Creating Serverless APIs With AWS Lambda and API Gateway

Save time & money by serving API calls with on-demand computing

Today’s Agenda (2 Hours Total)

- Introductions (Hi!)
- Definitions
- Introduction to Lambda
- Introduction to API Gateway
- Break—10 minutes
- Implementing an API function in practice
- Business implications of serverless APIs
- Break—10 minutes
- Next Steps for creating serverless APIs

About Me, About You

About Me—Why Should I Be Here?

- Hi! I’m Bill “Spruke” Boulden.
- @Spruke on Twitter
- linkedin.com/in/spruke
- Dad taught me BASIC at 5 years old
- 20+ years development experience moving from VB through .NET through Perl now to Node.js
- Currently Chief Technology Officer of ClearView Social, a social network employee advocacy solution
- We’ve run 100% of this business’s tech solely on AWS since day one!

About You—Why Should You Be Here?

- Moderate or higher proficiency in Node.js or Python
- Very basic understanding of HTML5 and Javascript in the DOM—enough to manipulate a simple form on a web page
- Very basic understanding of relational databases
- Very basic understanding of domain names
- Familiarity with API concepts such as statelessness and RESTfulness
- Have an AWS account set up and ready to go!
    - Costs will be miniscule today—in the single digit cents range—but there is no free tier usage levels for Lambda and API Gateway, so we will indeed be paying micropennies.

Does the above sound like you? Then we're ready to go.

Vocabulary Overview

Web API

- Web “Application Programming Interface”
- Providing URL “endpoints” (in most cases and in today’s case, via HTTP(S)) which can be called with “requests” to receive “responses”
- Historically built in frameworks known as SOAP or SOA using XML
- Today popularly built using RESTful approach using JSON

An HTML page or a mobile app, rendered on a consumer’s client device, doesn’t have access to the web’s database. How does it add a contact/get a list of customers/check the player’s score/upload a photo? By sending a request to a web API and doing something with the response it receives back.

“RESTful” Approach

- Stands for “REpresentational State Transfer”, but that’s not important
- More of an attitude than a protocol, encompasses many design principles
    - Separation of concerns between client and server
    - Statelessness
    - Cacheability
    - Proper usage of HTTP methods
    - Proper usage of HTTP status codes
- You don’t *have* to adhere to RESTful design principles while working in Lambda and API Gateway, but your life will be much easier if you do!

Separation of Concerns Between Client and Server

- The Client
  - Lives on a user's device over which you can assume no control
  - Probably an HTML document, thin desktop application, mobile app, or IoT software
  - Does not understand what happens behind the scenes outside of the request and response
  - Asks for more information from server by sending a request with HTTP method to API
  - Controls all presentation details
- The Server
  - Knows nothing of presentation
  - Trusted environment over which the client has no power except what is sent in qequest
  - Has connectivity to databases, business logic, etc.
  - Probably a Lambda function being served behind an API Gateway :)
- A RESTful approach doesn't let the client care about business logic or database details, or the server care about UI or presentation concerns!

Statelessness

- The belief that every request/response cycle lives in a vacuum
- “If a request to add 2+2 is sent, the response says 4, even if the previous request asked for 3+3 or the server has been multiplying all day”
- This is in contrast to previous generations of APIs where operations had to be called in a particular sequence, such as a request to “open a session” followed by a request to “prime an object”, etc. and the server would be expected to “remember where it was”
- Notable exceptions:
    - Cookies are used to represent that a client is in a particular state or is an authenticated user, but since cookies are HTTP headers, that technically does not violate statelessness.
    - Due to container re-use in Lambda, you may accidentally violate statelessness

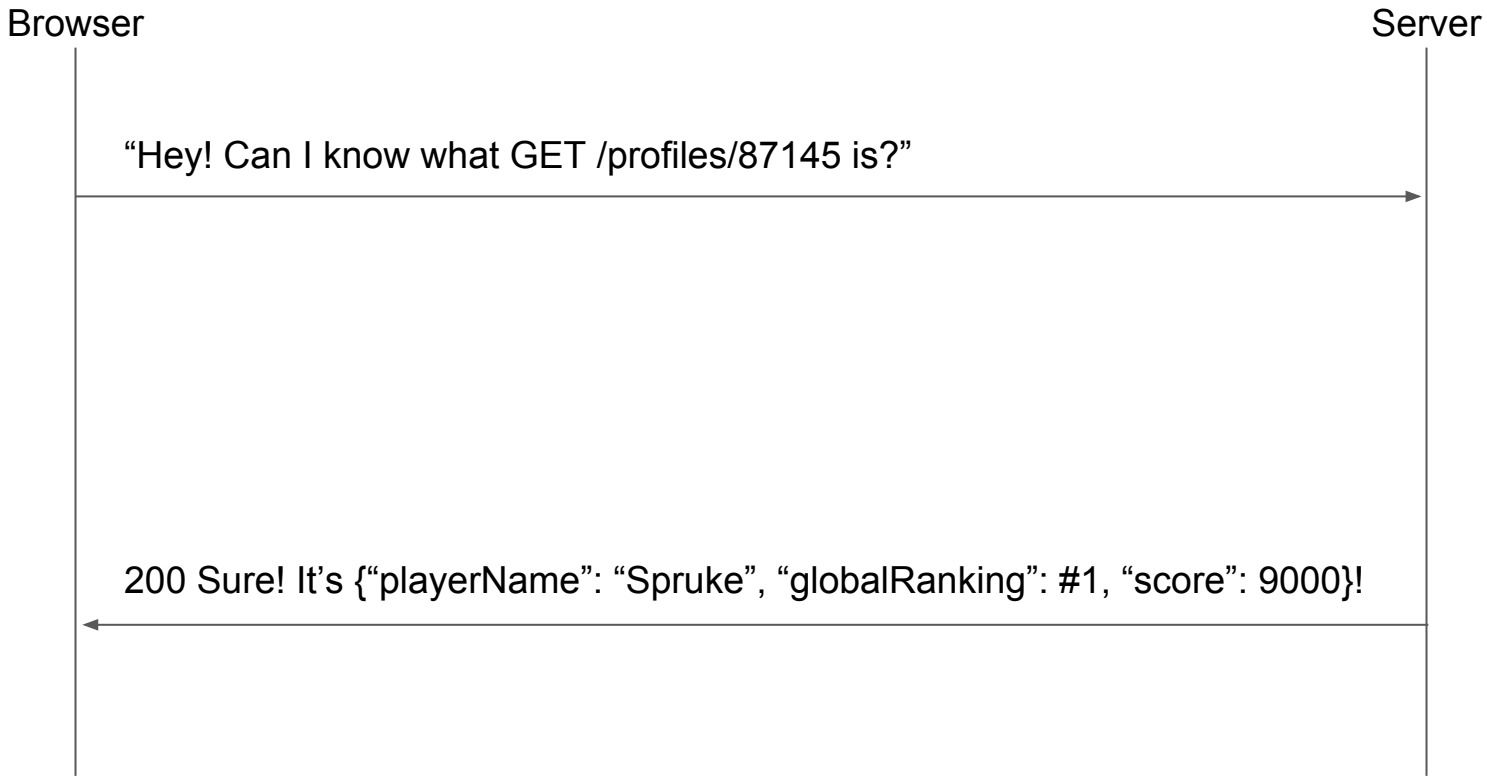
Cacheability

- Since requests are stateless, requests that do not create a change to data (for all practical purposes, mostly GETs) need not be re-executed soon
- Since every request lives in a vacuum and doesn’t affect anything else, if client Anon123 asks for GET /images/john.jpg and four seconds later client AnonABC asks for GET /images/john.jpg, it is safe to let middleware save Anon123’s response and hand the same thing back to AnonABC
- This is a place API Gateway and especially CloudFront work wonders

Proper Usage of HTTP Methods & Status Codes

- Correctly having the client invoke their request as an HTTP GET, POST, PUT, or DELETE will help API Gateway (although Lambda won't care)
- These have semantic differences, but also technical consequences, as GETs are expected to display cacheability and the other three are not
- Correctly labeling the response a status code such as 200 OK, 404 Not Found, 500 Internal Server Error, or 504 Gateway Timeout (get ready for these) will further assist API Gateway & later CloudFront

Sample (simplified) HTTP Flow Diagram



Introduction to “Compute On Demand As A Service”

AWS Lambda

- “On Demand Computing As A Service” or “Function As A Service”
- Execute code “in the cloud” without having to administer or configure a server or even a container, and pay by the resources consumed
- Your obligation to pay essentially begins when the CPU and RAM start to service “function handler() { …” and then ends at “return ____;”
- AWS Lambda alternatives that are conceptually similar include OpenWhisk, Google Cloud Functions, and Microsoft Azure

As opposed to?

- I want to set up a web server to answer requests such as GET /api/playerid
- Okay, I'll provision an EC2 virtual server (or multiples according to load)
- Now I'll install Apache and/or NGNIX
- Configure them, install libraries and dependencies such as modssl or modphp
- Hmmm, which users should have permissions to which directories?
- Oh darn, the server went down because of a memory problem
- Hey, some attack IP in China is taxing our resources trying dictionary attacks to gain access
- Dang, all I wanted was to run a few lines of code so I could return playerid records from the database

Business Case for Embracing Lambda for APIs

- Replace the deep and complicated world of server administration with just cutting straight to the code you want executed, plus light Lambda configuration
- Don’t pay for servers to sit idle, only pay by the exact amount of CPU and RAM your actions consume and not a byte-second more
- Lambdas are inherently scalable by default: the function can be called 10 times or 10,000,000 times
- Lambda lends itself so well to RESTful approach that it is intrinsically harder for teams to engage in bad API design

Signs an API Function Might Not Be Right for Lambda

- No ability to administer the environment means not all binaries can be available
- Maximum 1536MB memory consumption in one process puts a cap on solving certain kinds of problems or loading every library in the world
- Maximum 5 minute function runtime puts an upper bound on the kinds of long-running processes Lambda can serve
- The language is not supported yet (most notably PHP)

Hello World

Password

Sign In

Sign in to a different account

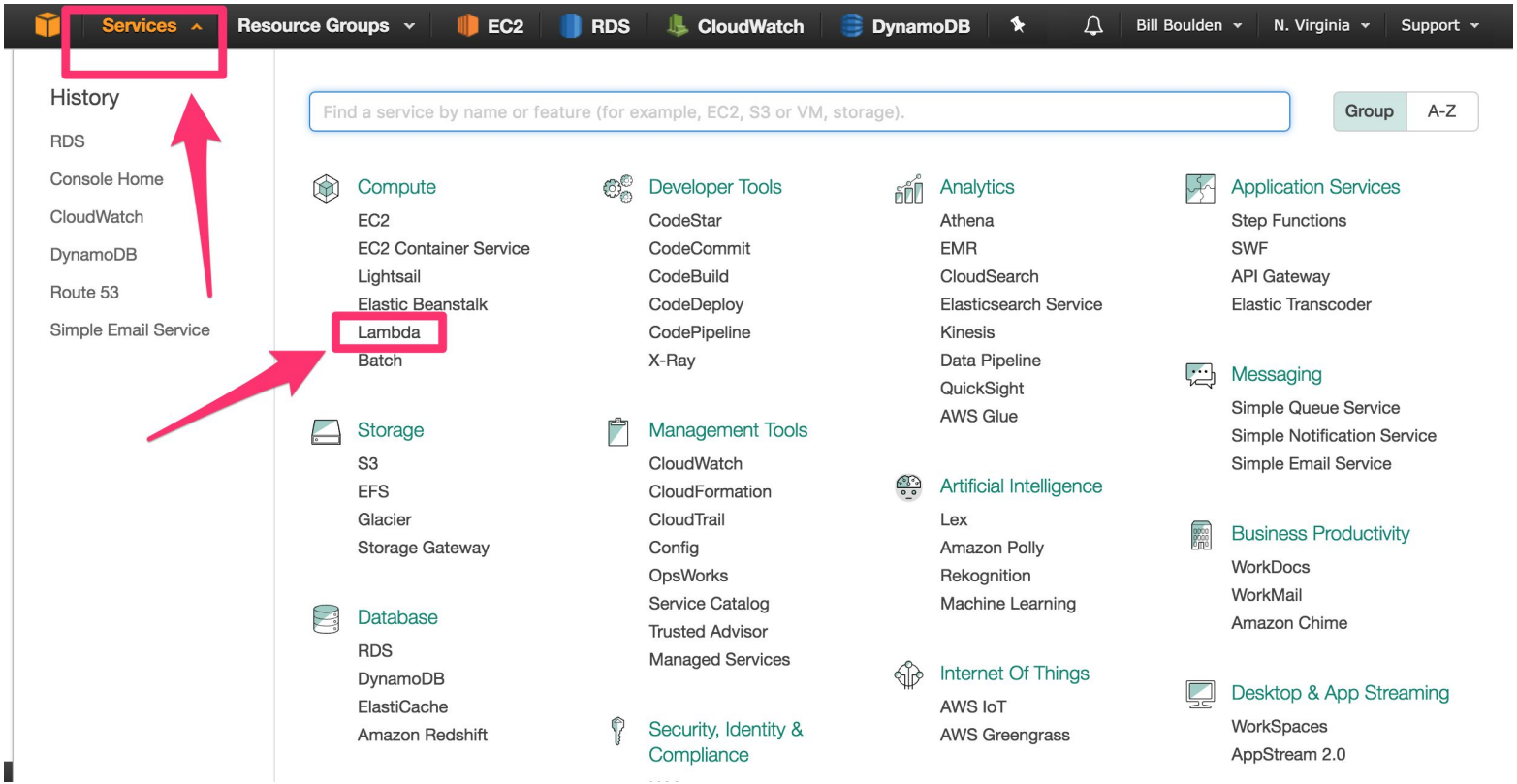
Forgot your password?

AWS Lambda

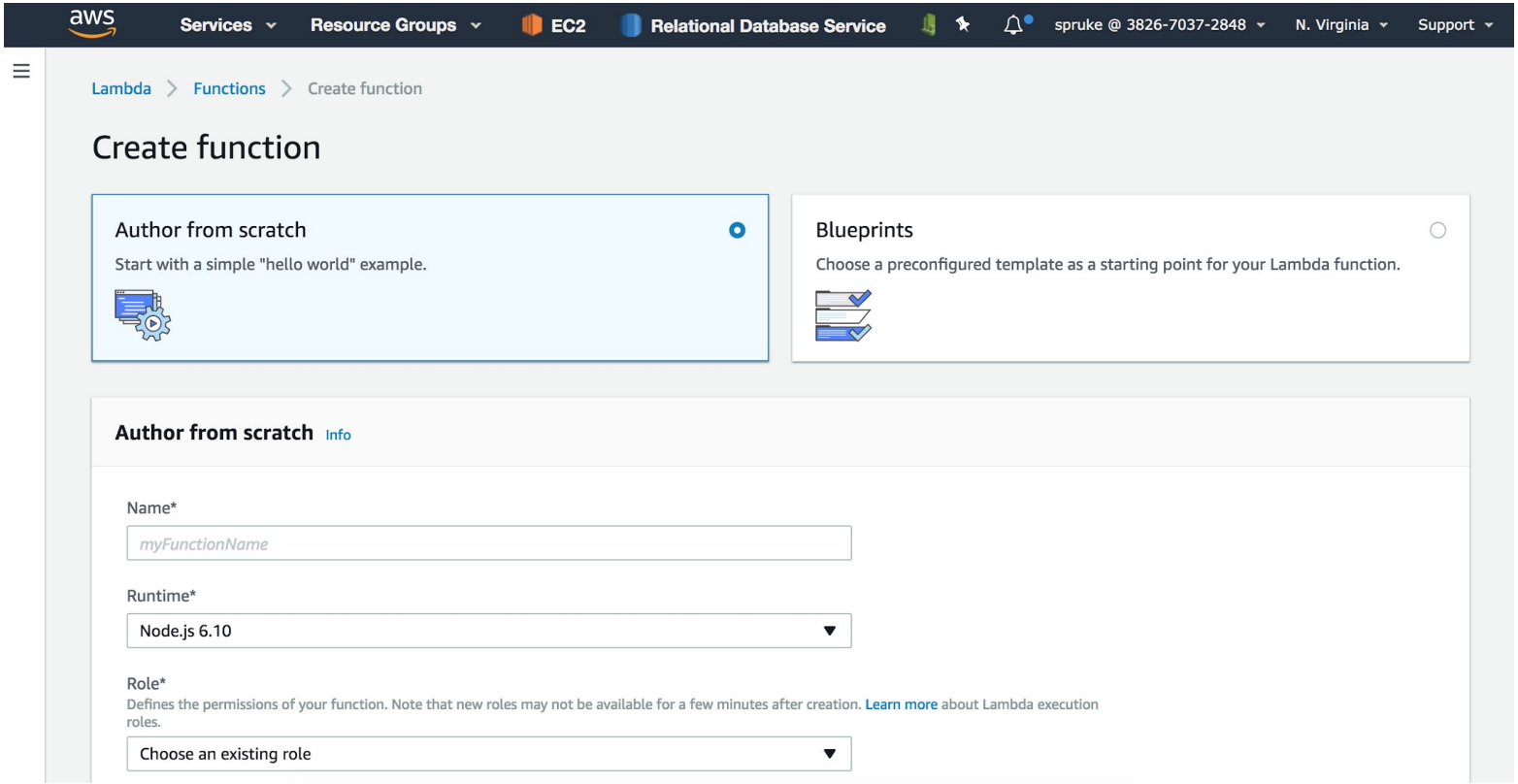
Run and scale code for Python, Node.js, Java, or C# without provisioning or managing servers

signin.aws.amazon.com

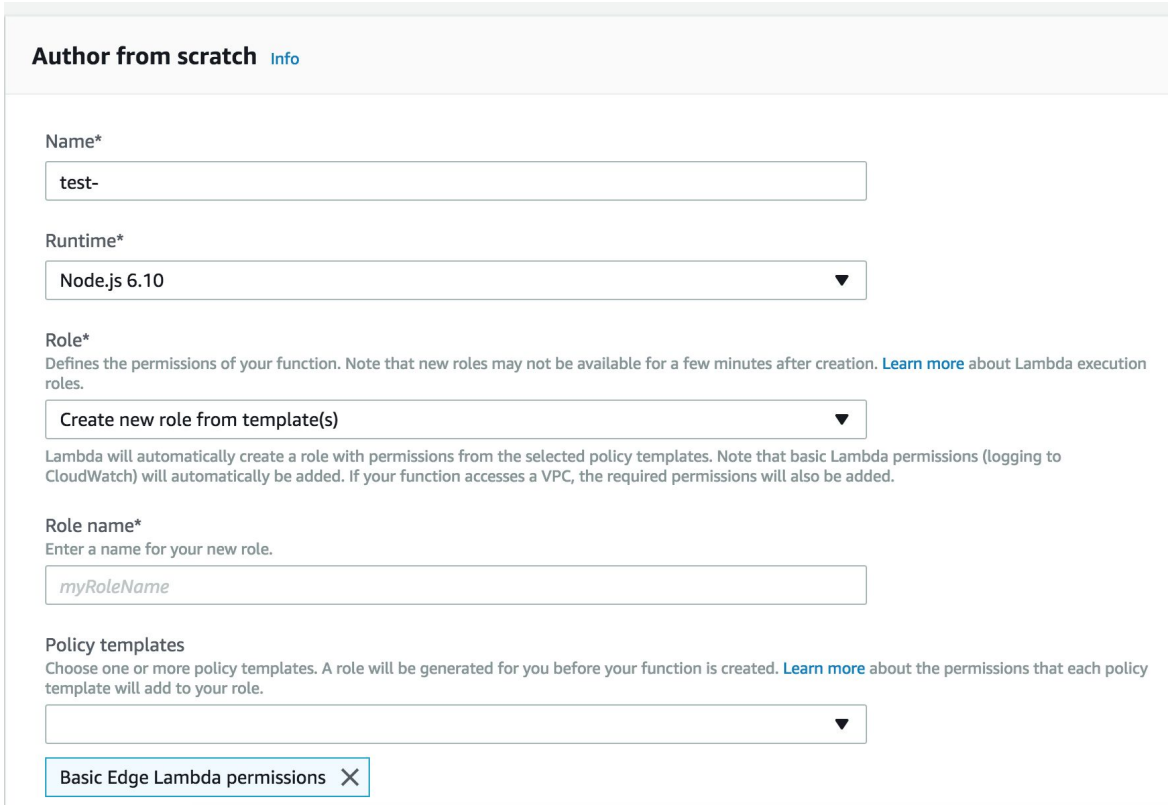
console.aws.amazon.com/lambda



“Create a Function”, then “Author from Scratch”

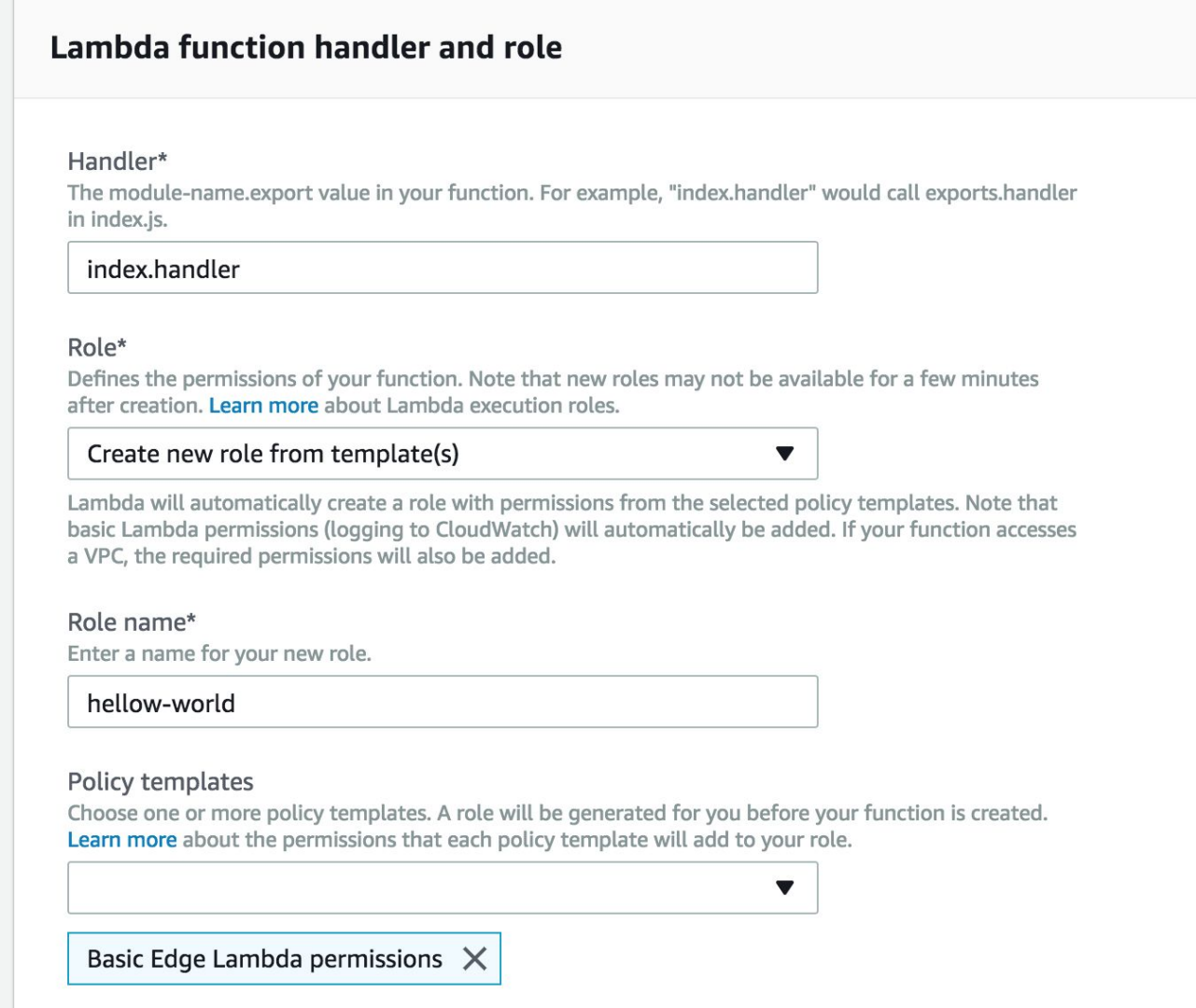


Guten tag, Velt. Hola, mundo.

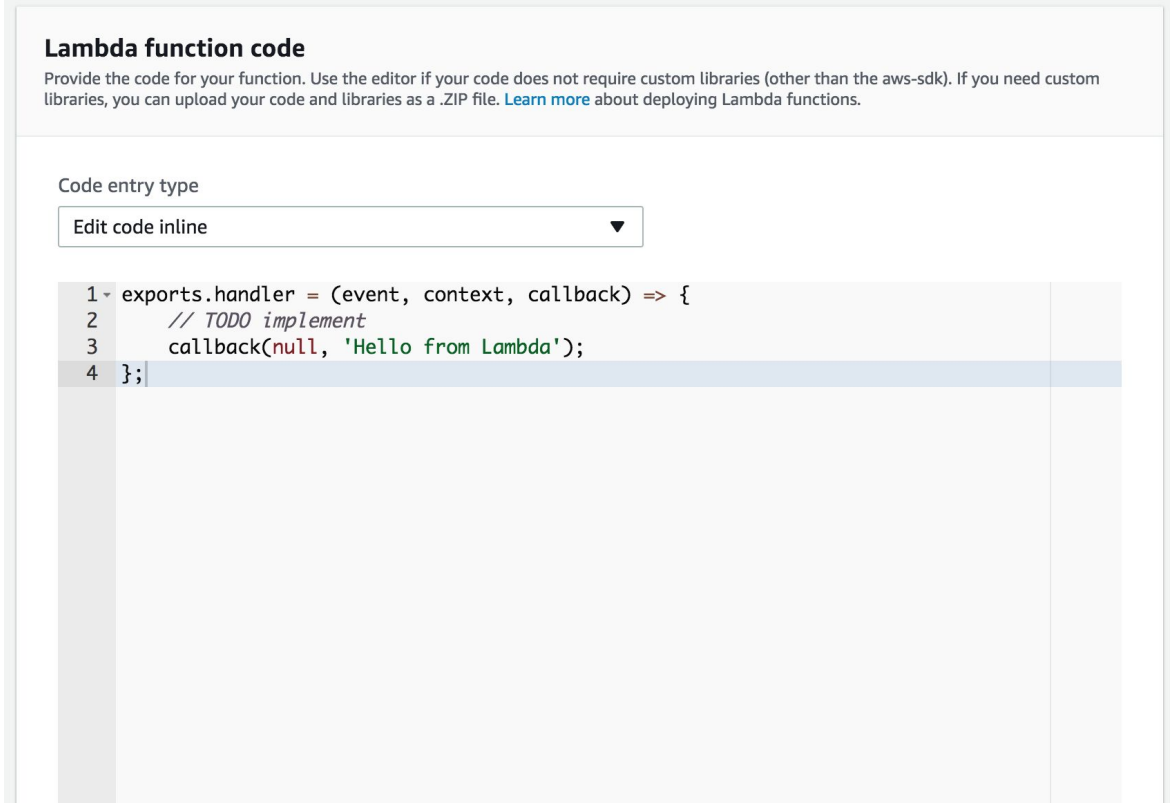


Roles

These will be a headache in advanced applications, once you begin connecting to a database or the like, but for now you can get by on Basic Lambda



Amazon’s way ahead of us with its default content



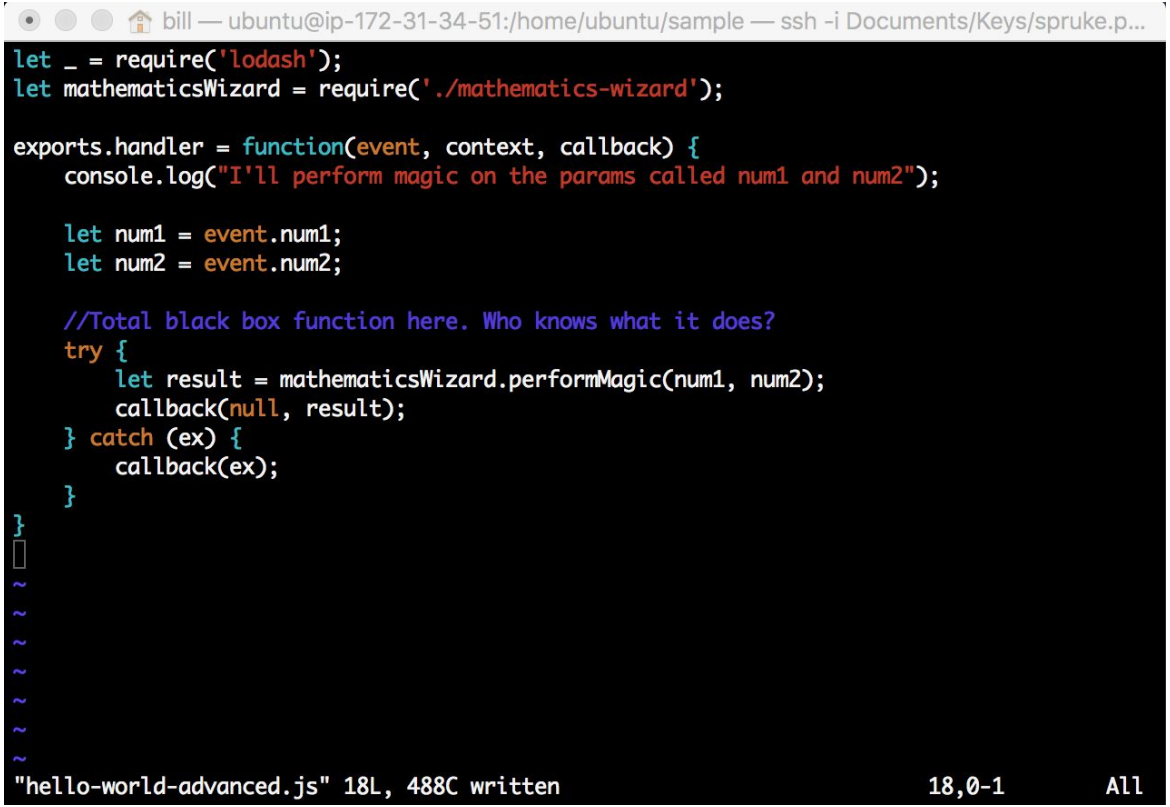
Now Let’s Create, and Test

Although, as a Hello World, it can’t really fail. ;)

Hello World, part 2

This time, with the console, using the parameters, using multiple files, and more details that are reminiscent of applied problem-solving

hello-world-advanced.js



Let’s Review Everything Going On Here

- We’ve included a node.js library, the popular Lodash
- We’ve included a library of our own, “mathematics-wizard.js”
- **event**—contains data passed into the Lambda as parameters. Can be JSON data from the invocation, or, if the Lambda is “triggered”, will contain details of the triggering event such as the SNS notification, DynamoDB event, etc.
- **context**—contains environment data about the Lambda such as .getRemainingTimeInMillis() or .memoryLimitInMB
- **callback**—in the node fashion, a function to call once you’re finished is provided, and takes the first parameter as failure information and the second as success information

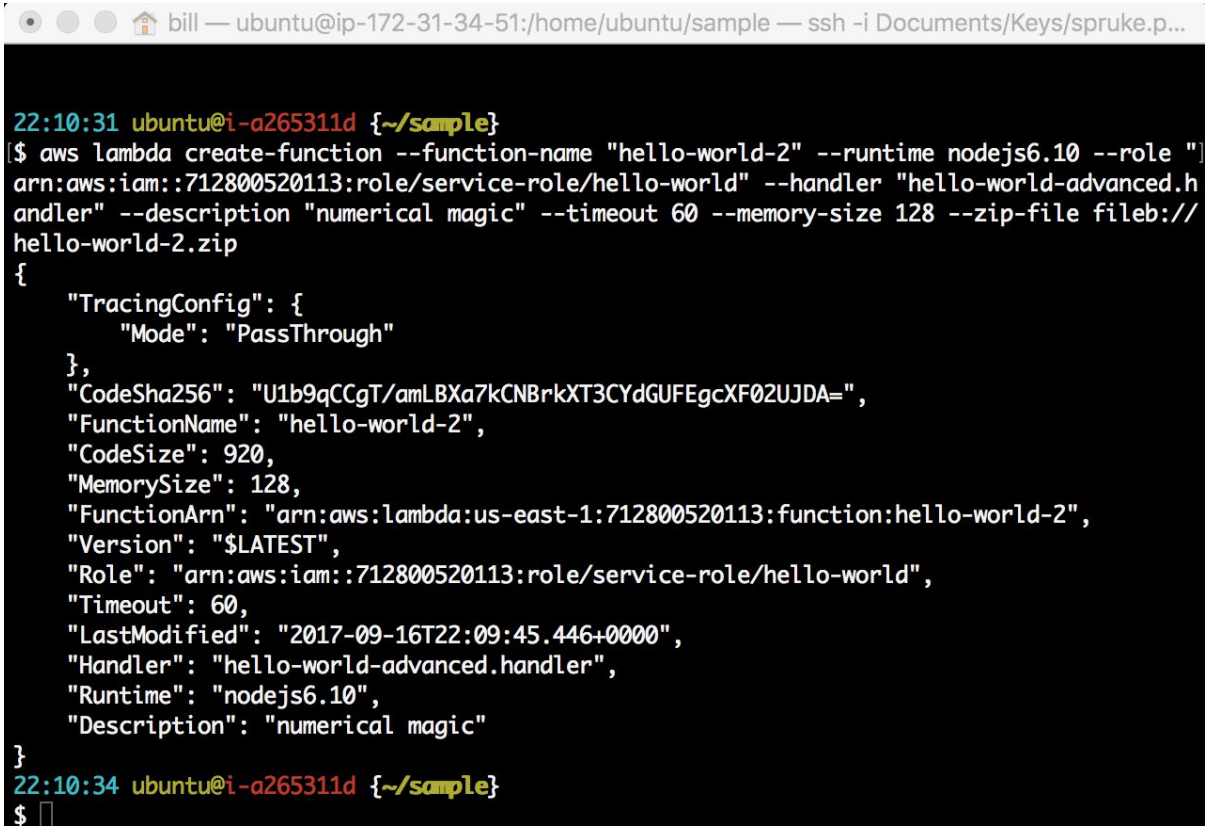
mathematics-wizard.js

Finally, npm install lodash in this directory...

Bundle this into a zip.

Deploy to Lambda With the Command Line Interface

Let's break down everything that's happening here, because there's a lot going on.

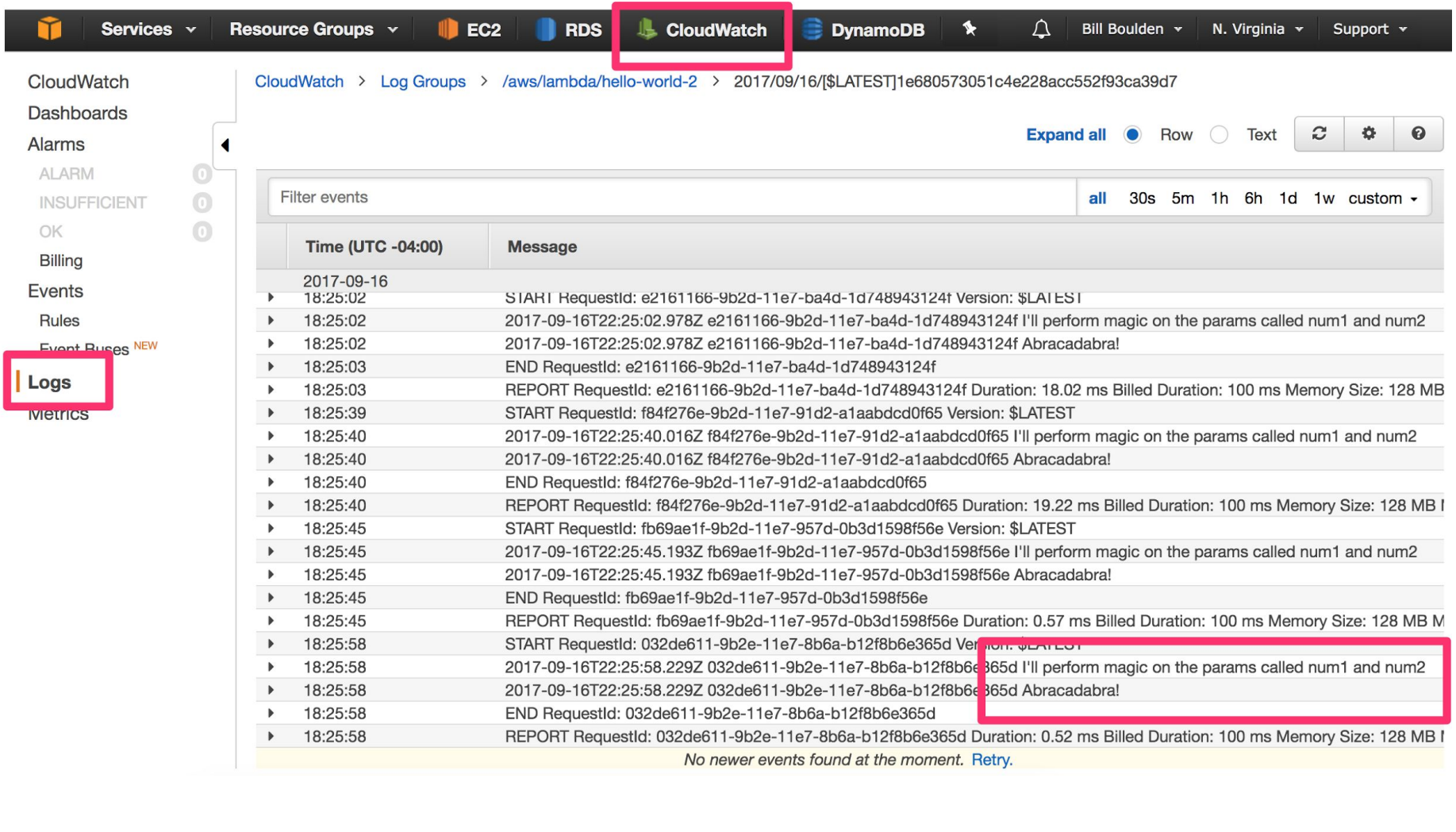


Invoke from CLI

Result is honestly not that helpful, because the output is base64 encoded.

To get non-base-64 logs, go look at the log that has been created for this function in CloudWatch for you.

Hacky command: `aws lambda invoke --function-name "hello-world-2" --payload='{"num1": 2, "num2": 2}' --log-type Tail --query=LogResult /tmp/answer.txt | sed -e 's/"//g' | base64 --decode`

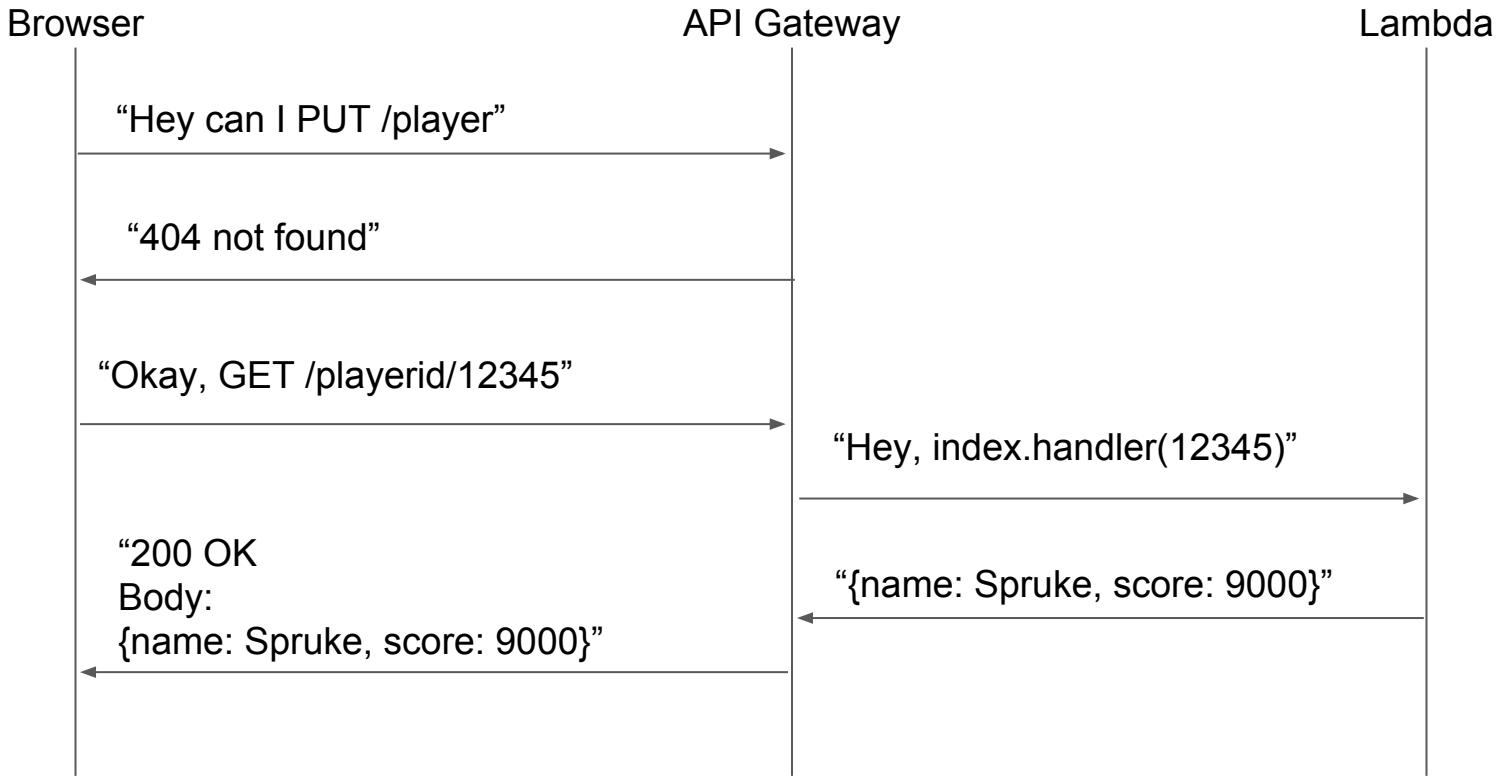

Services
Resource Groups
EC2
RDS
CloudWatch
DynamoDB
Bill Boulden
N. Virginia
Support
CloudWatch
Dashboards
Alarms
ALARM 0
INSUFFICIENT 0
OK 0
Billing
Events
Rules
Event Buses NEW
Logs
Metrics
CloudWatch > Log Groups > /aws/lambda/hello-world-2 > 2017/09/16/[$LATEST]1e680573051c4e228acc552f93ca39d7
Expand all
Row
Text
Filter events
all 30s 5m 1h 6h 1d 1w custom
Time (UTC -04:00)
Message
2017-09-16
18:25:02 START RequestId: e2161166-9b2d-11e7-ba4d-1d748943124f Version: $LATEST
18:25:02 2017-09-16T22:25:02.978Z e2161166-9b2d-11e7-ba4d-1d748943124f I'll perform magic on the params called num1 and num2
18:25:02 2017-09-16T22:25:02.978Z e2161166-9b2d-11e7-ba4d-1d748943124f Abracadabra!
18:25:03 END RequestId: e2161166-9b2d-11e7-ba4d-1d748943124f
18:25:03 REPORT RequestId: e2161166-9b2d-11e7-ba4d-1d748943124f Duration: 18.02 ms Billed Duration: 100 ms Memory Size: 128 MB
18:25:39 START RequestId: f84f276e-9b2d-11e7-91d2-a1aabdcd0f65 Version: $LATEST
18:25:40 2017-09-16T22:25:40.016Z f84f276e-9b2d-11e7-91d2-a1aabdcd0f65 I'll perform magic on the params called num1 and num2
18:25:40 2017-09-16T22:25:40.016Z f84f276e-9b2d-11e7-91d2-a1aabdcd0f65 Abracadabra!
18:25:40 END RequestId: f84f276e-9b2d-11e7-91d2-a1aabdcd0f65
18:25:40 REPORT RequestId: f84f276e-9b2d-11e7-91d2-a1aabdcd0f65 Duration: 19.22 ms Billed Duration: 100 ms Memory Size: 128 MB
18:25:45 START RequestId: fb69ae1f-9b2d-11e7-957d-0b3d1598f56e Version: $LATEST
18:25:45 2017-09-16T22:25:45.193Z fb69ae1f-9b2d-11e7-957d-0b3d1598f56e I'll perform magic on the params called num1 and num2
18:25:45 2017-09-16T22:25:45.193Z fb69ae1f-9b2d-11e7-957d-0b3d1598f56e Abracadabra!
18:25:45 END RequestId: fb69ae1f-9b2d-11e7-957d-0b3d1598f56e
18:25:45 REPORT RequestId: fb69ae1f-9b2d-11e7-957d-0b3d1598f56e Duration: 0.57 ms Billed Duration: 100 ms Memory Size: 128 MB
18:25:58 START RequestId: 032de611-9b2e-11e7-8b6a-b12f8b6e365d
18:25:58 2017-09-16T22:25:58.229Z 032de611-9b2e-11e7-8b6a-b12f8b6e365d I'll perform magic on the params called num1 and num2
18:25:58 2017-09-16T22:25:58.229Z 032de611-9b2e-11e7-8b6a-b12f8b6e365d Abracadabra!
18:25:58 END RequestId: 032de611-9b2e-11e7-8b6a-b12f8b6e365d
18:25:58 REPORT RequestId: 032de611-9b2e-11e7-8b6a-b12f8b6e365d Duration: 0.52 ms Billed Duration: 100 ms Memory Size: 128 MB
No newer events found at the moment. Retry.

Introduction to API Gateway

API Gateway Is an API Management Tool or ”Facade”

- You explicitly and clearly define the rules for the content of the requests and responses, and it will enforce and manage them
- Transforms HTTP requests into mapped JSON, etc. for the backend
- Isolates the backend from the wild west of the HTTP wilderness
- Explicitly disallows all headers, methods, response codes, paths, etc. that are not allowed by the configuration
    - Will help by default avoid (although nothing ever proofs one completely against) security oversights such as injections or unanticipated paths
- Notable alternatives include Apigee, recently purchased by Google

Sample (simplified) API Gateway + Lambda Flow

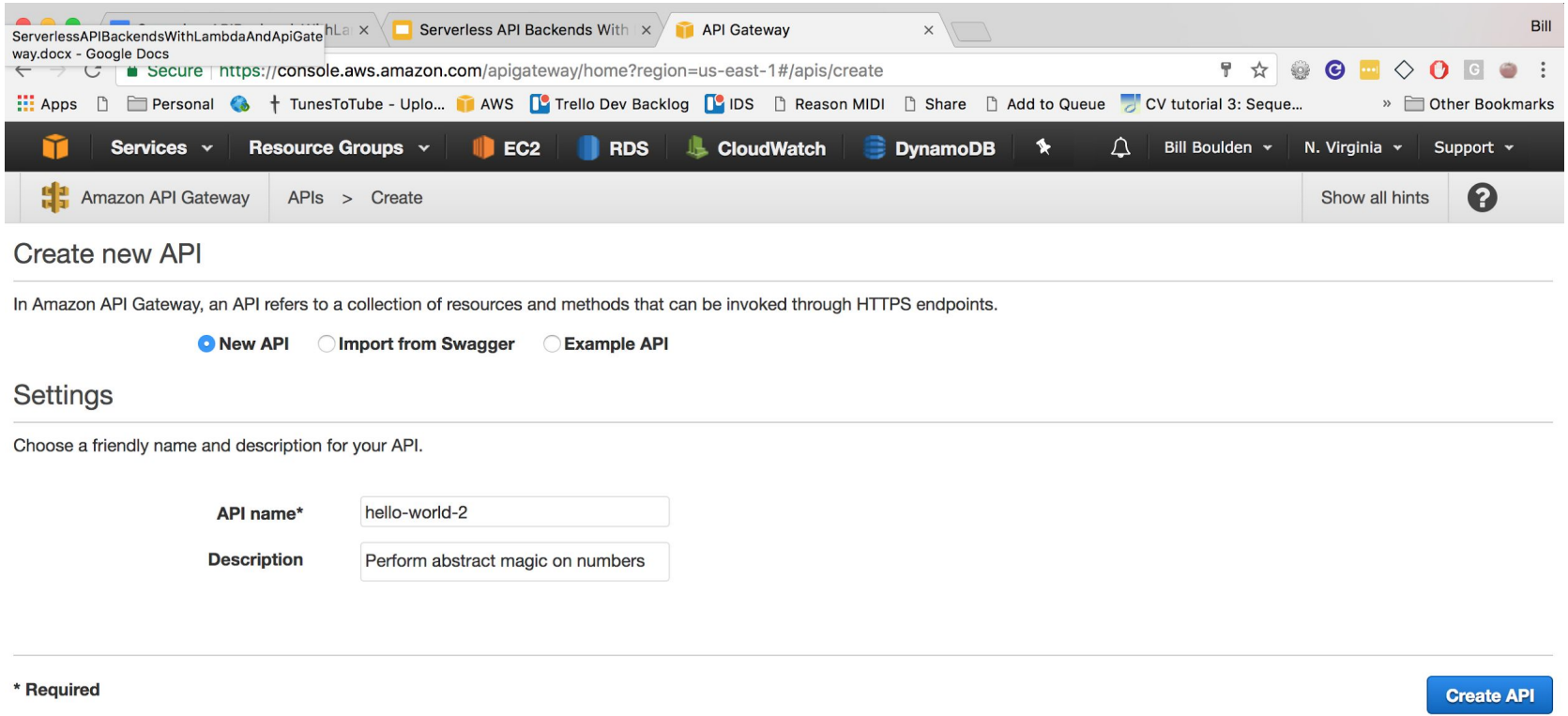


How Does API Gateway Help Us Create a Serverless API Model?

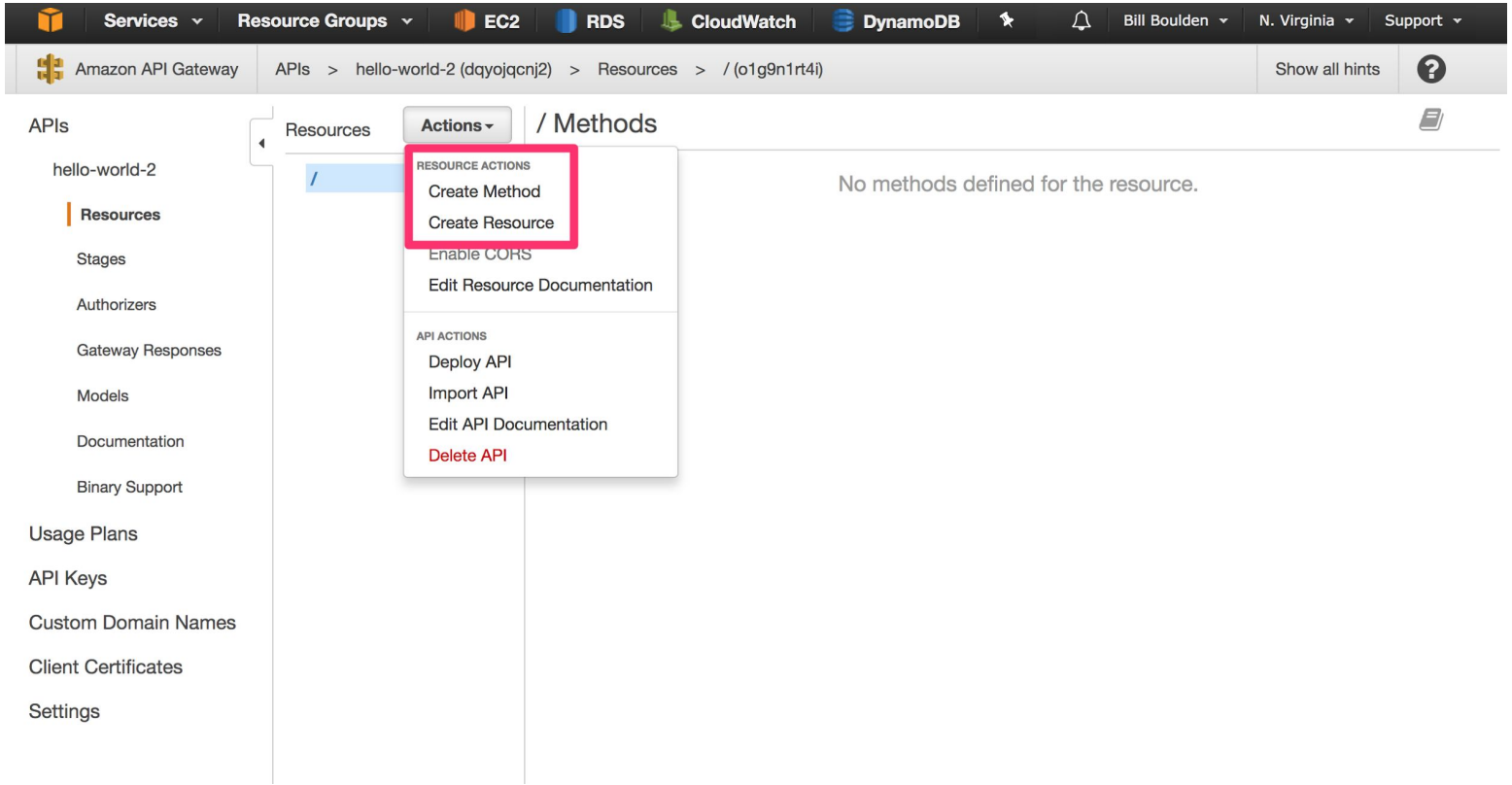
- Lambdas are “Function As A Service”, but you’re not going to let the Internet just call Functions for you… that’s a Bad Idea™ :)
- Need this intermediary layer to translate between the languages of HTTP/headers/methods and JSON/code/objects
- What does this “replace” from the traditional serverful model?
    - The NGNIX or Apache config that routes some paths to a running process
    - The cgi-bin directory structure or the express process that routes some functions to be handlers for some calls

Hooking It Up in front of Hello World 2

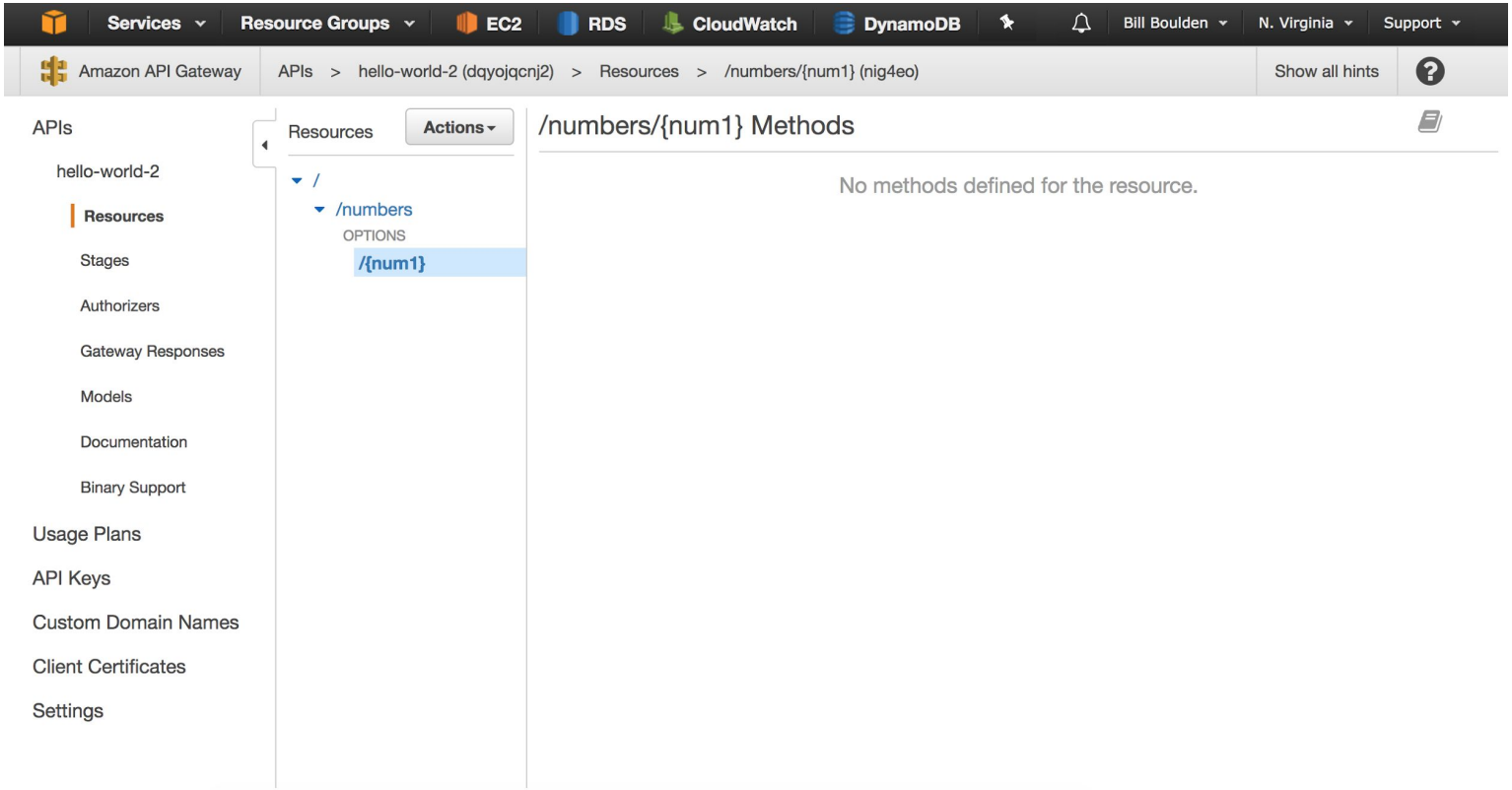
Create A New API Gateway



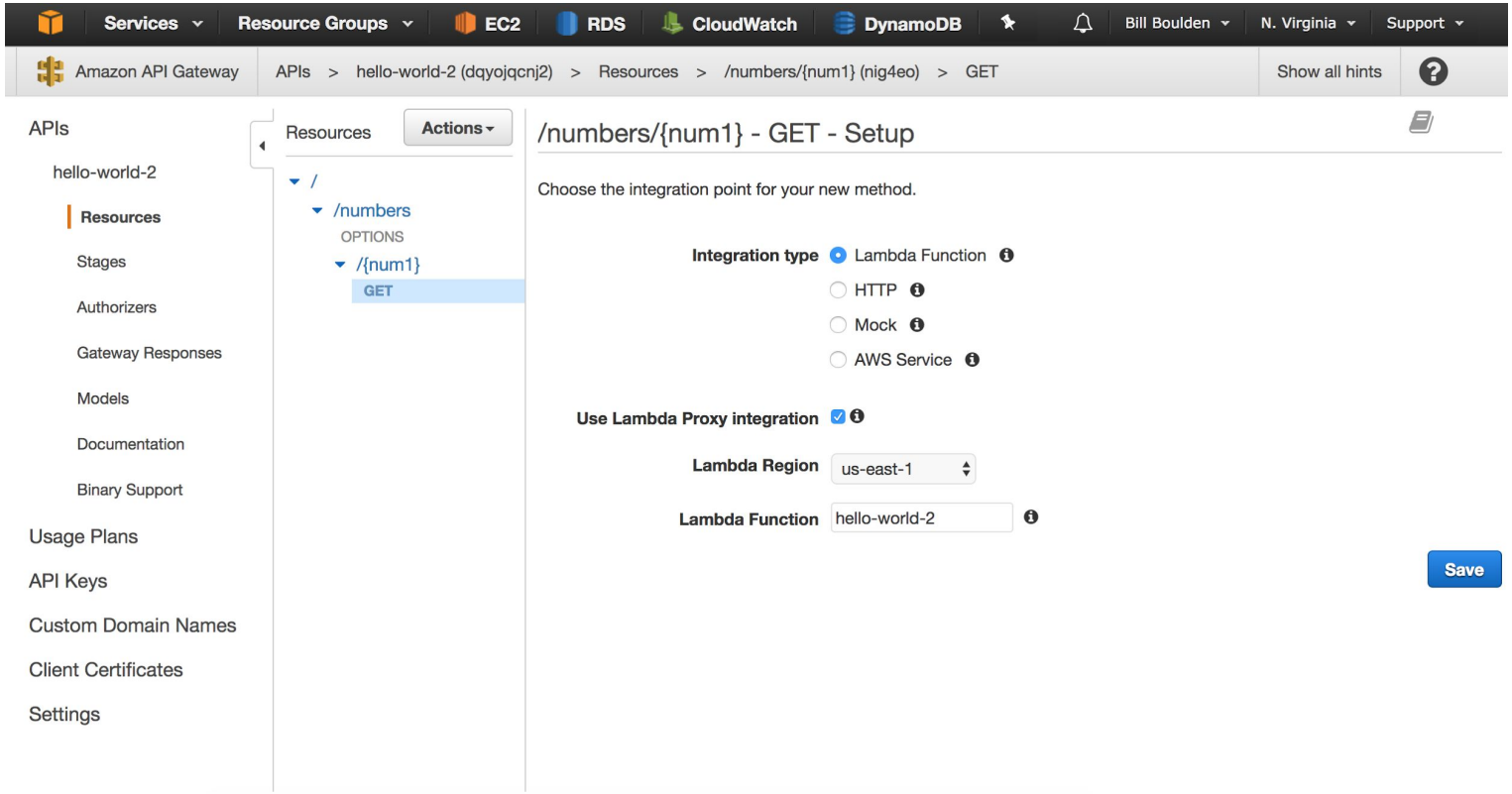
Methods & Resources



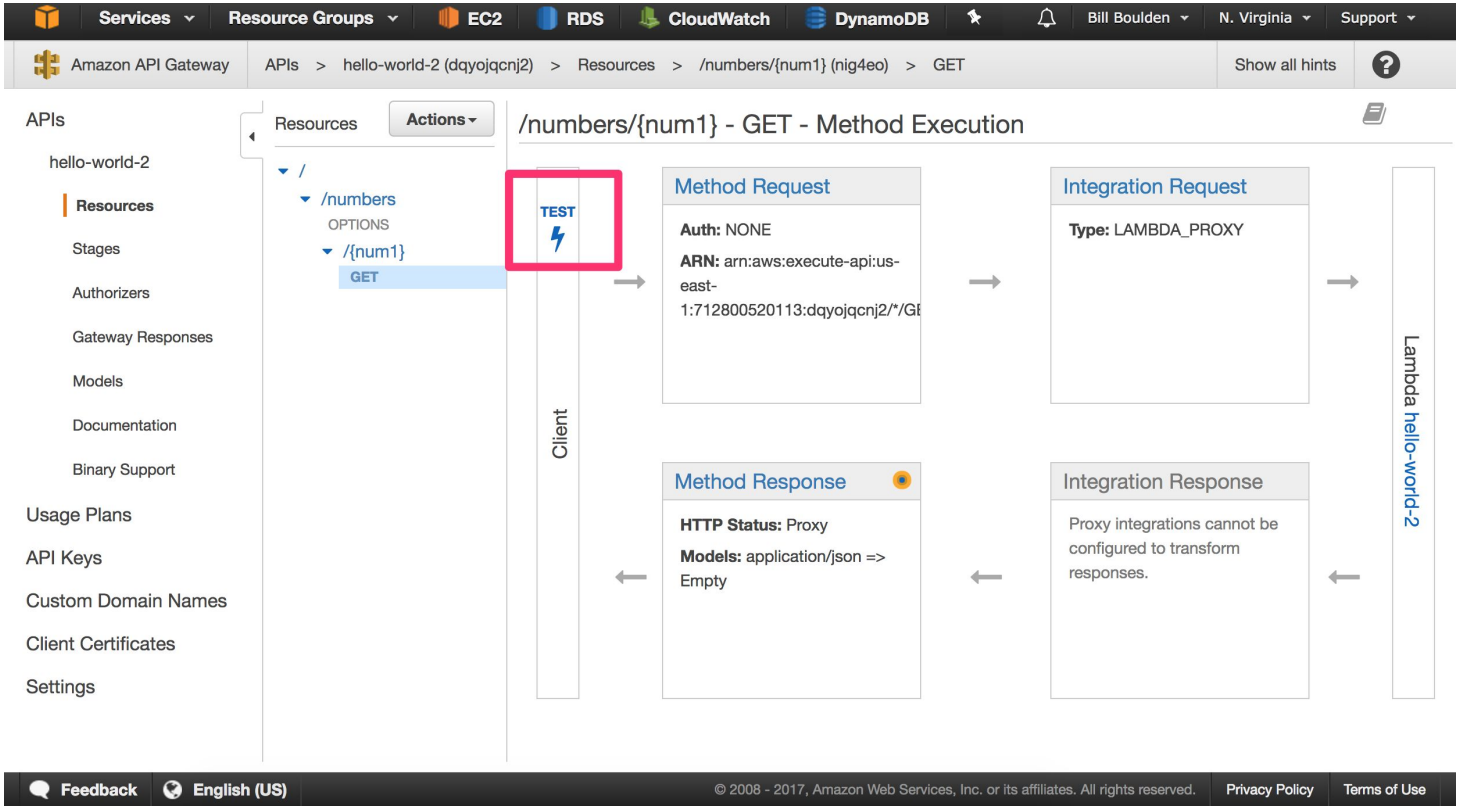
Create Resource /numbers, under that, /{num1}



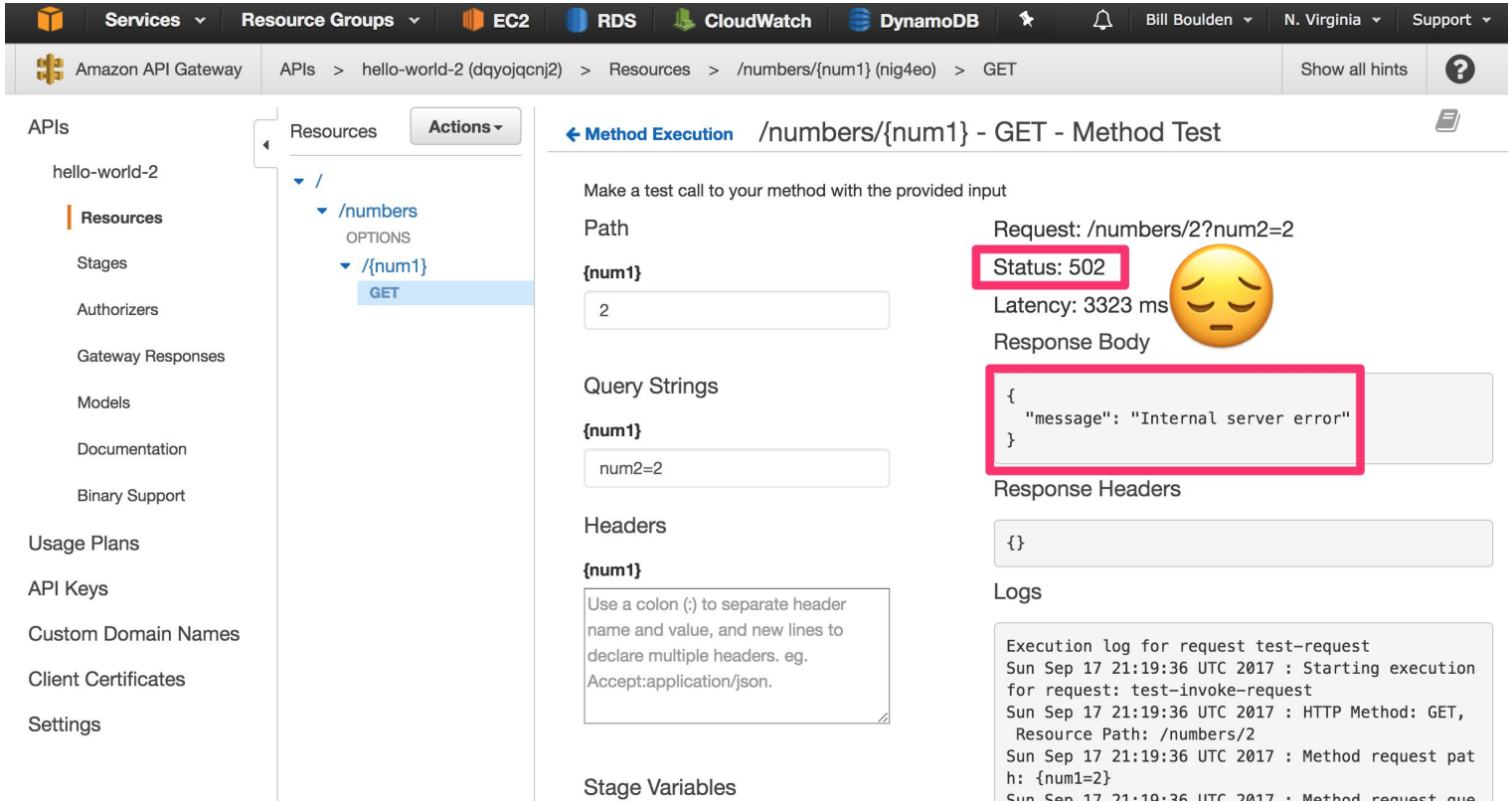
And Then Give {num1} Path a GET Method



Neat. Now Let's Try It Out.



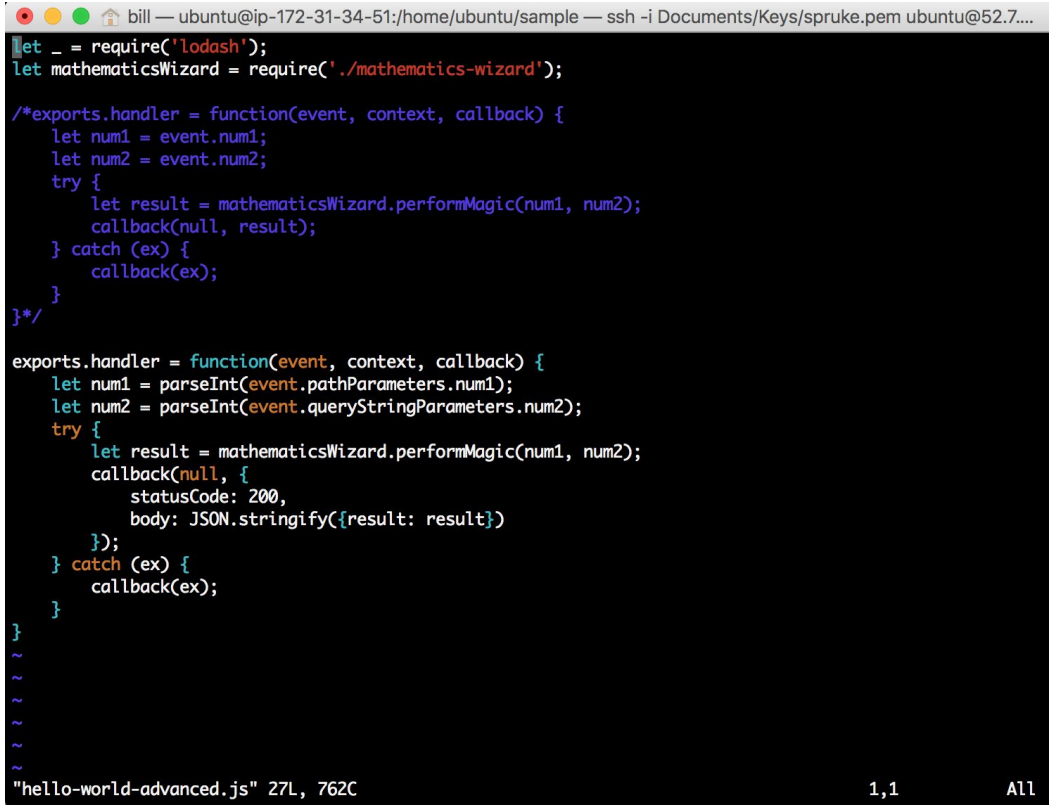
Not So Hot. Log Says “Malformed Proxy Response”



Using “Lambda Proxy Mode”

- If you’re going to use Lambda Proxy Mode, API Gateway won’t map parameters for you
- Instead, it will deliver “event” with a dump of request info, and expect a specific format back in response
- We’re going to do this Lambda Proxy Mode way first, and then the more detailed way next as a deep dive :)

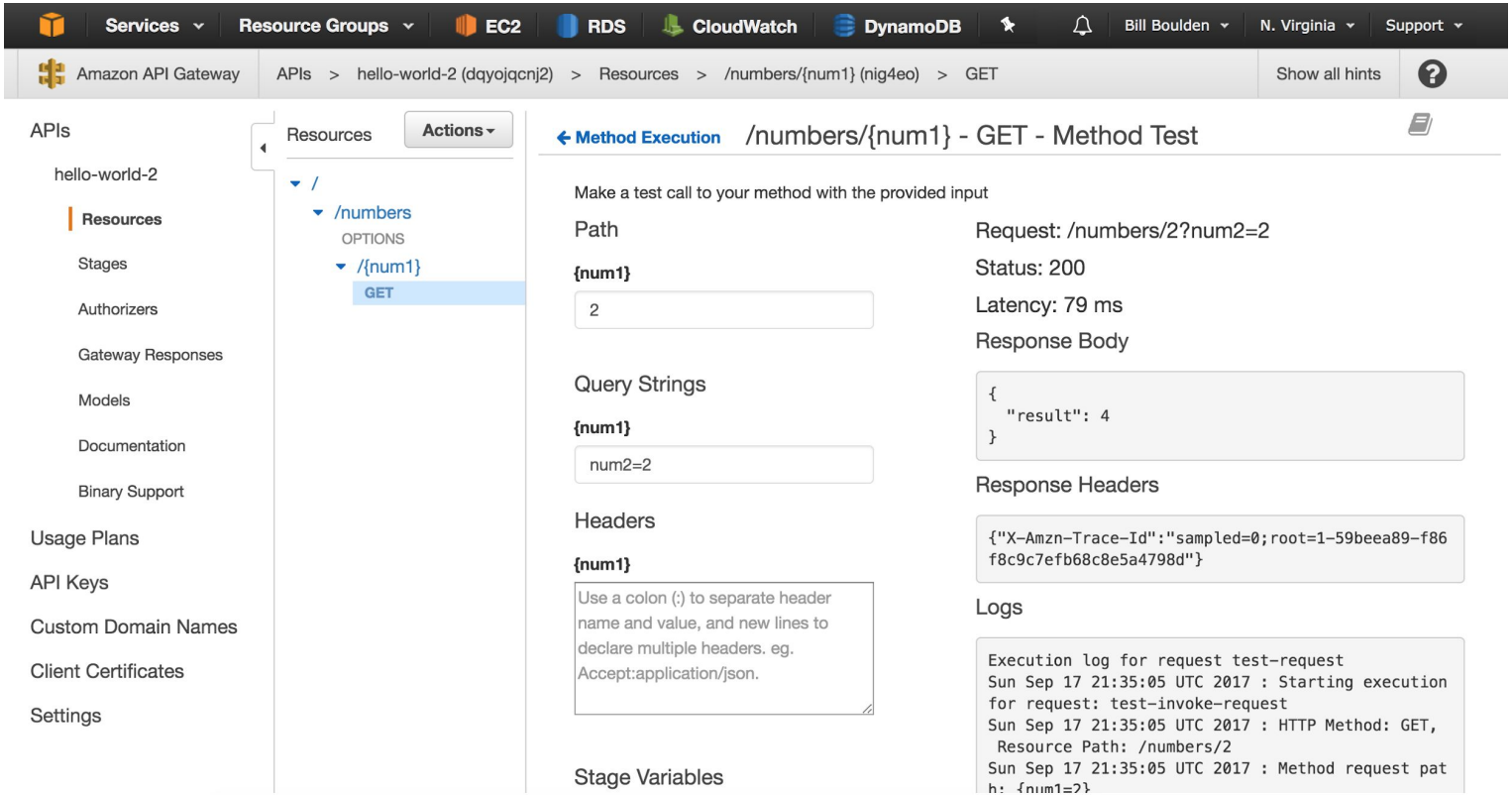
Outfitting Our Function for the Specific Proxy Use



Notice Some Changes, and Reupload

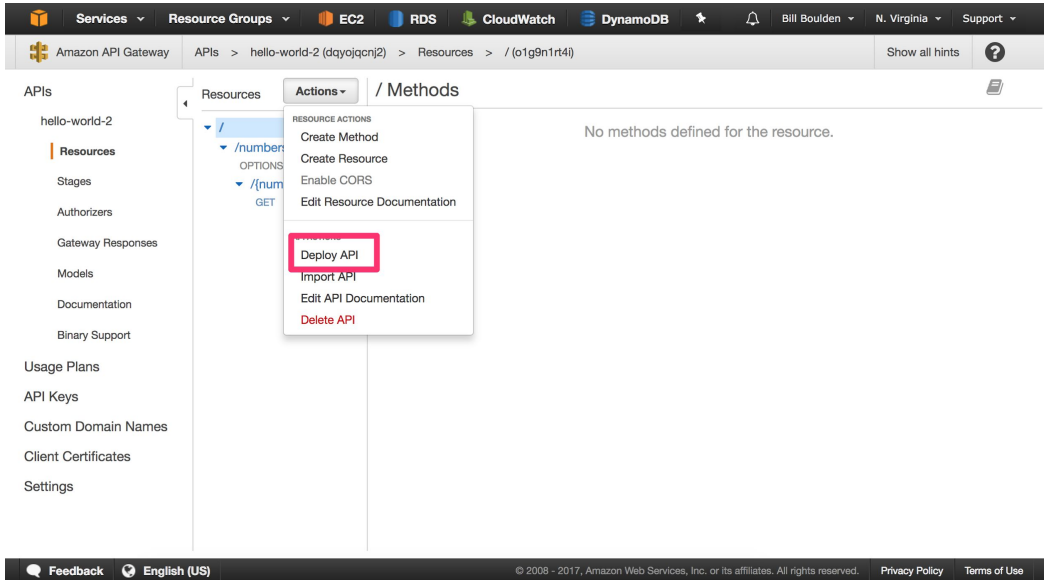
- **event** now contains some request-type objects with further details
- The return callback passes an object rather than a single value, with statusCode and Body at a minimum (optional: headers, encoding, etc)
- Delete your local zip file and recreate it with new contents
- Then update the Lambda with:
  ```
  aws lambda update-function-code --function-name
  "hello-world-2" --zip-file fileb://hello-world-2.zip
  ```

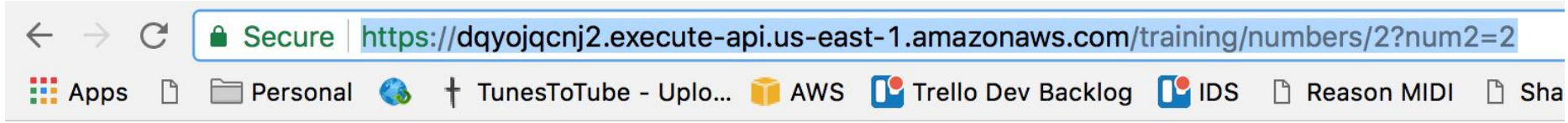
Now, on the Next Test, It's All Good!



To Use This API In The Wild...

- Use the format https://{api-id}.execute-api.{region}.amazonaws.com/{stage}
- A stage will be created once you Deploy API




Secure
https://dqyojqcnj2.execute-api.us-east-1.amazonaws.com/training/numbers/2?num2=2
Apps
Personal
TunesToTube - Uplo...
AWS
Trello Dev Backlog
IDS
Reason MIDI
Sha


```
{
    result: 4
}
```

What Did We Just Learn?

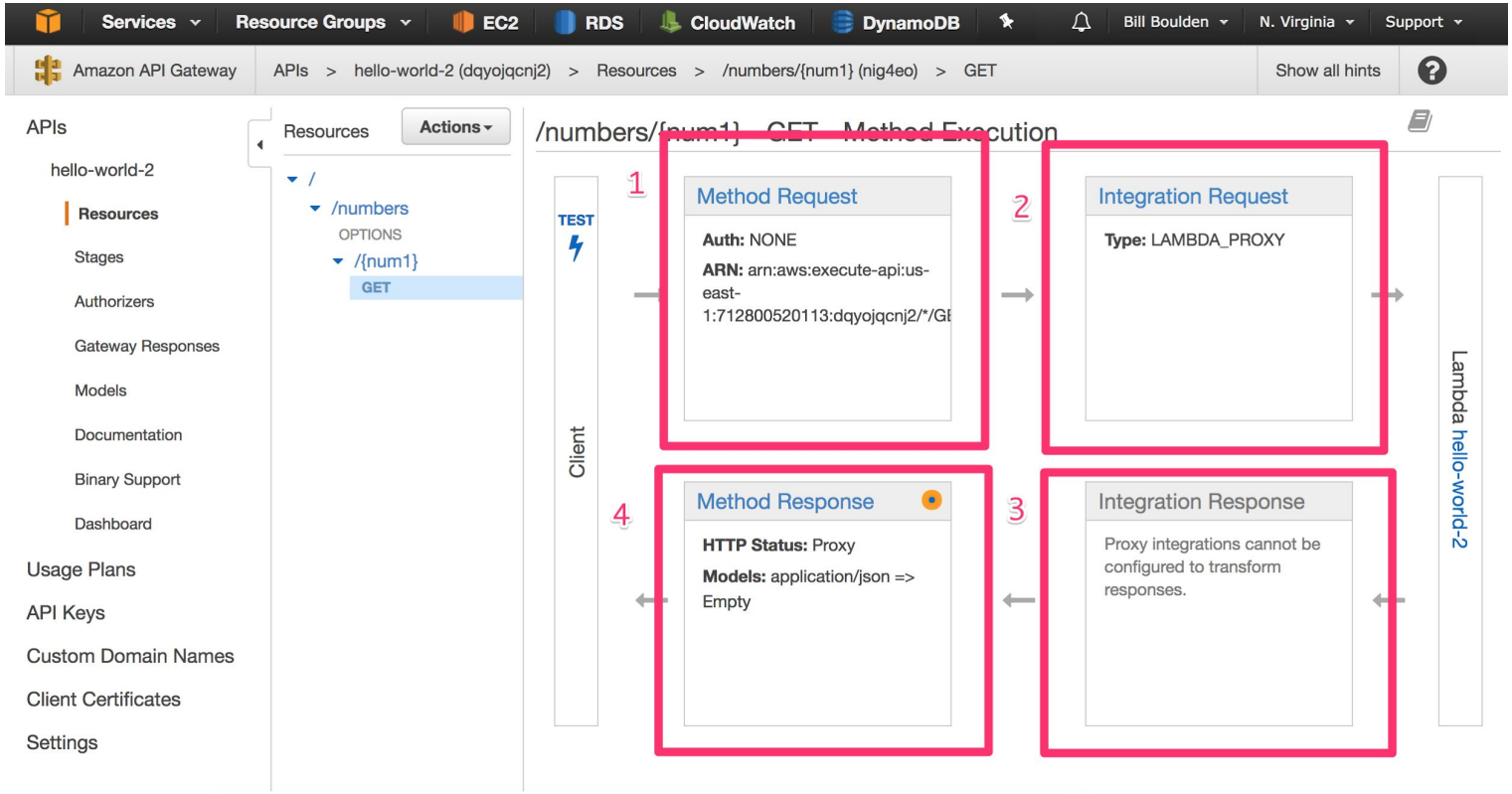
- We created a web endpoint that accepts GET requests and then does computation behind the scenes to return a response… all without running a server
- We used one query string parameter and one path parameter
- However, we are forwarding the entire request payload to the endpoint, so not taking full advantage of API Gateway's ability to isolate the interface of the API to the outside world, from the Lambda code running behind the scenes

Let's go back and restore our Lambda to the isolated, pure style it had before

bill — ubuntu@ip-172-31-34-51:/home/ubuntu/sample — ssh -i Documents/Keys/spruke.pem ubuntu@52.7....

```
let _ = require('lodash');
let mathematicsWizard = require('./mathematics-wizard');

exports.handler = function(event, context, callback) {
    let num1 = event.num1;
    let num2 = event.num2;
    try {
        let result = mathematicsWizard.performMagic(num1, num2);
        callback(null, result);
    } catch (ex) {
        callback(ex);
    }
};

/*exports.handler = function(event, context, callback) {
    let num1 = parseInt(event.pathParameters.num1);
    let num2 = parseInt(event.queryStringParameters.num2);
    try {
        let result = mathematicsWizard.performMagic(num1, num2);
        callback(null, {
            statusCode: 200,
            body: JSON.stringify({result: result})
        });
    } catch (ex) {
        callback(ex);
    }
};*/
~
~
~
~
~
~
"hello-world-advanced.js" 27L, 764C written                    5,5           All
```

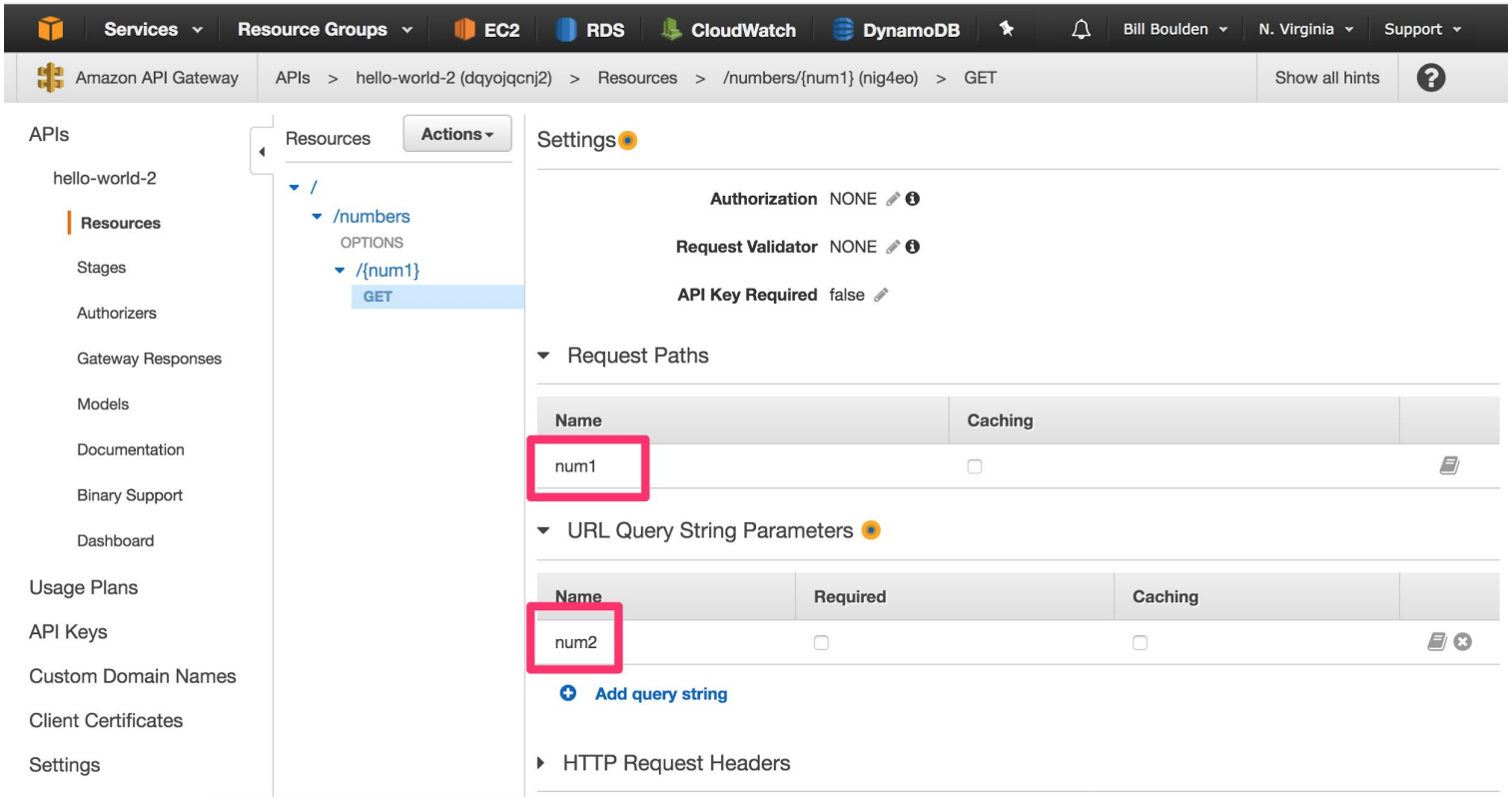
We're Going To Manually Configure In This Order



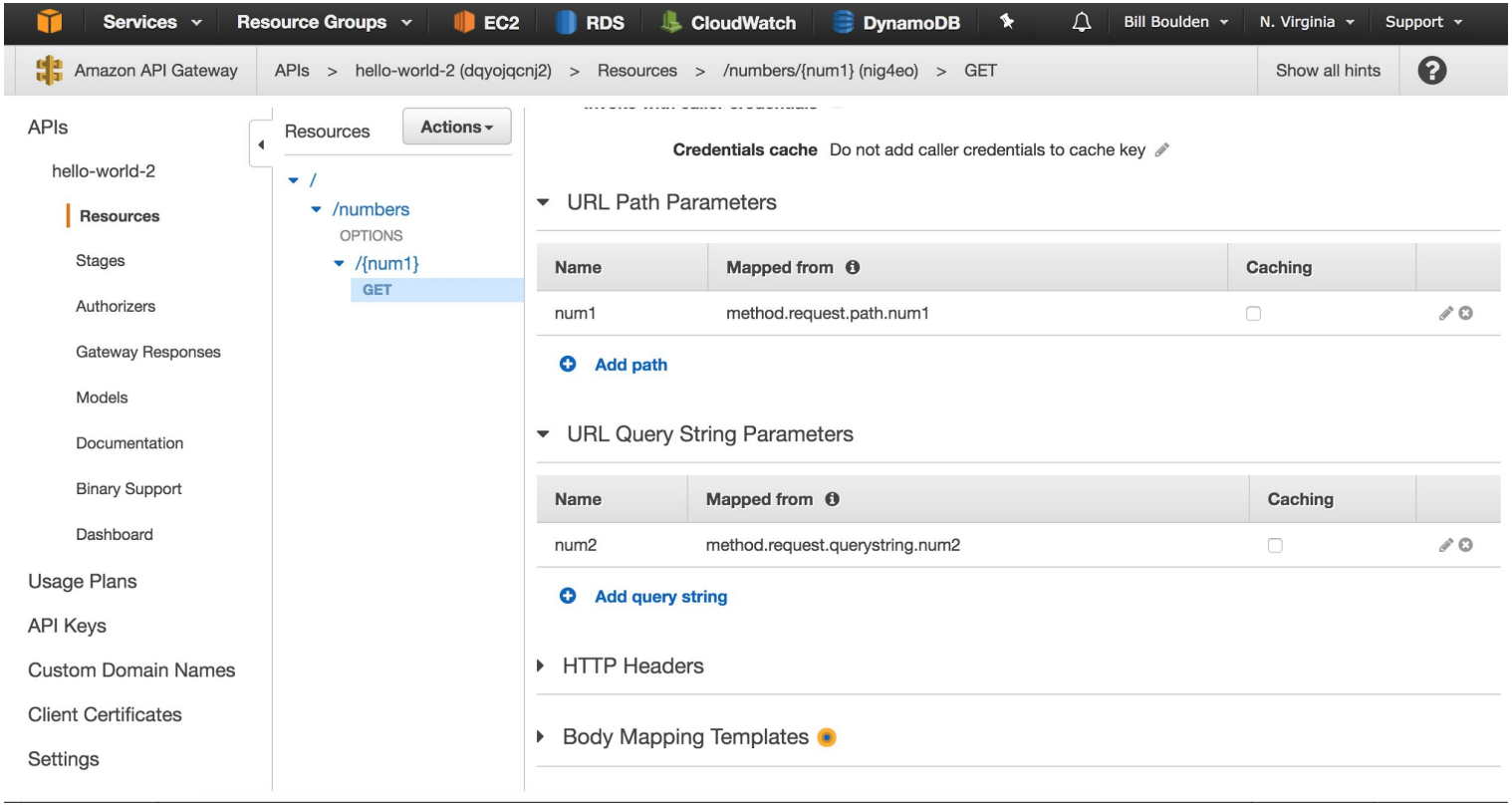
The Four Parts of the Lifecycle

1. Method Request: how I deal with the HTTP from the outside world, parsing it into API Gateway information
2. Integration Request: how I prepare the request for the backend, parsing API Gateway information into JSON or event information
3. Integration Response: how I prepare the response from the backend, transforming the object returned into API Gateway information
4. Method Response: how I send an HTTP response back to the caller, transforming the API Gateway information into an HTTP message

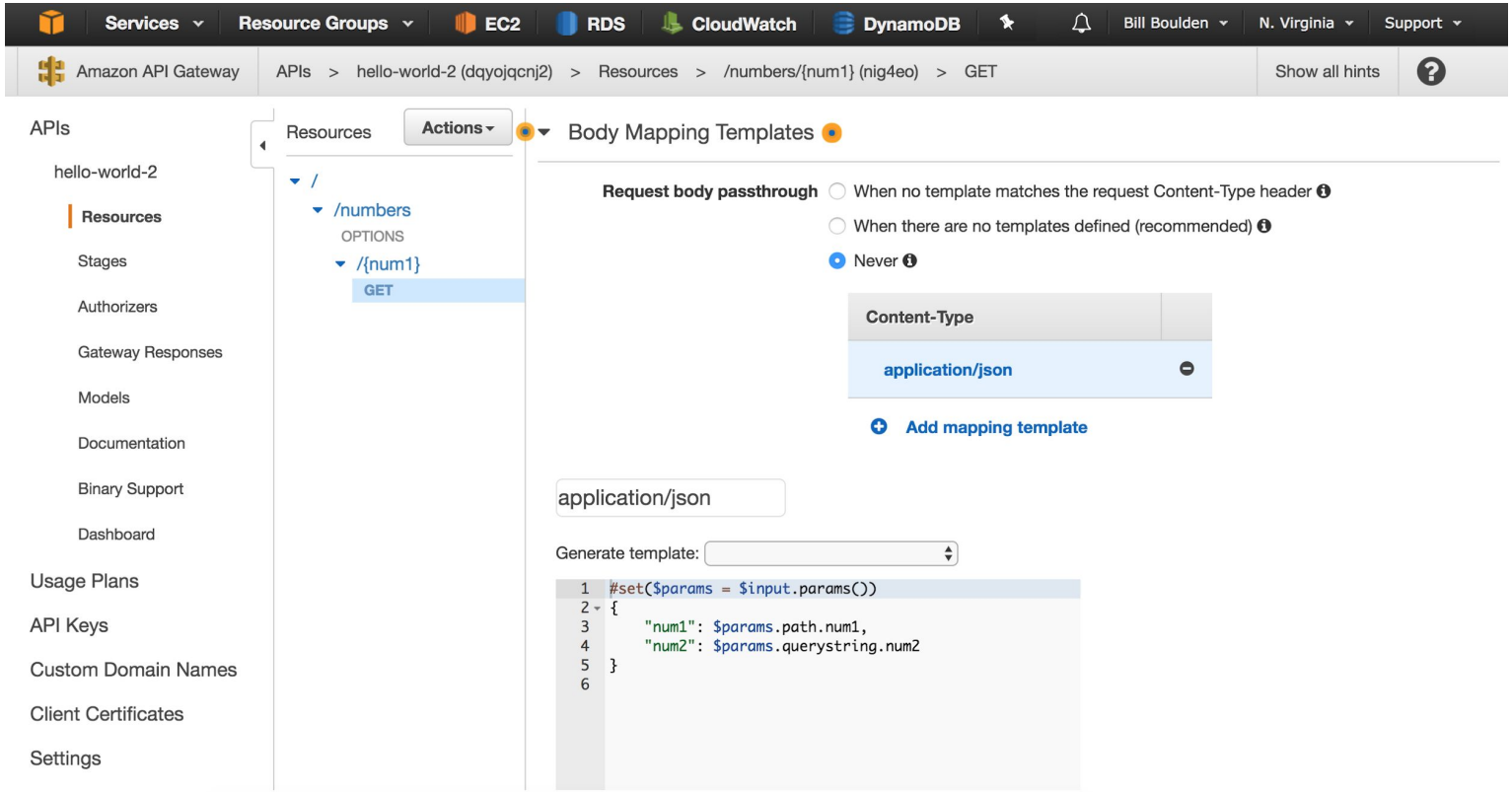
Method Request



Integration Request



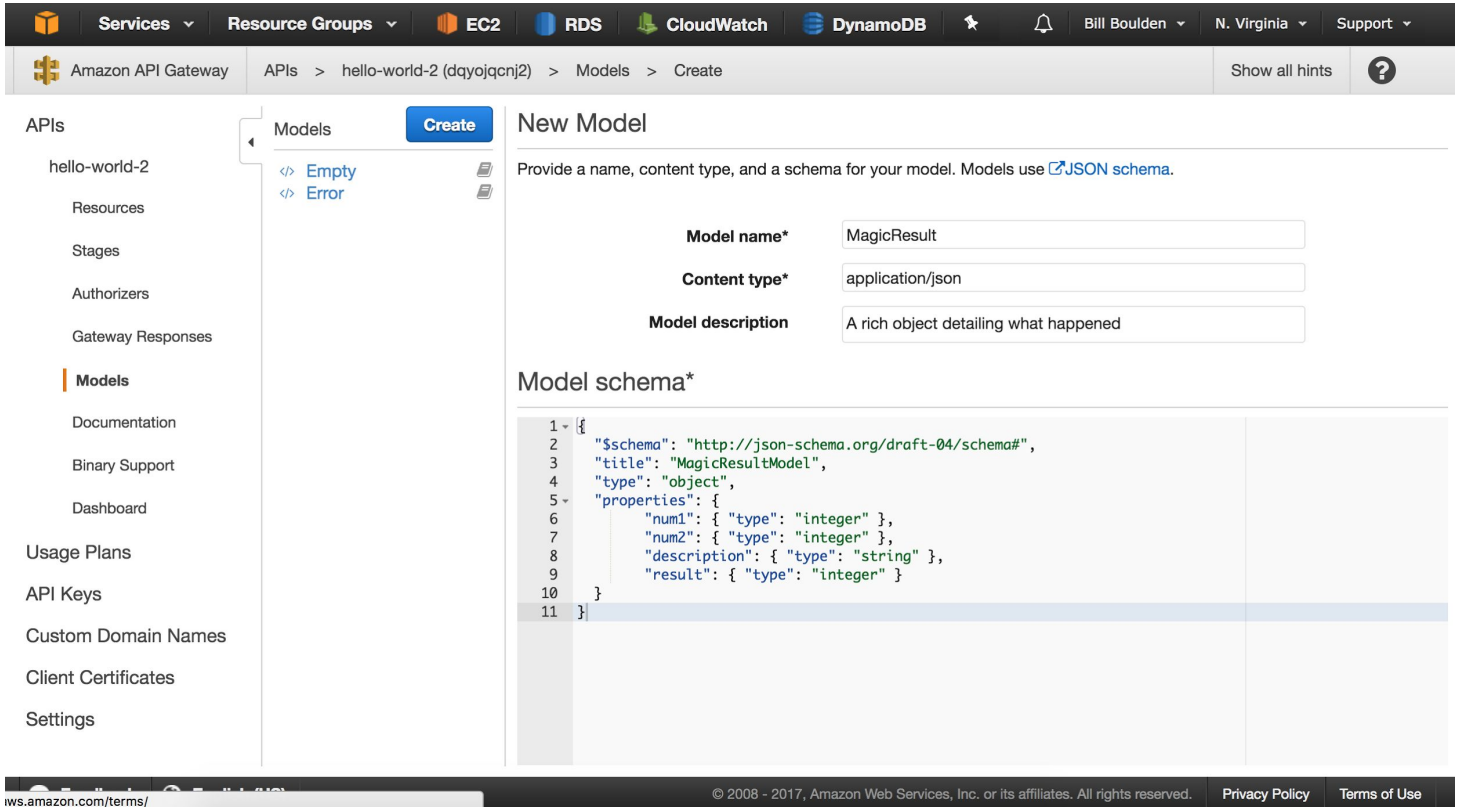
Integration Request pt 2



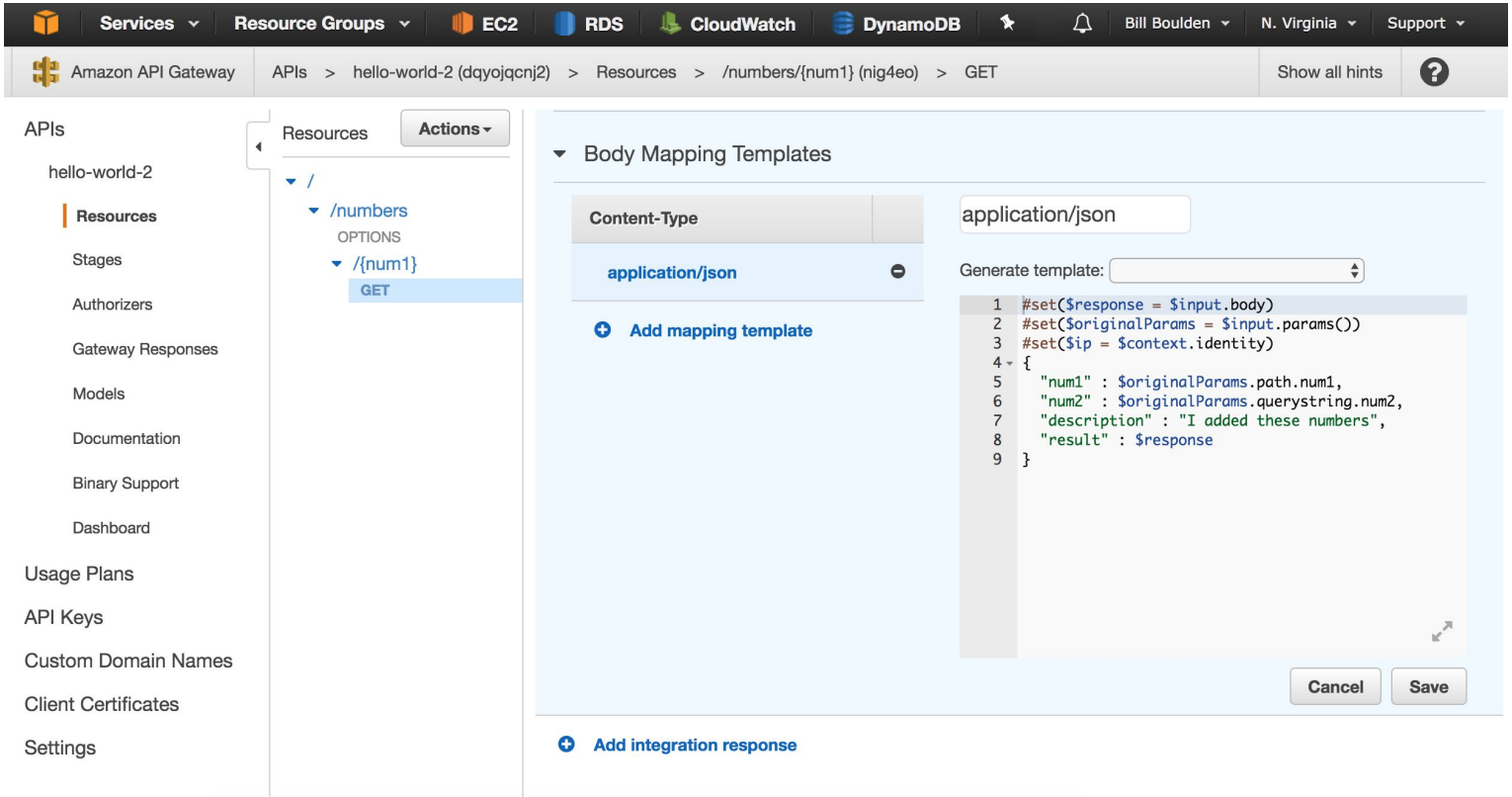
What Was That?

- Apache “Velocity Template Language”
- *Very* useful reference: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-mapping-template-reference.html
- We clearly defined a variable meant to come from path, and one meant to come from querystring
- We mapped them into a nice, neat object so Lambda can be agnostic as to how it received its calling parameters
- Any and all other headers, parameters, querystrings, body, etc. disregarded
- “Security” against development oversights :)

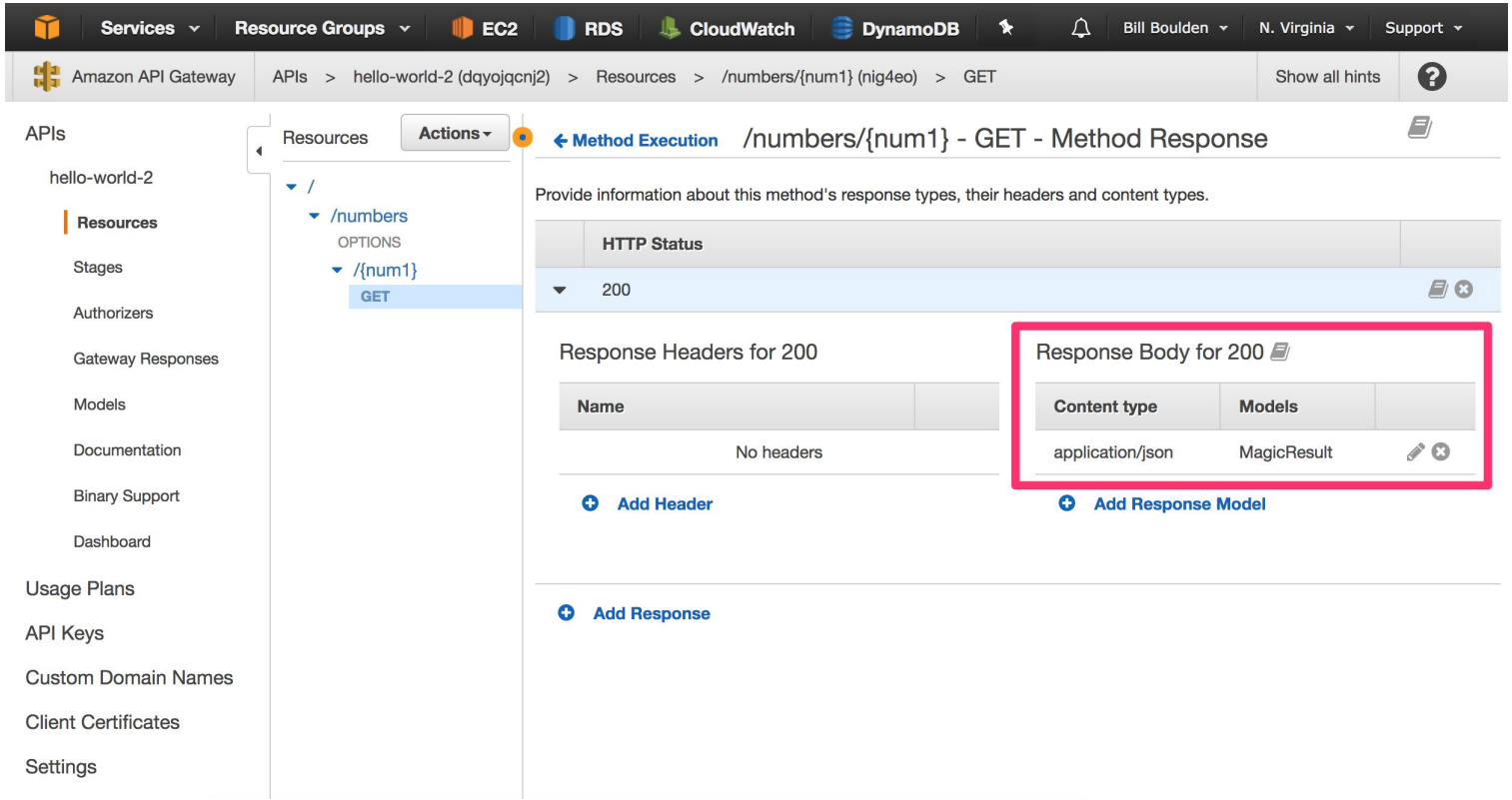
We Have to Make an Output Model before We Continue

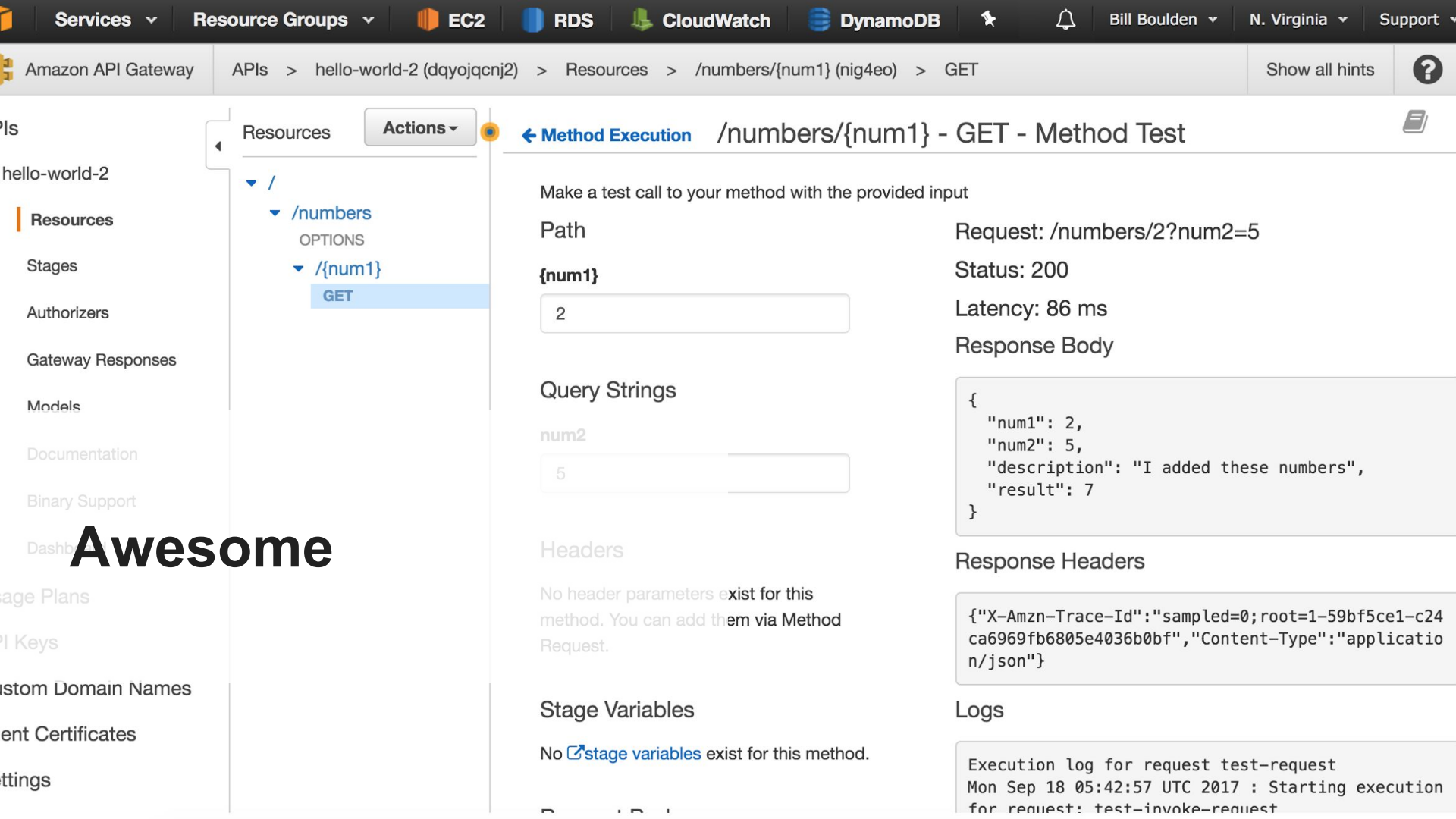


Integration Result Maps Response Specifically to Model



And Finally, the Response Model




Services
Resource Groups
EC2
RDS
CloudWatch
DynamoDB
Bill Boulden
N. Virginia
Support
Amazon API Gateway
APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo) > GET
Show all hints
hello-world-2
Resources
Stages
Authorizers
Gateway Responses
Models
Documentation
Binary Support
Resources
Actions
/
/numbers
OPTIONS
/{num1}
GET
Method Execution
/numbers/{num1} - GET - Method Test
Make a test call to your method with the provided input
Path
{num1}
2
Query Strings
num2
5
Headers
No header parameters exist for this method. You can add them via Method Request.
Stage Variables
No stage variables exist for this method.
Request: /numbers/2?num2=5
Status: 200
Latency: 86 ms
Response Body
{
"num1": 2,
"num2": 5,
"description": "I added these numbers",
"result": 7
}
Response Headers
{"X-Amzn-Trace-Id":"sampled=0;root=1-59bf5ce1-c24ca6969fb6805e4036b0bf","Content-Type":"application/json"}
Logs
Execution log for request test-request
Mon Sep 18 05:42:57 UTC 2017 : Starting execution for request: test-invoke-request
Awesome

10 Minute Break

Putting It All Together: Implementing an API Function In Practice

What Shall We Build?

- Wanted to choose a sample project for us that was general and standalone
- Was not feeling a standard CRUD app such as Books & Authors or *shudder* Yet Another Image Resizer
- Would not want to involve RDS/database as that would expand the scope of this training to unnecessarily include RDS
- Would not want to involve a Facebook or Twitter app as that would require you all to generate your own app keys and tokens for the purposes of the training
- The answer is obviously a meme: a dog rater :)

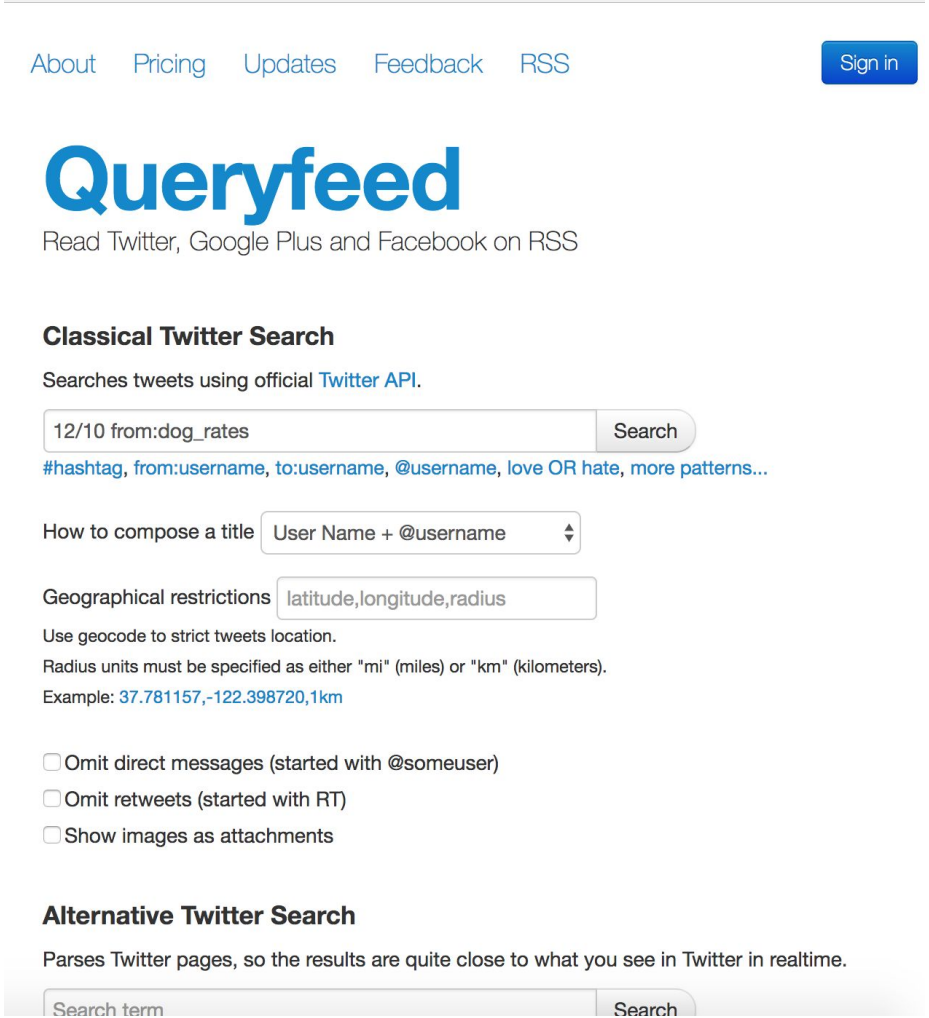
Some meme background

- **WeRateDogs** is a popular Twitter account that takes pictures of cute “doggos” or “puppers” and ranks them on a scale from 1-10
- Except—because all doggos are Good Boys, every doggo scores way over 10, usually 11-15.
- https://twitter.com/dog_rates
- 3.5 million followers
- Has never rated a doggo less than 10 except as a gag to later revise its rating to be higher
- Coined the slang term “h*ckin”

Queryfeed

I didn't want you to have to get a Twitter app authorized to use the official Twitter API, so we'll be doing things quick and easy with Queryfeed, a website that scrapes Twitter as publicly available and returns the results as RSS.

This will give us a chance to include some third party node modules for added practicality, too, so we can see what that would look like in practice.

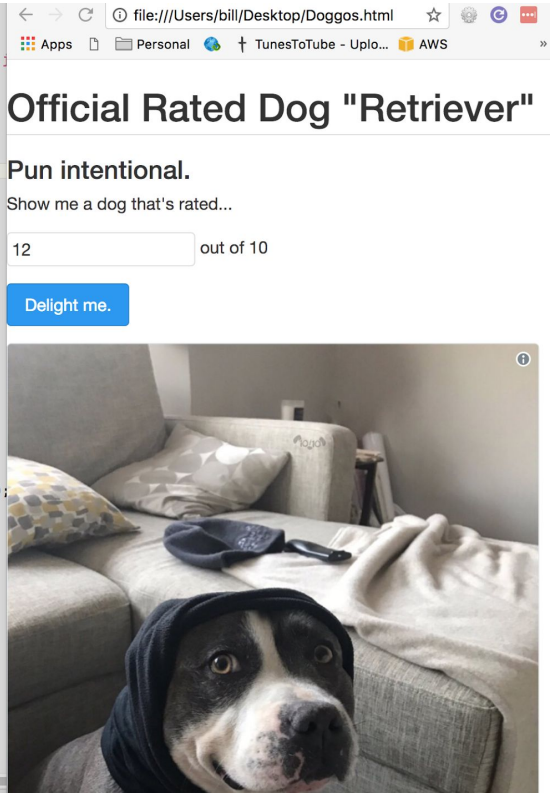


Base Page

The contents of this page will only be skimmed here, as they are elementary jQuery. In essence, you ask for a doggo of a certain rating in the textbox, and it changes a in the page to become an embedded picture of a doggo rated that score.

(Remember, no ratings below 10! All doggos are exceptionally good.)

```
<html>
<head>
<!-- Authorship: Bill "Spruke" Boulden, not that it's anything to be proud of. -->
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.j
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/marx/2.0.7/marx.min.css" />
<script type="text/javascript" src="https://platform.twitter.com/widgets.js"></script>

</head>
<body>

<h1>Official Rated Dog "Retriever"</h1>
<h3>Pun intentional.</h3>

<p>Show me a dog that's rated...</p>
<p><input type="number" id="rating"></input> out of 10</p>
<p><button id="retrieve">Delight me.</button>

<div id="embed-tweet-here">
</div>

<script type="text/javascript">

$(document).ready(function() {
    $('#retrieve').click(function () {
        $('#embed-tweet-here').empty();
        $.get('https://3t2l2ug9si.execute-api.us-east-1.amazonaws.com/prod/doggos?rating='
            + $('#rating').val())
            .done(function(json) {
                let tweetNumber = json.tweet.match(/\/(\d+)$/)[1];
                console.log(tweetNumber);
                twttr.widgets.createTweet(tweetNumber, document.getElementById('embed-tweet-here'));
            });
    });
});

</script>
</body>
</html>
```

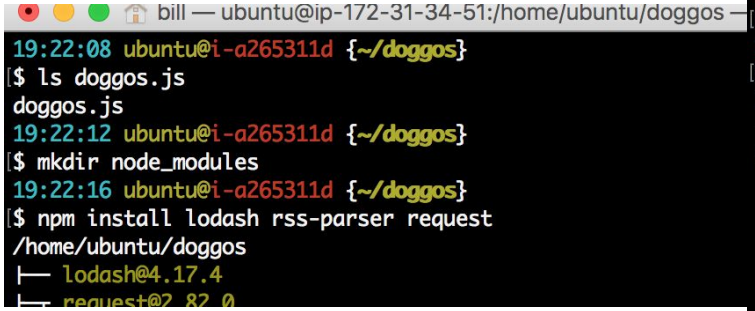


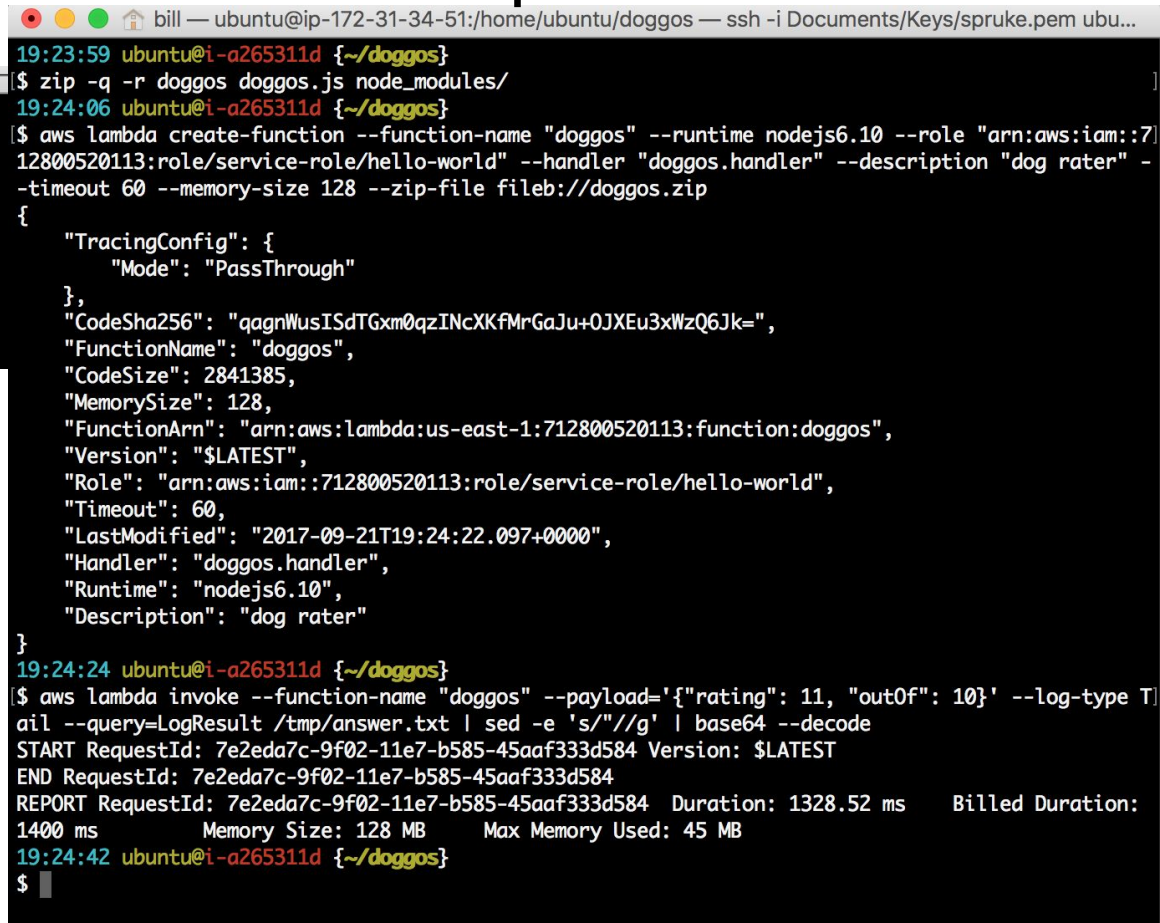
Lambda Function: doggos.js

bill — ubuntu@ip-172-31-34-51:/home/ubuntu/doggos — ssh -i Documents/Keys/spruke.pem ubu...

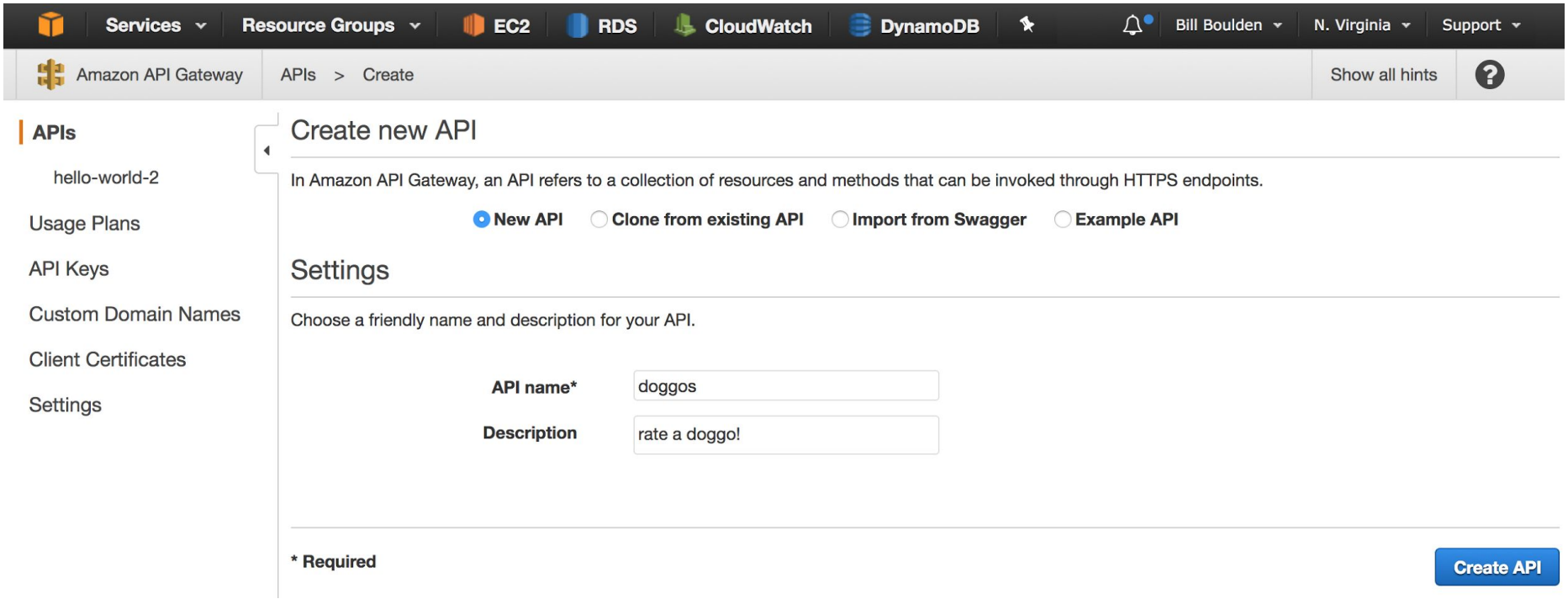
```
let _ = require('lodash');
let parser = require('rss-parser');
let request = require('request');

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

    let rating = event.rating;
    let outOf = event.outOf;

    if (rating < 10) {
        return callback("What are you asking for-some kind of... bad? dog? No such thing");
    }

    request('https://queryfeed.net/twitter?q='+rating+'%2F'+outOf+
            '+from%3Adog_rates&title-type=user-name-both&geocode=',
        function(error, response, body) {
            if (error) {
                return callback("Error fetching Queryfeed:" + error);
            }
            parser.parseString(body, function(err, goodies) {
                if (err) {
                    return callback("Error parsing feed RSS:" + error);
                }
                let doggoTweets = goodies.feed.entries;
                if (doggoTweets.length == 0) {
                    return callback("No doggos found at that level of goodness.");
                }
                return callback(null, { tweet: _.sample(doggoTweets).link });
            });
        });

}
                                                                    1,1           All
```

Everything Else Needed To Get It Up There

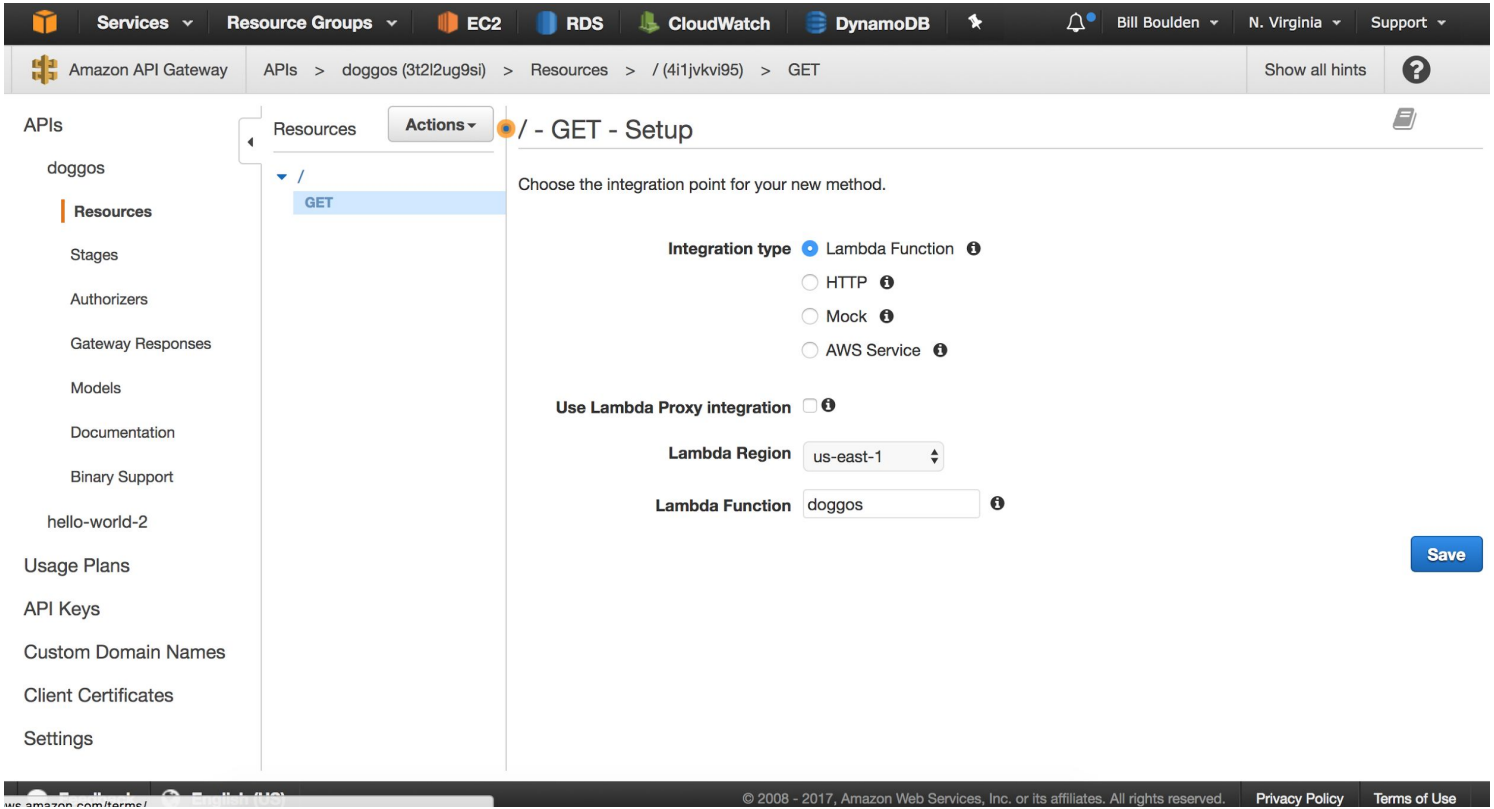




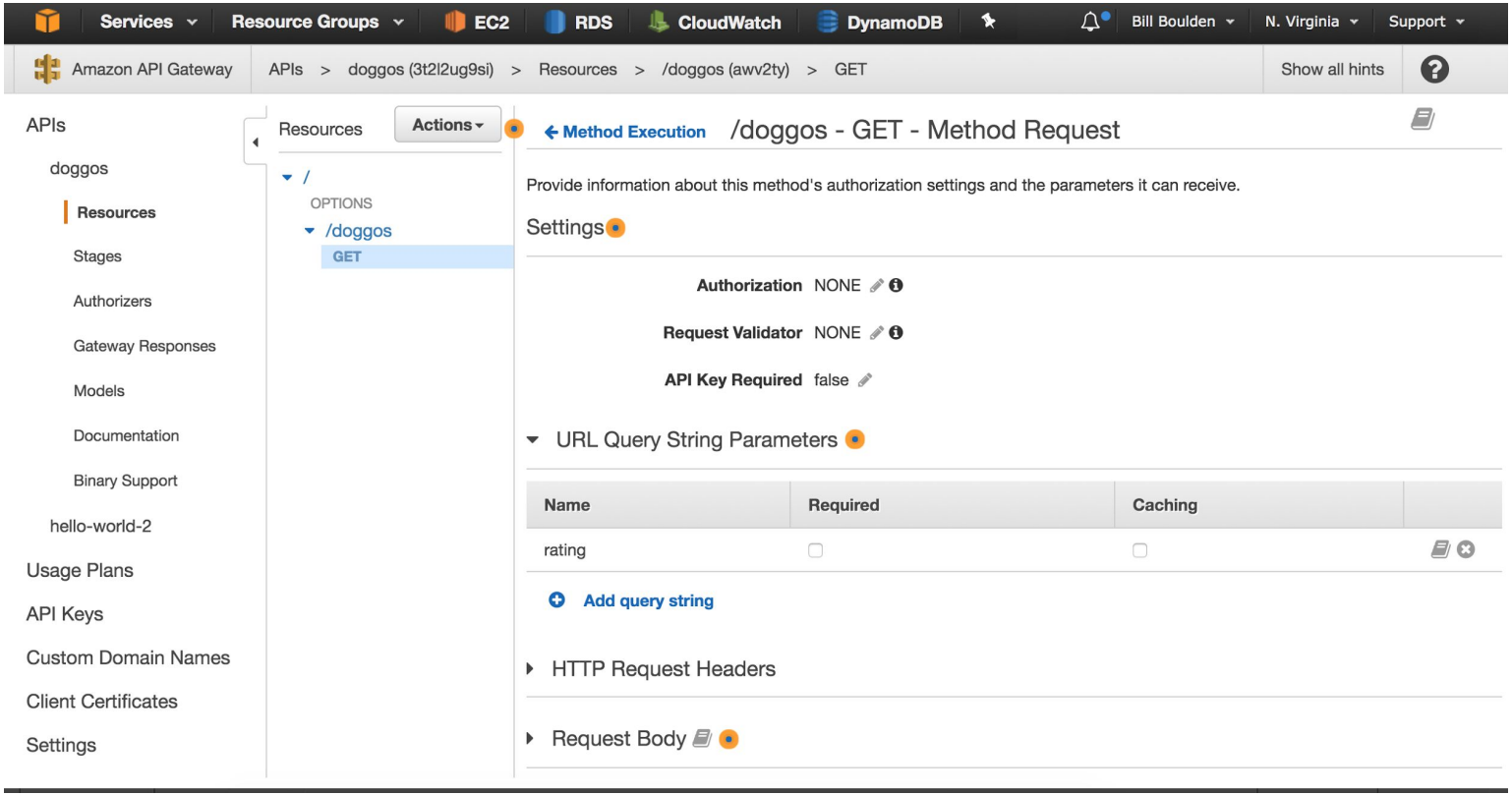
Start Building An API Gateway!



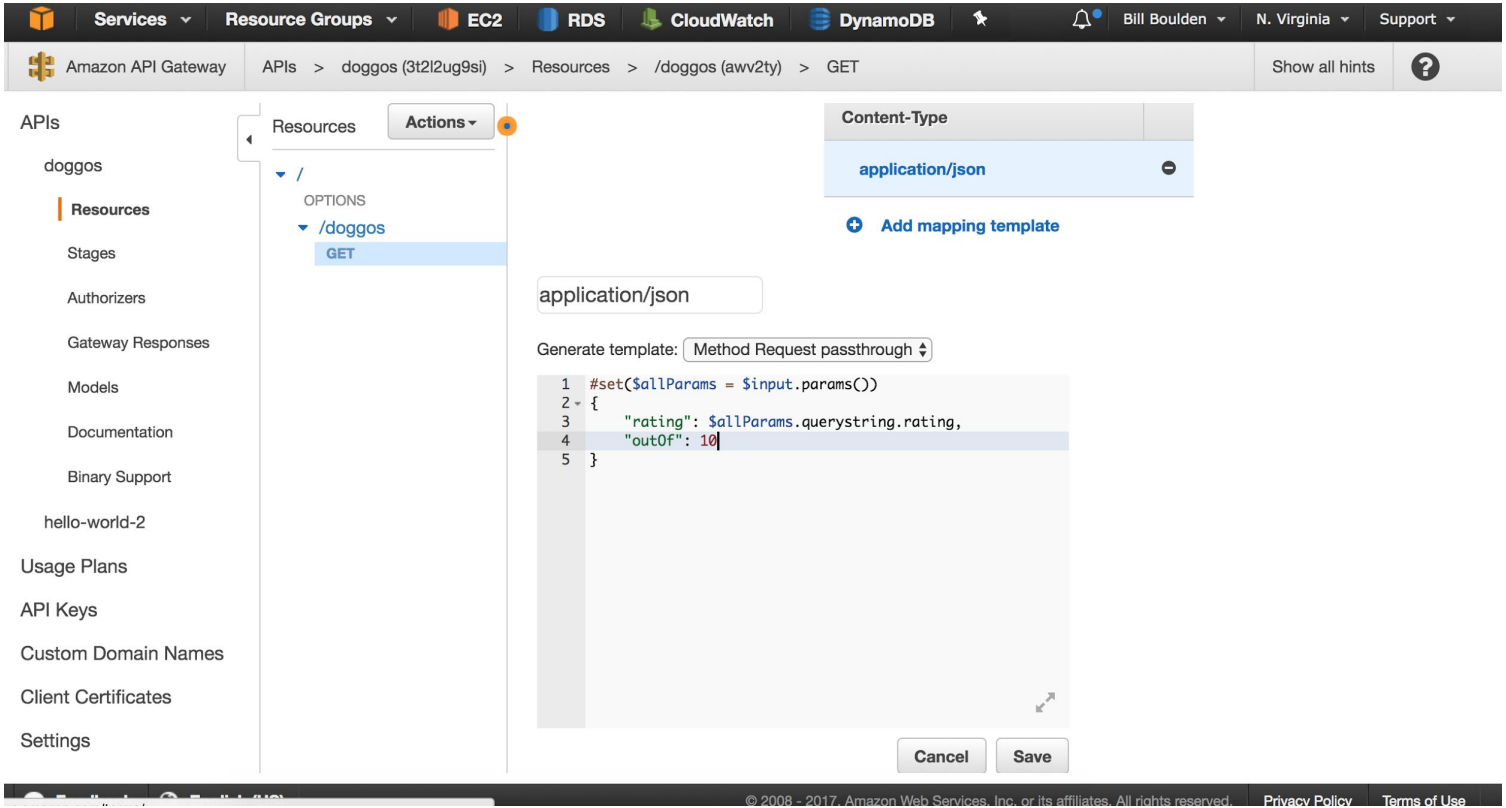
API Gateway, cont.



Extract Rating Parameter On The Way In

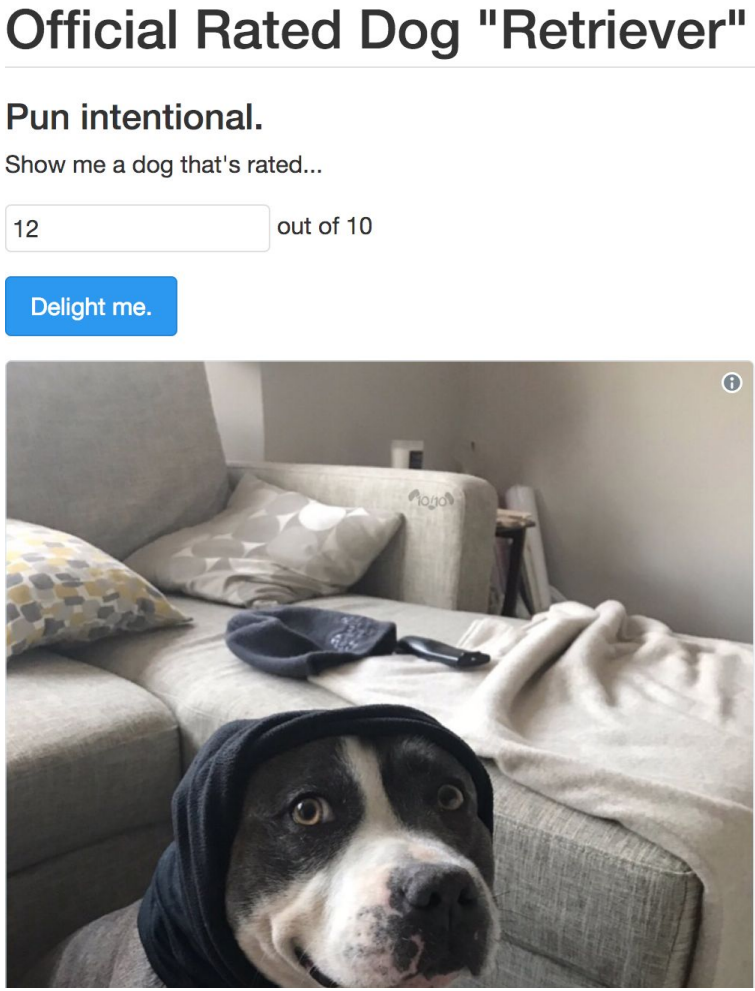


And Map It to the Lambda While Hardcoding OutOf



Final steps

- Deploy API
- Enable CORS on both top resource and sub resource
- Take the URL of deployed API and work it into the base page’s JavaScript
- Enjoy doggo love!!
- For bonus points, build in custom method response template to handle errors and give them a non-200 code



Business Implications of Serverless APIs

Literal Cash Cost Comparisons

- Assuming that your application is using anything less than 100% of the EC2's full resources, Lambda is dramatically cheaper
- It's usually not wise to engineer an application to keep all its EC2 web server instances running at close to 100% unless you're really doing some advanced resource management
- Source: https://www.trek10.com/blog/lambda-cost/

Total Cost of Ownership Comparisons

Additional Lambda “Charges”

- Educating development team and learning new design patterns
- Kind of a pain to develop rapidly due to package-upload-deploy-invoke cycle
    - Can be mitigated with scripts or Serverless Framework
    - Once supporting libraries grow large enough, no avoiding annoying 10-20 second “redeploy to test” cycle
- Code reuse can be tricky without some form of Symlinking or git subtree or private package server

Additional EC2 “Charges”

- Servers/networks can and will be attacked constantly, need some knowledgeable security people and their time/input
- Scaling takes extra effort and does not happen automatically; needs configuration which can be right or wrong
- Time spent managing packages, libraries, or at least time spent managing packaging manager/Puppet/Ansible
- Every second of CPU time and MB of RAM not used is lost opportunity

Keeping Lambda Costs Down

- Check those CloudWatch logs often!
- Logs can report on MB used… if you can reasonably believe it is never using the last 64 MB of RAM, shave it off!
- Cost is directly Seconds x Megabytes (get ready to hear the term pico-dollars-per-byte-second or the like a lot) so every MB taken off of the overhead is as good as a speed improvement!

Bad Fits For Lambda

- Infrequently called APIs (less than one call per 5 minutes)
  - Lambda keeps 0 code execution containers up and ready if your API has not been called
  - The first call to arrive makes a container need to “spin up” from scratch, has an unpleasant 2-5 second delay
  - Containers stay alive for several minutes afterward and are kept alive by renewed calls
  - APIs which are continually in use will respond quickly at all times
  - If your API endpoint is called once an hour, then it will always take 2-5 seconds plus execution time to respond, which is Not A Great Performance Gain
- Long running endpoints
  - The maximum allowable timeout of 5 minutes makes this a bad fit for something like a “download your entire database export file” endpoint because there really is just nothing that can be done for an operation that takes > 5 minutes and can’t be itemized further

10 Minute Break Part II: Electric Boogaloo

Next Steps & Extra Credit

Extra Credit!

In this section, I’m going to touch on several very cool technologies and tangentially related things you can do as you get into serverless API creation.

I won’t be able to go into it in-depth because I have tried to teach everything in this course very thoroughly, and these next topics cannot be skimmed over; to do them justice I’d have to deep-dive them, and then this would be a day-long course.

I’ll touch on what they are and where you can learn more.

Serverless Framework

- https://serverless.com/
- Middleware toolset that lets you define a serverless.yml with many implementation details in it, saving you from hooking up the Lambdas and API Gateways
- You'll get ridiculous timesavers like `sls deploy` and `sls invoke` and `sls logs`
- Vibrant open-source community, actively maintained

```
functions:
  usersCreate: # A Function
    handler: users.create # The file and module for this specific function
    name: ${self:provider.stage}-lambdaName # optional, Deployed Lambda na
    description: My function # The description of your function.
    memorySize: 512 # memorySize for this specific function.
    runtime: nodejs6.10 # Runtime for this specific function. Overrides th
    timeout: 10 # Timeout for this specific function.  Overrides the defau
    role: arn:aws:iam::XXXXXX:role/role # IAM role which will be used for
    onError: arn:aws:sns:us-east-1:XXXXXX:sns-topic # Optional SNS topic a
    awsKmsKeyArn: arn:aws:kms:us-east-1:XXXXXX:key/some-hash # Optional KM
    environment: # Function level environment variables
      functionEnvVar: 12345678
    tags: # Function specific tags
      foo: bar
    vpc: # Optional VPC. But if you use VPC then both subproperties (secur
      securityGroupIds:
        - securityGroupId1
        - securityGroupId2
      subnetIds:
        - subnetId1
        - subnetId2
    events: # The Events that trigger this Function
      - http: # This creates an API Gateway HTTP endpoint which can be use
          path: users/create # Path for this endpoint
          method: get # HTTP method for this endpoint
          cors: true # Turn on CORS for this endpoint, but don't forget to
          private: true # Requires clients to add API keys values in the `
          authorizer: # An AWS API Gateway custom authorizer function
            name: authorizerFunc # The name of the authorizer function (mu
            arn:  xxx:xxx:Lambda-Name # Can be used instead of name to ref
            resultTtlInSeconds: 0
            identitySource: method.request.header.Authorization
            identityValidationExpression: someRegex
      - s3:
```

Using CloudFront + Route 53 to Front API Gateway

Nobody wants their API's official address to be k89374hf4.amazon.whatever.aws.com!! :D

To put a custom domain name such as api.foobarmysite.com on your API, first create a CloudFront Distribution that has your published Deploy stage as its Origin, and be sure to whitelist any headers and methods you see yourself using (Cookies, Content-Type, PUT, POST, etc…)

Then set up a Route 53 DNS record that CNAME's your pretty url, api.myfoobarsite.com, to the k89492381.cloudfront.net.

Live Demo: Using CloudWatch to Pull Metrics