How to Work with Web APIs

Published December 3, 2020 by Sean Chou

I've been recently working with a number of web-based APIs using Catalytic, and while it's an incredibly powerful feature, it can also be challenging to troubleshoot problems. It's worth doing however because once a Web API action has been completed, we should be able to share it broadly with the Community and on Showcase, and I hope to be able to do that in great volume soon.

We already have a number of good resources that you should familiarize yourself with:

- Help at https://help.catalytic.com/docs/general-api-information/
- Notion document overview (internal only) at https://www.notion.so/catalytic/Web-API-90423833b97b4594a83549979063837a

What makes working with Web APIs challenging? There are a number of potential challenges:

- Authentication can be challenging since there are many different approaches. And occasionally, the same API provides multiple approaches. Some may require OAUTH style authentication. Others may just require a special key within a header.
- The mechanics of working with different APIs differ. Some are a direct GET/POST and response. Others might be asynchronous requiring a GET/POST and then provide a web hook for a response later. Yet others might require a GET/POST and then provide a link to then check for a response after a period of time. Many different approaches.
- The format, encoding, and content of a GET/POST can vary dramatically. Generating the content with proper formatting is often the bulk of the work.
- It can be hard to understand what data is being passed back and forth between Catalytic and the API, which makes it harder to understand exactly where the problem is occurring. Is the authentication incorrect? Are the headers incorrect? Are the parameters or is the body correct? Is the response being correctly received? And so forth.

This document is intended to provide an approach to systematically work with and troubleshoot Web APIs. To get the most out of this document, you should have a basic understanding of how an HTTP POST works including headers, body, encoding, and responses.

Building out a QuickChart Workflow

I'm going to use a simple API that QuickChart provides for generating charts. It's a flexible API that accepts GET as well as POST, and can respond directly with an image, or a URI to an image. For the purposes of this document, I'll be using a POST because I generally think of it as more secure and easier to work with larger content, and receiving a binary file directly so it's stored securely and safely in the Catalytic platform.

I will be using the POST endpoint:

`https://quickchart.io/chart/`

The endpoint expects body content to be in the form of:

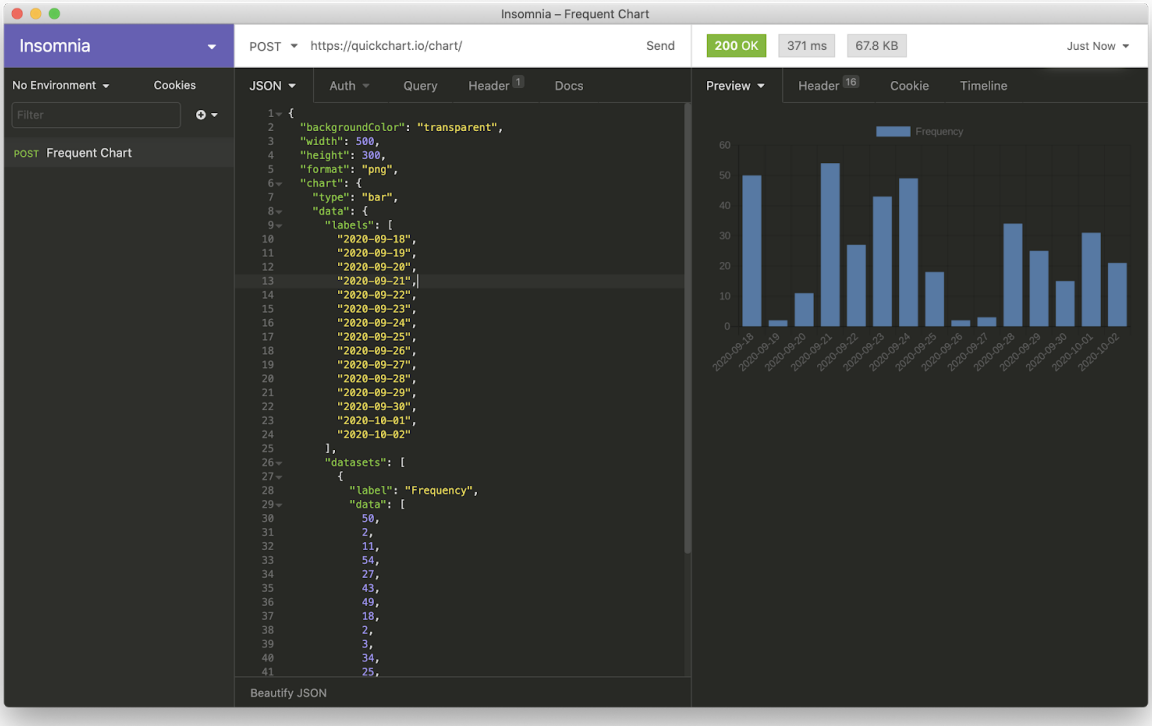
```
{
  "backgroundColor": "transparent",
  "width": 500,
  "height": 300,
  "format": "png",
  "chart": {...},
}
```

Where the chart content will be in the form of:

```
{
  type: 'bar',                                  // Show a bar chart
  data: {
    labels: [2012, 2013, 2014, 2015, 2016],     // Set X-axis labels
    datasets: [{
      label: 'Users',                           // Create the 'Users' dataset
      data: [120, 60, 50, 180, 120]             // Add data to the chart
    }]
  }
}
```

Step 1. Test it all manually first.

Before doing anything else, create a manual test using Insomnia (what I will be using this document) or Postman.

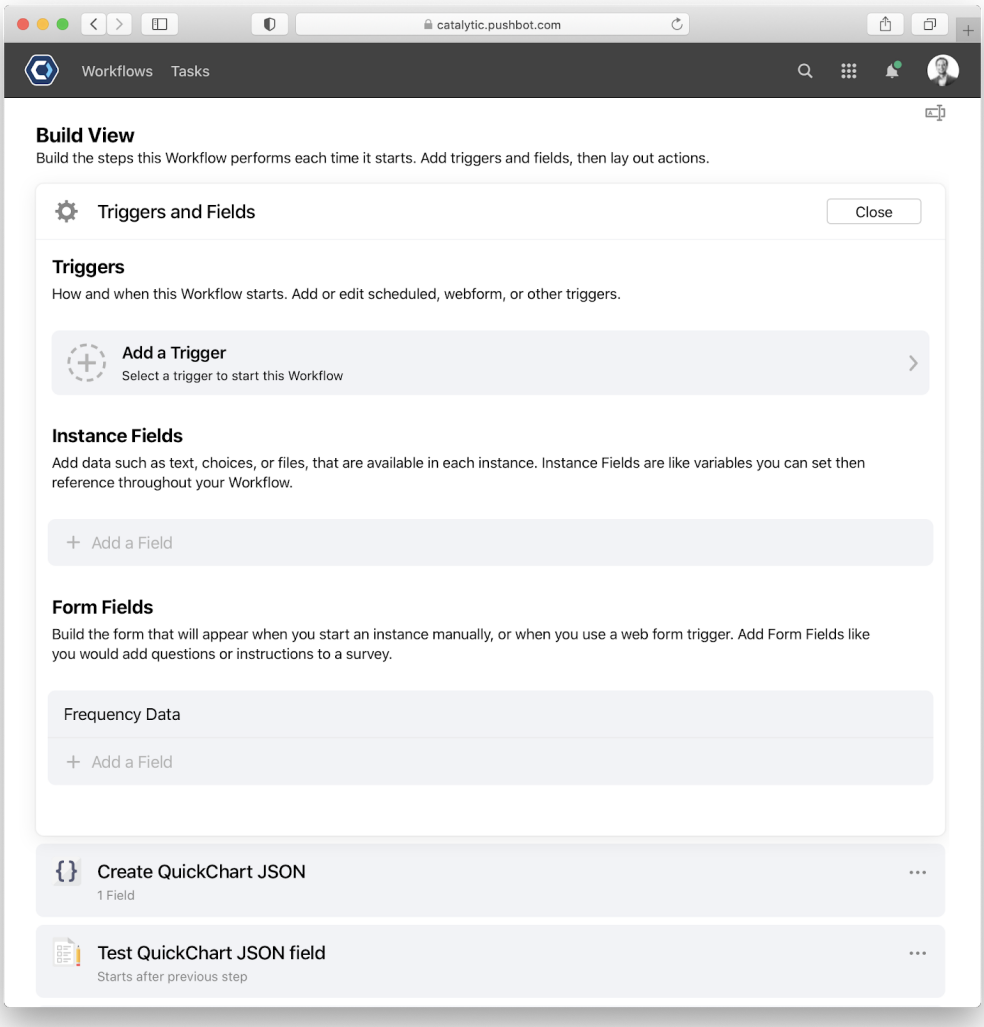


Now, I can see that the endpoint is working. The connection is fine on my side. The content is correct, and the response is what I am expecting.

Step 2. Build up the content before the Web API action.

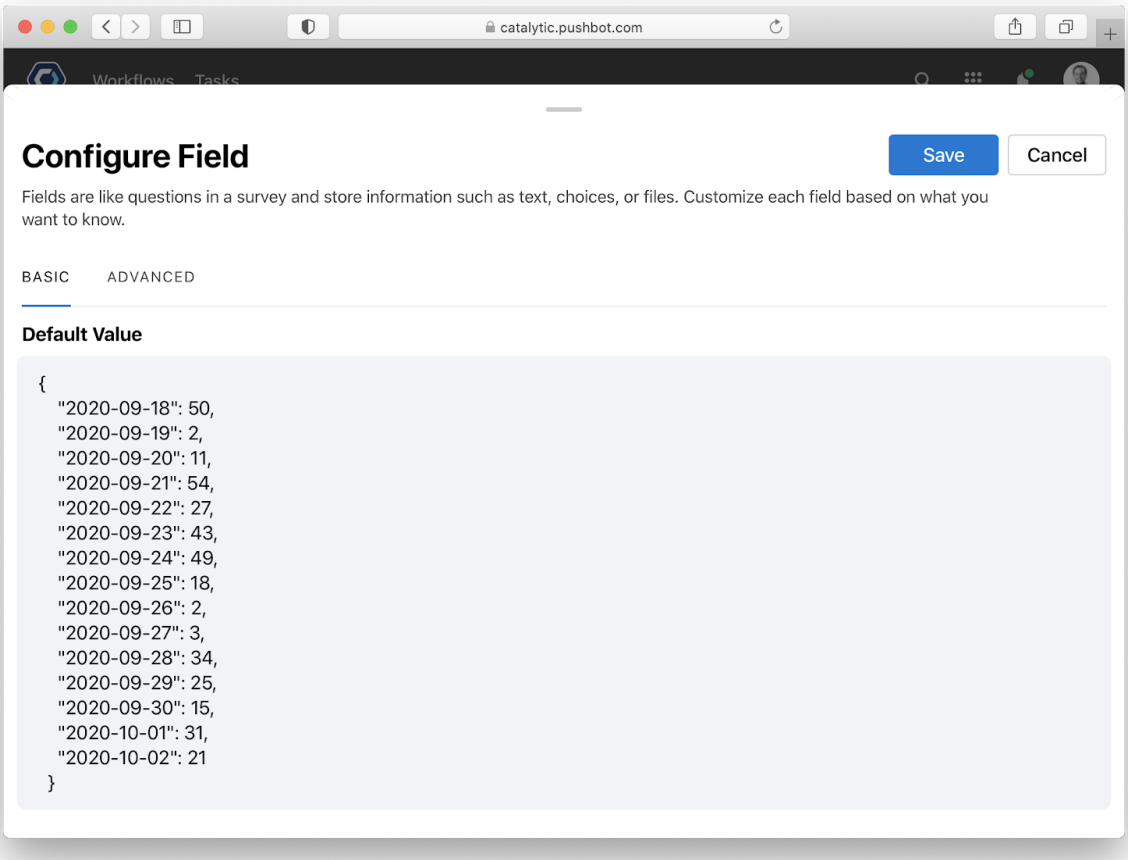
The majority of the work will likely come from creating the content. There are many ways to create content based on the specifics of the API, but it will likely involve some combination of working with data files and/or tables, data processing actions, and text processing actions. In this QuickChart example, we will use:

- A form field called ‘Frequency Data` which contains data that will be used to generate the chart.
- A `Field Formula` action to format the Frequency Data. I won’t get into the specifics of how the script works in this tutorial since this is focused on the Web API. But assume that the output is well-formed JSON that meets the QuickChart requirements.



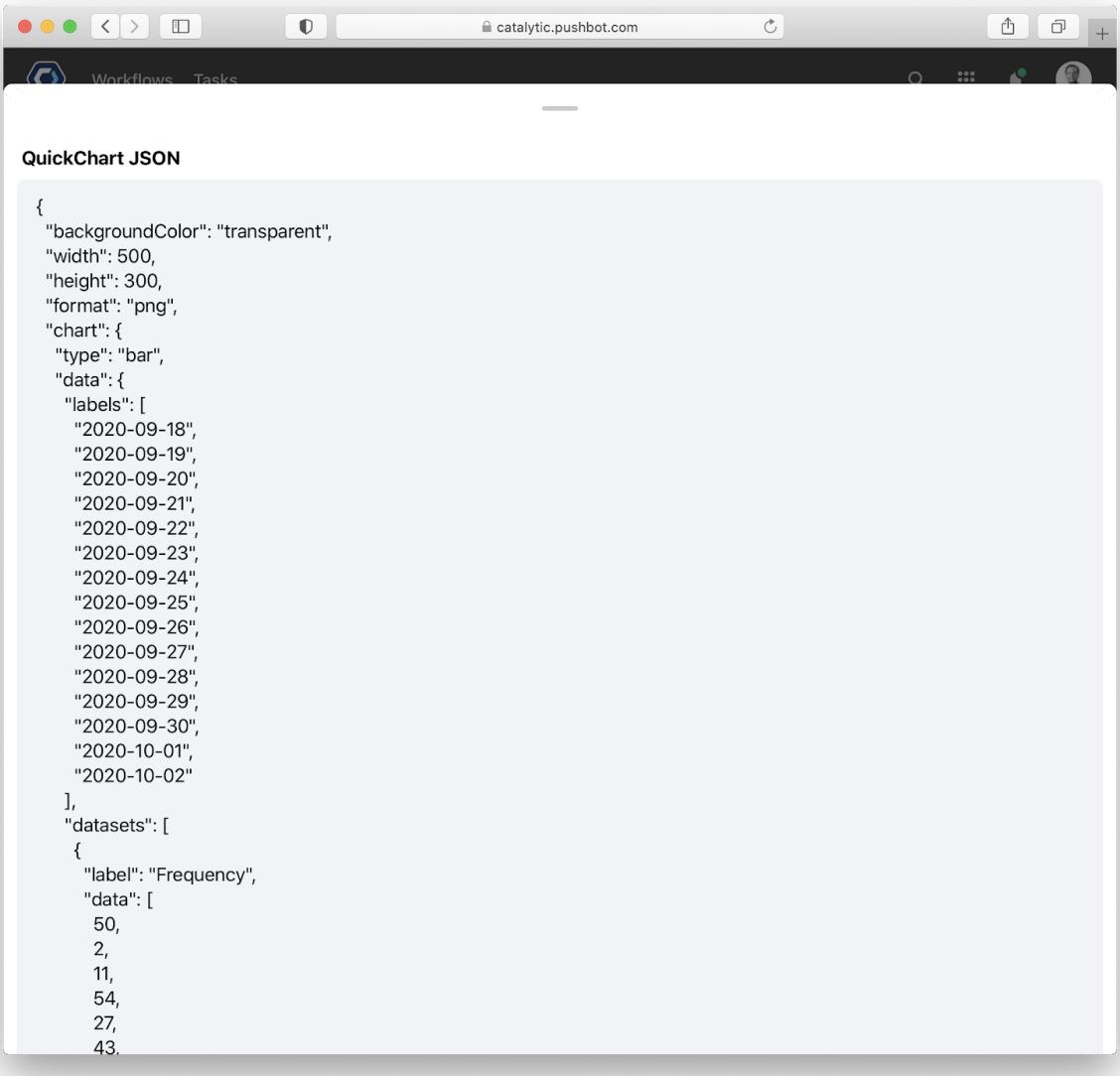
TIP: In the screenshot above, note that there is a `Task` action. It's not necessary for the workflow but it acts as a "breakpoint" to pause the workflow so we check the fields to see if the JSON output looks correct.

TIP: Although the Frequency Data field is intended to be filled out by a workflow upstream that will "call" this action, we can add a default value to make it easier to test. For example:

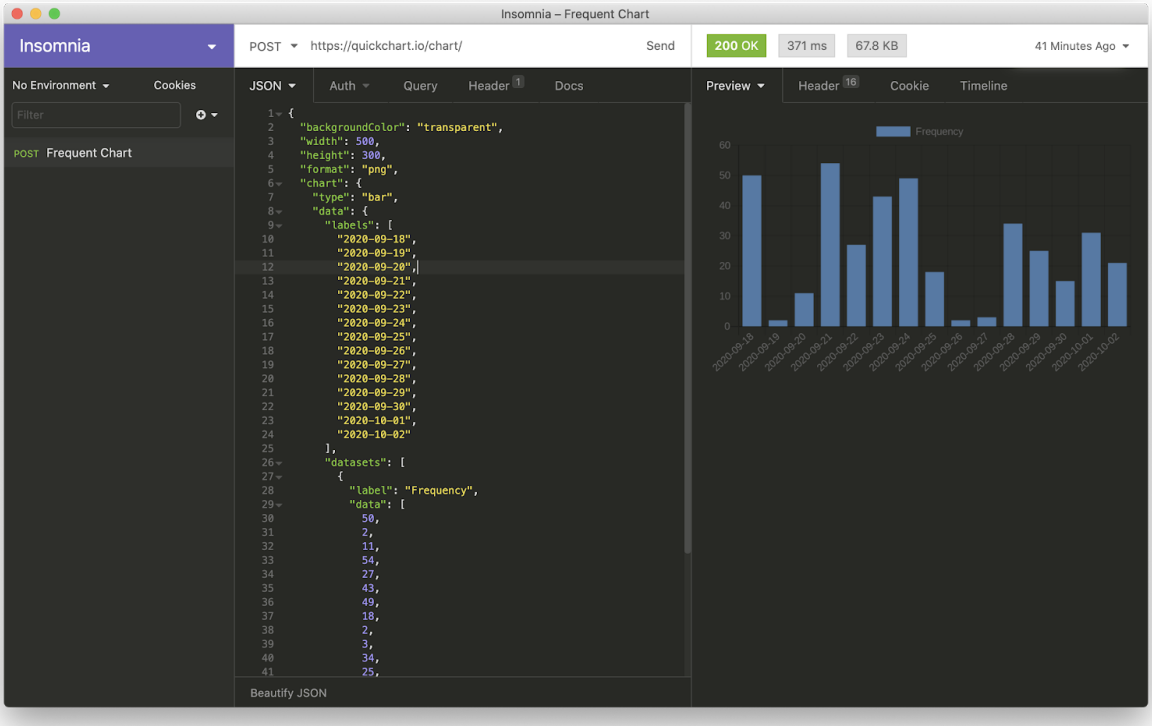


Step 3. Manually test the content.

Now that we believe the workflow is generating the content we want to post, we should test it manually before moving on. Run the workflow, copy the field content:



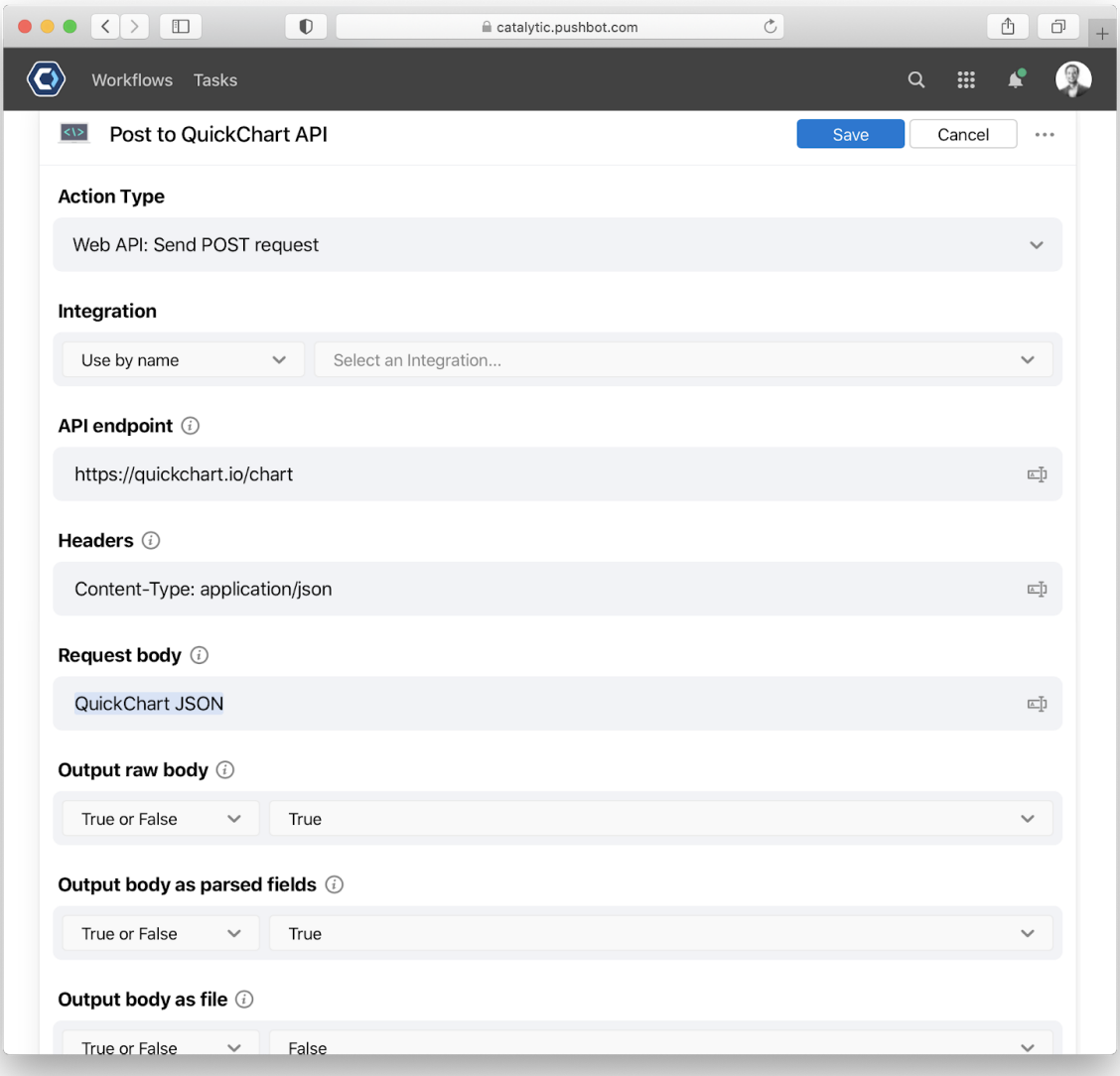
And paste it into Insomnia (or Postman):



If the result still looks good, proceed to the next step. Otherwise, our content is incorrect and we need to figure out what's happening with the content before proceeding.

Step 4. Add the Web API action.

Let's add the `Web API: Send POST request` action. It has a number of parameters but they are all documented in the help and easy to fill out.



The parameters and values for this particular endpoint are:

Parameter	Value	Comment
Integration		This parameter specifies a preconfigured integration. In this use case, we do not need to set

		up a separate OAUTH integration to use for this endpoint so we can leave it default.
API endpoint	https://quickchart.io/chart	This parameter specifies the URI endpoint of the API. In this case, we are using the URI provided by the QuickCharts documents and tested with Insomnia. Many API providers provide multiple endpoints so be sure you are using the correct one.
Headers	Content-Type: application/json	This parameter allows us to add any additional headers. This is frequently required for setting the content type and sharing an API key. In this case, we only need to set the Content-Type. If we were using a paid version of QuickChart, we would also pass along the API key.
Request body	{{QuickChart JSON}}	This parameter specifies the body of the HTTP POST. We will typically need to interpolate field values. If this is a string value, the results will be sent as a string. If this is a binary value, the results will be encoded based on the “File encoding format” parameter. If the value is mixed, the binary file will be converted to a string. In this case, we are providing a JSON value which is a string via an interpolated field. NOTE: There is currently a bug that prevents interpolation of fields from occurring if the file encoding format is set to anything other than ID.
Output raw body	False	This parameter controls whether the response body will be saved as an output field. If set to TRUE, the action will create a field called `prefix-Response--Body--Text`. In this case, we do not need the body of

		the response in a raw format, but we will need it as a file.
Output body as parsed fields	False	This parameter, if set to TRUE, will automap an XML or JSON response to output fields. In this case, we are not expecting an XML or JSON response and do not need them to be mapped.

Output body as file	True	This parameter, if set to TRUE, will save the response as a file, and will create the following fields:
		
		`prefix--Response--Body--File`
		
		`prefix--Response--Body--File name`
		
		`prefix--Response--Body--File size in bytes`
		
		`prefix--Response--Body--File extension`
		
		`prefix--Response--Body--File type`
		
		In this case, we are expecting a binary image file so we need to set this to true.
		
		This setting can also be useful for very large XML or JSON responses that exceed our field size.

Output headers	False	This parameter, if set to TRUE, will save the headers in one field per header in the following convention: `prefix--Response--Header--key name` In this case, we will not need them but this can be useful for debugging.
File encoding format	ID	This parameter controls the file encoding format of a binary file in the Request body parameter. The valid values are:

		ID - Provides the file ID on the Catalytic platform; mostly useful for internal Catalytic APIs Link - Provides a URI to the file which some API endpoints are able to use in lieu of directly passing the file Base64 - Encodes the file with Base64 Binary - Encodes the file with binary NOTE: As of Dec 3, 2020, there is currently a bug where if any value other than ID is specified, the Request body will not be interpolated. In this case, we are not sending a binary file so we will leave it at the default value.
Disable certificate validation	False	This parameter allows the action to ignore invalid certificates. Note that this is a security risk so be sure you know what you are doing if set this to TRUE. In this case, we are fine with the default.
Include headers with redirect	False	This parameter allows the action to send the headers when redirected to the new endpoint. Otherwise, if redirected, the headers will not be included. This is a security precaution to ensure headers are only sent to the direct endpoint by default. Note that this is a security risk so be sure you know what you are doing if you set this to TRUE. In this case, we are fine with the default.
Override the default allowed TLS/SSL ciphers	False	This parameter allows the action to accept less secure ciphers. Note that this is a security risk so be sure you know what you are doing if you set this to TRUE. In this case, we are fine with the default.

Now that we've configured this action, we can run the Workflow. Once it's complete (you may have to mark a task complete if you've left it as a breakpoint), check for the response from the action. If it looks good, congratulations, you've set up the Web API action!

If it doesn't seem to be working, welcome to the majority of first attempts. Failure and issues are to be expected because, as mentioned in the introduction of this document, there are a lot of potential failure points.

Here are some tools to help diagnose problems:

Postman Echo

This is a lifesaver. Change your endpoint to:

```
https://postman-echo.com/post
```

Test the workflow and look at the response. Postman Echo is a simple service that, as the name suggests, echoes back what was sent to it. It's a great way to check if the content we think we are sending is really what we think we are sending.

JSON Lint / Beautifier / Validator

JSON can make your eyeballs bleed. Is there an extra brace somewhere? Is a quote missing?

There are a lot of services that will validate and beautify your JSON for you. I use the instant answer that shows up when you search for "json lint".