How to Upload a Binary File to Amazon S3 Bucket via API Gateway

Parth Trambadiya Follow

Aug 5 · 8 min read

Photo by Tetsuya Tomomatsu on Unsplash

I have seen many people are facing an issue in uploading a binary file from their REST API to the Amazon S3 Bucket. In every web and mobile application, it's common and good to provide users with the feature of upload binary files such as PDFs, Images, Videos, GIFs, etc to the S3 Bucket of our application.

Normally, all server-based applications or server-side technologies are providing this kind of facility, but usually, binary files are quite larger than text-based files, so the upload process of binary files may take up network I/O and CPU time of your server. And in these cases, you need to keep track of the state transfer of the upload process — whether it failed, succeeded or needs to be re-uploaded.

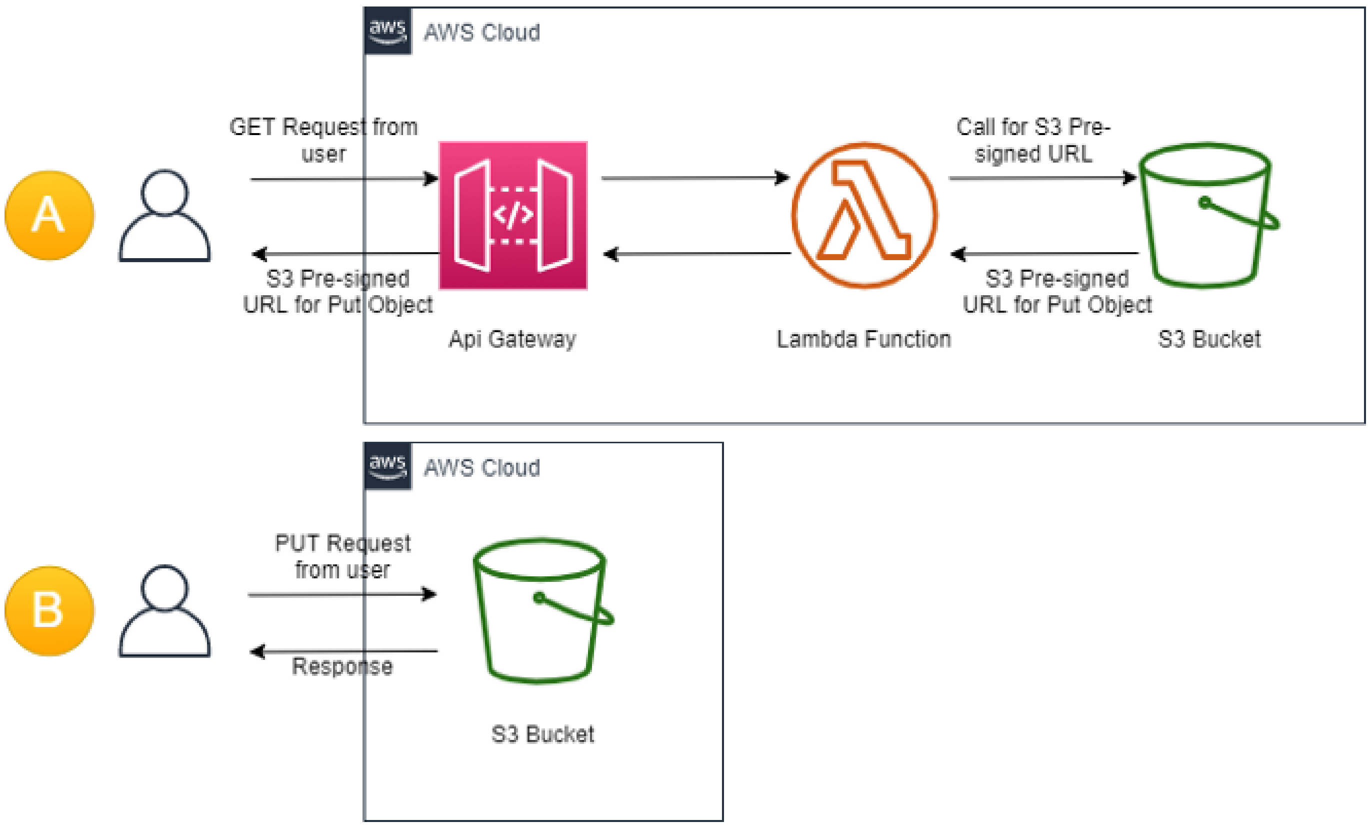
In the end, the result of these types of processes may take your application down or result in a late response in case of high traffic on your applications. So finally, your customers might be unhappy.

So you might have the question — is it possible to avoid proxy of those requests through your application server or is there any serverless solution for this?

Yes, there is one serverless solution: you can upload your binary file directly to the Amazon S3 Bucket. In this blog, I am going to show how you can upload a binary file into your S3 Bucket.

Architecture

To do this, there is a simple one architecture and I have divided that architecture in two small part,



First part of architecture A in the API Gateway will receive GET request from the user, GET method of API Gateway is integrated with lambda function, so API Gateway will pass that GET Request to Lambda function and Lambda function will call the S3 Bucket using method of AWS SDK(i.e. In JavaScript SDK(NodeJS), using getSignedUrl() or getSignedUrlPromise()). and lambda will send back response with S3 Pre-SignedUrl to API Gateway and API Gateway to User.

Second part of architecture B, once application got Pre-signedUrl then application can perform PUT request on that URL to upload binary file before expiry of pre-signedurl.

I will show implementation of this concept using two methods

1. Using AWS Serverless Application Modal(SAM) template
2. From AWS Console

Using AWS SAM Template

First, install AWS SAM CLI on your machine, you can find installation steps from official documentation of AWS SAM,

Installing the AWS SAM CLI

AWS SAM provides you with a command line tool, the AWS SAM CLI, that makes it easy for you to create and manage...

docs.aws.amazon.com

Git Repository: https://github.com/ParthTrambadiya/binary-file-upload-s3

Clone AWS SAM Template from my GitHub Repository, and perform below commands.

```
cd binary-file-upload-s3
```

To build SAM Template

```
sam build
```

To deploy cloudformation stack on AWS

```
sam deploy --guided
```

if you have set multiple profile of AWS IAM users and you want to deploy in one specific profile's AWS Account:

```
sam deploy --guided --profile <profile-name>
```

> ***Note:*** *In Lambda function of this stack(you will find lambda function code under* **functions** *folder of git repo,* **upload.js***), must change* **ContentType** *according your application. For reference see this:*

Common MIME types - HTTP | MDN

Here is a list of MIME types, associated by type of documents, ordered by their common extensions. Two primary MIME...

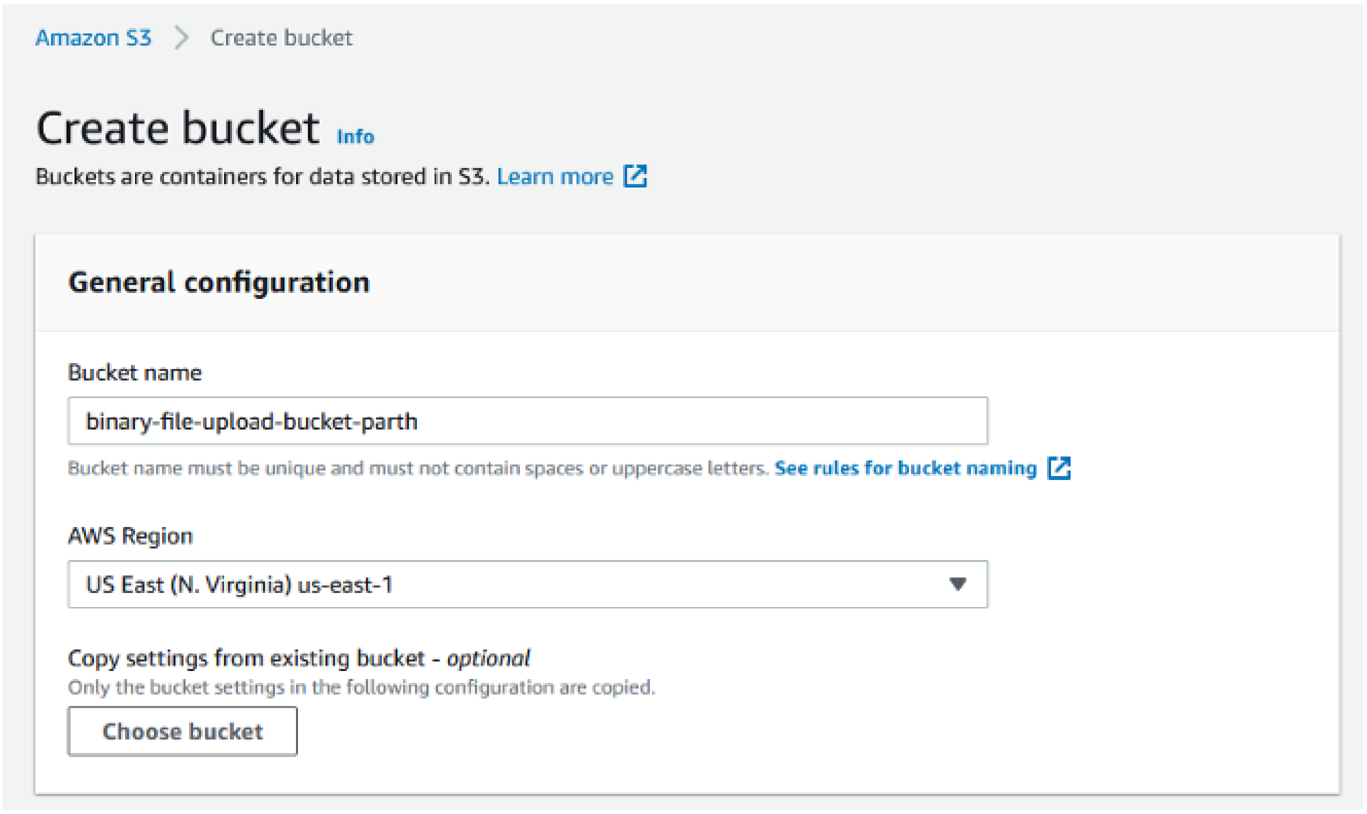
developer.mozilla.org

. . .

If you want don't to see implementation using AWS Console then you can directly jump on ***Test*** part of this article.

From AWS Console

First, login or signup into your AWS Console, and go to S3.



Amazon S3 > Create bucket

Create bucket Info

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name

binary-file-upload-bucket-parth

Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming

AWS Region

US East (N. Virginia) us-east-1

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Create a new S3 bucket by clicking on ***Create bucket***, your bucket must be unique globally, bucket names must be between 3 and 63 characters long, bucket names can consist only of lowercase letters, numbers, dots (.), and hyphens (-), bucket names must begin and end with a letter or number, bucket names must not be formatted as an IP address (for example, 192.168.5.4). For more information about Bucket Naming Rule visit below link:

Bucket naming rules

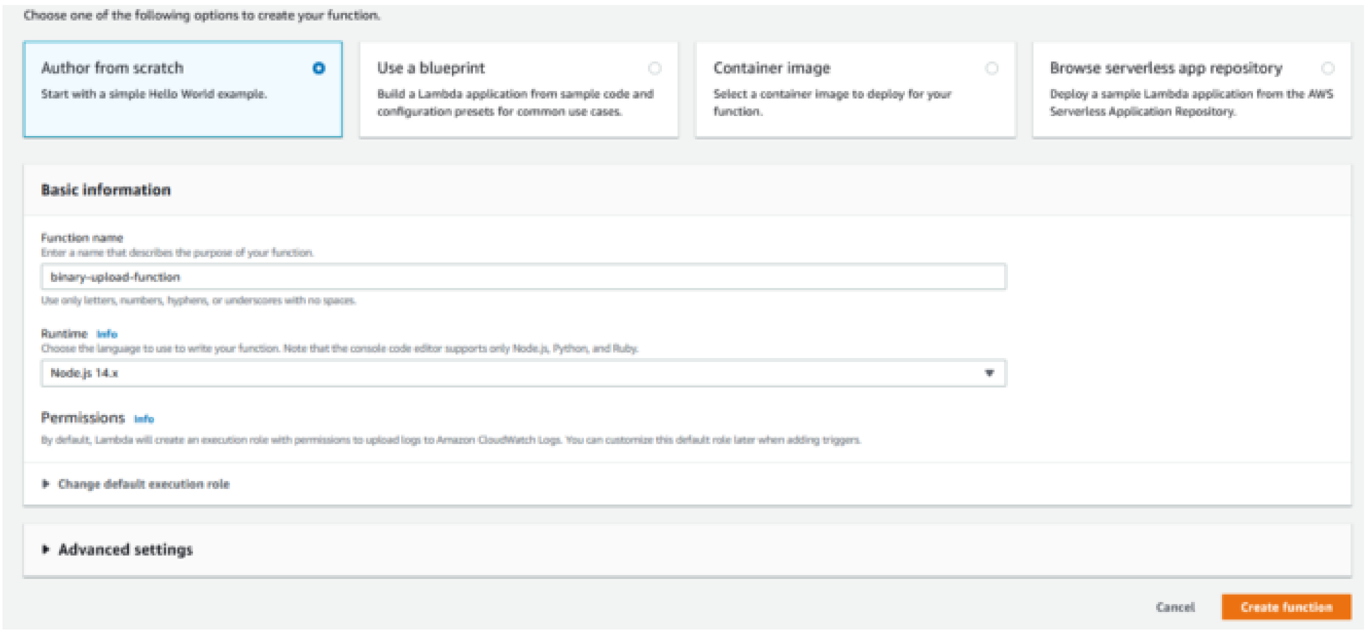
The following rules apply for naming buckets in Amazon S3: Bucket names must be between 3 and 63 characters long...

docs.aws.amazon.com

Choose bucket region according your project, and leave all other options as it is or default just for demo purpose or you can change according your requirements, and now click on ***Create bucket*** button.

After creation of Amazon S3 Bucket go to AWS Lambda Function.

Create a new lambda function from Orange button "Create Function" at top-right corner, in creation of lambda function choose ***Author From Scratch***.



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.

Container image
Select a container image to deploy for your function.

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.
binary-upload-function
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 14.x

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▸ Change default execution role

▸ **Advanced settings**

In basic information, write ***Function name***, according your requirements, ***Runtime***, choose latest version of NodeJS(i.e. Node.Js 14.x). Now leave all other options default and click on ***Create Function***.

Once you have created function, copy and paste below code into newly created lambda function.

```
'use strict'

const AWS = require('aws-sdk')
const s3 = new AWS.S3({
    signatureVersion: 'v4'
})
```

```javascript
})

const responseSucceeded = {
    statusCode: 200,
    headers: {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Credentials': true
    },
    body: JSON.stringify({})
};

module.exports.handler = function(event, context, callback) {
    console.log('Event ', event);

    const bucketName = event.queryStringParameters.bucketName;
    const fileName = event.queryStringParameters.fileName;
    const expiryTime = event.queryStringParameters.expiryTime;
    const contentType = event.queryStringParameters.contentType;

    console.log('File Name: ', fileName)
    console.log('Expiry Time: ', expiryTime)

    const s3Params = {
        Bucket: bucketName,
        Key: fileName,
        Expires: Number(expiryTime),
        ContentType: contentType
    }

    console.log('Params: ', s3Params)
    const uploadURL = s3.getSignedUrl('putObject', s3Params)

    console.log(uploadURL)
    responseSucceeded.body = JSON.stringify(uploadURL)
    callback(null, responseSucceeded);
}
```

binary-file-upload.js hosted with ♡ by GitHub view raw

You have created S3 bucket, so go and get name of bucket which you created in above step and write or paste at ***Bucket*** option in s3Params.

> ***Note:*** *In Lambda function, must change* **ContentType** *according your application. For reference see this:*

Common MIME types - HTTP | MDN

Here is a list of MIME types, associated by type of documents, ordered by their common extensions. Two primary MIME...

developer.mozilla.org

developer.mozilla.org

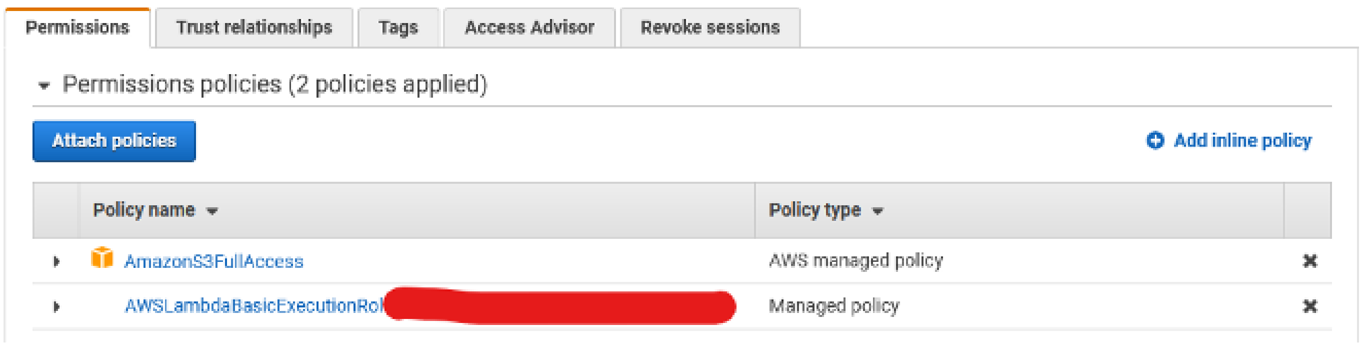
At the our s3Pramas will look like below, in our case.

```
const s3Params = {

Bucket: binary-file-upload-bucket-parth,

Key,

Expires: 300,

ContentType: 'video/mp4',

}
```

We have configured our lambda function, but we need to give permission to lambda to access our s3 bucket.

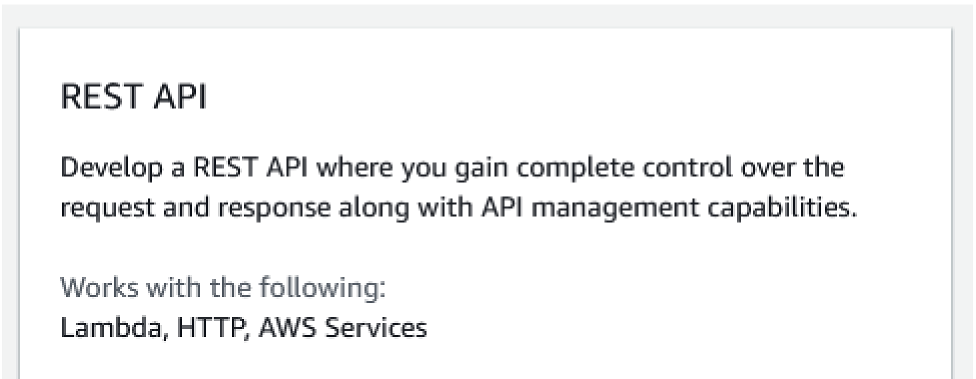
Now go to Configuration tab of lambda function and click on permission tab at left side bar, and in **Execution Role** section click on role name, so you will redirect to IAM Console.

On IAM Console click on attach policies under permission tab, and search AWS Managed Policy `AmazonS3FullAccess` and attach this policy to that IAM Role.



Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
AWSLambdaBasicExecutionRol[redacted]	Managed policy

Now go to API Gateway,

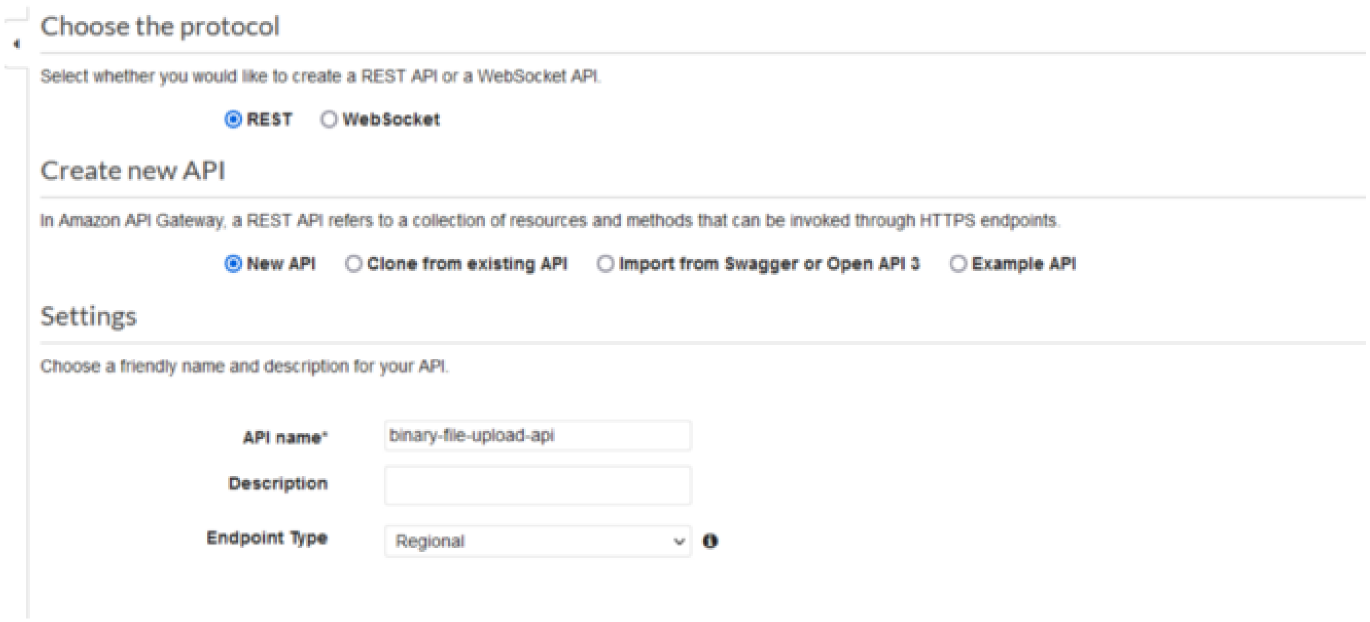


REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

Create REST API by clicking on ***Build*** button of REST API card.


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.

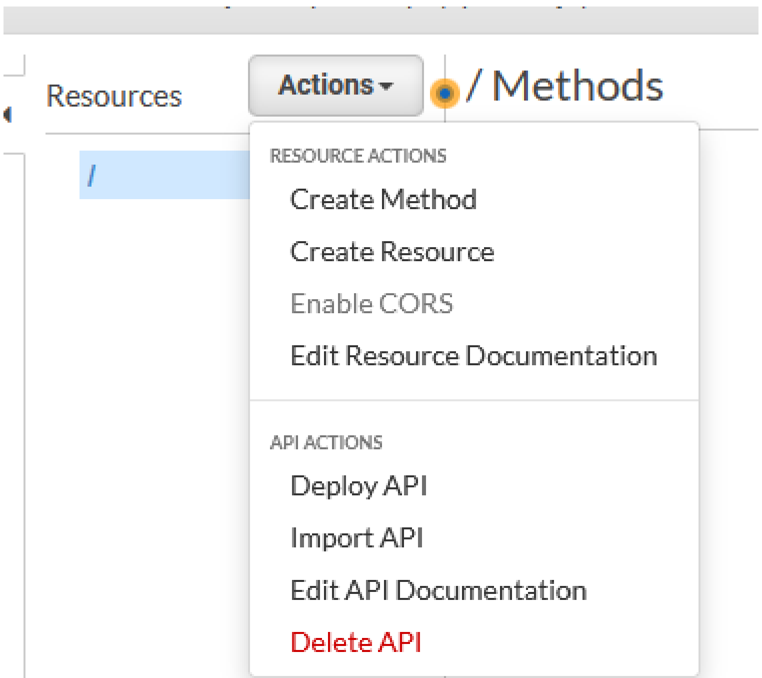
API name*	binary-file-upload-api
Description	
Endpoint Type	Regional

In Protocol choose REST, we are creating new API so choose new API , and in setting write API name according to your project and Endpoint Type Choose ***Regional***, you can also choose Edge-Optimized of you want. For more information about Endpoint visit below link:

Choose an endpoint type to set up for an API Gateway API

An API endpoint type refers to the hostname of the API. The API endpoint type can be edge-optimized, regional, or...

docs.aws.amazon.com

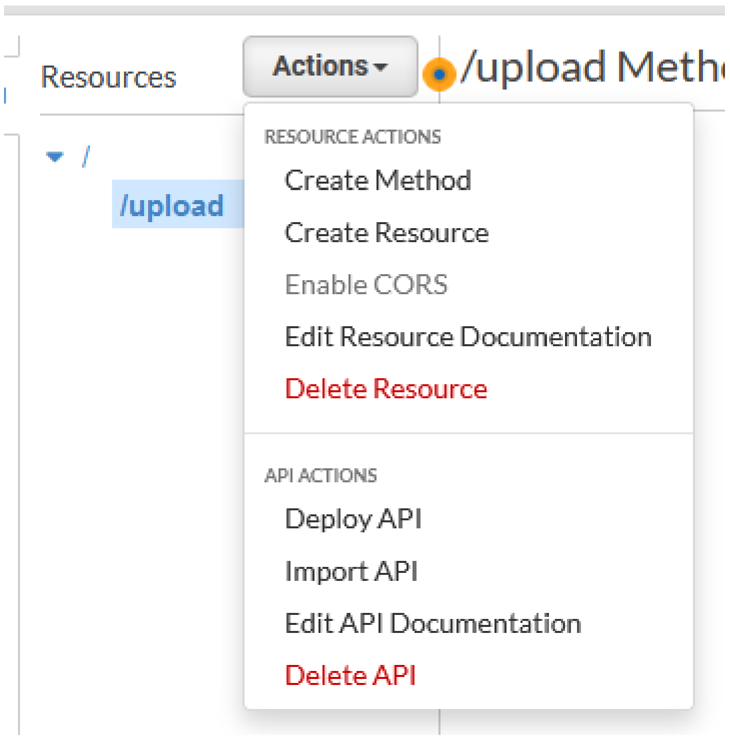


In resources, create other sub resource with ***upload*** name from action button and click on create resource.

Resource Name: upload

Resource Path: upload

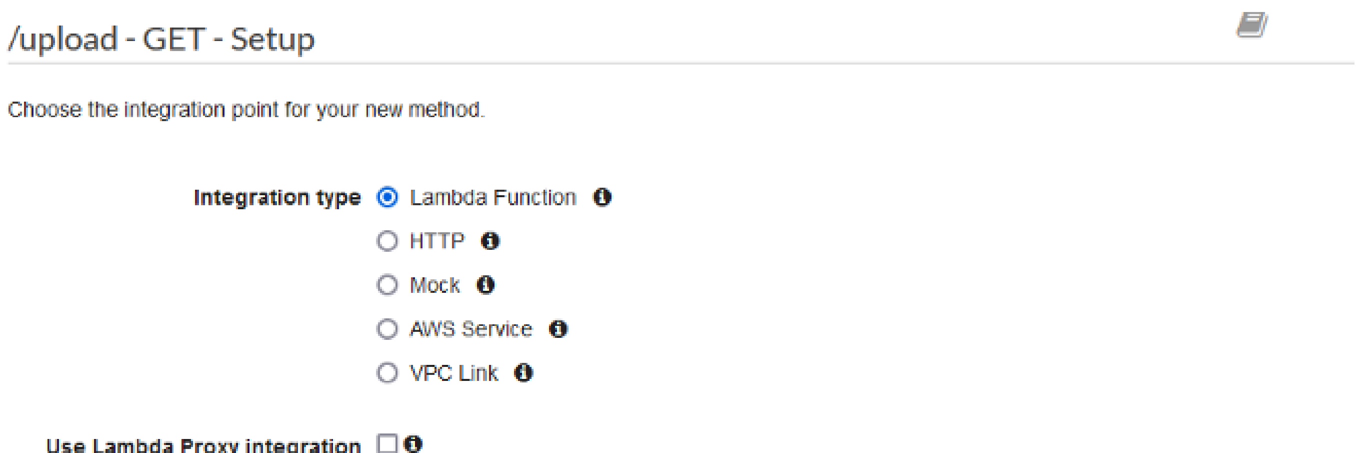
Click on create resource.



Click on ***/upload*** resource and then click on action button and create method.

Choose ***GET*** method and click on button with right sign.

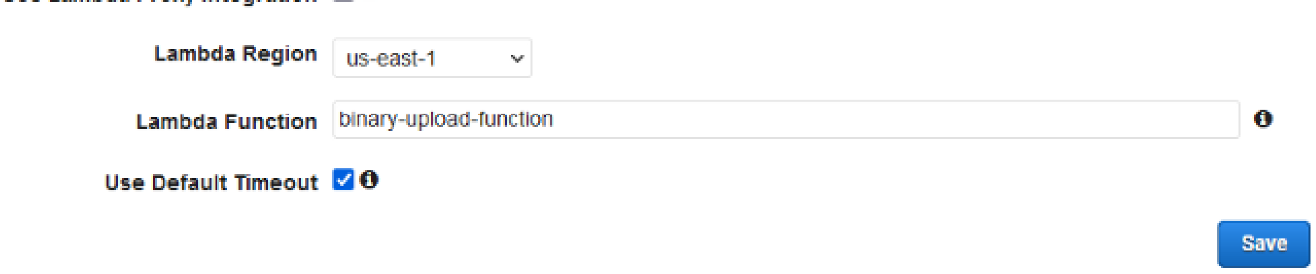
Once you have crated **GET** method then go to inside that method. you will see this kind of screen.


/upload - GET - Setup

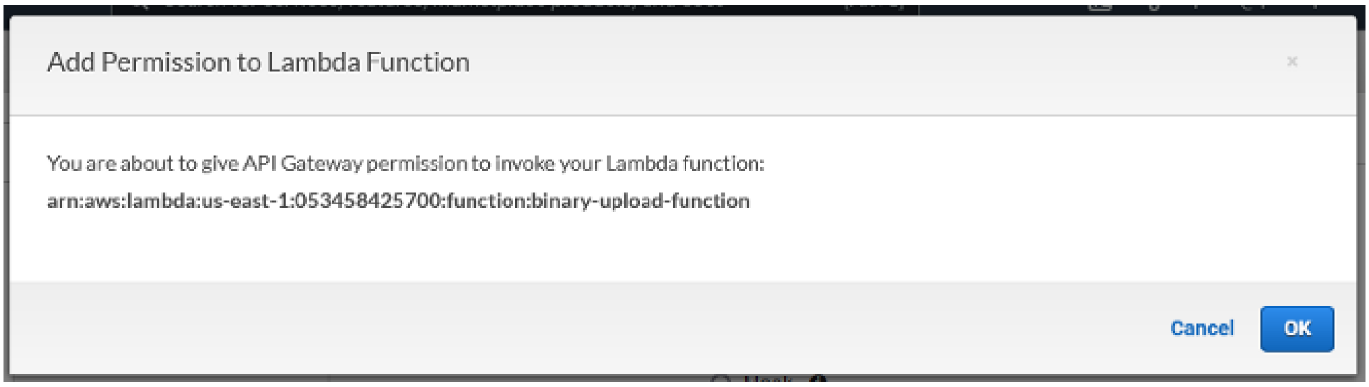
Choose the integration point for your new method.

Integration type [✓ Lambda Function] HTTP Mock AWS Service VPC Link

Use Lambda Proxy integration ☐


Lambda Region: us-east-1
Lambda Function: binary-upload-function
Use Default Timeout: ✓

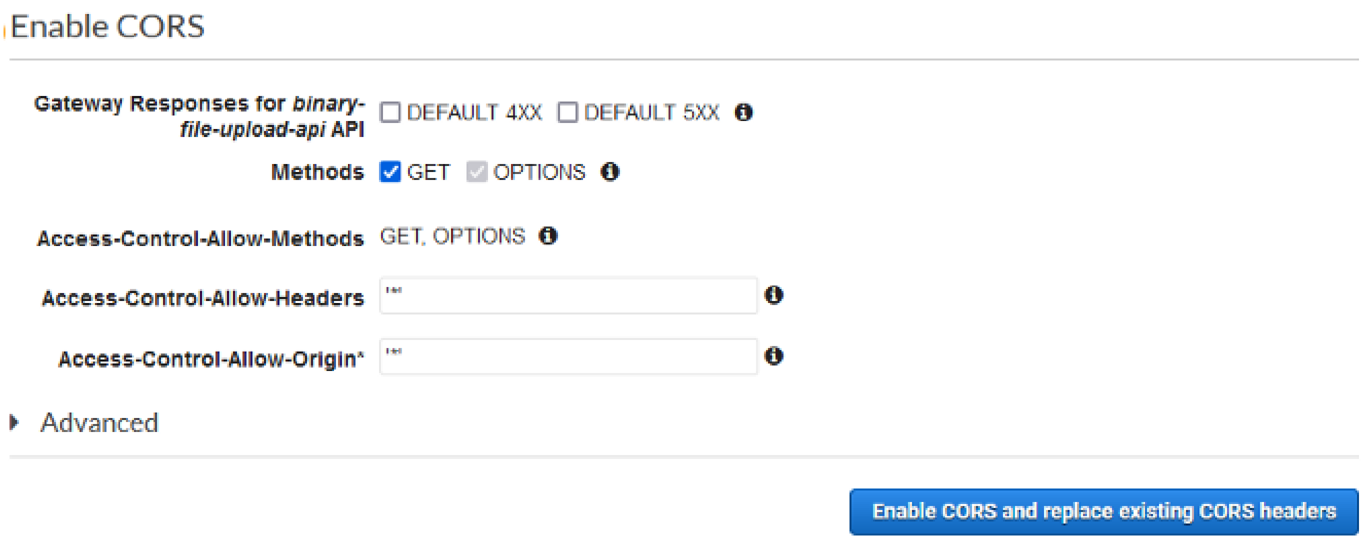
In ***Integration Type***, choose Lambda function, in ***Lambda Region*** choose AWS region where your lambda function is available, and in ***Lambda Function*** type lambda function name(which we created in above step) and select from menu. Now click on Save.


Add Permission to Lambda Function

You are about to give API Gateway permission to invoke your Lambda function:
arn:aws:lambda:us-east-1:053458425700:function:binary-upload-function

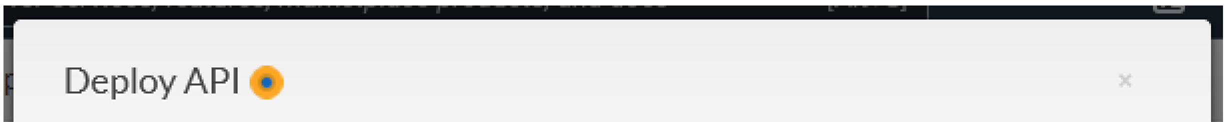
It will ask for above permission, so click on Ok.

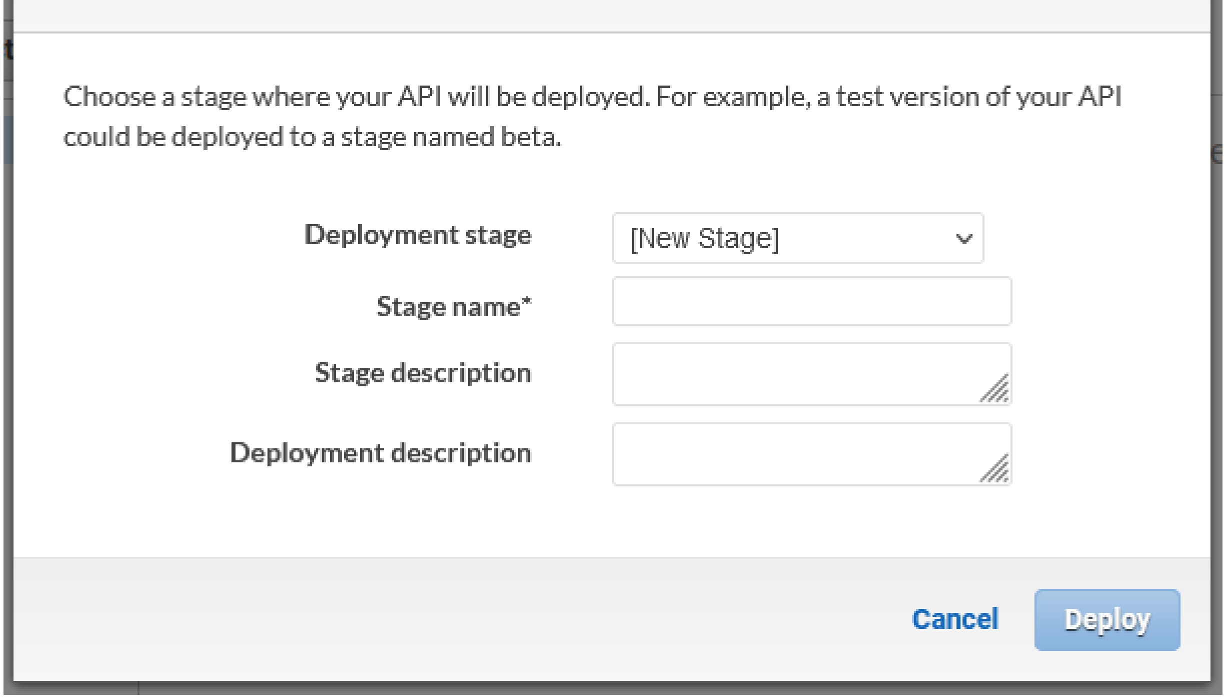
Now click again on ***/upload*** resource and then click on action, and then click on ***Enable CORS***.


Enable CORS

Gateway Responses for *binary-file-upload-api* API	☐ DEFAULT 4XX ☐ DEFAULT 5XX
Methods	✓ GET ✓ OPTIONS
Access-Control-Allow-Methods	GET, OPTIONS
Access-Control-Allow-Headers	'*'
Access-Control-Allow-Origin*	'*'

In **Accecc-Control-Allow-Headers** remove all existing content and type just `'*'` and leave other option default and click on Enable CORS and replace… button, and then press on Yes, replace existing values.




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*:
Stage description:
Deployment description:

Now click on your root resource `/` and then click on that Action button and choose `Deploy API` fill-up all details and click on Deploy, now will get endpoint of this API.

```
https://api-id.execute-api.aws-region.amazonaws.com/dev
```

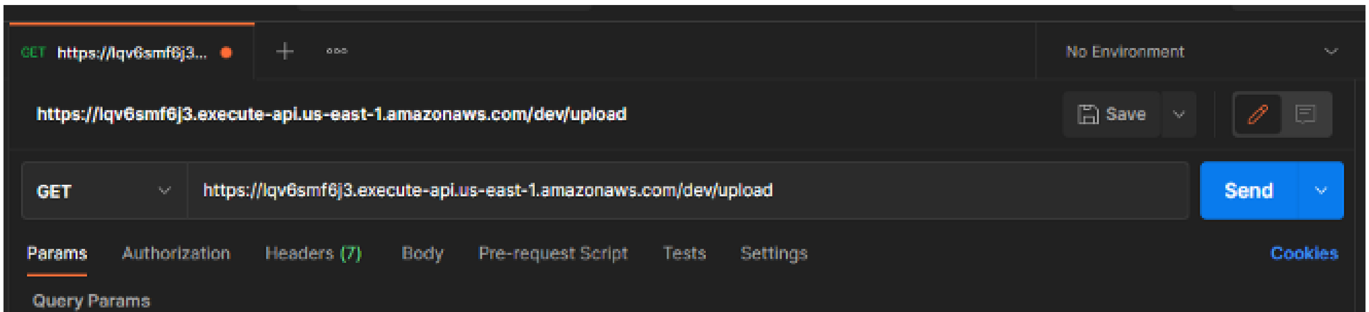
So, now you have configured all setup, now this turn of testing, I am going to show you API testing by **Postman**. Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs — faster.

Testing

First, get your API Endpoint from API Gateway Console and append your resource name into that endpoint so your API endpoint will look likes,

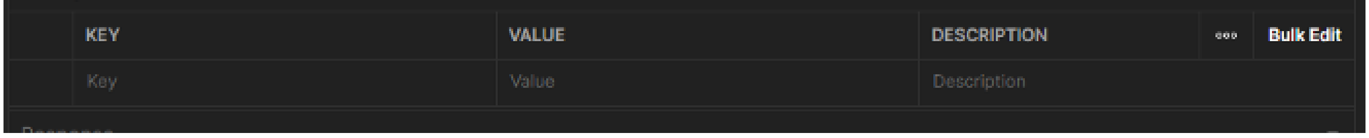
```
https://api-id.execute-api.aws-region.amazonaws.com/dev/upload
```

Now go to Postman Desktop Application and create GET Method and copy and paste above endpoint in postman, which look likes,

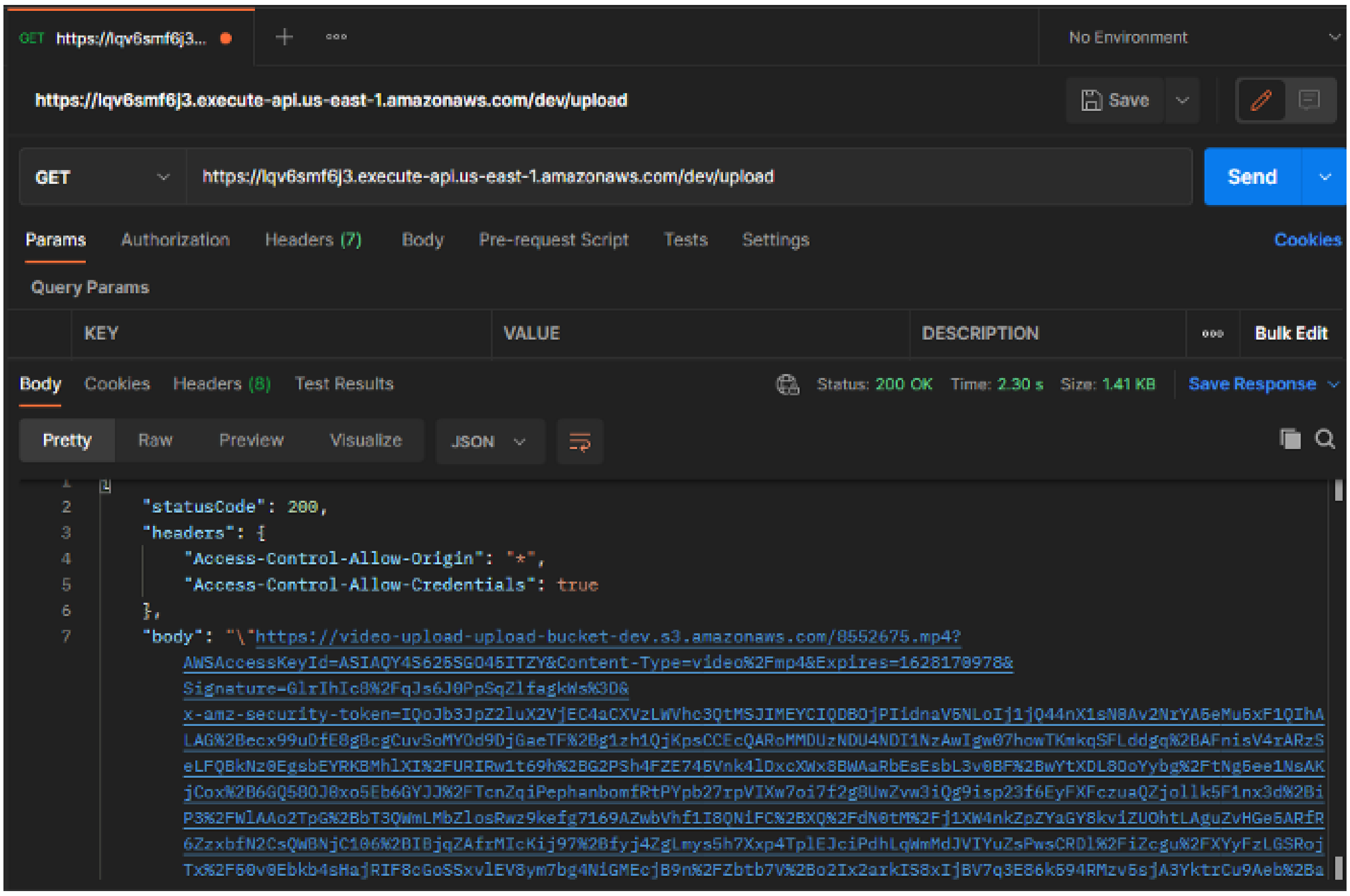

GET https://lqv6smf6j3...

https://lqv6smf6j3.execute-api.us-east-1.amazonaws.com/dev/upload

GET https://lqv6smf6j3.execute-api.us-east-1.amazonaws.com/dev/upload



Now fire the request by clicking on Send button, if you have configured perfectly as I shown you then you will get Success response from API.



https://lqv6smf6j3.execute-api.us-east-1.amazonaws.com/dev/upload

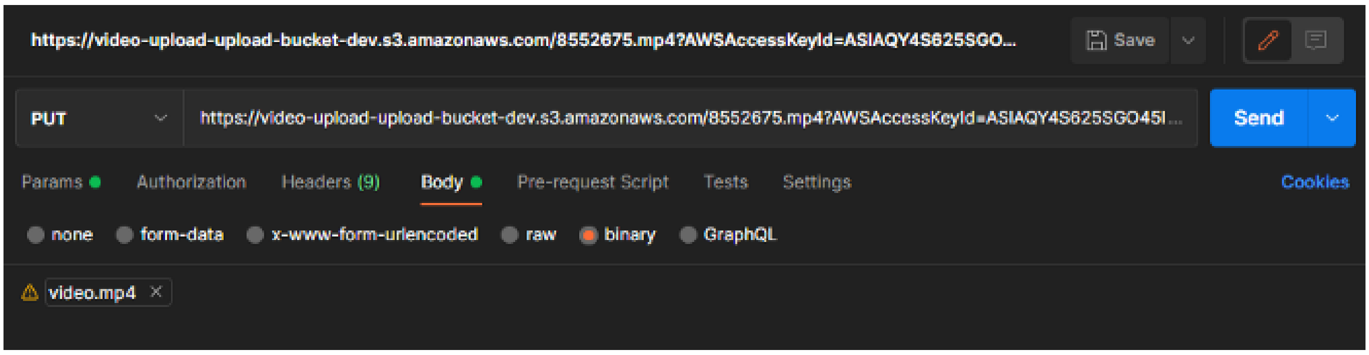
GET https://lqv6smf6j3.execute-api.us-east-1.amazonaws.com/dev/upload

Status: 200 OK Time: 2.30 s Size: 1.41 KB

```
{
    "statusCode": 200,
    "headers": {
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Credentials": true
    },
    "body": "\"https://video-upload-upload-bucket-dev.s3.amazonaws.com/8552675.mp4?
        AWSAccessKeyId=ASIAQY4S625SGO45ITZY&Content-Type=video%2Fmp4&Expires=1628170978&
        Signature=GlrIhIc8%2FqJs6J0PpSqZlfagkWs%3D&
        x-amz-security-token=IQoJb3JpZ2luX2VjEC4aCXVzLWVhc3QtMSJIMEYCIQDBOjPIidnaV6NLoIj1jQ44nX1sN8Av2NrYA6eMu6xF1QIhA
        LAG%2Becx99uDfE8gBcgCuvSoMYOd9DjGaeTF%2Bg1zh1QjKpsCCEcQARoMMDUzNDU4NDI1NzAwIgw07howTKmkqSFLddgq%2BAFnisV4rARzS
        eLFQBkNz0EgsbEYRKBMhlXI%2FURIRw1t69h%2BG2PSh4FZE745Vnk4lDxcXWx8BWAaRbEsEsbL3v0BF%2BwYtXDL8OoYybg%2FtNg5ee1NsAK
        jCox%2B6GQ580J8xo5Eb6GYJJ%2FTcnZqiPephanbomfRtPYpb27rpVIXw7oi7f2g8UwZvw3iQg9isp23f6EyFXFczuaQZjollk5F1nx3d%2Bi
        P3%2FWlAAo2TpG%2BbT3QWmLMbZlosRwz9kefg7169AZwbVhf1I8QNiFC%2BXQ%2FdN0tM%2Fj1XW4nkZpZYaGY8kviZUOhtLAguZvHGe5ARfR
        6ZzxbfN2CsQWBNjC106%2BIBjqZAfrMIcKij97%2Bfyj4ZgLmys5h7Xxp4TplEJciPdhLqWmMdJVIYuZsPwsCRDl%2FiZcgu%2FXYyFzLGSRoj
        Tx%2F60v0Ebkb4sHajRIF8cGoSSxvlEV8ym7bg4NiGMEcjB9n%2FZbtb7V%2Bo2Ix2arkIS8xIjBV7q3E86k594RMzv5sjA3YktrCu9Aeb%2Ba
```

In body part of success response you will get s3 pre-signedurl for putObject and it will valid for next X minutes that you have written in your lambda code.

Now copy and that URL only and create new PUT method in postman, and paste in URL. In **Body** tab choose binary and select binary file from your machine.



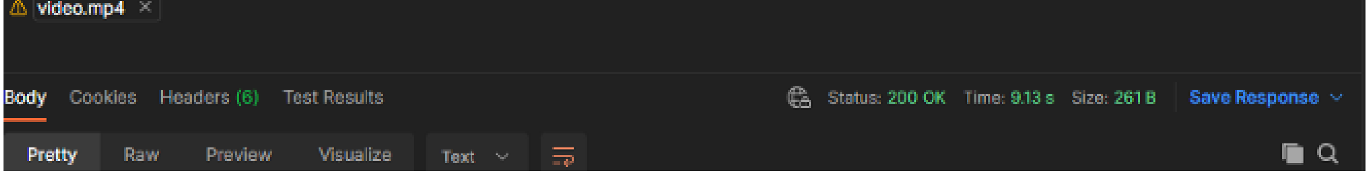
https://video-upload-upload-bucket-dev.s3.amazonaws.com/8552675.mp4?AWSAccessKeyId=ASIAQY4S625SGO...

PUT https://video-upload-upload-bucket-dev.s3.amazonaws.com/8552675.mp4?AWSAccessKeyId=ASIAQY4S625SGO45I...

none   form-data   x-www-form-urlencoded   raw   [✓ binary]   GraphQL

video.mp4

And now fire request by clicking on Send button, it will take time based on size of your binary file.



If you get status: 200 that means your binary file is successfully uploaded in S3 bucket, so you can check in Amazon S3.

So, that's it. Thanks for reading, and please follow and tell me other problems you may have faced or are facing so that I can create a solution for those problems also.

More content at ***plainenglish.io***

AWS AWS Lambda Cloud Computing Programming Coding