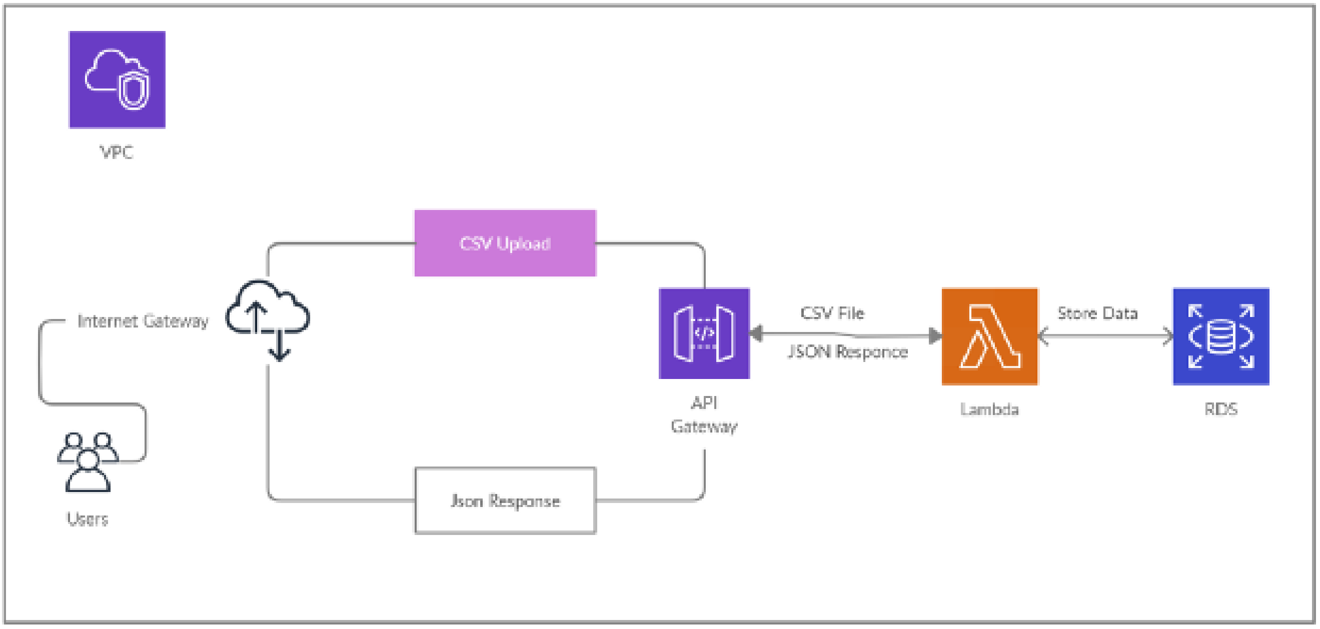
1) Building Serverless API

Upload CSV files to RDS using AWS API Gateway with AWS Lambda

Mohitsojitraa
Jul 22 · 9 min read

How to use the AWS API Gateway endpoint with Python AWS Lambda backend to allow uploads of CSV files to your cloud environment.



Architecture Diagram

Table of Contents

Introduction

When we needed to give our customers the ability to send binary files to our cloud application, I had to find a stable and secure way to upload the files, keeping in mind that I wanted all our internet-facing environment to be managed by AWS, and separated as much as possible from our internal cloud environment.

AWS API Gateway binary support makes it possible to send requests with any content type. In order to make that possible, the data is encoded in base64, so some manipulation is required before the data can be handled or stored for future use.

The ability to create an AWS Lambda function as an internal handler makes this manipulation and the integration of the requests with our cloud environment resources much easier.

Requirements

This guide only assumes that you have AWS as your cloud provider and that you have access to the API Gateway and Lambda management consoles.

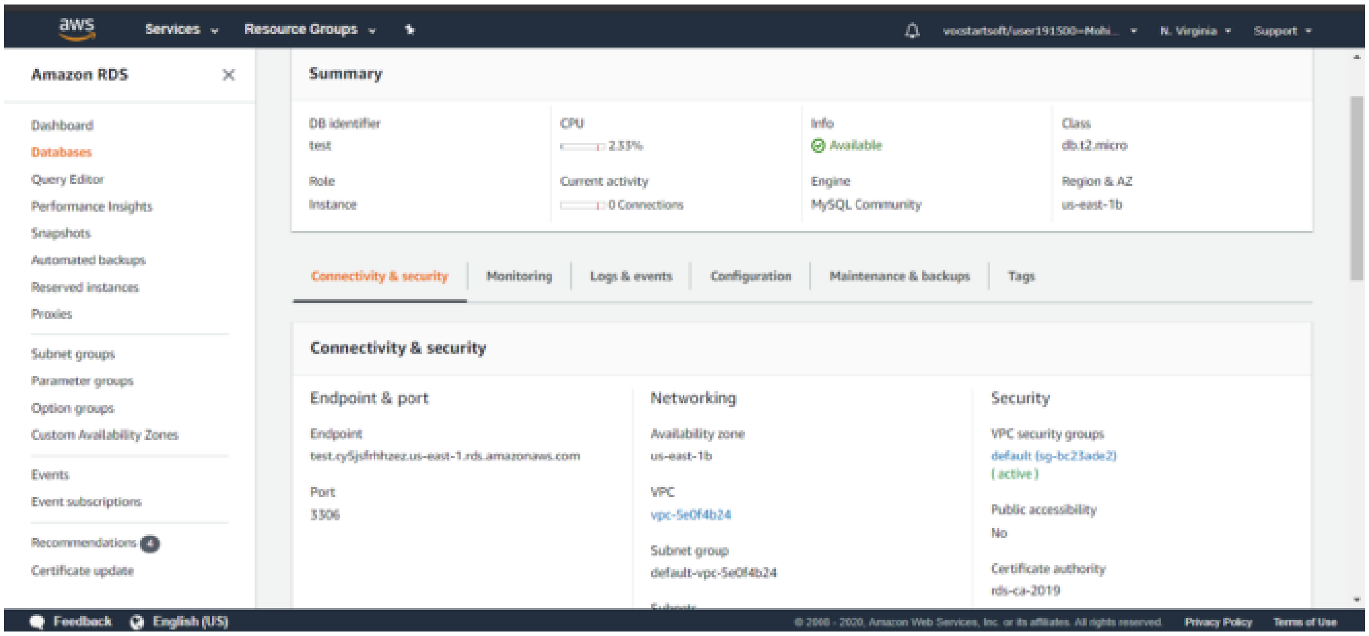
The solution is using an AWS Lambda function written in Python, but a similar solution is possible for other languages supported by AWS Lambda.

RDS and lambda function must be in the same VPC

Setting up the RDS Instances

Open the Services menu and select RDS.

Click on *Create Databases*, choose *Mysql Database*, select *MySQL,* and type a name. Click to finish. (Your database must not be publicly accessible)



Summary

DB identifier	CPU	Info	Class
test	2.33%	Available	db.t2.micro
Role	**Current activity**	**Engine**	**Region & AZ**
Instance	0 Connections	MySQL Community	us-east-1b

Connectivity & security

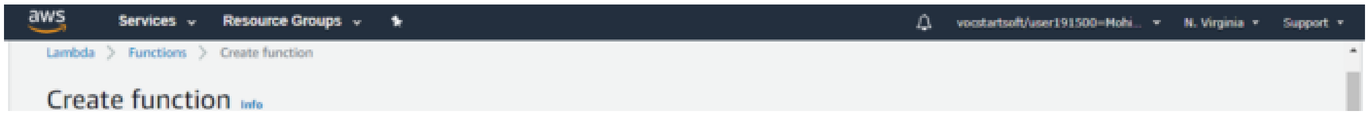
Endpoint & port	Networking	Security
Endpoint: test.cy5jsfrhhzez.us-east-1.rds.amazonaws.com	Availability zone: us-east-1b	VPC security groups: default (sg-bc23ade2) (active)
Port: 3306	VPC: vpc-5e0f4b24	Public accessibility: No
	Subnet group: default-vpc-5e0f4b24	Certificate authority: rds-ca-2019

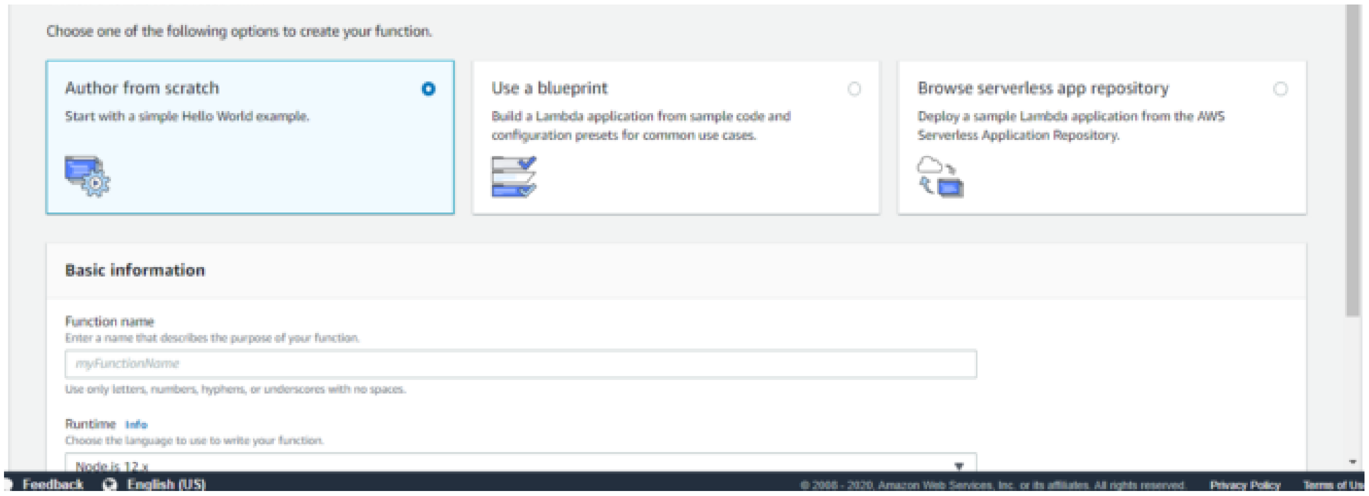
Writing the AWS Lambda function

We will start by creating the AWS Lambda function that handles our endpoint incoming requests.

Open the AWS Management Console, and from the Services menu select Lambda.

In the Lambda page click on *Create function*. Choose *Author from scratch*, type a name, and select *Python 3.6* or *Python 3.7* runtime. Expand the *Permissions* section, and choose to *create a new role with basic Lambda permissions & RDS Full access*. This will set up CloudWatch Logs for your Lambda function, so you'll be able to track executions. However, we also need RDS permissions to write our files. We will handle that later. Click on *Create function* to finish.





Choose one of the following options to create your function.

[✓ Author from scratch]
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Browse serverless app repository
Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name
Enter a name that describes the purpose of your function.
myFunctionName
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.
Node.js 12.x

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

Writing the function code

Once your function is created, you will see the function management screen, where you handle configuration, test the function, and of course write the code.

The Python Lambda code editor comes with a default *lambda_function.py* file with a *lambda_handler* method. We will use this default file and method for our example.

In the code editor, delete the content of the *lambda_function.py* file, and type the following code instead :

```
import json
import base64
import csv
import sys
import logging
import pymysql
from io import StringIO
rds_host  = "test.cy5jsfrhhzez.us-east-1.rds.amazonaws.com"
name = "admin"
password = "testdatabase"
db_name = "userdetails"

try:
    conn = pymysql.connect(rds_host, user=name, passwd=password,
db=db_name, connect_timeout=5)
except pymysql.MySQLError as e:
    logger.error("ERROR: Unexpected error: Could not connect to MySQL
instance.")
    logger.error(e)
    sys.exit()
    logger.info("SUCCESS: Connection to RDS MySQL instance
succeeded")
```

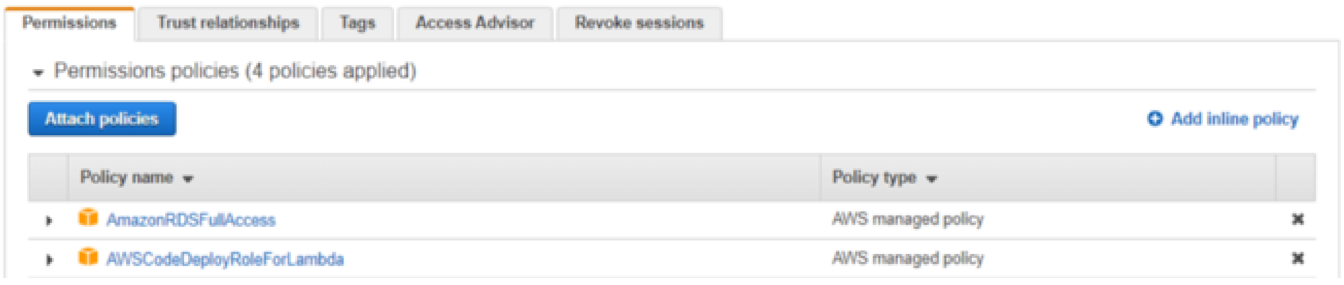
```
def lambda_handler(event, context):
    file_content = str(base64.b64decode(event['content']))
    #csv_data = csv.reader(file_content)
    file_path = 'mohit.csv'
    output = file_content.split('\\r')
    x = addData(output[4])

    return {
        'statusCode': 200,
        'body': {
            'Record Added': x
        }
    }

def addData(output):
    i = 0
    #return output
    data = output.split('\\n')
    with conn.cursor() as cur:
        for row in data[2:len(data)-1]:
            i = i + 1
            data = row.split(',')
            #return data[1]
            query = 'insert into userdetail (name, email,contact)
values("'+data[0]+'","'+data[1]+'","'+data[2]+'")'
            cur.execute(query)
        conn.commit()
    return i
            # if i== (len(data)-2):
            #     return len(output)-1
```

Granting RDS access to the function

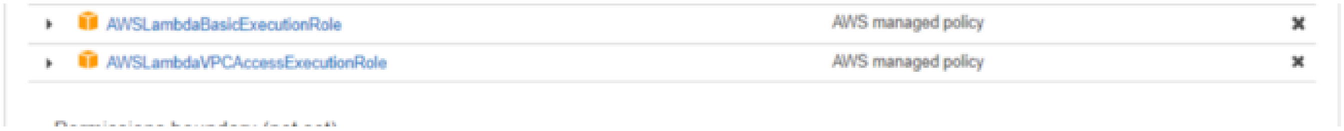
For our Lambda function to work, we need to grant it access to RDS.

In the function management page, go to the *Execution role* section, and click on *View the … role on the IAM console*.

In the role IAM page click on *Attach policies*. Select the *AmazonRDSFullAccess* policy and click on *Attach policy*.



Policy name	Policy type
AmazonRDSFullAccess	AWS managed policy
AWSCodeDeployRoleForLambda	AWS managed policy



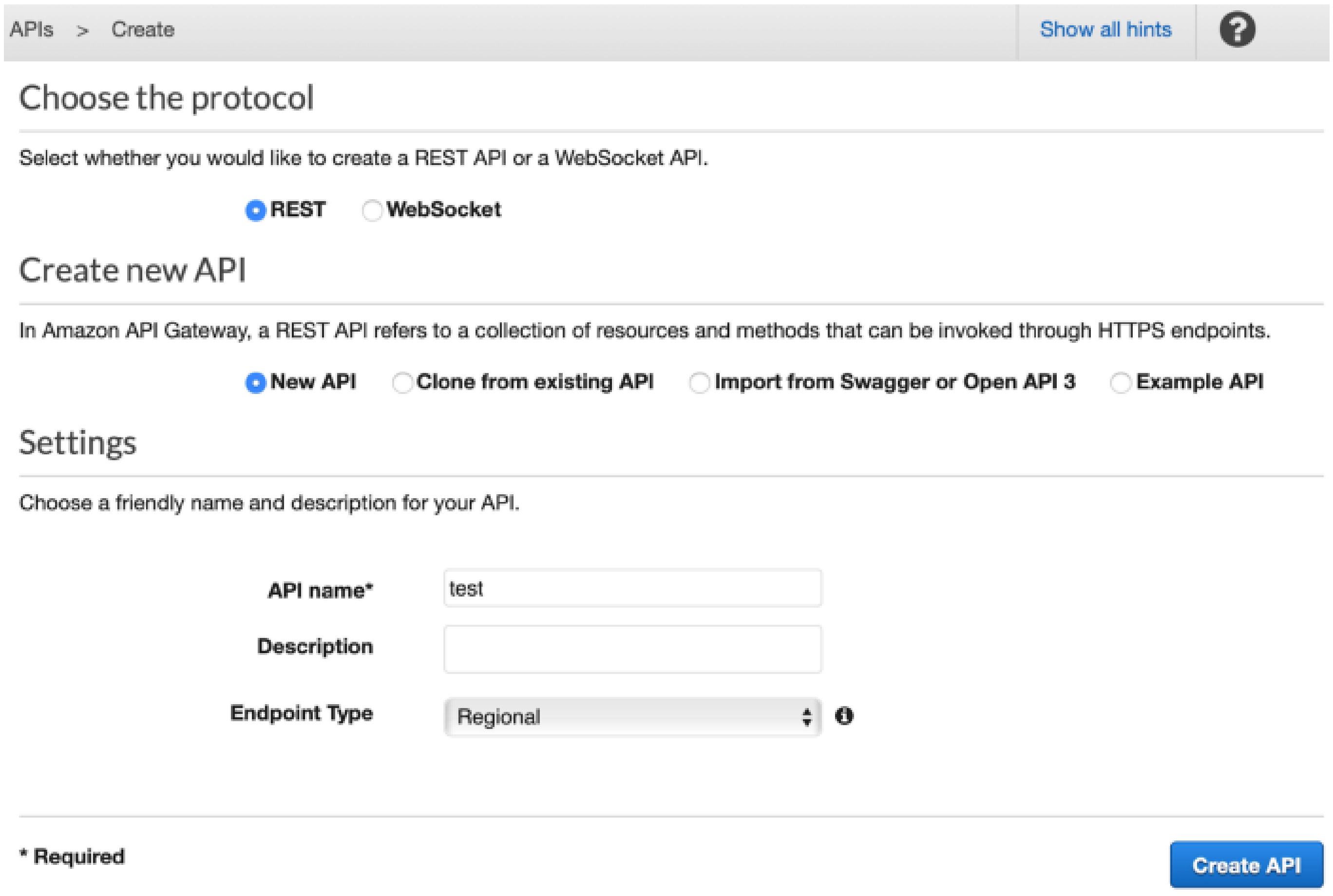
AWSLambdaBasicExecutionRole	AWS managed policy
AWSLambdaVPCAccessExecutionRole	AWS managed policy

Setting up the API Gateway

Creating the API Gateway endpoint

Open the Services menu and select API Gateway.

Click on *Create API*, choose *REST*, select *New API* and type a name. Click on *Create API* to finish.



Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

[✓ REST] WebSocket

Create new API

In Amazon API Gateway, a REST 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 or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

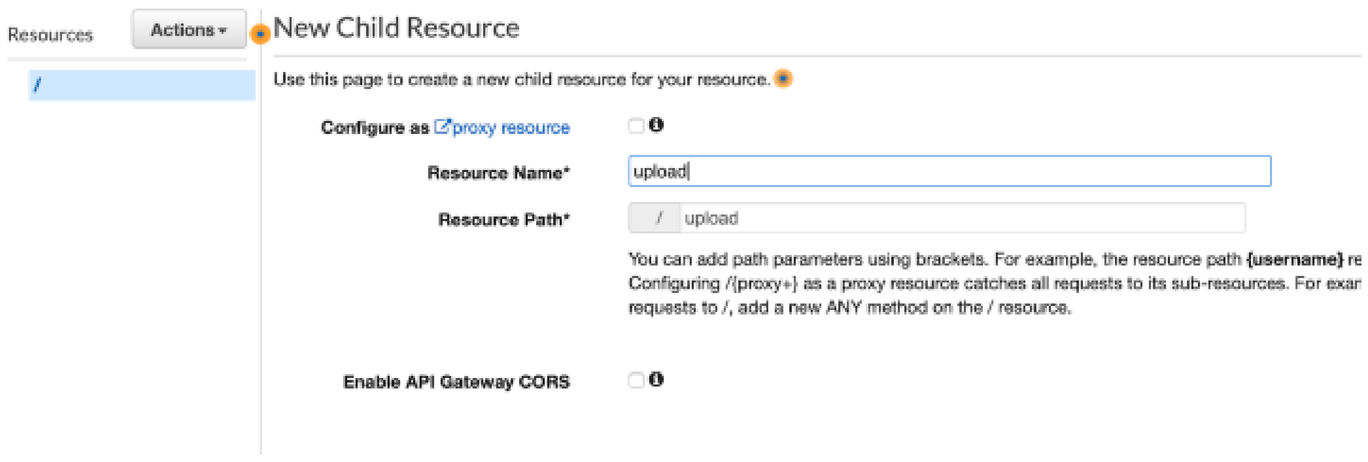
API name*	test
Description	
Endpoint Type	Regional

* Required

You will be redirected to the resources page. Here you can define your endpoint paths (defined here as resources), and the behavior of the method for each path.

Setting up a POST method endpoint

In the *Resources* page, open the *Actions* menu and choose *Create Resource*. Type a resource name and path (for example 'upload'), and click on *Create Resource*.



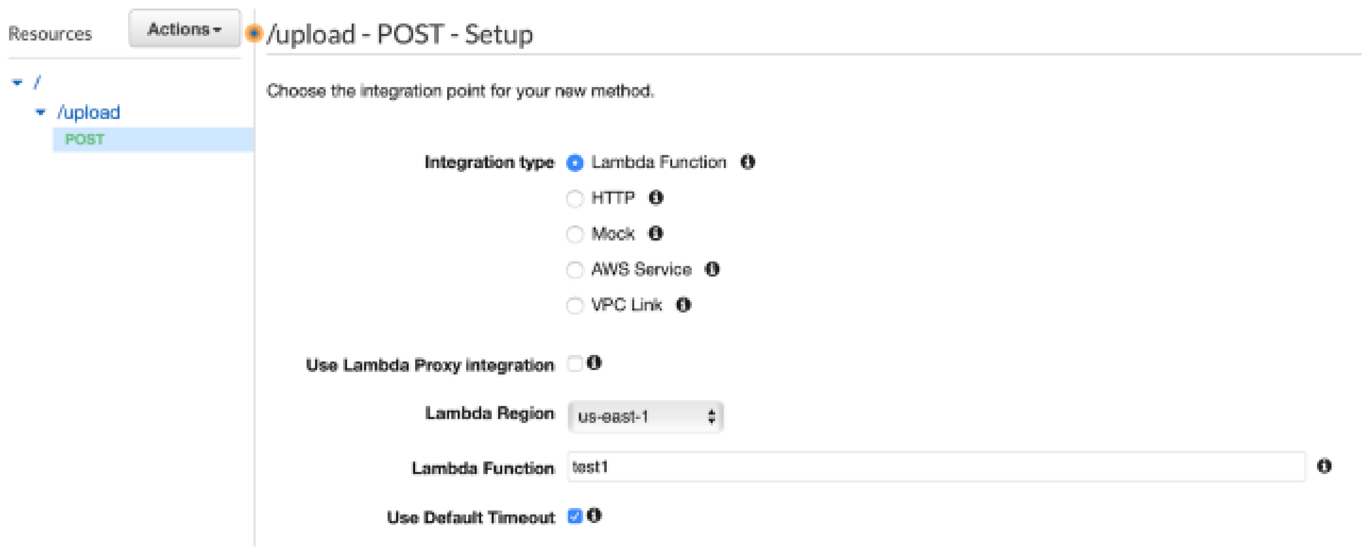
New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource	☐
Resource Name*	upload
Resource Path*	/ upload
	You can add path parameters using brackets. For example, the resource path {username} re Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For exar requests to /, add a new ANY method on the / resource.
Enable API Gateway CORS	☐

Then, open the *Actions* menu again, and this time choose *Create Method*. In the opened dropdown select *POST*, and click on the small *V* icon to confirm.

In the opened form, choose *Lambda Function* as the integration type, and select your newly created Lambda function by typing its name or ARN identifier. Click on *Save*, and confirm that you allow API Gateway to invoke your Lambda function.



/upload - POST - Setup

Choose the integration point for your new method.

Integration type	[✓ Lambda Function] HTTP Mock AWS Service VPC Link
Use Lambda Proxy integration	☐
Lambda Region	us-east-1
Lambda Function	test1
Use Default Timeout	☑

You will then be redirected to the *Method Execution* configuration page of your newly created method, where you can handle the flow of the request to the gateway and from the gateway to your environment (in this case, Lambda function), and the response from your environment to the gateway and back to the requester.

You can also test your method execution here, but since we handle binary content, this is not really feasible

Accepting binary data of certain types

We want to configure our endpoint to accept the binary content of certain types through our newly created method. We will take as an example PDF files, but the following guide applies for any type of content, and for multiple types of content as well.

Open your method *Integration Request*, and expand the *Mapping Templates* section.

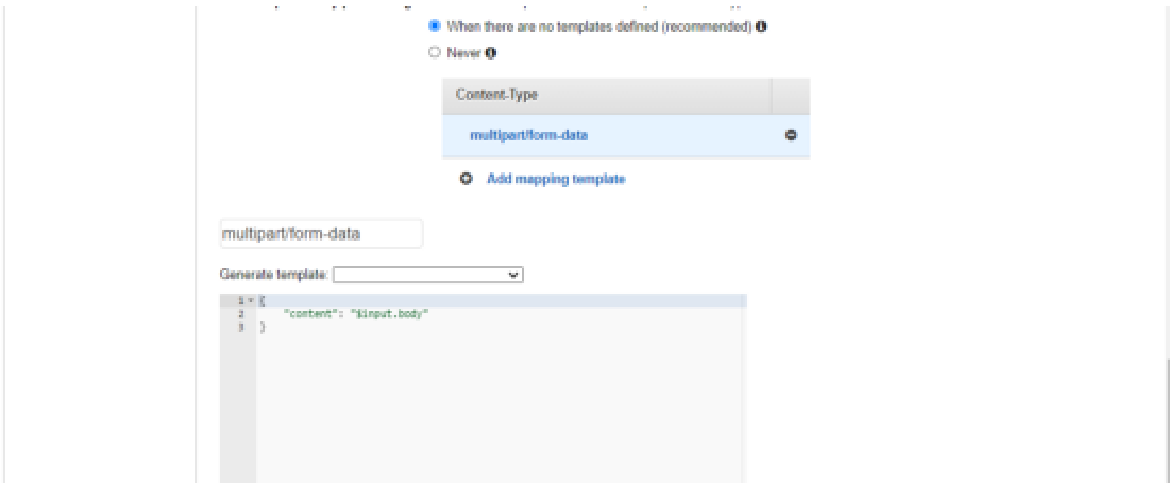
In *Request body passthrough* choose *When there are no templates defined*.

Then, for each of your media types click on *Add mapping template*. In the *Content-Type* box type your media type identifier (for example, for PDF type *application/pdf*), and click on the small *V* icon.

In the template box type the following, then click *Save*:

```
{
    "content": "$input.body"
}
```

This will tell your endpoint to pass the body of your request (the binary content) to the Lambda function inside the *content* property. The content will be passed encoded as base64. Recall that in our Lambda function we got the file content by accessing the event *content* property and decoding it as base64.



[✓ When there are no templates defined (recommended)] Never

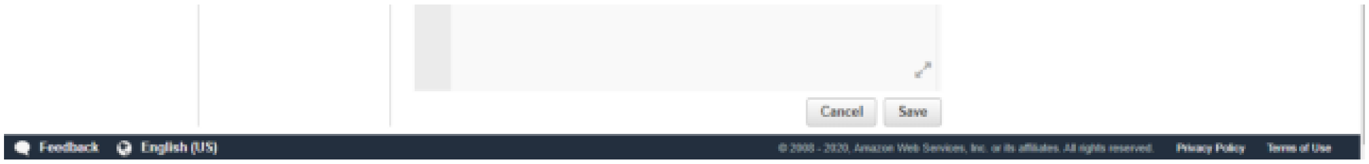
Content-Type
multipart/form-data

Add mapping template

multipart/form-data

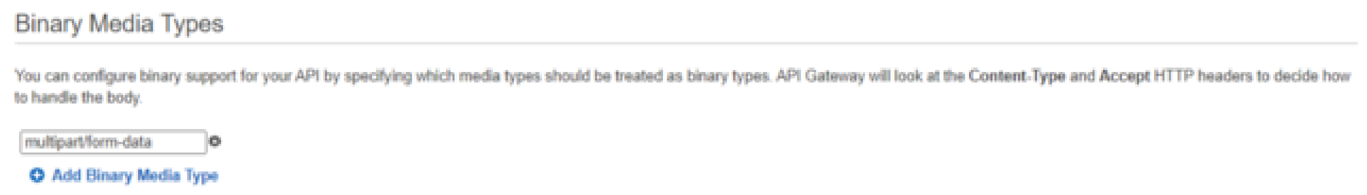
Generate template:

```
{
    "content": "$input.body"
}
```


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

Next we need to add binary media types support to our endpoint. This will determine which media types will be treated as binary (and encoded as base64 to be processed in our Lambda function).

In the endpoint left menu click on *Settings*, then in the *Binary Media Types* section, for each of your content types click on *Add Binary Media Type* and type your content identifier. Click on *Save Changes* to finish.



Binary Media Types

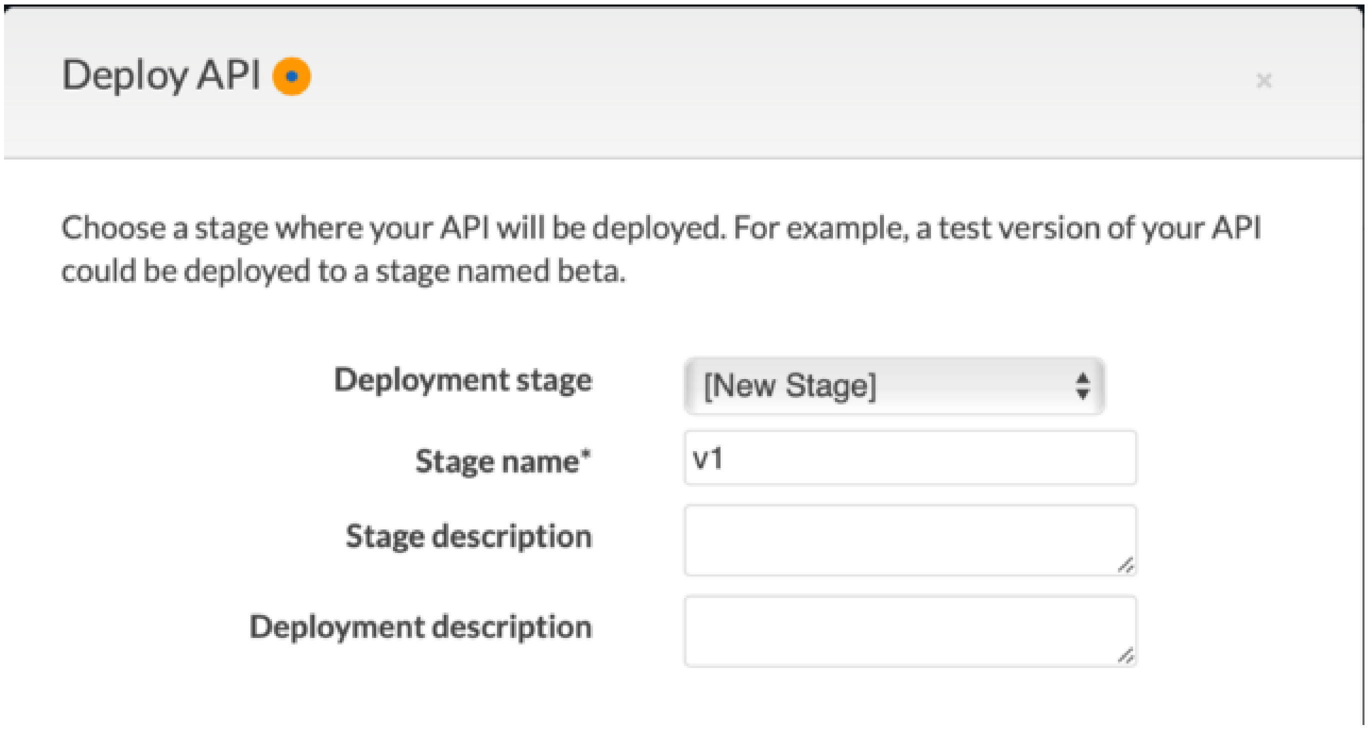
You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the **Content-Type** and **Accept** HTTP headers to decide how to handle the body.

multipart/form-data

Deploying the endpoint

From the *Actions* menu choose *Deploy API*.

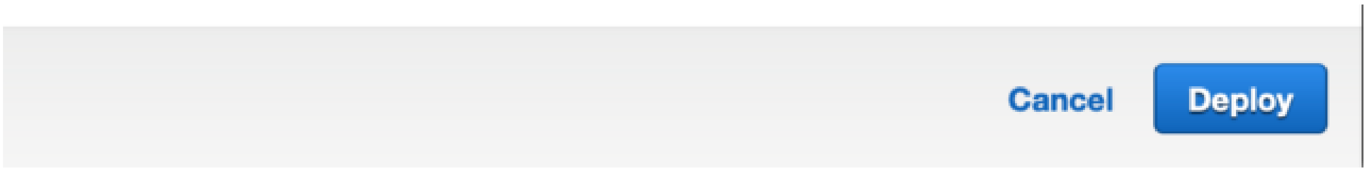
Select *[New Stage]* as the deployment stage, type a stage name (e.g. v1), and click *Deploy*.



Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	v1
Stage description	
Deployment description	



You will be redirected to the *Stages* page, and at the top, you will see your endpoint *Invoke URL*.

It's important to note that while your endpoint is managed by AWS, it is not secured out-of-the-box. Once you have deployed your endpoint it becomes publicly accessible, and any request to the invocation URL with the correct method and data will invoke your Lambda function. I am planning to cover endpoint security in my next post.

2) Host Static Serverless Website using S3

> *http://www.mohitsojitra.ml*

Most websites are becoming static websites which means they run zero server-side code and consist of only HTML, CSS, and JavaScript. With no server-side code to run, there is no reason to host them on a traditional server.

By using the static website hosting feature on an S3 bucket, we host static websites for one to two dollars a month and scale to handle millions of users.

With the background out of the way, are you hungry to learn more about AWS by completing a practical example? Then let's jump in.

1. Naming Your S3 Bucket

Before you begin hosting your awesome static website out of S3, you need a bucket first. For this blog post, it is **critical** that your bucket has the same name as your domain name.

If your website domain is www.my-awesome-site.com, then your bucket name must be www.my-awesome-site.com.

The reasoning for this has to do with how requests are routed to S3. The request comes into the bucket, and then S3 uses the Host header in the request to route to the appropriate bucket.

2. Configuring Your S3 Bucket for Static Website Hosting

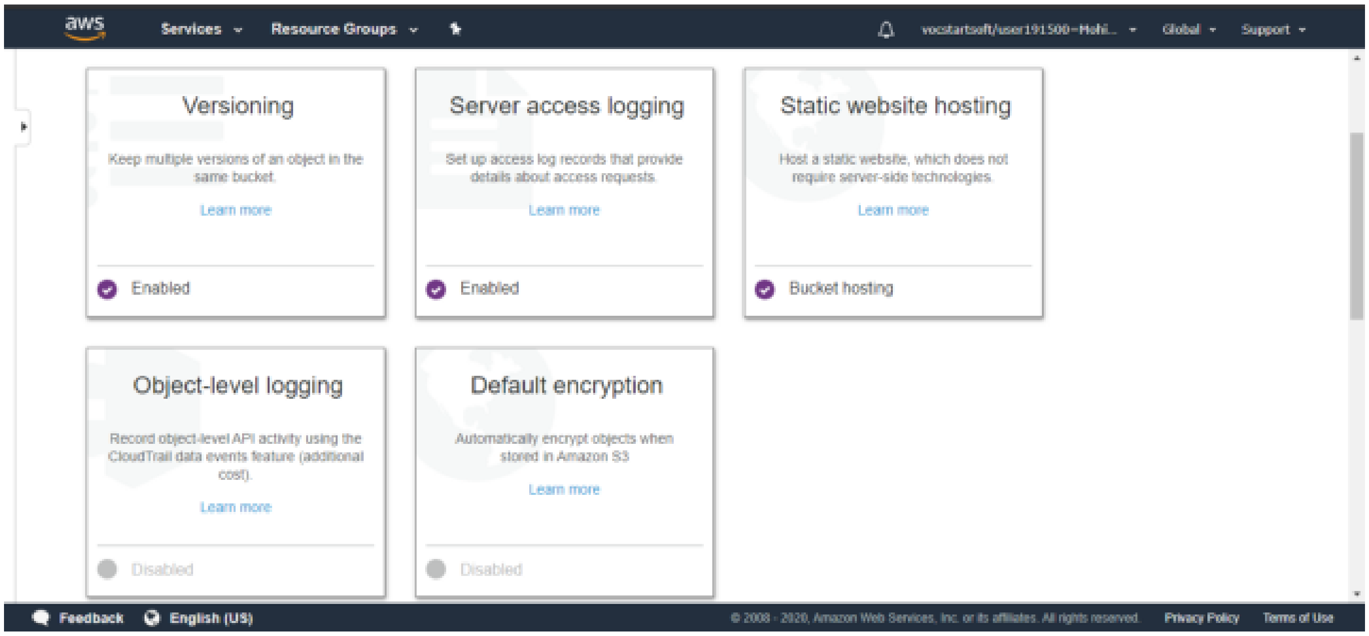
(Bucket must be public)

Make Sure that S3 bucket is public and it has only read-only publicly accessible

Alright, you have your bucket. It has the same name as your domain name, yes? Time to configure the bucket for static website hosting.

Guess what? Turning on static website hosting for your bucket is as simple as a few clicks in the AWS Console.

- Navigate to S3 in the AWS Console.
- Click into your bucket.
- Click the "Properties" section.
- Click the "Static website hosting" option.
- Select "Use this bucket to host a website".
- Enter "index.html" as the index document.


aws
Services
Resource Groups
vocstartsoft/user191500=Mohi...
Global
Support
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Versioning
Keep multiple versions of an object in the same bucket.
Learn more
Enabled

Server access logging
Set up access log records that provide details about access requests.
Learn more
Enabled

Static website hosting
Host a static website, which does not require server-side technologies.
Learn more
Bucket hosting

Object-level logging
Record object-level API activity using the CloudTrail data events feature (additional cost).
Learn more
Disabled

Default encryption
Automatically encrypt objects when stored in Amazon S3
Learn more
Disabled

Benefits

In five simple and easy steps, you have learned how to host your static website out of AWS S3. Not to mention you scored some benefits from moving your static website to S3.

Low cost — Hosting a website in S3 does not incur extra charges. You are paying standard S3 prices on getting requests and Data Transfer out of the bucket when a user visits your site.

- GET Requests cost $0.004 per 10,000 requests
- Data Transfer Out cost $0.090 per GB (up to 10 TB / month)

A Cost breakdown example: Let's say that www.my-awesome-site.com loads 20 resources. The total size of those resources per visit is 1MB. The average total monthly visits is 20,000. Then we estimate the total cost of S3 on a monthly basis at around **$1.96** per month.

Not long ago, you paid $10/month, so $2 is worth it.

Maintenance — Your static website now resides in S3. There is no longer any server-side code to maintain and no web servers to configure and keep up to date.

Scale — S3 is high availability and durable service that AWS maintains. If your website goes from 10 users a day to 10 million, S3 scales your website automatically.

Security — There is no server running that you maintain. Thus you avoid making configuration errors that make you vulnerable to attacks. You are still responsible for the security of your bucket**. Remember your website bucket is public!**

3) Building web application using Python(Flask) and Dynamo DB on AWS