

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



A vertical black bar is positioned on the left side of the white band. The background to the right of the white band features a repeating pattern of dark gray circles.

**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 minuscule 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 qeuest
 - 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

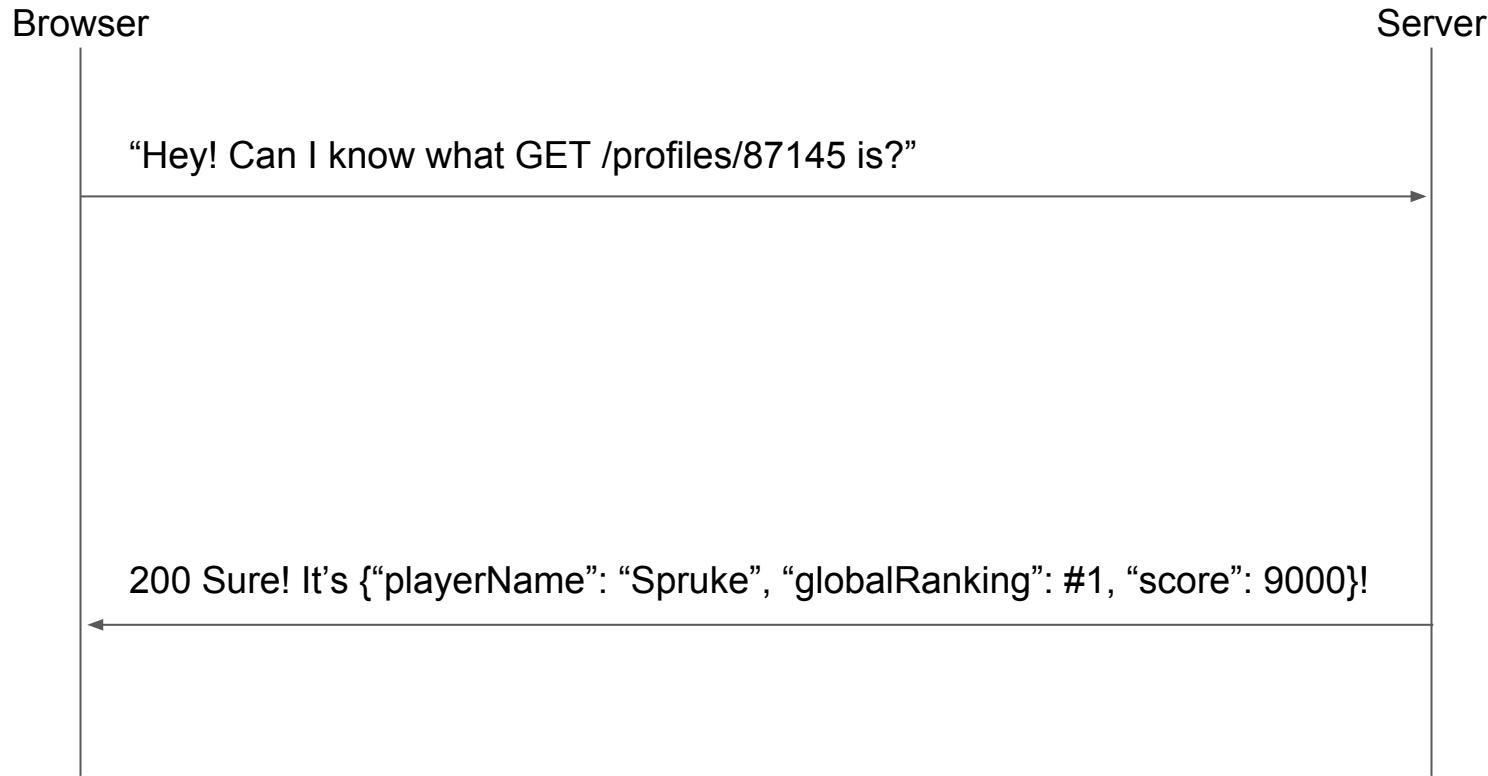
Cacheability

- 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
- Hmm, 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

Build Applications With AWS Lambda

Password

 ...

Sign In

[Sign in to a different account](#)

[Forgot your password?](#)



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

The screenshot shows the AWS Lambda console homepage. The top navigation bar includes links for Services, Resource Groups, EC2, RDS, CloudWatch, DynamoDB, and more. Below the navigation is a search bar and a sidebar with links for History, RDS, Console Home, CloudWatch, DynamoDB, Route 53, and Simple Email Service. The main content area is organized into a grid of service categories:

- Compute:** EC2, EC2 Container Service, Lightsail, Elastic Beanstalk, Lambda, Batch.
- Developer Tools:** CodeStar, CodeCommit, CodeBuild, CodeDeploy, CodePipeline, X-Ray.
- Analytics:** Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis, Data Pipeline, QuickSight, AWS Glue.
- Application Services:** Step Functions, SWF, API Gateway, Elastic Transcoder.
- Storage:** S3, EFS, Glacier, Storage Gateway.
- Management Tools:** CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog, Trusted Advisor, Managed Services.
- Artificial Intelligence:** Lex, Amazon Polly, Rekognition, Machine Learning.
- Messaging:** Simple Queue Service, Simple Notification Service, Simple Email Service.
- Business Productivity:** WorkDocs, WorkMail, Amazon Chime.
- Database:** RDS, DynamoDB, ElastiCache, Amazon Redshift.
- Internet Of Things:** AWS IoT, AWS Greengrass.
- Security, Identity & Compliance:** (partially visible).
- Desktop & App Streaming:** WorkSpaces, AppStream 2.0.

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

Screenshot of the AWS Lambda "Create function" wizard.

The top navigation bar shows: AWS, Services (dropdown), Resource Groups (dropdown), EC2, Relational Database Service, three small icons, spruke @ 3826-7037-2848 (dropdown), N. Virginia (dropdown), and Support (dropdown).

The breadcrumb navigation shows: Lambda > Functions > Create function.

Create function

Author from scratch Start with a simple "hello world" example.


Blueprints Choose a preconfigured template as a starting point for your Lambda function.


Author from scratch [Info](#)

Name*

Runtime*

Role*
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

Guten tag, Velt. Hola, mundo.

Author from scratch [Info](#)

Name*

Runtime*

▼

Role*
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

▼

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*
Enter a name for your new role.

▼

Policy templates
Choose one or more policy templates. A role will be generated for you before your function is created. [Learn more](#) about the permissions that each policy template will add to your role.

▼

Basic Edge Lambda permissions [X](#)

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

Lambda function handler and role

Handler*

The module-name.export value in your function. For example, "index.handler" would call exports.handler in index.js.

index.handler

Role*

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

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*

Enter a name for your new role.

hellow-world

Policy templates

Choose one or more policy templates. A role will be generated for you before your function is created. [Learn more](#) about the permissions that each policy template will add to your role.

Basic Edge Lambda permissions X

Amazon's way ahead of us with its default content

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 ▾

```
1 - exports.handler = (event, context, callback) => {
2     // TODO implement
3     callback(null, 'Hello from Lambda');
4 };
```

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...

```
bill — ec2-user@ip-172-31-43-91:~/sample — ssh -o ServerAliveInterval=10 -i Documents...
20:03:21 ec2-user@i-0eec534816d99986 {~/sample}
[$ vi hello-world-advanced.js
20:10:00 ec2-user@i-0eec534816d99986 {~/sample}
[$ vi mathematics-wizard.js
20:13:20 ec2-user@i-0eec534816d99986 {~/sample}
[$ npm install lodash
/home/ec2-user/sample
└─ lodash@4.17.4

npm [WARN] enoent ENOENT: no such file or directory, open '/home/ec2-user/sample/package.json'
npm [WARN] sample No description
npm [WARN] sample No repository field.
npm [WARN] sample No README data
npm [WARN] sample No license field.
20:13:26 ec2-user@i-0eec534816d99986 {~/sample}
$ ]
```

Bundle this into a zip.

```
 bill — ec2-user@ip-172-31-43-91:~/sample — ssh -o ServerAliveInterval=10 -i Documents...
20:17:21 ec2-user@i-0eec534816d99986 {~/sample}
[$ zip hello-world-2 *
 adding: hello-world-advanced.js (deflated 44%)
 adding: mathematics-wizard.js (deflated 22%)
 adding: node_modules/ (stored 0%)
20:17:27 ec2-user@i-0eec534816d99986 {~/sample}
$ ]
```

Deploy to Lambda With the Command Line Interface

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

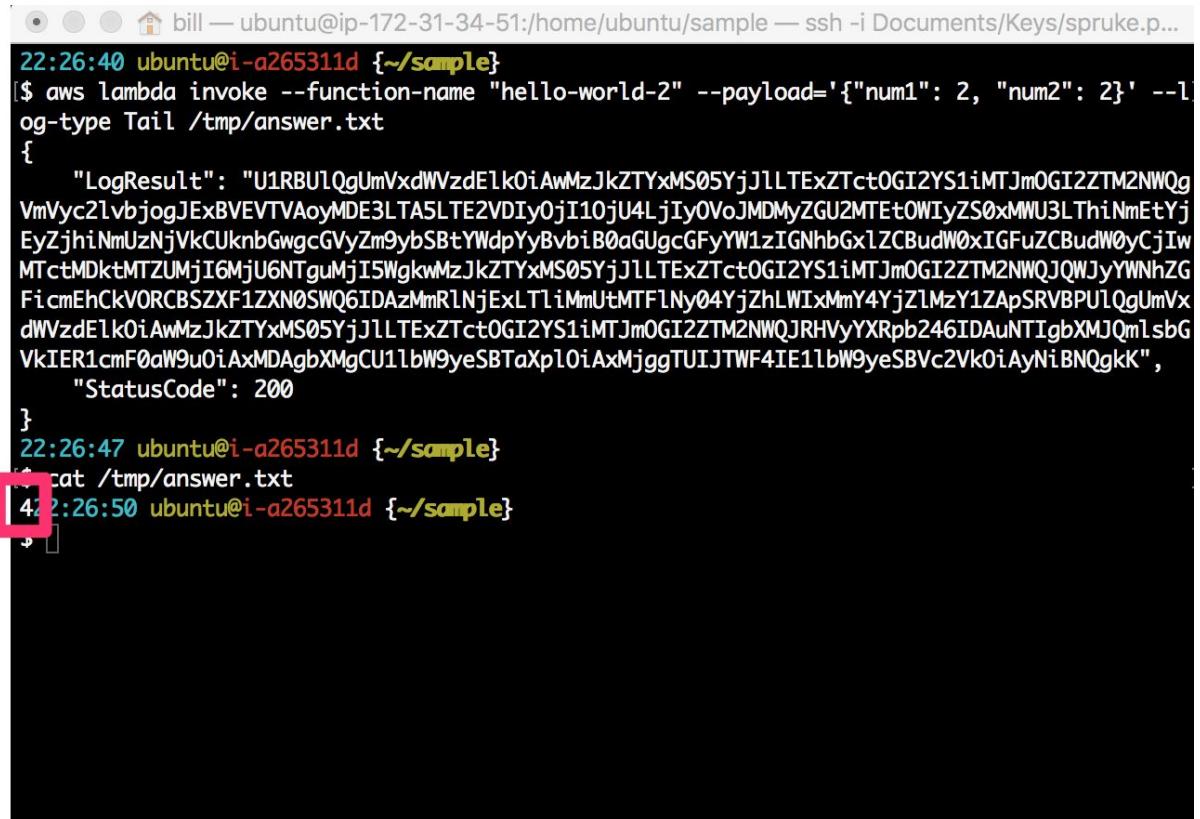
```
bill — ubuntu@ip-172-31-34-51:/home/ubuntu/sample — ssh -i Documents/Keys/spruke.p...  
  
22:10:31 ubuntu@i-a265311d {~/sample}  
[$ aws lambda create-function --function-name "hello-world-2" --runtime nodejs6.10 --role "arn:aws:iam::712800520113:role/service-role/hello-world" --handler "hello-world-advanced.handler" --description "numerical magic" --timeout 60 --memory-size 128 --zip-file fileb://hello-world-2.zip  
{  
    "TracingConfig": {  
        "Mode": "PassThrough"  
    },  
    "CodeSha256": "U1b9qCCgT/amLBXa7kCNBrkXT3CYdGUFEgcXF02UJDA=",  
    "FunctionName": "hello-world-2",  
    "CodeSize": 920,  
    "MemorySize": 128,  
    "FunctionArn": "arn:aws:lambda:us-east-1:712800520113:function:hello-world-2",  
    "Version": "$LATEST",  
    "Role": "arn:aws:iam::712800520113:role/service-role/hello-world",  
    "Timeout": 60,  
    "LastModified": "2017-09-16T22:09:45.446+0000",  
    "Handler": "hello-world-advanced.handler",  
    "Runtime": "nodejs6.10",  
    "Description": "numerical magic"  
}  
22:10:34 ubuntu@i-a265311d {~/sample}  
$ █
```

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
```



The screenshot shows a terminal window on an Ubuntu system (ip-172-31-34-51). The user is running the command:

```
$ aws lambda invoke --function-name "hello-world-2" --payload='{"num1": 2, "num2": 2}' --log-type Tail /tmp/answer.txt
```

The output is a large, base64-encoded string:

```
{"LogResult": "U1RBULQgUmVxdWVzdElk0iAwMzJkZTYxMS05YjJ1LTExZTct0GI2YS1iMTJm0GI2ZTM2NWQgVmVyc2lvbjogJExBVEVTVAoyMDE3LTASLTER2VDIy0jI10ju4LjIy0VoJMDMyZGU2MTEtOWiyZS0xMWU3LThiNmEtYjEyZjhINmUzNjVCKUknbGwgcGVyZm9ybSBtYWdpYyBvbIB0aGugcGFyYW1zIGNhbGxlZCBudW0xIGFuZCBudW0yCjIwMTctMDktMTZUMjI6MjU6NTguMjI5WgkwMzJkZTYxMS05YjJ1LTExZTct0GI2YS1iMTJm0GI2ZTM2NWQJQWJyYWNhZGFicmEhCkVORCBSZXF1ZXN0SWQ6IDAzMmR1NjExLTlMmUtMTF1Ny04YjZhLWIxMmY4YjZ1MzY1ZApsRVBPULQgUmVx dWVzdElk0iAwMzJkZTYxMS05YjJ1LTExZTct0GI2YS1iMTJm0GI2ZTM2NWQJRHVYXRpB246IDAuNTIgbXMJQmlsbGVkIER1cmF0aW9uOiAxMDAgbXMgCU1lbW9yeSBTaXpl0iAxMjggTUIJTWF4IE1lbW9yeSBVc2Vk0iAyNiBNQgkK", "StatusCode": 200}
```

At the bottom of the terminal, there is a red box highlighting the command:

```
$ cat /tmp/answer.txt
```

Services ▾ | Resource Groups ▾ | EC2 | RDS | **CloudWatch** | DynamoDB | Bill Boulden ▾ | N. Virginia ▾ | Support ▾

CloudWatch Dashboards Alarms ALARM INSUFFICIENT OK 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

Time (UTC -04:00) Message

Time (UTC -04:00)	Message
2017-09-16	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-a1aababcd0f65 Version: \$LATEST
▶ 18:25:40	2017-09-16T22:25:40.016Z f84f276e-9b2d-11e7-91d2-a1aababcd0f65 I'll perform magic on the params called num1 and num2
▶ 18:25:40	2017-09-16T22:25:40.016Z f84f276e-9b2d-11e7-91d2-a1aababcd0f65 Abracadabra!
▶ 18:25:40	END RequestId: f84f276e-9b2d-11e7-91d2-a1aababcd0f65
▶ 18:25:40	REPORT RequestId: f84f276e-9b2d-11e7-91d2-a1aababcd0f65 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 Version: \$LATEST
▶ 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

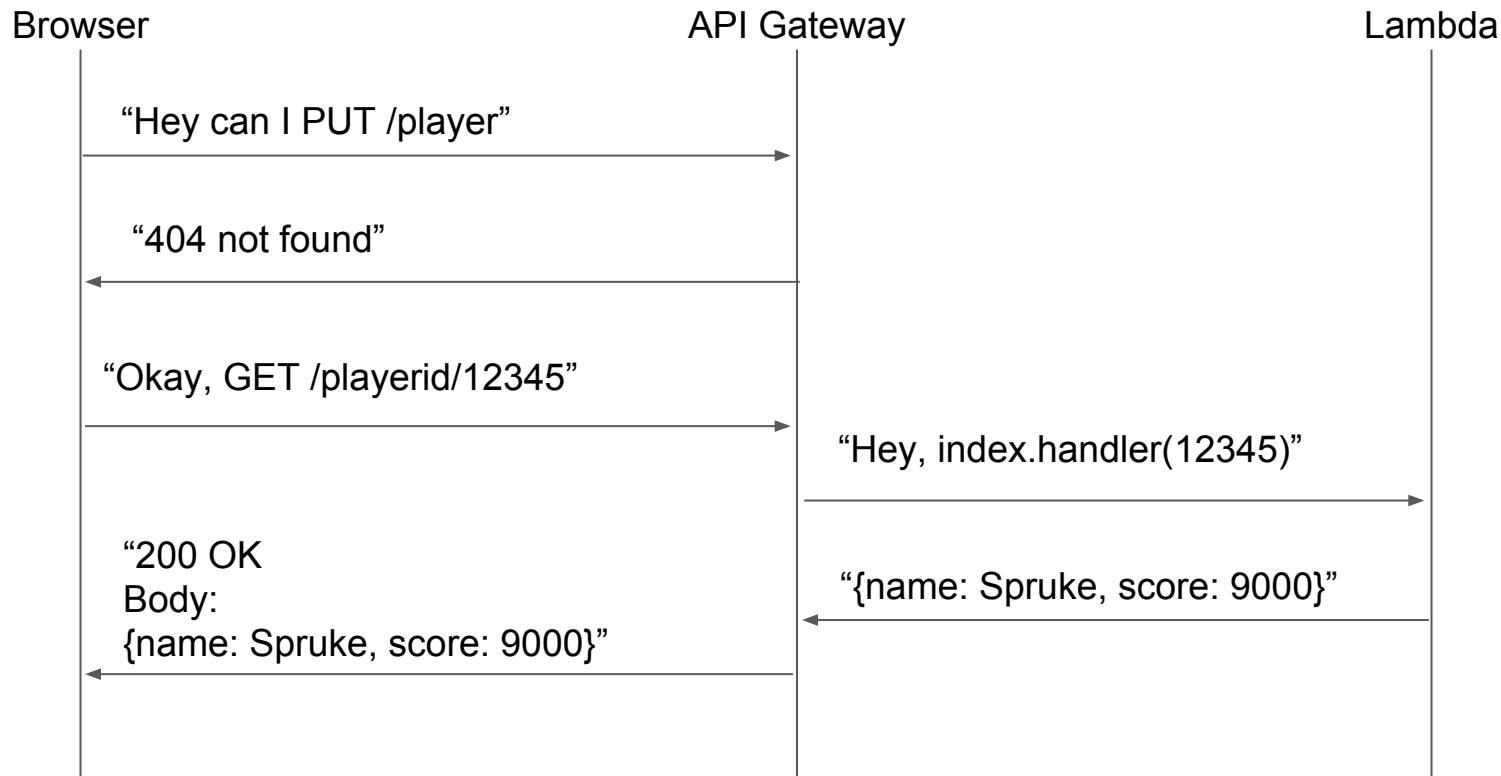
No newer events found at the moment. [Retry](#).

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



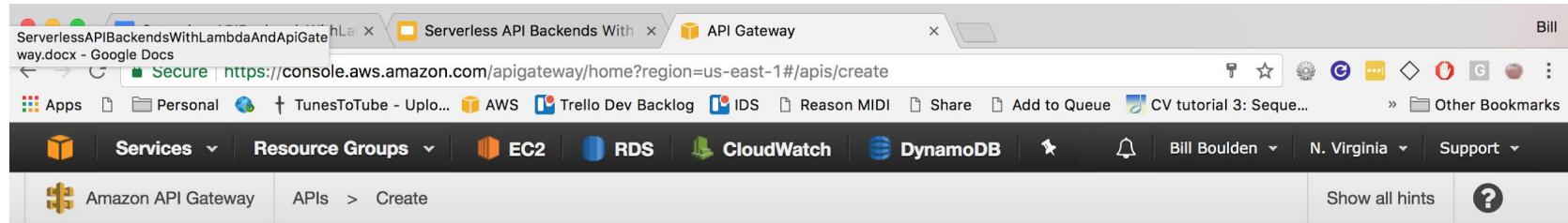
How Does API Gateway Help Us Create a Serverless API Model?

- 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



Create new API

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

New API Import from Swagger Example API

Settings

Choose a friendly name and description for your API.

API name*

hello-world-2

Description

Perform abstract magic on numbers

* Required

Create API

Methods & Resources

The screenshot shows the AWS API Gateway console interface. The top navigation bar includes links for Services (with a dropdown arrow), Resource Groups (with a dropdown arrow), EC2, RDS, CloudWatch, DynamoDB, a user icon for Bill Boulden, and account information for N. Virginia. Below the navigation is a breadcrumb trail: APIs > hello-world-2 (dqyojqcnj2) > Resources > / (o1g9n1rt4i). On the right side of the header are links for Show all hints and a help icon.

The left sidebar contains a tree view of API resources:

- APIs
- hello-world-2
 - Resources** (selected, indicated by an orange border)
 - Stages
 - Authorizers
 - Gateway Responses
 - Models
 - Documentation
 - Binary Support
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates
- Settings

The main content area has a title / Methods. Below it, a message states "No methods defined for the resource." To the left of this message is a dropdown menu labeled Actions. The menu is divided into two sections: RESOURCE ACTIONS and API ACTIONS. The RESOURCE ACTIONS section contains "Create Method" and "Create Resource", which are highlighted with a red rectangular box. The API ACTIONS section contains "Enable CORS", "Edit Resource Documentation", "Deploy API", "Import API", "Edit API Documentation", and "Delete API".

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

The screenshot shows the Amazon API Gateway console interface. The top navigation bar includes links for Services (with a dropdown arrow), Resource Groups (with a dropdown arrow), EC2, RDS, CloudWatch, DynamoDB, a user icon for Bill Boulden, a location dropdown for N. Virginia, and Support (with a dropdown arrow). Below the navigation bar, the breadcrumb trail indicates the current path: APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo). There are also links for Show all hints and a help icon.

The left sidebar contains the following navigation items:

- APIs
- hello-world-2
- Resources** (selected)
- Stages
- Authorizers
- Gateway Responses
- Models
- Documentation
- Binary Support
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates
- Settings

The main content area is titled "/numbers/{num1} Methods". It shows a tree structure of resources under the path "/numbers/{num1}":

- /
- /numbers (OPTIONS method available)
- {num1} (highlighted with a blue background)

A message in the center states "No methods defined for the resource." A small edit icon is located in the top right corner of the main content area.

And Then Give {num1} Path a GET Method

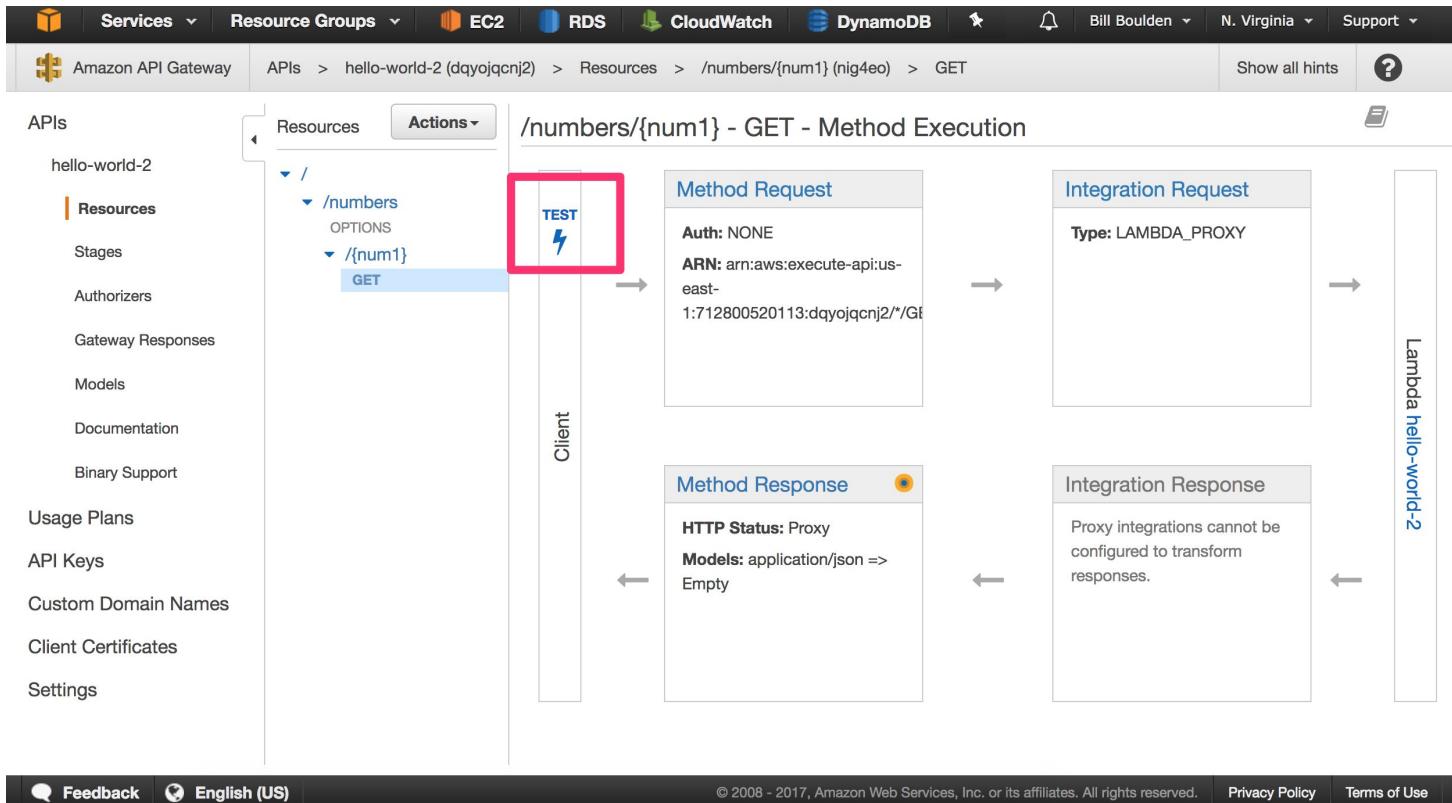
The screenshot shows the AWS API Gateway console interface. The top navigation bar includes links for Services (with a dropdown arrow), Resource Groups (with a dropdown arrow), EC2, RDS, CloudWatch, DynamoDB, a star icon, a bell icon, Bill Boulden (with a dropdown arrow), N. Virginia (with a dropdown arrow), and Support (with a dropdown arrow). Below the navigation bar, the breadcrumb trail indicates the current path: APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo) > GET. There are also links for Show all hints and a question mark icon.

The left sidebar contains the following navigation items:

- APIs
- hello-world-2
- Resources** (selected)
- Stages
- Authorizers
- Gateway Responses
- Models
- Documentation
- Binary Support
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates
- Settings

The main content area is titled "/numbers/{num1} - GET - Setup". It displays the path /numbers/{num1} and the method GET. A sub-section titled "Choose the integration point for your new method." is present. The "Integration type" section shows "Lambda Function" selected (radio button is checked). Other options include "HTTP" (radio button is unselected), "Mock" (radio button is unselected), and "AWS Service" (radio button is unselected). Below this, the "Use Lambda Proxy integration" checkbox is checked (checkbox is checked). The "Lambda Region" dropdown is set to "us-east-1". The "Lambda Function" dropdown is set to "hello-world-2". At the bottom right of the main content area is a blue "Save" button.

Neat. Now Let's Try It Out.



Not So Hot. Log Says “Malformed Proxy Response”

The screenshot shows the AWS API Gateway Method Execution interface for a GET request to /numbers/{num1} with num1=2. The response status is 502, and the response body contains an internal server error message. The logs show the execution starting and failing with a 502 error.

APIs

hello-world-2

Resources

Stages

Authorizers

Gateway Responses

Models

Documentation

Binary Support

Usage Plans

API Keys

Custom Domain Names

Client Certificates

Settings

Actions

Resources

Actions

Method Execution /numbers/{num1} - GET - Method Test

Make a test call to your method with the provided input

Path

{num1}

2

Request: /numbers/2?num2=2

Status: 502

Latency: 3323 ms

Response Body

{
 "message": "Internal server error"
}

Response Headers

{}

Logs

Execution log for request test-request
Sun Sep 17 21:19:36 UTC 2017 : Starting execution
for request: test-invoke-request
Sun Sep 17 21:19:36 UTC 2017 : HTTP Method: GET,
Resource Path: /numbers/2
Sun Sep 17 21:19:36 UTC 2017 : Method request pat
h: {num1=2}
Sun Sep 17 21:19:36 UTC 2017 : Method request que

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!

The screenshot shows the AWS API Gateway console with the following details:

- Services**: EC2, RDS, CloudWatch, DynamoDB
- Resource Groups**: N/A
- Region**: Bill Boulder, N. Virginia
- Support**: N/A

The navigation path is: APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo) > GET.

The left sidebar shows the API structure:

- APIs
- hello-world-2
 - Resources
 - /
 - /numbers
 - OPTIONS
 - /[num1]
 - GET
 - Stages
 - Authorizers
 - Gateway Responses
 - Models
 - Documentation
 - Binary Support
 - Usage Plans
 - API Keys
 - Custom Domain Names
 - Client Certificates
 - Settings

The main panel displays the **Method Execution** results for the **/numbers/{num1} - GET** test:

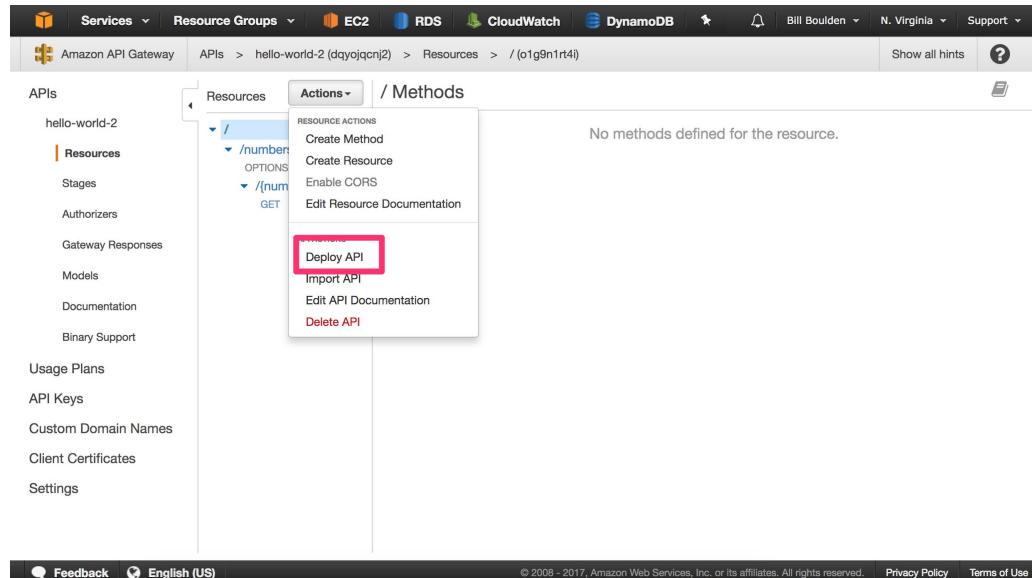
- Path**: Request: /numbers/2?num2=2
- Status**: 200
- Latency**: 79 ms
- Response Body**:

```
{ "result": 4 }
```
- Query Strings**: {num1}: num2=2
- Headers**: {num1}: Accept:application/json
- Stage Variables**: (empty)
- Response Headers**: { "X-Amzn-Trace-Id": "sampled=0;root=1-59beea89-f86f8c9c7efb68c8e5a4798d" }
- Logs**:

```
Execution log for request test-request
Sun Sep 17 21:35:05 UTC 2017 : Starting execution
for request: test-invoker-request
Sun Sep 17 21:35:05 UTC 2017 : HTTP Method: GET,
Resource Path: /numbers/
Sun Sep 17 21:35:05 UTC 2017 : Method request path: {num1=2}
```

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**

We're Going To Manually Configure In This Order

The screenshot shows the AWS API Gateway Method Execution interface for the path `/numbers/{num1}` with the `GET` method selected. The interface is divided into four main sections, each outlined with a red box and numbered 1 through 4:

- Method Request**:
Auth: NONE
ARN: arn:aws:execute-api:us-east-1:712800520113:dqyojqcnj2/*/Get
This section is labeled with a red box and the number 1.
- Integration Request**:
Type: LAMBDA_PROXY
This section is labeled with a red box and the number 2.
- Method Response**:
HTTP Status: Proxy
Models: application/json => Empty
This section is labeled with a red box and the number 4.
- Integration Response**:
Proxy integrations cannot be configured to transform responses.
This section is labeled with a red box and the number 3.

The interface also includes a sidebar on the left with various API management options like APIs, Resources, Stages, and Lambda functions, and a vertical sidebar on the right labeled "Lambda hello-world-2".

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

The screenshot shows the AWS API Gateway Settings page for a specific method request. The URL path is `/numbers/{num1}` and the HTTP method is `GET`. The `Authorization` is set to `NONE`, `Request Validator` is `NONE`, and `API Key Required` is `false`.

Request Paths:

Name	Caching
num1	<input type="checkbox"/>

URL Query String Parameters:

Name	Required	Caching
num2	<input type="checkbox"/>	<input type="checkbox"/>

HTTP Request Headers:

+ Add query string

Integration Request

The screenshot shows the AWS API Gateway console with the path `APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo) > GET`. The left sidebar lists various API management features like APIs, Resources, Stages, and Models. The main panel displays the integration request configuration for the `GET` method of the `/numbers/{num1}` resource.

Credentials cache: Do not add caller credentials to cache key [Edit](#)

URL Path Parameters:

Name	Mapped from ?	Caching
num1	method.request.path.num1	<input type="checkbox"/> Edit Delete

[+ Add path](#)

URL Query String Parameters:

Name	Mapped from ?	Caching
num2	method.request.querystring.num2	<input type="checkbox"/> Edit Delete

[+ Add query string](#)

HTTP Headers:

Body Mapping Templates: [?](#)

Integration Request pt 2

The screenshot shows the AWS API Gateway console with the following details:

- Services:** EC2, RDS, CloudWatch, DynamoDB
- User:** Bill Boulden, N. Virginia
- API:** Amazon API Gateway, APIs > hello-world-2 (dqyojqcnj2) > Resources > /numbers/{num1} (nig4eo) > GET
- Actions:** Body Mapping Templates
- Request body passthrough:** Never (selected)
- Content-Type:** application/json
- Add mapping template:** application/json
- Generate template:** dropdown menu
- Template Content:** (highlighted)

```
1 #set($params = $input.params())
2 {
3     "num1": $params.path.num1,
4     "num2": $params.querystring.num2
5 }
```

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

The screenshot shows the Amazon API Gateway console. The top navigation bar includes 'Services' (dropdown), 'Resource Groups' (dropdown), 'EC2', 'RDS', 'CloudWatch', 'DynamoDB', a bell icon, 'Bill Boulden' (dropdown), 'N. Virginia' (dropdown), and 'Support' (dropdown). Below the navigation is a breadcrumb trail: 'Amazon API Gateway' > 'APIs' > 'hello-world-2 (dqyojqcnj2)' > 'Models' > 'Create'. On the left, a sidebar menu lists 'APIs', 'Resources', 'Stages', 'Authorizers', 'Gateway Responses', 'Models' (which is selected and highlighted in orange), 'Documentation', 'Binary Support', 'Dashboard', 'Usage Plans', 'API Keys', 'Custom Domain Names', 'Client Certificates', and 'Settings'. The main content area is titled 'New Model' and contains instructions: 'Provide a name, content type, and a schema for your model. Models use JSON schema.' It has three input fields: 'Model name*' (set to 'MagicResult'), 'Content type*' (set to 'application/json'), and 'Model description' (set to 'A rich object detailing what happened'). Below these is a section titled 'Model schema*' containing a JSON schema definition:

```
1 {  
2     "$schema": "http://json-schema.org/draft-04/schema#",  
3     "title": "MagicResultModel",  
4     "type": "object",  
5     "properties": {  
6         "num1": { "type": "integer" },  
7         "num2": { "type": "integer" },  
8         "description": { "type": "string" },  
9         "result": { "type": "integer" }  
10    }  
11 }
```

At the bottom of the page, there are links for 'aws.amazon.com/terms/' (with a lock icon), '© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.', 'Privacy Policy', and 'Terms of Use'.

Integration Result Maps Response Specifically to Model

The screenshot shows the AWS Management Console with the API Gateway service selected. The navigation bar includes links for Services (EC2, RDS, CloudWatch, DynamoDB), Resource Groups, and various AWS regions (Bill Boulden, N. Virginia). The main content area displays the configuration for the 'hello-world-2' API, specifically for the '/numbers/{num1}' resource under the 'GET' method.

APIs sidebar:

- hello-world-2
- Resources** (selected)
- Stages
- Authorizers
- Gateway Responses
- Models
- Documentation
- Binary Support
- Dashboard
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates
- Settings

Resources section:

- /
- /numbers
- OPTIONS
- /numbers/{num1}
- GET** (selected)

Body Mapping Templates section:

Content-Type: application/json

application/json

Generate template:

```
1 #set($response = $input.body)
2 #set($originalParams = $input.params())
3 #set($ip = $context.identity)
4 {
5   "num1" : $originalParams.path.num1,
6   "num2" : $originalParams.querystring.num2,
7   "description" : "I added these numbers",
8   "result" : $response
9 }
```

Buttons: Cancel, Save

Add integration response

And Finally, the Response Model

The screenshot shows the AWS Management Console with the following navigation path: Services > Resource Groups > EC2 > RDS > CloudWatch > DynamoDB > Bill Boulder > N. Virginia > Support. The main area displays the API Gateway interface for the 'hello-world-2' API.

APIs sidebar:

- hello-world-2
- Resources** (selected)
- Stages
- Authorizers
- Gateway Responses
- Models
- Documentation
- Binary Support
- Dashboard
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates
- Settings

Method Execution /numbers/{num1} - GET - Method Response

Provide information about this method's response types, their headers and content types.

HTTP Status

200

Response Headers for 200

Name	Value
No headers	

Add Header

Response Body for 200

Content type	Models
application/json	MagicResult

Add Response Model

Add Response



Pls

hello-world-2

Resources

Stages

Authorizers

Gateway Responses

Models

Documentation

Binary Support

Awesome

Usage Plans

API Keys

Custom Domain Names

Cert Certificates

Settings

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

Request: /numbers/2?num2=5

{num1}

Status: 200

2

Latency: 86 ms

Query Strings

Response Body

num2

5

```
{  
  "num1": 2,  
  "num2": 5,  
  "description": "I added these numbers",  
  "result": 7  
}
```

Headers

Response Headers

No header parameters exist for this method. You can add them via Method Request.

```
{"X-Amzn-Trace-Id":"sampled=0;root=1-59bf5ce1-c24ca6969fb6805e4036b0bf","Content-Type":"application/json"}
```

Stage Variables

Logs

No stage variables exist for this method.

Execution log for request test-request
Mon Sep 18 05:42:57 UTC 2017 : Starting execution for request: test-invoke-request



A large white triangle is positioned in the upper right corner of the slide, containing the text "10 Minute Break". The background is black with a uniform grid of light gray circles.

**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”

tifications Messages 

Search Twi

131 2.2K 19K

WeRateDogs™ (Oct 3)  @dog_rates · Sep 13
This is Charlie. He's addicted to broccoli. It's tearing his family apart. Won't admit he has a problem. 11/10 we're here for you Charlie



katie

171 7.1K 33K

WeRateDogs™ (Oct 3)  @dog_rates · Sep 12
Here's a doggo who's rather h*ckin startled by the moving stairs. Puppreciates the assistance good sir. 14/10

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.

Queryfeed

Read Twitter, Google Plus and Facebook on RSS

Classical Twitter Search

Searches tweets using official [Twitter API](#).

#hashtag, from:username, to:username, @username, love OR hate, more patterns...

How to compose a title

Geographical restrictions

Use geocode to strict tweets location.

Radius units must be specified as either "mi" (miles) or "km" (kilometers).

Example: [37.781157,-122.398720,1km](#)

Omit direct messages (started with @someuser)

Omit retweets (started with RT)

Show images as attachments

Alternative Twitter Search

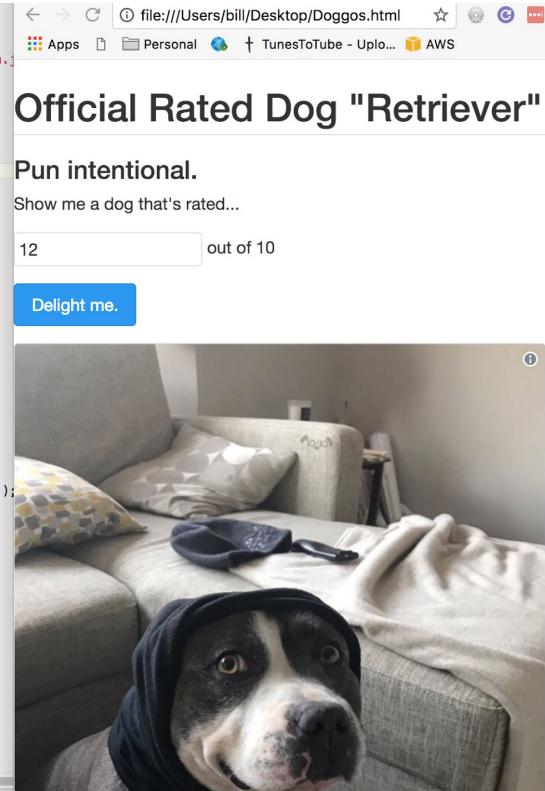
Parses Twitter pages, so the results are quite close to what you see in Twitter in realtime.

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 <div> in the page to become an embedded picture of a doggo rated that score.

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

```
1  <html>
2  <head>
3  <!-- Authorship: Bill "Spruke" Boulden, not that it's anything to be proud of. -->
4  <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js">
5  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/marx/2.0.7/marx.min.css" />
6  <script type="text/javascript" src="https://platform.twitter.com/widgets.js"></script>
7
8  </head>
9  <body>
10 <h1>Official Rated Dog "Retriever"</h1>
11 <h3>Pun intentional.</h3>
12
13 <p>Show me a dog that's rated...</p>
14 <p><input type="number" id="rating"></input> out of 10</p>
15 <p><button id="retrieve">Delight me.</button>
16
17 <div id="embed-tweet-here">
18 </div>
19
20 <script type="text/javascript">
21
22 $(document).ready(function() {
23   $('#retrieve').click(function () {
24     $('#embed-tweet-here').empty();
25     $.get('https://3t2l2ug9si.execute-api.us-east-1.amazonaws.com/prod/doggos?rating='
26       + $('#rating').val())
27     .done(function(json) {
28       let tweetNumber = json.tweet.match(/\/(\d+)/$)[1];
29       console.log(tweetNumber);
30       twtr.widgets.createTweet(tweetNumber, document.getElementById('embed-tweet-here'));
31     });
32   });
33 });
34
35
36 </script>
37 </body>
38 </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

```
bill — ubuntu@ip-172-31-34-51:/home/ubuntu/doggos — ssh -i Documents/Keys/spruke.pem ubu...
19:22:08 ubuntu@i-a265311d {~/doggos}
[$ ls doggos.js
doggos.js
19:22:12 ubuntu@i-a265311d {~/doggos}
[$ mkdir node_modules
19:22:16 ubuntu@i-a265311d {~/doggos}
[$ npm install lodash rss-parser request
/home/ubuntu/doggos
└─ lodash@4.17.4
└─ request@2.87.0
19:23:59 ubuntu@i-a265311d {~/doggos}
[$ zip -q -r doggos doggos.js node_modules/
19:24:06 ubuntu@i-a265311d {~/doggos}
[$ aws lambda create-function --function-name "doggos" --runtime nodejs6.10 --role "arn:aws:iam::712800520113:role/service-role/hello-world" --handler "doggos.handler" --description "dog rater" --timeout 60 --memory-size 128 --zip-file fileb://doggos.zip
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "qagnWusISdTGxm0qzINcXKfMrGaJu+OJXEu3xWzQ6Jk=",
    "FunctionName": "doggos",
    "CodeSize": 2841385,
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-east-1:712800520113:function:doggos",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::712800520113:role/service-role/hello-world",
    "Timeout": 60,
    "LastModified": "2017-09-21T19:24:22.097+0000",
    "Handler": "doggos.handler",
    "Runtime": "nodejs6.10",
    "Description": "dog rater"
}
19:24:24 ubuntu@i-a265311d {~/doggos}
[$ aws lambda invoke --function-name "doggos" --payload='{"rating": 11, "outOf": 10}' --log-type TAIL --query=LogResult /tmp/answer.txt | sed -e 's//"/g' | base64 --decode
START RequestId: 7e2eda7c-9f02-11e7-b585-45aaf333d584 Version: $LATEST
END RequestId: 7e2eda7c-9f02-11e7-b585-45aaf333d584
REPORT RequestId: 7e2eda7c-9f02-11e7-b585-45aaf333d584 Duration: 1328.52 ms Billed Duration: 1400 ms Memory Size: 128 MB Max Memory Used: 45 MB
19:24:42 ubuntu@i-a265311d {~/doggos}
$
```

Start Building An API Gateway!

The screenshot shows the Amazon API Gateway interface. At the top, there's a navigation bar with icons for Services, Resource Groups, EC2, RDS, CloudWatch, and DynamoDB, along with user information for Bill Boulden and location N. Virginia. Below the navigation bar, the main title is "APIs > Create". On the left, a sidebar lists "APIs", "hello-world-2", "Usage Plans", "API Keys", "Custom Domain Names", "Client Certificates", and "Settings". The main content area is titled "Create new API" and contains a descriptive paragraph: "In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints." Below this are four radio button options: "New API" (selected), "Clone from existing API", "Import from Swagger", and "Example API". The "Settings" section is titled "Choose a friendly name and description for your API." It includes fields for "API name*" (containing "doggos") and "Description" (containing "rate a doggo!"). At the bottom left is a note "* Required", and at the bottom right is a blue "Create API" button.

Services | Resource Groups | EC2 | RDS | CloudWatch | DynamoDB | Bill Boulden | N. Virginia | Support

Amazon API Gateway | APIs > Create | Show all hints | ?

APIs

hello-world-2

Usage Plans

API Keys

Custom Domain Names

Client Certificates

Settings

Create new API

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

New API Clone from existing API Import from Swagger Example API

Settings

Choose a friendly name and description for your API.

API name* doggos

Description rate a doggo!

* Required

Create API

API Gateway, cont.

The screenshot shows the AWS API Gateway console interface. At the top, there's a navigation bar with links for Services, Resource Groups, EC2, RDS, CloudWatch, DynamoDB, and other account details. Below the navigation bar, the URL indicates the path: APIs > doggos (3t2l2ug9si) > Resources > / (411jvkvi95) > GET.

The main area is titled "o/ - GET - Setup". On the left, a sidebar lists various API-related options: APIs, Resources (which is selected and highlighted in orange), Actions, Stages, Authorizers, Gateway Responses, Models, Documentation, Binary Support, hello-world-2, Usage Plans, API Keys, Custom Domain Names, Client Certificates, and Settings.

In the center, the path "o/ - GET" is displayed, with "GET" being the selected method. A callout bubble says "Choose the integration point for your new method." Below this, the "Integration type" section shows "Lambda Function" selected (indicated by a blue circle). Other options include "HTTP", "Mock", and "AWS Service".

Below the integration type, there are two checkboxes: "Use Lambda Proxy integration" (unchecked) and "Lambda Region" (set to "us-east-1"). Under "Lambda Function", the name "doggos" is entered. At the bottom right, a large blue "Save" button is visible.

At the very bottom of the page, there's a footer with links for "Privacy Policy" and "Terms of Use".

Extract Rating Parameter On The Way In

The screenshot shows the AWS API Gateway console with the following details:

- Top Navigation:** Services, Resource Groups, EC2, RDS, CloudWatch, DynamoDB, Bill Boulden, N. Virginia, Support.
- Breadcrumbs:** APIs > doggos (3t2l2ug9si) > Resources > /doggos (awv2ty) > GET
- Left Sidebar:** APIs, doggos, Resources (selected), Stages, Authorizers, Gateway Responses, Models, Documentation, Binary Support, hello-world-2, Usage Plans, API Keys, Custom Domain Names, Client Certificates, Settings.
- Current View:** Method Execution for /doggos - GET - Method Request.
- Method Settings:**
 - Authorization: NONE
 - Request Validator: NONE
 - API Key Required: false
- URL Query String Parameters:**

Name	Required	Caching
rating	<input type="checkbox"/>	<input type="checkbox"/>

+ Add query string
- HTTP Request Headers:** (List item)
- Request Body:** (List item)

And Map It to the Lambda While Hardcoding OutOf

The screenshot shows the AWS Lambda API Gateway Mapping Template editor. The URL in the browser bar is `https://console.aws.amazon.com/apigateway/v2/resources/3t2l2ug9si/resources/doggos/methods/awv2ty`. The left sidebar lists APIs, Stages, and other resources. The main pane shows a GET method for the path `/doggos`. The response content type is set to `application/json`. The mapping template is defined as:

```
1 #set($allParams = $input.params())
2 {
3     "rating": $allParams.querystring.rating,
4     "outOf": 10
5 }
```

At the bottom right are `Cancel` and `Save` buttons.

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

Official Rated Dog "Retriever"

Pun intentional.

Show me a dog that's rated...

12

out of 10

Delight me.



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 name  
    description: My function # The description of your function.  
    memorySize: 512 # memorySize for this specific function.  
    runtime: nodejs6.10 # Runtime for this specific function. Overrides the default runtime.  
    timeout: 10 # Timeout for this specific function. Overrides the default timeout.  
    role: arn:aws:iam::XXXXXXXX:role/role # IAM role which will be used for this function.  
    onError: arn:aws:sns:us-east-1:XXXXXX:sns-topic # Optional SNS topic arn.  
    awsKmsKeyArn: arn:aws:kms:us-east-1:XXXXXX:key/some-hash # Optional KMS key arn.  
    environment: # Function level environment variables  
      functionEnvVar: 12345678  
    tags: # Function specific tags  
      foo: bar  
  vpc: # Optional VPC. But if you use VPC then both subproperties (securityGroupIds and subnetIds) are required.  
    securityGroupIds:  
      - securityGroupId1  
      - securityGroupId2  
    subnetIds:  
      - subnetId1  
      - subnetId2  
  events: # The Events that trigger this Function  
    - http: # This creates an API Gateway HTTP endpoint which can be used to trigger this function.  
      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 add the appropriate headers.  
      private: true # Requires clients to add API keys values in the `x-api-key` header.  
      authorizer: # An AWS API Gateway custom authorizer function  
        name: authorizerFunc # The name of the authorizer function (must be defined in the provider section)  
        arn: xxx:xxx:Lambda-Name # Can be used instead of name to refer to the ARN.  
        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

