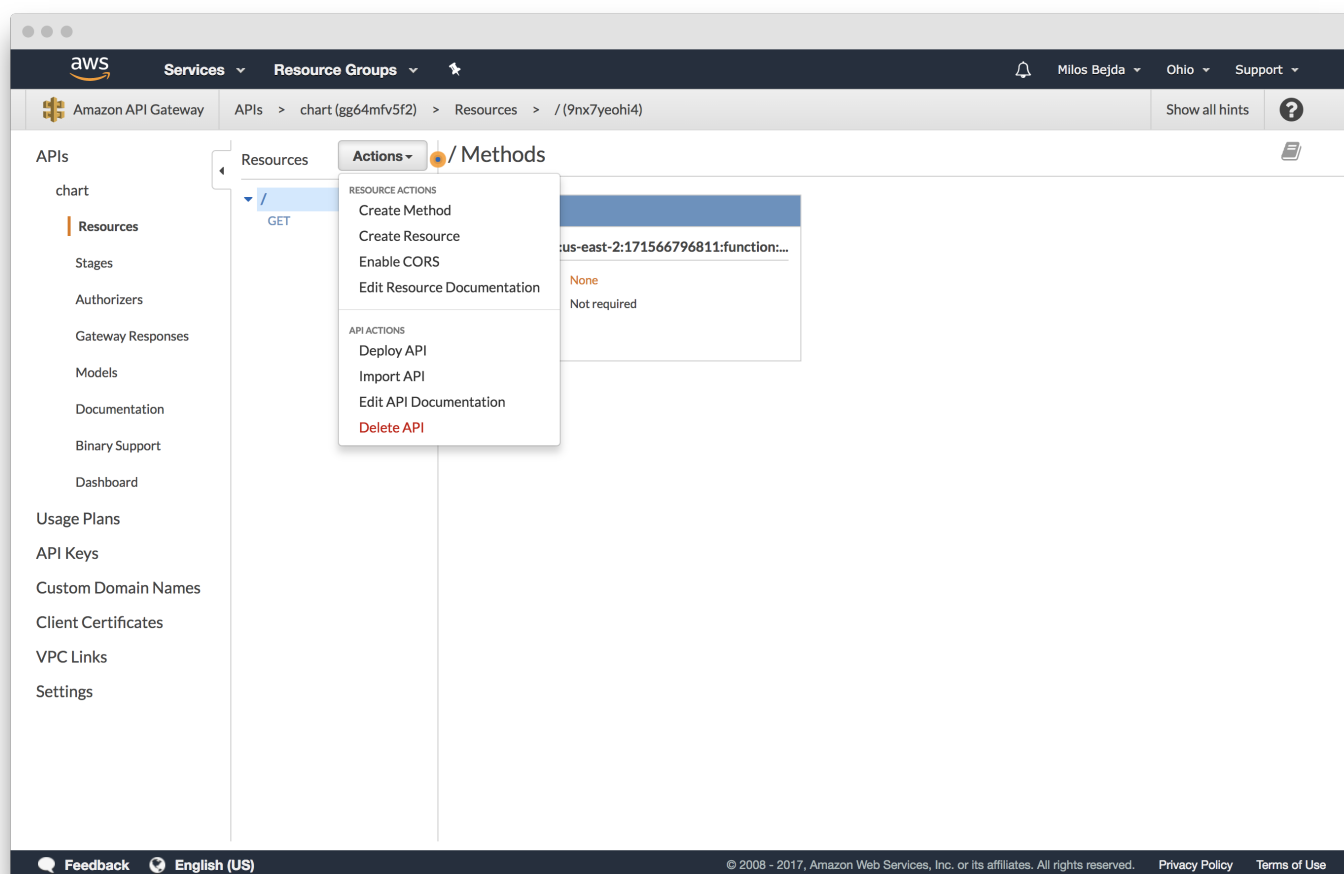
You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the Content-Type and Accept HTTP headers to decide how to handle the body.

Binary media types

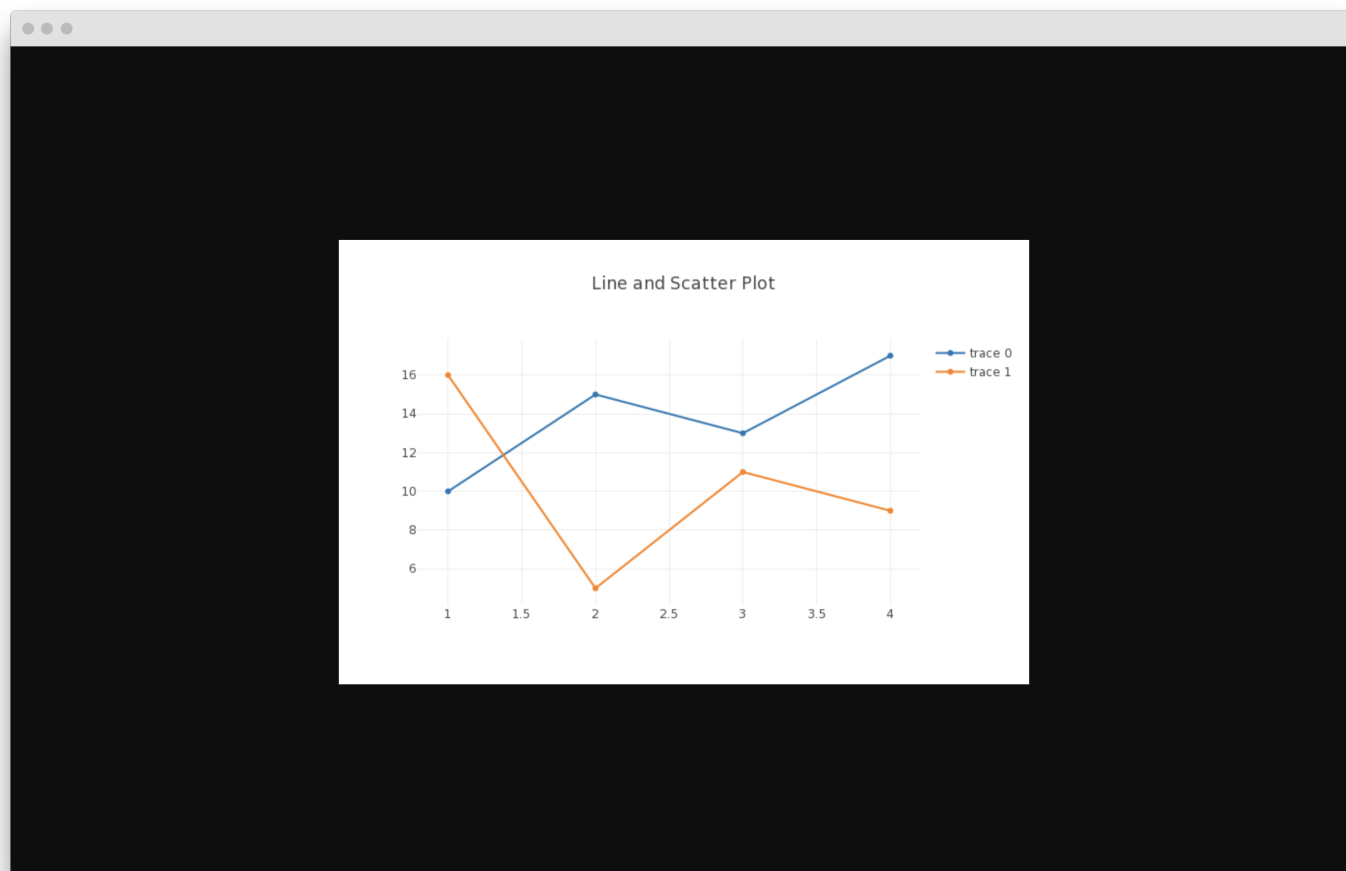
- image/png

[Edit](#)

That should be it! Time to deploy the API and try it out.



Navigate to the API endpoint to invoke the lambda function. The function renders the plot.ly chart and returns it in base64 encoded format. The API gateway then converts that base64 encoding to binary and outputs it to the page.



Conclusion

NPM Dependency: <https://www.npmjs.com/package/plotlychartexport>

(<https://www.npmjs.com/package/plotlychartexport>)

Github : <https://github.com/mbejda/plotlychartexport>

(<https://github.com/mbejda/plotlychartexport>)

If you have any feedback on this tutorial, you are stuck or I'm missing something, send me a tweet and let me know. @notmilobejda (<https://twitter.com/notMiloBejda>).

Happy Hacking

■ © 2020 Milos Bejda - All Things Automated

(<https://www.mbejda.com>) All rights reserved.

Coder Ghost Theme created by Milos Bejda

(<https://www.mbejda.com>)