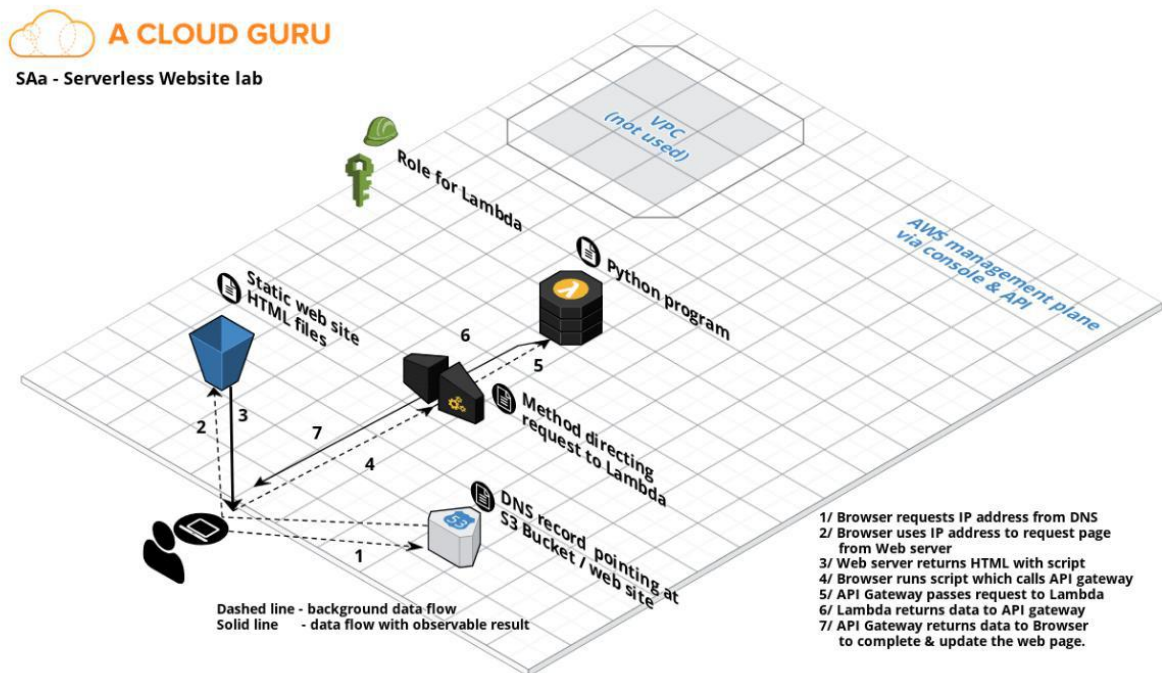


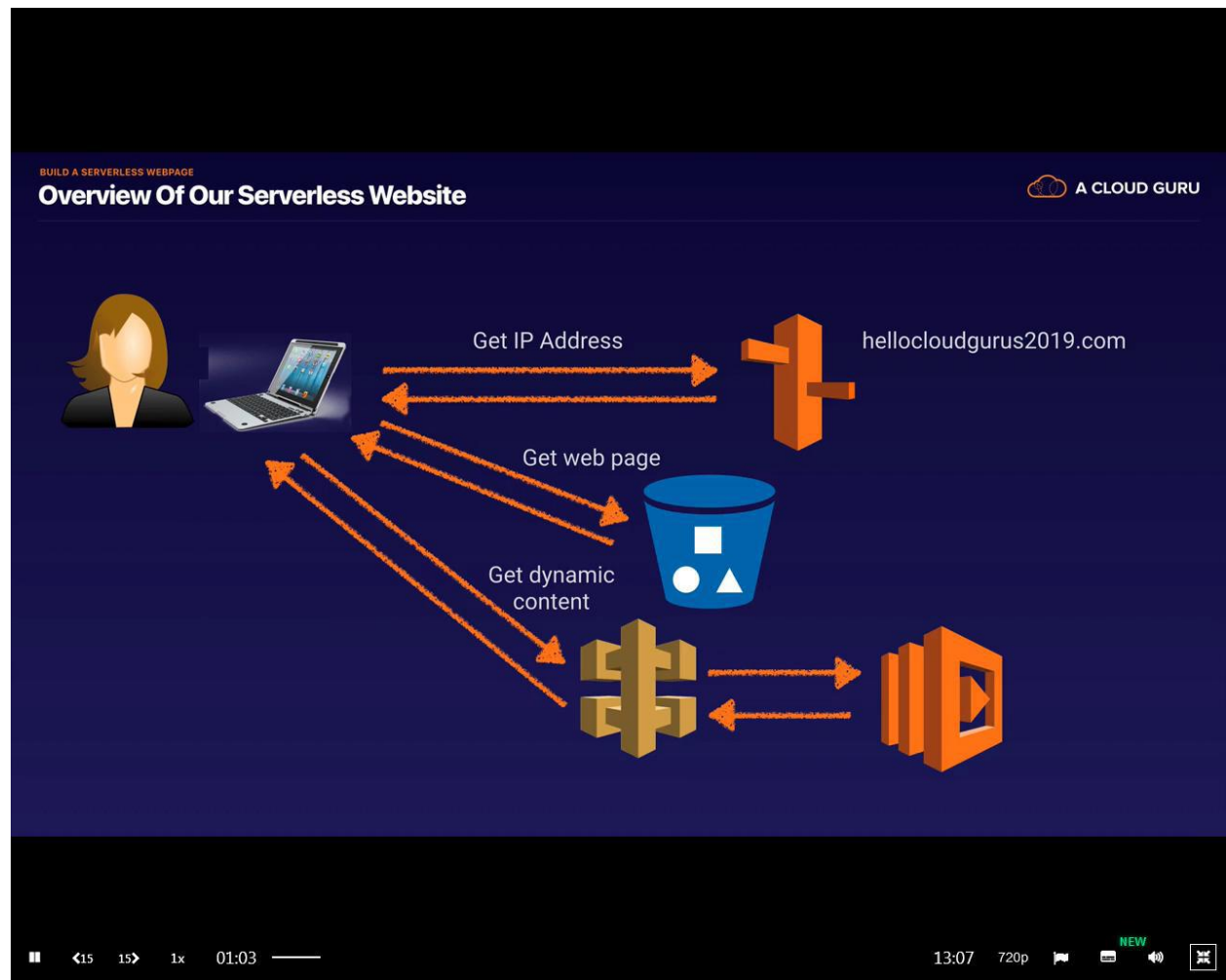
Build a Serverless Web page with API Gateway & Lambda

Architecture Involved:



Desired Result:

Our user types our website URL into their browser and hits enter. Our user is essentially sending a query to Route53 (Amazon's Domain Name System), which will respond back with the bucket address for our website (because as you will see, we are hosting our webpage using an S3 bucket). Our user will be directed to our index.html page. On our index.html page we will have a button which will prompt dynamic content. When the button is clicked, a request is sent through API Gateway, which sends the request to a Lambda function. The lambda function will take the data and return a result to API Gateway, which will then return the result to our user.



Section 1) Let's create our Lambda function

Login to the AWS console

Select the region that you want. For this project I selected the Singapore region.

Click **Services** and then click **Lambda** (which is located under compute)

Click the **Create function** button

Select **Author from scratch**

Name: MyFirstLambdaFunction ← give your function a name

Runtime: ✓ Python 3.8 ← select Python 3.8 (the latest version of Python)

Role: ✓ Create a new role from AWS policy templates ← select create a new role from AWS policy templates

Role name: MyLambdaExecutionRole ← give your role a name

Policy templates: ✓ Simple microservice permissions

Click **Create function**

Copy the python code below:

```
import json
```

```
def lambda_handler(event, context):
```

```
    print("In lambda handler")
```

```
    resp = {
```

```
        "statusCode": 200,
```

```
        "headers": {
```

```
            "Access-Control-Allow-Origin": "*",
```

```
        },
```

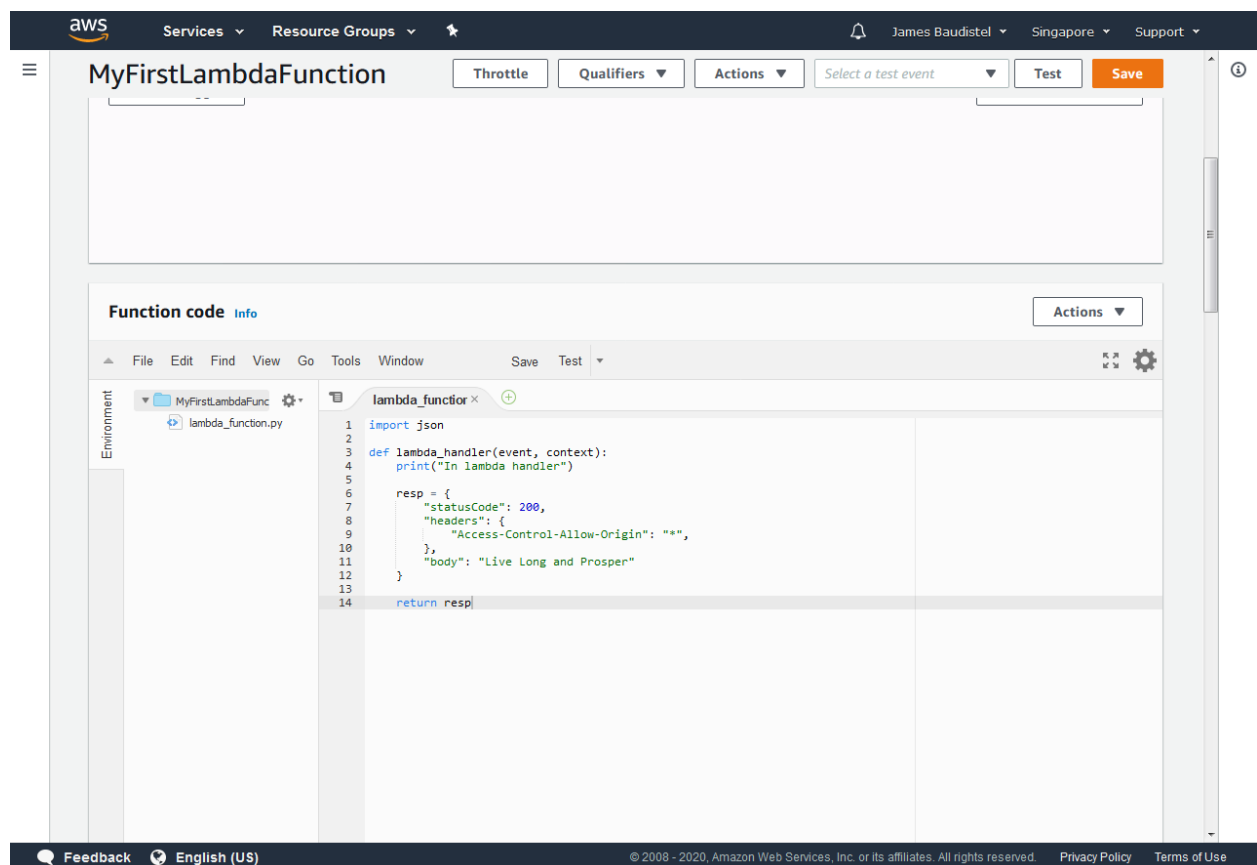
```
        "body": "Live Long and Prosper"
```

```
    }
```

```
    return resp
```

Delete the code that is initially present in the **Function code** section and instead paste the copied code into the **Function code** section:

See the below screenshot for clarification



Then click **File** and click **Save**

Also, click **Save** located next to where it says Test ←do not forget this step ★ alternatively you can click the orange save button. The orange button should then turn grey and become unclickable.

Scroll down to where it says Basic settings and click Edit (see screenshot)

The screenshot displays the AWS Lambda console interface for a function named 'MyFirstLambdaFunction'. At the top, the AWS logo and navigation menu are visible, including 'Services', 'Resource Groups', and user information 'James Baudistel' in 'Singapore'. The function name 'MyFirstLambdaFunction' is prominently displayed, followed by buttons for 'Throttle', 'Qualifiers', 'Actions', and a 'Test' button with a 'Save' button next to it. Below this, a 'No tags' section indicates no tags are associated with the function, with a 'Manage tags' button. The main configuration area is divided into three sections: 'Basic settings', 'Monitoring tools', and 'VPC'. The 'Basic settings' section includes fields for 'Description' (currently empty), 'Runtime' (set to 'Python 3.6'), 'Handler' (set to 'lambda_function.lambda_handler'), 'Memory (MB)' (set to '128'), and 'Timeout' (set to '0 min 3 sec'). The 'Monitoring tools' section shows 'Logs and metrics (Default)' as 'Enabled' and 'Active tracing' as 'Not enabled'. The 'VPC' section indicates 'No VPC configuration' and that the function is not connected to a VPC. A 'File system' section at the bottom has an 'Add file system' button. The footer contains a 'Feedback' link, 'English (US)' language selection, and copyright information for Amazon Web Services, Inc. from 2008 to 2020.

aws Services Resource Groups James Baudistel Singapore Support

MyFirstLambdaFunction

Throttle Qualifiers Actions Select a test event Test Save

No tags
No tags associated with this function.
Manage tags

Basic settings

[Info](#) [Edit](#)

| | |
|--------------------------------|-------------|
| Description | Runtime |
| - | Python 3.6 |
| Handler Info | Memory (MB) |
| lambda_function.lambda_handler | 128 |
| Timeout | |
| 0 min 3 sec | |

Monitoring tools

[Info](#) [Edit](#)

| | |
|----------------------------|----------------|
| Logs and metrics (Default) | Active tracing |
| Enabled | Not enabled |

VPC

[Info](#) [Edit](#)

No VPC configuration
This function isn't connected to a VPC.
[Edit](#)

File system

[Info](#) [Add file system](#)

Feedback English (US) © 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Description: My First Lambda Function ←give it a description

Then click the orange save button

(see screenshot below)

Edit basic settings

Basic settings [Info](#)

Description - optional
My First Lambda Function

Runtime
Python 3.6

Handler [Info](#)
lambda_function.lambda_handler

Memory (MB)
Your function is allocated CPU proportional to the memory configured.
128 MB

Timeout
0 min 3 sec

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
service-role/MyLambdaExecutionRole

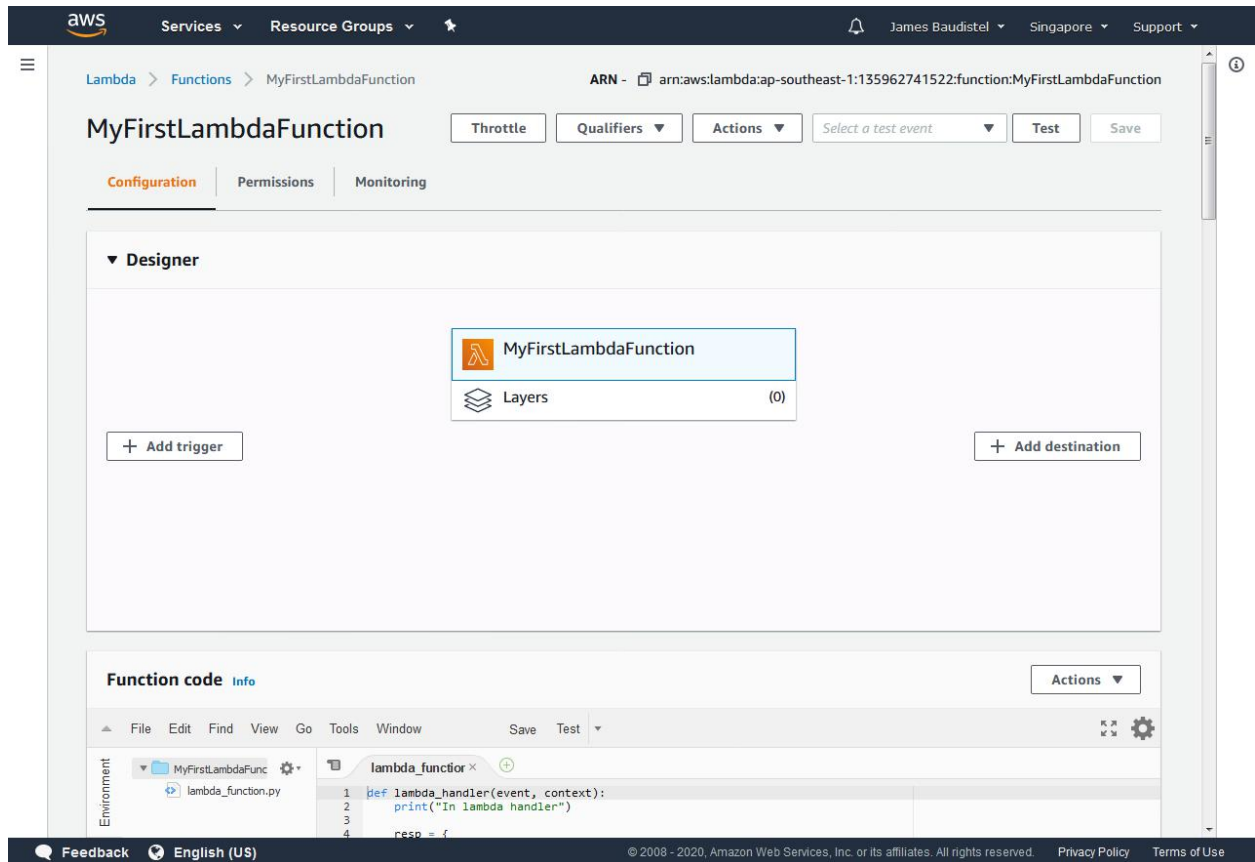
[View the MyLambdaExecutionRole role](#) on the IAM console.

Cancel Save

Now let's configure a trigger for our function:

Click the **Add trigger** button

(See the screenshot below)



Trigger Configuration: ✓ API Gateway ← select API Gateway as the trigger for your lambda function

API: ✓ Create an API ← select Create an API because we want to create a new one

API type: ✓ Rest API ← select Rest API

Security: ✓ IAM ← select IAM


(See the screenshot below for clarification)

aws Services Resource Groups James Baudistel Singapore Support

Lambda > Add trigger

Add trigger

Trigger configuration

 **API Gateway**
api application-services aws serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

API
Create a new API or attach an existing one.

Create an API

API type

☐ HTTP API
Create an HTTP API.

☒ **REST API**
Create a REST API.

Security
Configure the security mechanism for your API endpoint.

IAM

Use IAM permissions to authorize access to your API. Users are required to include authentication signatures in their HTTP calls.

▼ Additional settings

API name
Choose a name for your API. API names don't need to be unique.

MyFirstLambdaFunction-API

Deployment stage
The name of your API's deployment stage.

default

Feedback English (US) © 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Now scroll down and click the **Add** button at the bottom of the page
(see the below screenshot)

Configure the security mechanism for your API endpoint.

IAM

Use IAM permissions to authorize access to your API. Users are required to include authentication signatures in their HTTP calls.

▼ Additional settings

API name
Choose a name for your API. API names don't need to be unique.

MyFirstLambdaFunction-API

Deployment stage
The name of your API's deployment stage.

default

Cross-origin resource sharing (CORS)
CORS is required to call your API from a webpage that isn't hosted on the same domain. To enable CORS for a REST API, set the Access-Control-Allow-Origin header in the response object that you return from your function code.

☐ **Enable metrics and error logging**
Record latency and error metrics in Amazon CloudWatch, and log errors in Amazon CloudWatch Logs. Standard CloudWatch and CloudWatch Logs pricing applies.

Binary media types
Specify response types that API Gateway should treat as binary data instead of text. To treat all responses as binary data, use */*. If you enter one or more specific types, you must also set the Content-Type header in the response object that you return from your function code.

image/png Remove

Add

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

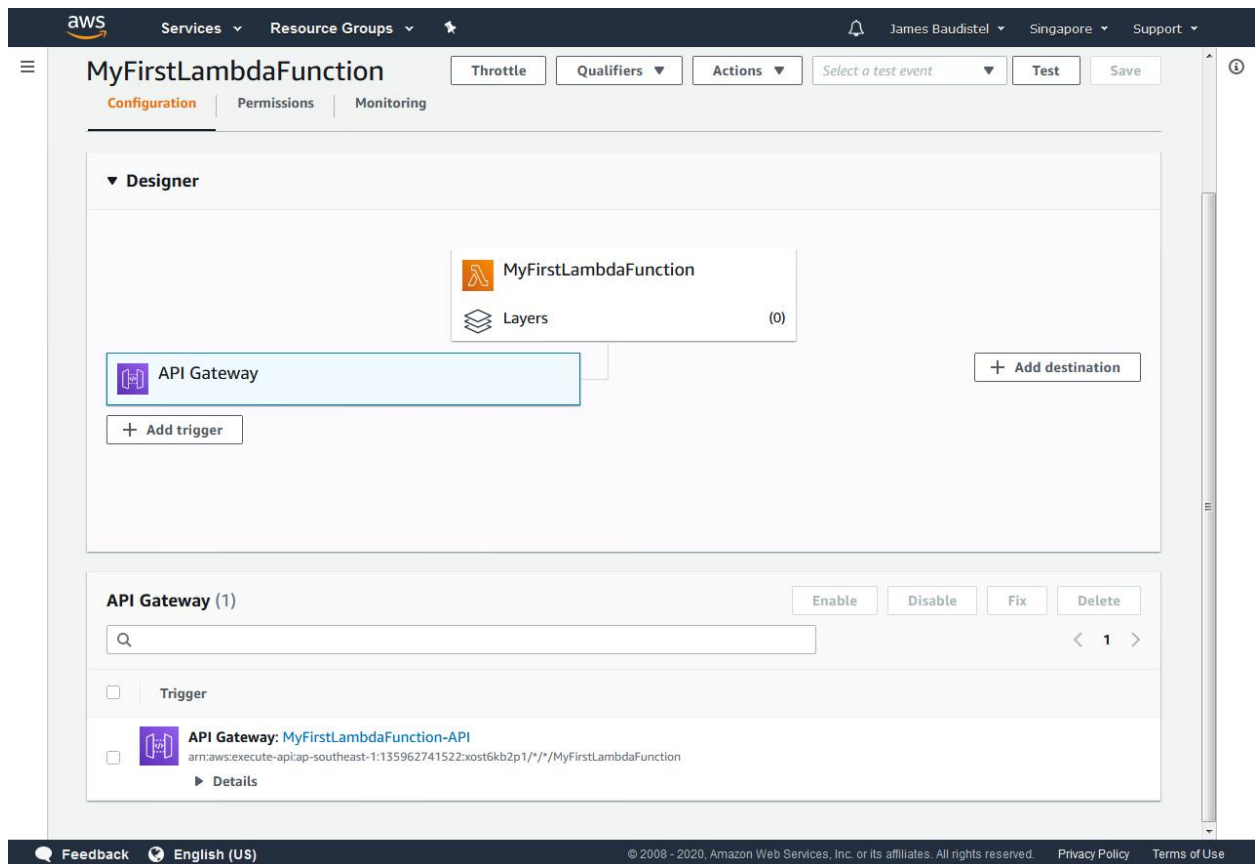
Feedback English (US) © 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

The trigger MyFirstLambdaFunction-API was successfully added to function MyFirstLambdaFunction. The function is now receiving events from the trigger.

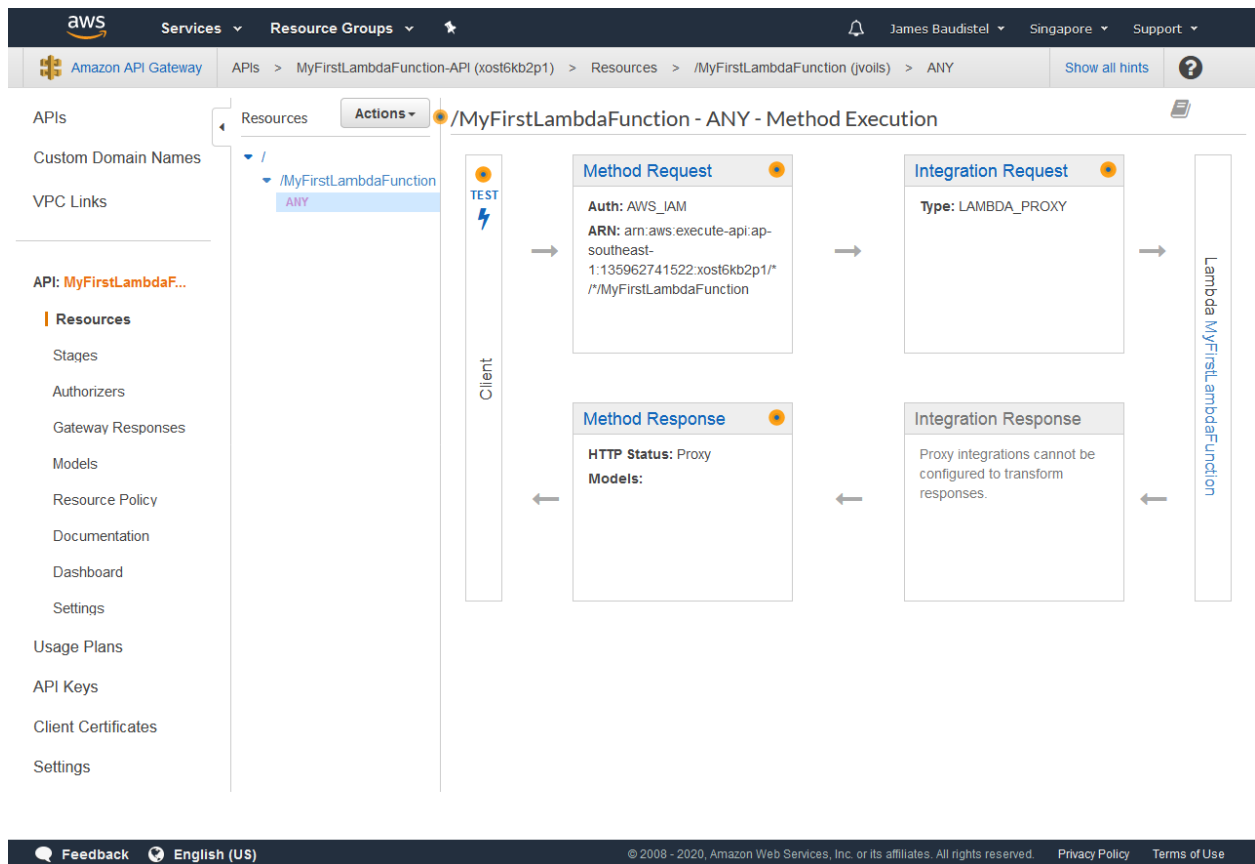
In otherwords, our API Gateway is now a trigger for the Lambda function we created. It has created a new API endpoint.

Scroll down and click **MyFirstLambdaFunction-API**

(per the screenshot below)



This will open up API Gateway
(per the screenshot below)



Select **ANY** and then click the **Actions** button and click **Delete Method**

Then click **Actions** and click **Create Method** and then Select **GET** (because we want to create a Get request) and click the ✓ checkmark symbol to confirm your selection

Integration Type: Lambda Function

Use Lambda Proxy Integration: ✓ ← select that you want to Use Lambda Proxy Integration

Lambda Region: ap-southeast-1 ← choose the region from which you created your lambda function. I created my function in the Singapore region, hence ap-southeast-1

Lambda Function: MyFirstLambdaFunction ← type the name of our function which is MyFirstLambdaFunction (it will likely prompt once you start typing)

Then click **Save** (This will give API Gateway permission to invoke the Lambda function)

Then click **OK**

(See the below screenshot for clarification)

The screenshot displays the AWS API Gateway console interface. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information. The breadcrumb trail shows the path: APIs > MyFirstLambdaFunction-API (xost6kb2p1) > Resources > /MyFirstLambdaFunction (jvoils) > GET. The left sidebar contains a navigation menu with categories like APIs, Custom Domain Names, and VPC Links, and a specific section for 'API: MyFirstLambdaF...' with sub-items like Resources, Stages, and Authorizers. The main content area is titled '/MyFirstLambdaFunction - GET - Setup' and prompts the user to 'Choose the integration point for your new method.' The configuration options include: 'Integration type' set to 'Lambda Function' (with options for HTTP, Mock, AWS Service, and VPC Link); 'Use Lambda Proxy integration' checked; 'Lambda Region' set to 'ap-southeast-1'; 'Lambda Function' set to 'MyFirstLambdaFunction'; and 'Use Default Timeout' checked. A blue 'Save' button is located at the bottom right of the configuration area. The footer contains a 'Feedback' link, 'English (US)' language selector, and copyright information for Amazon Web Services, Inc.

Make sure GET is highlighted and then click Actions and click Deploy API (per the screenshot below)

aws Services Resource Groups James Baudistel Singapore Support

Amazon API Gateway APIs > MyFirstLambdaFunction-API (xost6kb2p1) > Resources > /MyFirstLambdaFunction (jvoils) > GET Show all hints ?

APIs
Custom Domain Names
VPC Links

API: MyFirstLambdaF...

Resources

- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

Resources

Actions

- METHOD ACTIONS
 - Edit Method Documentation
 - Delete Method
- RESOURCE ACTIONS
 - Create Method
 - Create Resource
 - Enable CORS
 - Edit Resource Documentation
 - Delete Resource
- API ACTIONS
 - Deploy API
 - Import API
 - Edit API Documentation
 - Delete API

/MyFirstLambdaFunction - GET - Method Execution

Method Request

Auth: NONE
ARN: arn:aws:execute-api:ap-southeast-1:135962741522:xost6kb2p1/* /GET/MyFirstLambdaFunction

Integration Request

Type: LAMBDA_PROXY

Method Response

HTTP Status: Proxy
Models: application/json => Empty

Integration Response

Proxy integrations cannot be configured to transform responses.

Lambda MyFirstLambdaFunction

Feedback English (US) © 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Deployment Stage: ✓ Default ← select default

Deployment Description: MyProdDeployment ← give the deployment a description

Then click **Deploy**



★Important Step★

Now click **default** and Click **GET** ← **make sure to do this step before your proceed** ★★★

Then click the Invoke URL (which should say something like

<https://xost6kb2p1.execute-api.ap-southeast-1.amazonaws.com/default/MyFirstLambdaFunction>

)

(See the screenshot below for clarification)

Upon clicking on the Invoke URL, your screen should say Live Long and Prosper

Now copy that Invoke URL link into your clipboard and paste it into a new notepad file just so you have it for later when we need it. It should look something like this (but yours will be slightly different):

`https://13jevc7hud.execute-api.ap-southeast-1.amazonaws.com/default/MyFirstLambdaFunction`

Now using a code editor such as Visual Studio Code, Brackets, etc. create a file called index.html and paste in the following code:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
function myFunction() {  
  
    var xhttp = new XMLHttpRequest();  
  
    xhttp.onreadystatechange = function() {  
  
        if (this.readyState === 4 && this.status === 200) {  
  
            document.getElementById('my-demo').innerHTML = this.responseText;  
  
        }  
  
    };  
  
    xhttp.open('GET', 'YOUR API ENDPOINT HERE', true);  
  
    xhttp.send();  
  
}
```

</script>

</head>

<body>

<div align="center">

<h1>

dif-tor heh smusma

</h1>

<button onclick="myFunction()">Translate Vulcan</button>


```

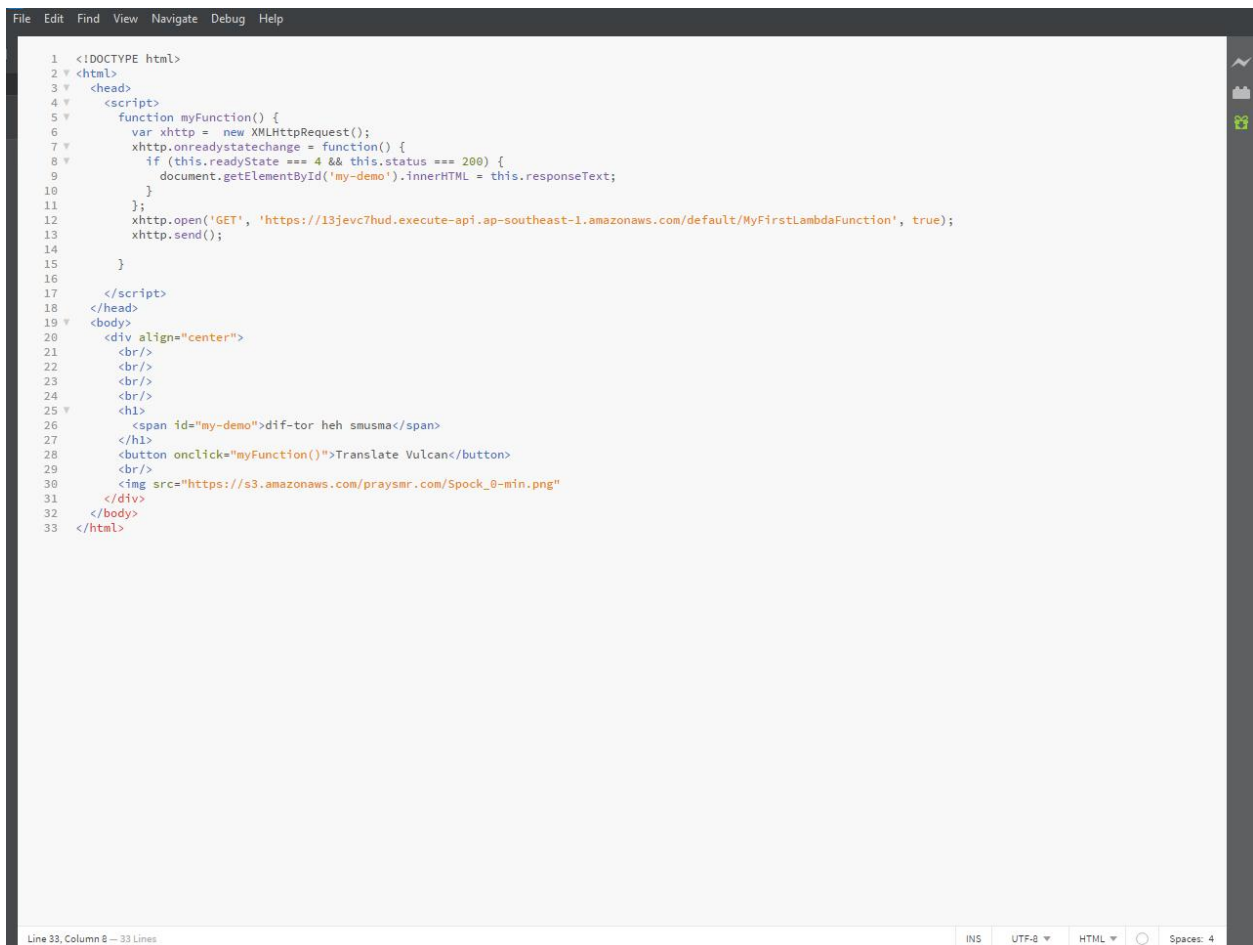

</body>

</html>
```

★★★Important Note: On Line 12 of the code, where it says 'Put Your Endpoint API here', replace that with the invoke URL that you copied. Make sure your invoke URL is surrounded by single quotes. It will look similar to this:

```
xhttp.open('GET', 'https://13jevc7hud.execute-api.ap-southeast-1.amazonaws.com/default/MyFirstLambdaFunction', true);
```

★★★See the below screenshot for clarification and note line 12 in particular. That is where you paste your invoke URL surrounded by single quotes.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <script>
5     function myFunction() {
6         var xhttp = new XMLHttpRequest();
7         xhttp.onreadystatechange = function() {
8             if (this.readyState === 4 && this.status === 200) {
9                 document.getElementById('my-demo').innerHTML = this.responseText;
10            }
11        };
12        xhttp.open('GET', 'https://13jevc7hud.execute-api.ap-southeast-1.amazonaws.com/default/MyFirstLambdaFunction', true);
13        xhttp.send();
14    }
15
16
17 </script>
18 </head>
19 <body>
20     <div align="center">
21         <br/>
22         <br/>
23         <br/>
24         <h1>
25
26         <span id="my-demo">dif-tor heh smusma</span>
27     </h1>
28     <button onclick="myFunction()">Translate Vulcan</button>
29
30     
32 </body>
33 </html>
```

Line 33, Column 8 — 33 Lines

INS UTF-8 HTML Spaces: 4

Now save the file that we just created. ★ Make sure it is saved as index.html

Now let's create an error.html page in case our user types the url with an incorrect path

Create a file called error.html using a code editor such as VSC or Brackets and paste in the following code.

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8">

    <title>Error</title>

  </head>

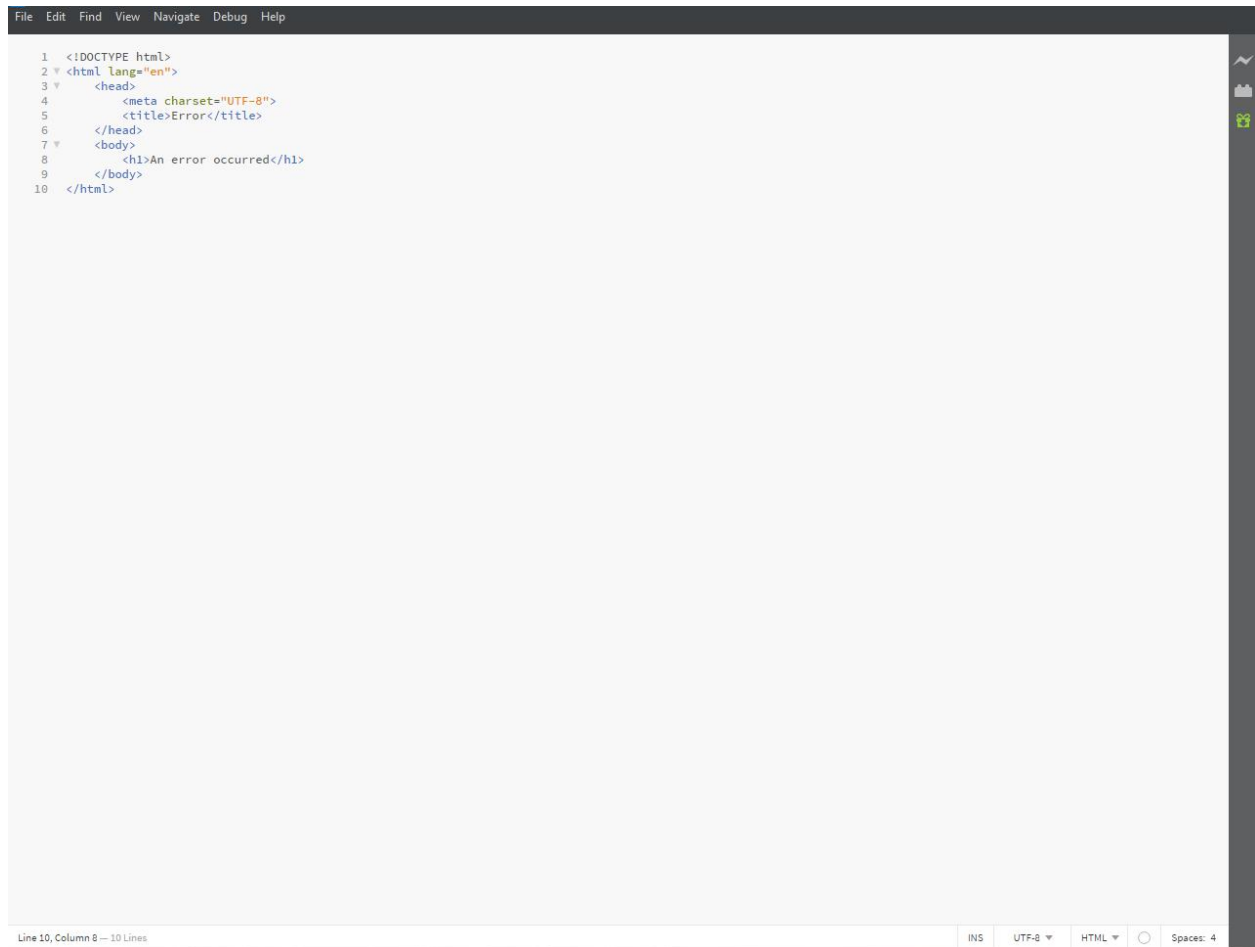
  <body>

    <h1>An error occurred</h1>

  </body>

</html>
```

★ See the below screenshot for clarification

A screenshot of a code editor window with a dark theme. The editor displays an HTML template for an error page. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Error</title>
6 </head>
7 <body>
8   <h1>An error occurred</h1>
9 </body>
10 </html>
```

The editor has a menu bar at the top with 'File', 'Edit', 'Find', 'View', 'Navigate', 'Debug', and 'Help'. On the right side, there are icons for a search, a file explorer, and a run button. At the bottom, a status bar shows 'Line 10, Column 8 — 10 Lines', 'INS', 'UTF-8', 'HTML', and 'Spaces: 4'.

★ Make sure it is saved as error.html

Section 2) Now we will purchase a domain name (this costs about \$12)

(If you do not want to purchase a domain name feel free to skip this step, but make sure to create a bucket in S3 as described in this Section. If you do not purchase a domain name, then the name you give to the bucket you create in this section will NOT matter)

Go to the AWS console

Click **Services** and Click **Route 53** which is located under Networking and Content Delivery

Go to Register Domain and type in the domain you would like to purchase and then click the **check** button to see if it is available.

★★★ If the domain name is available you want to also make sure that the bucket name is available in S3. So click **Services**, then click **S3**. Then click **Create bucket**. If the URL you are trying to purchase is

m3mes.com then name your bucket m3mes.com ← make sure to name your bucket **EXACTLY** the same as the URL you are going to purchase (including the .com portion)!!! So if your domain is going to be **m3mes.com** then you need to create a bucket name called **m3mes.com**

(If you do not want to purchase a domain name then you can name your bucket whatever you want so long as it's available)

Then select a region for your bucket and click **create**.

Once you have created your bucket, click **Services** and click **Route 53**. Type in the domain name you wish to purchase and click check. If it is available, it will be in your shopping cart. Then scroll down and click **Continue** (in order to continue with the checkout process). Simply follow the prompts to complete the transaction.

Section 3) Once you have your new domain name go back to S3 by clicking **Services** and then click **S3**.

Click the bucket you created in Section 2 of this lesson. Then click the **permissions** tab. Then click the **Block public access** tab.

Click **Edit**

Uncheck the box that says Block all public access (because we want the public to be able to access our site)

Then click **Save**

Type confirm (to confirm) and click **Confirm**

(See the screenshot below for clarification)

Overview

Properties

Permissions

Management

Access points

Block public access

Access Control List

Bucket Policy

CORS configuration

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.

☐ **Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Cancel

Save

Feedback

English (US)

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

Once that is complete, click the **Permissions** tab again and click the **Bucket Policy** button. As your bucket policy, paste in the following JSON code:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "YOUR ARN HERE/*"
    }
  ]
}
```

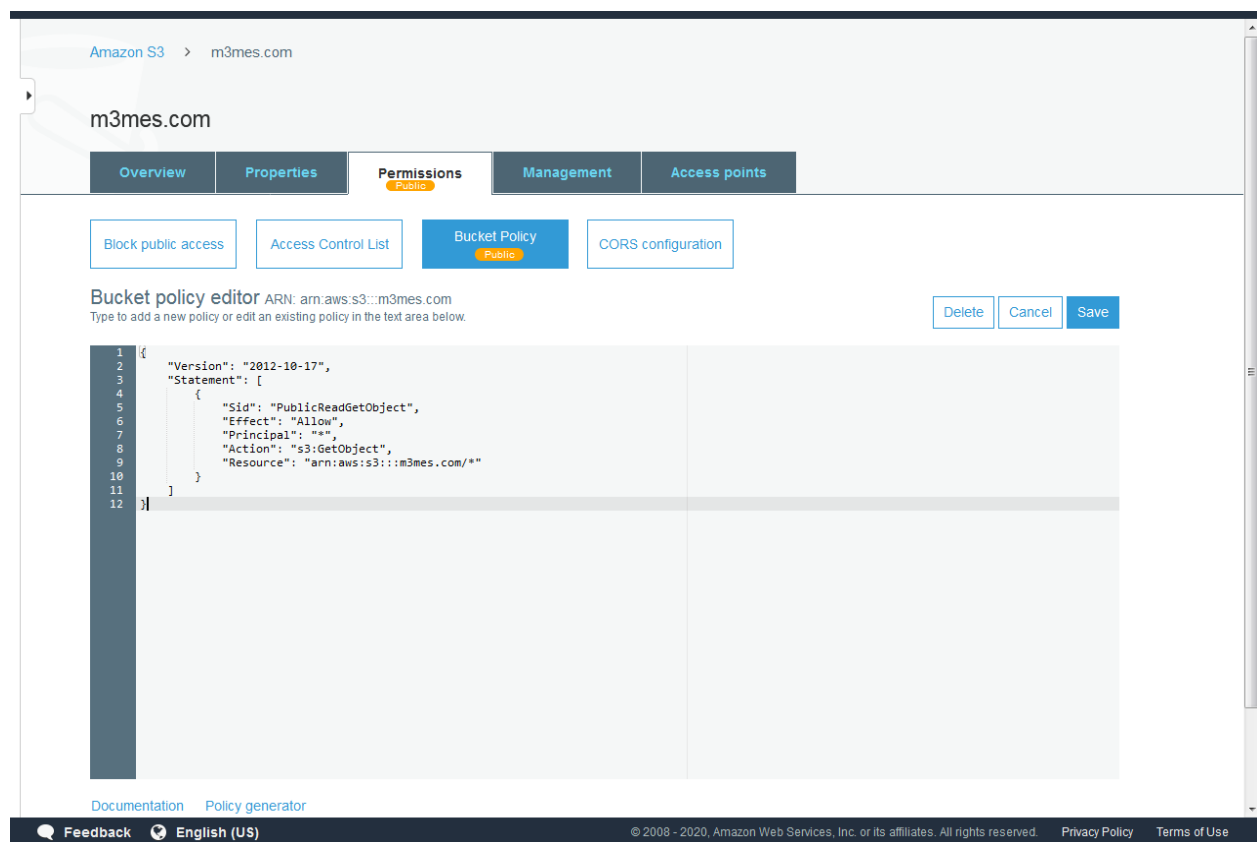
★★★On line 9 of the code where it says YOUR ARN HERE replace it with the ARN of the bucket you created. The ARN is located just below the Block public access tab where it says **Bucket policy editor**. So for the bucket that I created called m3mes.com the ARN is arn:aws:s3:::m3mes.com

Thus, line 9 of our JSON code will say "**Resource**": "arn:aws:s3:::m3mes.com/*"

(Your ARN will be different because your bucket name is different than the one I created)

Then click **Save** (and the word **public** will appear on your Bucket Policy button as well as on the Permissions tab)

★★★See the screenshot below for clarification and make sure to change line 9 of the JSON code to have your ARN (with a forward slash and asterisk at the end and surrounded by double quotes).



Now click the **Overview** Tab

Click **Upload**

Then click **Add files**

Select the index.html file as well as the error.html file that you created earlier and then click **upload**. Or upload each file one at a time. (See the below screenshot for clarification)



Once your files have been uploaded to your bucket,

Click the **Properties** tab and click **Static Website Hosting**

✓ **Use this bucket to host a website** ← Select Use this bucket to host a website

Index document: index.html ← type index.html

Error document: error.html ← type error.html

Then click **Save**

Click the **Overview** tab and click your **index.html** file

Scroll down to where it says **Object URL**

Then click on the Object URL or alternatively copy the Object URL and paste it into a new browser tab.

Your Object URL should look something like the following:

`https://s3.amazonaws.com/m3mes.com/index.html`

(although it will be different because your bucket name will be different)

Once you are at your URL you can test the lambda function by clicking the **Translate Vulcan** button

Section 4) If you bought a domain name we can hook up our domain name to the s3 bucket.

Click **Services** and click **Route 53**. Then click **Hosted Zones**.

Click your domain name and then click the **Create record** button (★leave the existing records alone)

Click **Simple Routing Policy** and click **Next**

Click **Define simple record**

Value/Route Traffic To: ✓ Alias to S3 website endpoint ←select Alias to S3 website

Choose Region: ✓ Select the Region that you created your S3 bucket in

Choose S3 bucket: ✓ Select the S3 bucket that you created

Record Type: A – Routes traffic to an ipv4 address and some AWS resources ← choose an A record (aka an Alias record)

Then click **Define simple record**

Then click **Create records**

You will now be able to visit your domain name, as it is now linked to your S3 bucket.

Then you can test your lambda function by clicking the **Translate Vulcan** button.

Mission complete!!! Below are important concepts and exam questions pertaining to Lambda and API Gateway.

Lambda Cheat Sheet

-**Lambdas** are serverless **functions**. You upload your code and it runs without you managing or provisioning any servers.

-Lambda is **serverless**. You don't need to worry about underlying architecture.

-Lambda is a good fit for short running tasks where you don't need to customize the OS environment. If you need long running tasks (>15 mins) and a Custom OS environment then consider using **Fargate**.

-There are **7 runtime language environments** officially supported by Lambda: **Ruby, Python, Java, NodeJS, C#, Powershell and Go**

-You pay per invocation (The **duration** and **the amount of memory used**) rounded up to the nearest 100 milliseconds and you pay based on amount of requests. First 1M requests per month are free

-You can adjust the duration timeout for up to **15 mins** and memory up to **3008 MB**

-You can trigger Lambdas from the SDK or multiple AWS services such as S3, API Gateway, DynamoDB

-Lambdas by default run in No VPC. To interact with some services you need to have your Lambda in the same VPC e.g. so in the case of RDS you would have to have your lambda in the same vpc as rds

-Lambda can scale to 1000 concurrent functions in seconds (1000 is the default, you can increase with AWS Service Limit increase)

-Lambdas have **Cold Starts**. If a function has not been recently executed there will be a delay.

API Gateway Cheat Sheet

-API Gateway is a solution for creating secure APIs in your cloud environment at any scale

-Create APIs that act as a front door for applications to access data, business logic, or functionality from back end services.

-API Gateway throttles api endpoints at **10,000** requests per second (can be increased via service request through AWS support)

-**Stages** allow you to have multiple published versions of your API e.g. prod, staging, QA

-Each Stage has an **invoke URL** which is the endpoint you use to interact with your API

-You can use a custom domain for your invoke URL

-You need to publish your API via Deploy API action. You choose which Stage you want publish your API.

-Resources are URLs eg /projects

-Resources can have child resources e.g. /projects/-id-/edit

-You defined multiples Methods on your Resources e.g. GET, POST, DELETE

-CORS issues are common with API Gateway, CORS can be enabled on all or individual endpoints

-Caching improves latency and reduces the amount of calls made to your endpoint

-Same Origin Policies help to prevent XSS attacks

-Same Origin Policies ignore tools like postman or curl

-CORS is always enforced by the client aka the browser

-You can require Authorization to your API via AWS Cognito or a custom Lambda

Practice Questions

The pharmaceutical company you work for has an aggressive schedule to bring a number of new products to market. You'd like to provide a library of metabolism assessment functions to application developers across the various teams so that each one doesn't have to write their own. Which AWS compute solution will be the most cost-effective to host this library?

A) Amazon EC2 Spot Instances

- B) AWS Lambda
 - C) Amazon Elastic Container Service
 - D) Amazon EC2 Reserved Instances
-

Explanation:

A microservices architecture provides the most cost-effective environment for a library of code, and the serverless nature of AWS Lambda allows you to do that with zero administration. You could host your microservices on Elastic Container Service, but the serverless nature of Lambda makes it more cost-effective. EC2 Spot and Reserved Instances provide lower cost options to EC2 On-Demand, but not lower than Lambda executions.

Resources

Answer: B) [AWS Lambda](#)

A startup clothing retailer has begun designing their online ordering application. The user interface will require presentation of four different screens to complete an order (product selection, shopping cart, payment, order confirmation). Orders can contain multiple line items. The application backend will need to scale seamlessly for seasonal spikes in demand. Which architecture will provide the most elastic and highest performing solution?

- A) Deploy EC2 instances, each with the capability to process all the steps of the ordering process. Submit interim order information to the backend after each ordering step and store it on EBS volumes for persistence. Implement Auto Scaling to handle the fluctuations in demand
 - B) Store all interim order information in browser cookies and submit it to a single EC2 instance at the end of the ordering process. Implement Auto Scaling to handle the fluctuations in demand
 - C) Deploy a different Lambda function to process each step of the ordering process. Submit interim order information to the backend after each ordering step and store it in Amazon DynamoDB. Index the data with a transaction ID cookie stored in the browser
 - D) Use a single Lambda function with the capability to process all the steps of the ordering process. Submit interim order information to the backend after each ordering step and store it in an Amazon RDS database. Keep track of order state in a database table
-

Explanation:

Deploying individual Lambda functions with different order processing capabilities provides scalability and performance at an atomic function level. A deployment package that includes all of the order processing logic will take longer to cold-start. DynamoDB will provide faster overall performance for interim order data than RDS. Storing all interim order data in browser cookies is not feasible, especially if the order contains multiple line items

Resources [Serverless Web Application](#)

Answer: C