BUILDING A SERVERLESS URL SHORTENER ON AWS

Imagine a Giant Library full of books with very long titles. But reading of long titles is tiring so the Librarian gives each book a short code instead. Now, if you want a book, you just tell the Librarian the short code and they find the full title for you. That is how a severless URL Shortener works.
A serverless URL is a simple tool that takes long URL and creates a short version. When one clicks the short link, they are redirected to the original long URL. A serverless system means you don't need to manage servers. Instead, AWS handles everything for you and you only pay for what you use. The URL shortener consists of:
Frontend (S3 + CloudFront): A simple UI for users to input long URLs and get short links.
API (API Gateway + Lambda): Handles URL shortening and redirection.
Database (DynamoDB): Stores mappings between short and long URLs.
CloudFront: A Content Delivery Network (CDN) that helps deliver contents like websites, images, APIs etc faster to users .

Here, the DynamoDB is like the **Library Catalog** where we store short codes and full book titles (URLs). The **Librarian** is AWS Lambda, which is the brain behind everything. When someone brings a new book (long URL), the Librarian writes a short code in the Catalog and when someone asks for a book by its short code, the Librarian looks up the full title and gives it to them.
The API Gateway acts as the **Request Counter**. The Librarian needs a way to receive a request. API Gateway is like the **Help Desk** in the Library where people ask for books.
S3 acts as the Bookshelf where special books are stored. Some books are really popular, so instead of looking them up every time, the Librarian puts few copies on the shelf (S3) and if someone asks for one, they can get it instantly.
CloudFront acts as a **Messenger**. If too many people keep asking for the same book, the Librarian gets tired. So, the Librarian hires a Messenger (CloudFront). If one asks for a book, the Messenger might already have a copy and deliver it fast.

This guide walks through the process of building a URL shortener using AWS services, including AWS Lambda, API Gateway, DynamoDB, S3 and Cloudfront.

PREREQUISITES

AWS Account
IAM Role with permissions for Lambda, DynamoDB, API Gateway, S3 and CloudFront

STEP 1: CREATE AN IAM ROLE FOR LAMBDA

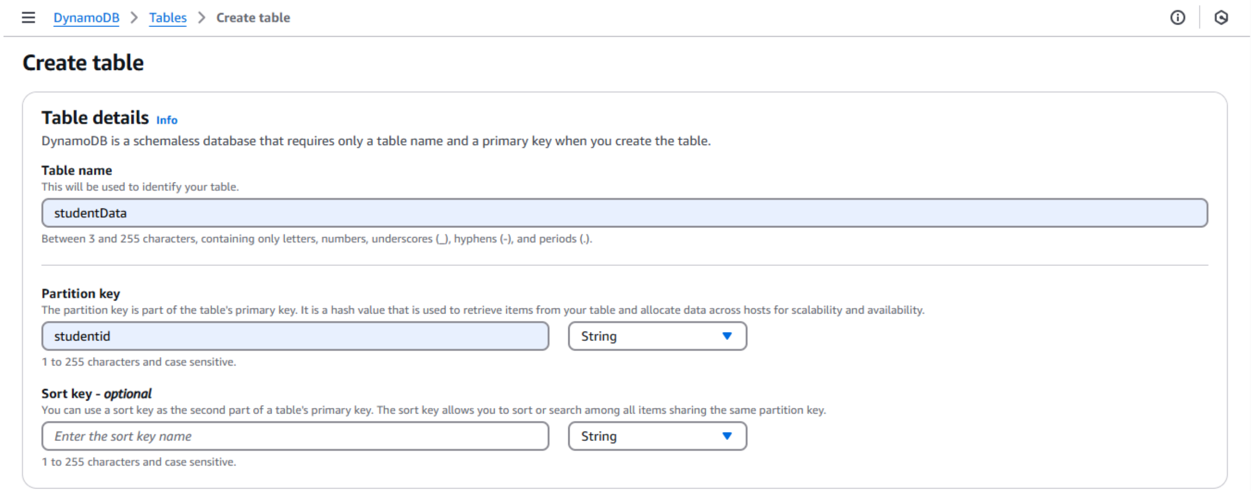
Since AWS Lambda will interact with multiple services, we need to create an IAM role with the necessary policies.

- ➢ Go to IAM in the AWS Console
- Click on Roles
- Create Role
- Select AWS Service
- Choose Lambda
- Click Next
- ➢ Attach the necessary policies
- AmazonDynamoDBFullAccess : for reading and writing URLs in DynamoDB
- AWSLambdaBasicExecutionRole: for Lambda logging to CloudWatch
- AmazonS3FullAccess : for reading and writing from S3
- CloudFrontFullAccess : for managing CloudFront distribution
- Click **Next**
- Add a name “**admin-lambda-role**”
- Click **Create Role.**

STEP 2: SETTING UP THE BACKEND (Lambda and DynamoDB)

We need a database to store both short and long URLs

- ➢ **Create a DynamoDB Table**
- Go to AWS Console
- Open AWS DynamoDB
- Click create table
- Set up Table Details
  - Table Name: studentData
  - Partition Key: studentid
  - Leave it as string



Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

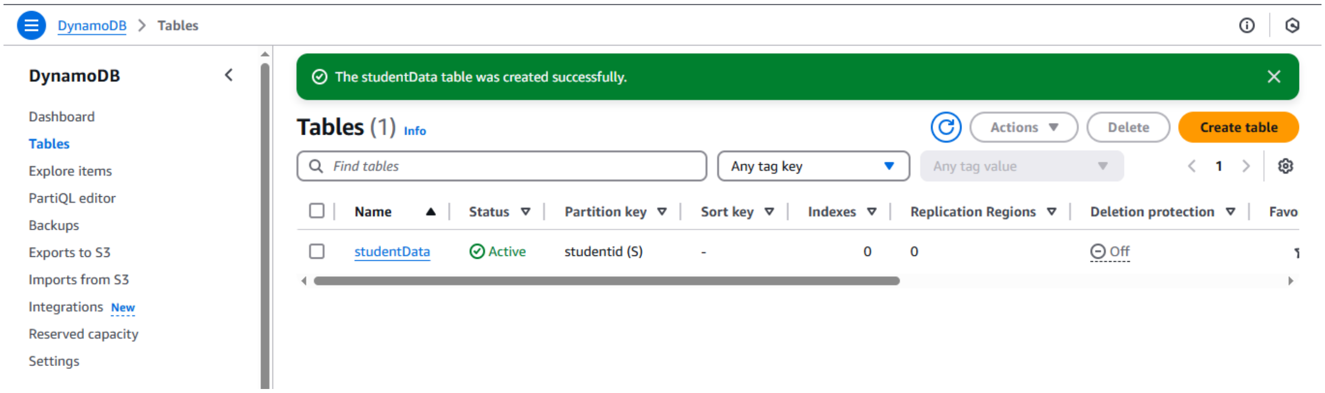
Table name
This will be used to identify your table.
studentData
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
studentid | String
1 to 255 characters and case sensitive.

Sort key - *optional*
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name | String
1 to 255 characters and case sensitive.

- Leave other settings as default
- Create Table

Now you have a database to store URLs.



The studentData table was created successfully.

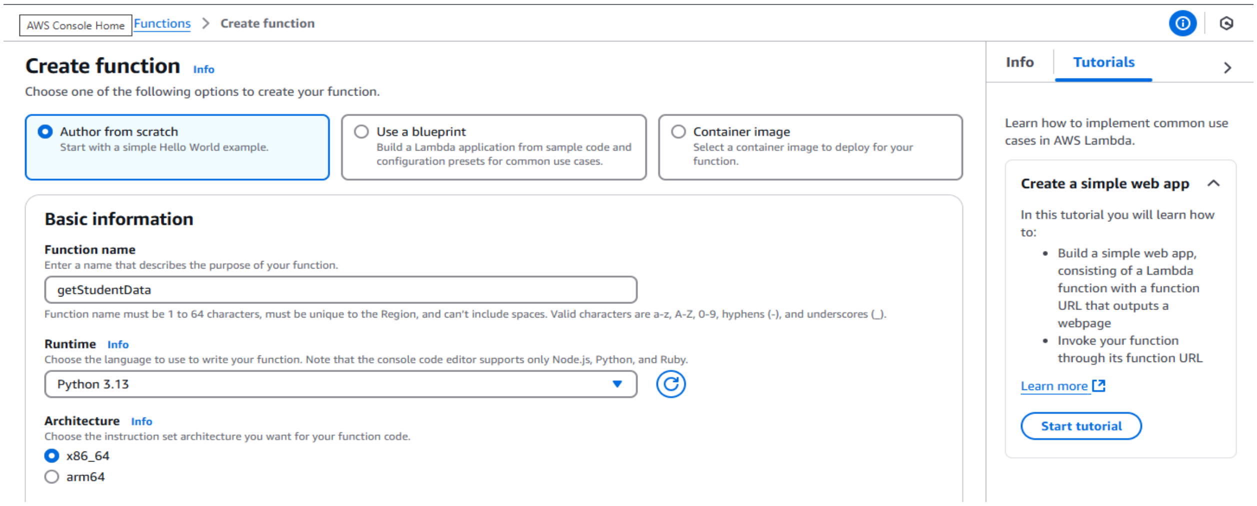
Tables (1) Info

□	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favo
□	studentData	Active	studentid (S)	-	0	0	Off	

➢ **Create a Lambda Function**

Lambda will handle URL shortening and redirection.

- Go to AWS Console
- Open AWS Lambda
- Click **Create function**
- Select **Author from scratch**
- Configure Lambda
    - Function Name: **getStudentData**
    - Runtime: **python 3.13**



AWS Console Home Functions > Create function

Create function Info

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.

Basic information

Function name
Enter a name that describes the purpose of your function.

getStudentData

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

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

Python 3.13

Architecture Info
Choose the instruction set architecture you want for your function code.

● x86_64
○ arm64

Info | Tutorials

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

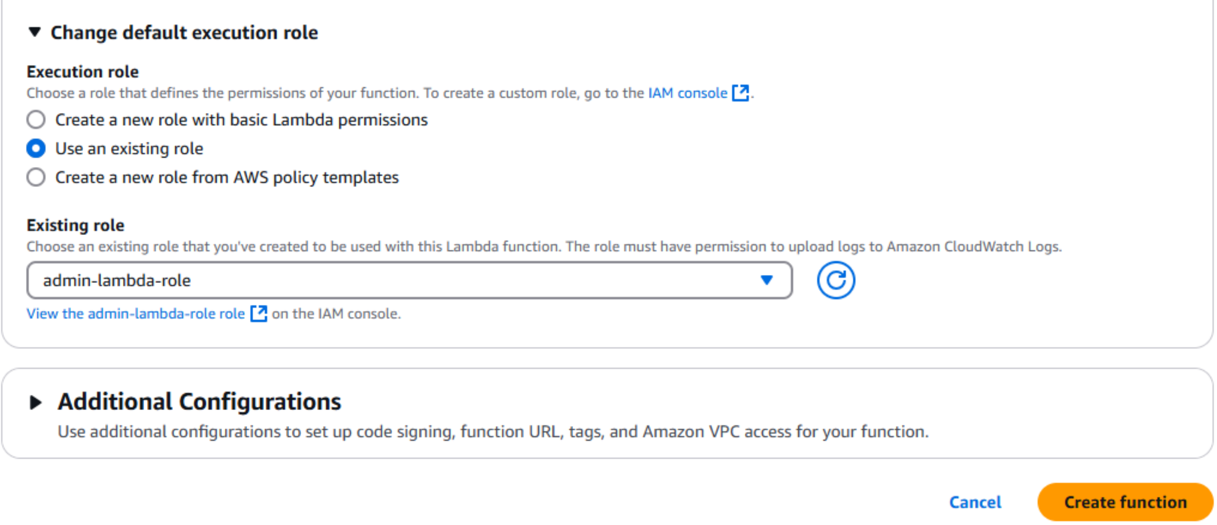
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

Start tutorial

- Permissions: choose use an existing role as a default execution role(select the Lambda role you had created)

- Click **Create Function**.



▼ **Change default execution role**

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

○ Create a new role with basic Lambda permissions
● 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.

admin-lambda-role

View the admin-lambda-role role on the IAM console.

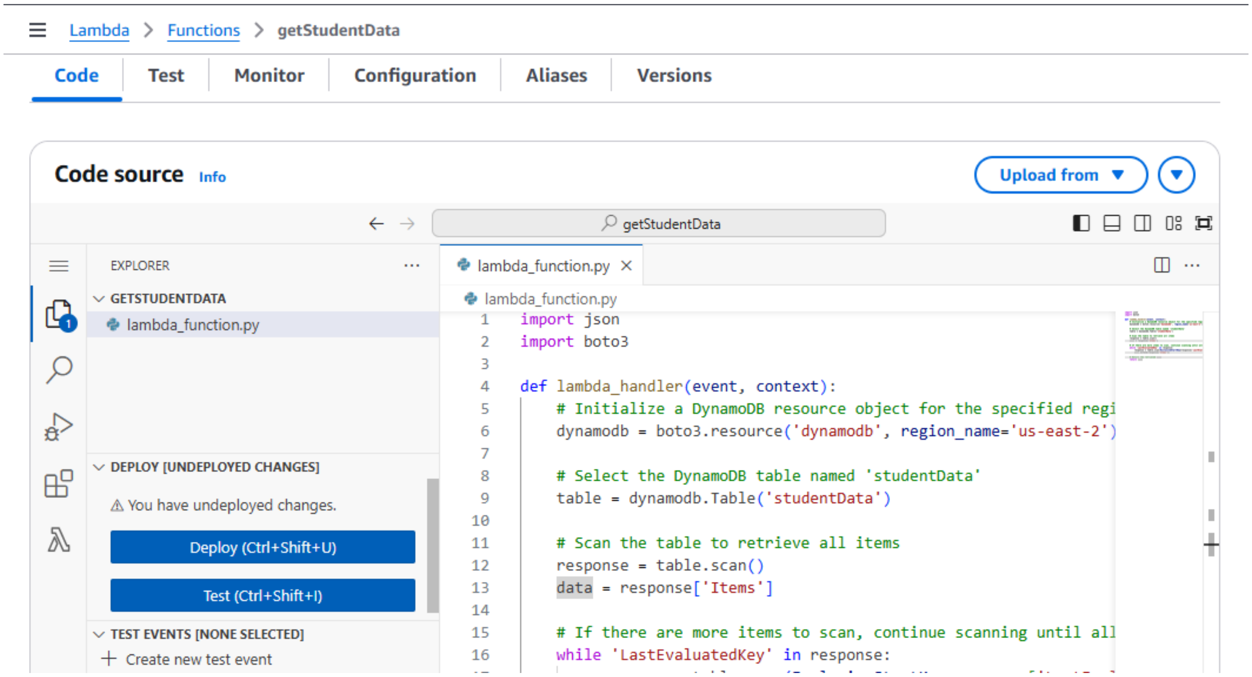
▶ **Additional Configurations**
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Cancel | Create function

Our Lambda function is ready.

- Add python code for Lambda
- Once the function is created
- Scroll down to **code source**
- Delete the default code and paste the getStudentsData.py code

```
import json
import boto3

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified region
    dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
    # Select the DynamoDB table named 'studentData'
    table = dynamodb.Table('studentData')

    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all items are retrieved
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    # Return the retrieved data
    return data
```



```
import json
import boto3

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified regi
    dynamodb = boto3.resource('dynamodb', region_name='us-east-2')

    # Select the DynamoDB table named 'studentData'
    table = dynamodb.Table('studentData')

    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all
    while 'LastEvaluatedKey' in response:
```

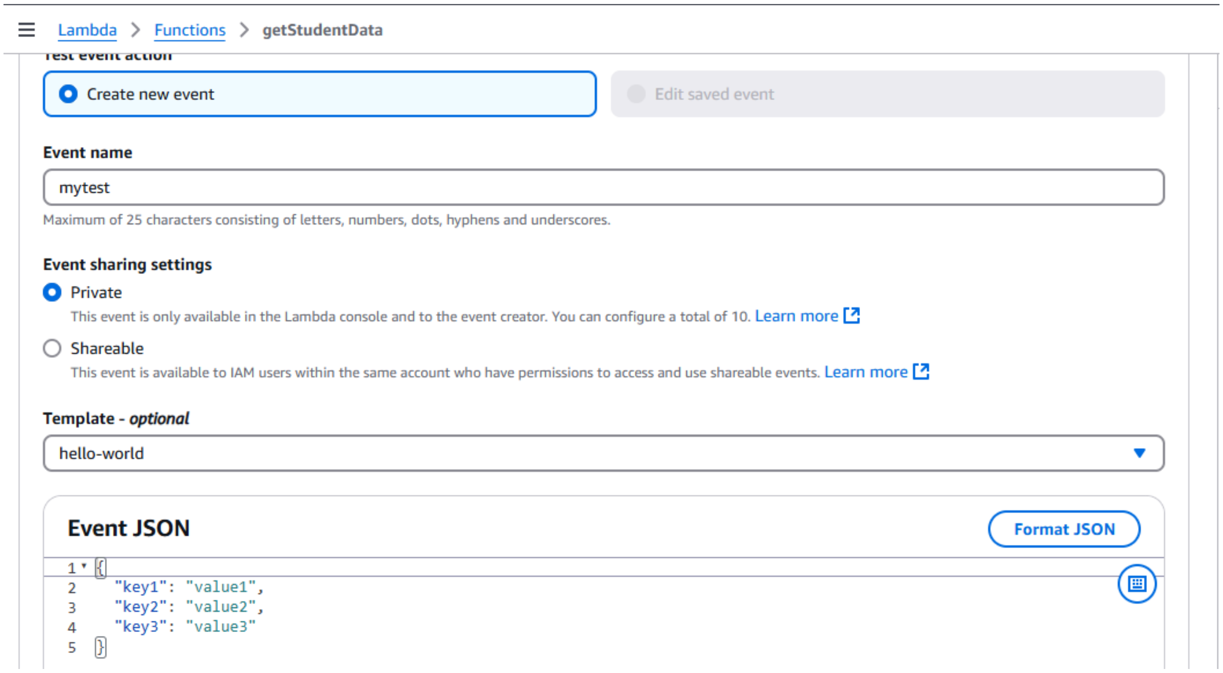
- Deploy Lambda by Clicking **Deploy**

Your Lambda function is now deployed

If you check DynamoDB, you will realize that there is no table.

- Go to Lambda function
- Create a new test event with the name “**mytest**”.
- Paste the code on the Event JSON

{
“Key 1”: “ Value1”,
“Key 2”: “value2”,
“Key 3”: “ value3”
}



Test event action

● Create new event ○ Edit saved event

Event name

mytest

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

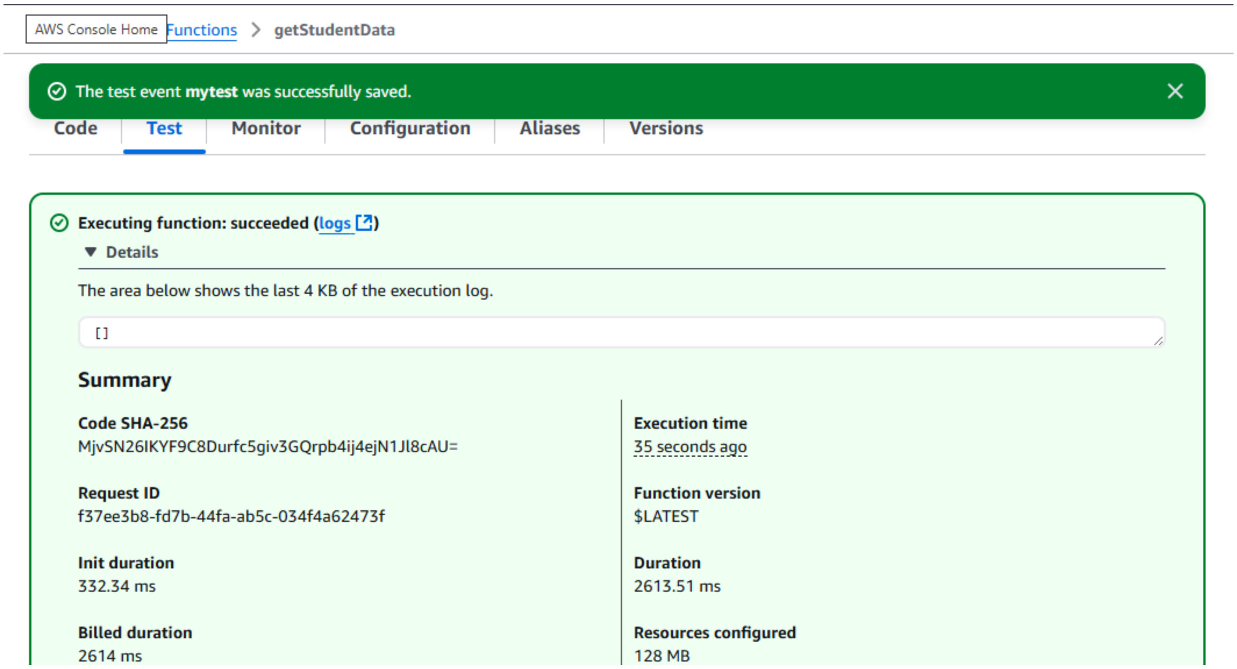
Template - *optional*

hello-world

Event JSON

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

- Save and test the “**newtest**” to invoke the Lambda function.
  This Lambda function will go to DynamoDB Table and try to retrieve data from the table but the response will give empty list so we will create another function for posting data and name it **insertStudentData**.



The test event **mytest** was successfully saved.

Executing function: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.

```
[]
```

Summary

Code SHA-256
MjvSN26IKYF9C8Durfc5giv3GQrpb4ij4ejN1Jl8cAU=

Execution time
35 seconds ago

Request ID
f37ee3b8-fd7b-44fa-ab5c-034f4a62473f

Function version
$LATEST

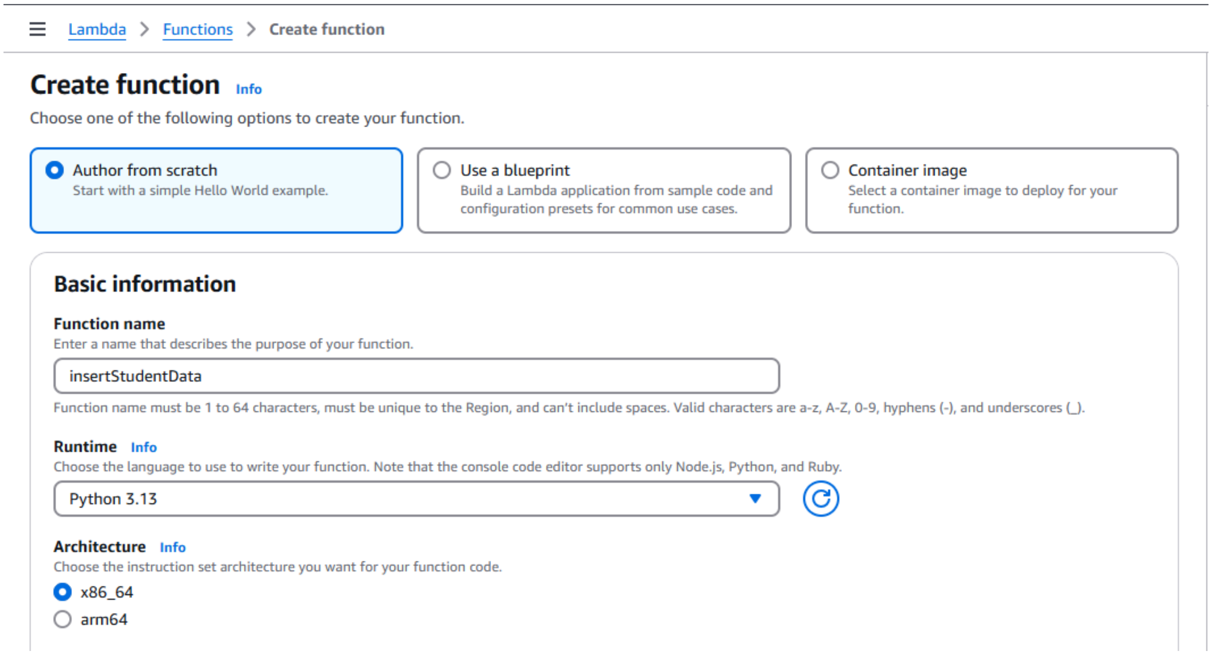
Init duration
332.34 ms

Duration
2613.51 ms

Billed duration
2614 ms

Resources configured
128 MB

- Go to functions and create a new function with the following details
  - Name: **insertStudentData**
  - Runtime: **python 3.13**



Create function

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.

Basic information

Function name
Enter a name that describes the purpose of your function.

insertStudentData

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

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

Python 3.13

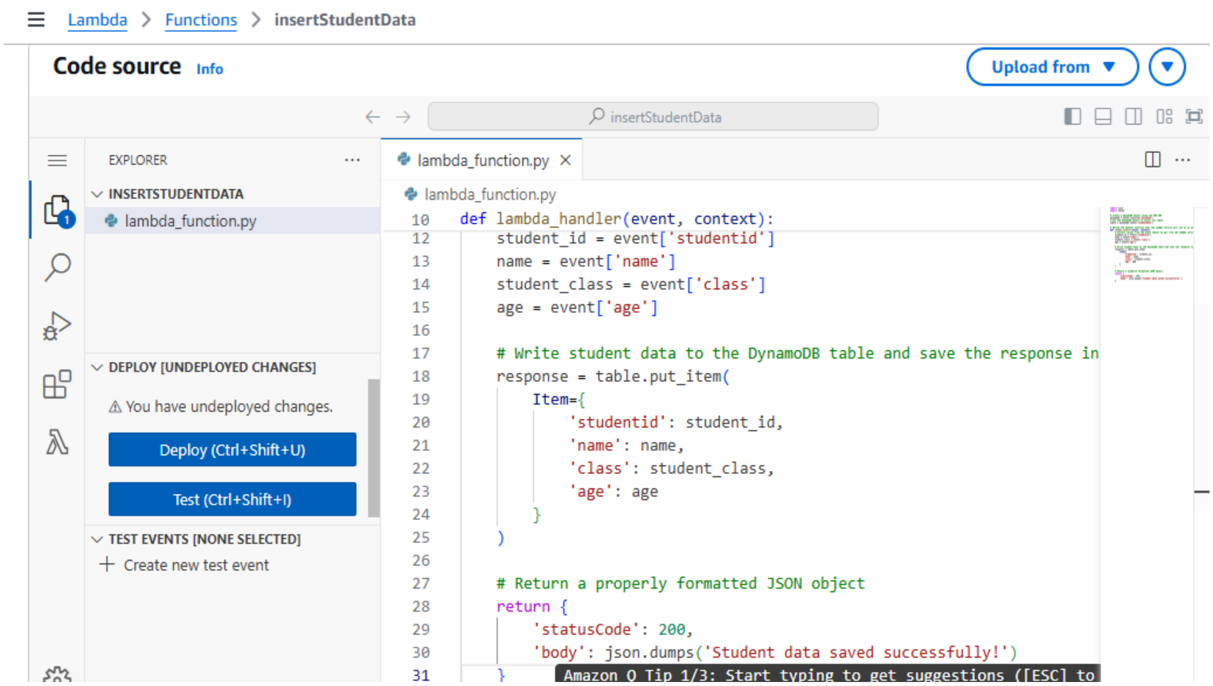
Architecture
Choose the instruction set architecture you want for your function code.

● x86_64
○ arm64

- Permissions: **choose create an existing role** (select the Lambda role you had created)

- Click **Create Function**.
- Copy and paste **insertStudentData.py** file and paste

```
import json
import boto3

# Create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# Use the DynamoDB object to select our table
table = dynamodb.Table('studentData')

# Define the handler function that the Lambda service will use as an entry point
def lambda_handler(event, context):
   # Extract values from the event object we got from the Lambda service and store in
variables
   student_id = event['studentid']
   name = event['name']
   student_class = event['class']
   age = event['age']

   # Write student data to the DynamoDB table and save the response in a variable
   response = table.put_item(
      Item={
         'studentid': student_id,
         'name': name,
         'class': student_class,
         'age': age
      }
   )

   # Return a properly formatted JSON object
   return {
      'statusCode': 200,
      'body': json.dumps('Student data saved successfully!')
   }
```

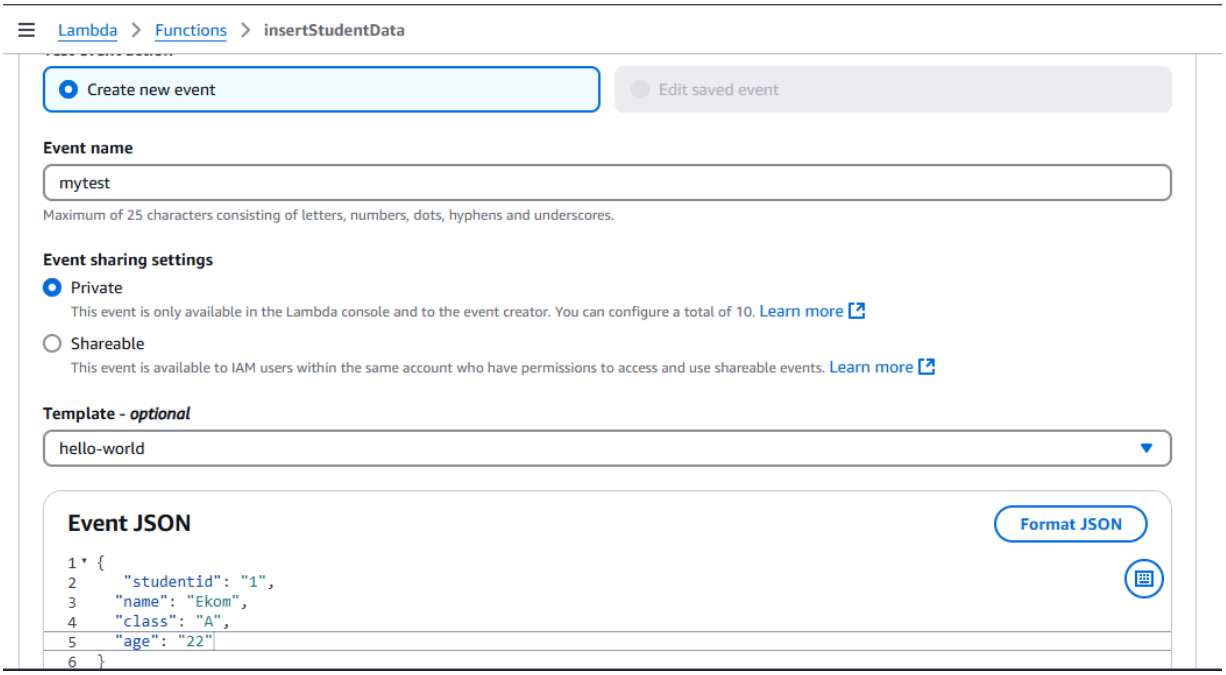

Lambda > Functions > insertStudentData
Code source Info
Upload from ▼
insertStudentData
EXPLORER
INSERTSTUDENTDATA
lambda_function.py
DEPLOY [UNDEPLOYED CHANGES]
⚠ You have undeployed changes.
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
+ Create new test event
lambda_function.py
Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to

```python
def lambda_handler(event, context):
    student_id = event['studentid']
    name = event['name']
    student_class = event['class']
    age = event['age']

    # Write student data to the DynamoDB table and save the response in
    response = table.put_item(
        Item={
            'studentid': student_id,
            'name': name,
            'class': student_class,
            'age': age
        }
    )

    # Return a properly formatted JSON object
    return {
        'statusCode': 200,
        'body': json.dumps('Student data saved successfully!')
    }
```

- Click on **deploy**
- Create test event with the name "**mytest**"
  We need to give students name, class, age and id
- Paste the following values. These are the data we will pass to Lambda function

```
{
  "studentid": "1",
  "name": "Ekom",
  "class": "A",
  "age": "22"
}
```

This is a part gotten from the inserStudentData.py that has been deployed



● Create new event
○ Edit saved event

Event name

mytest

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

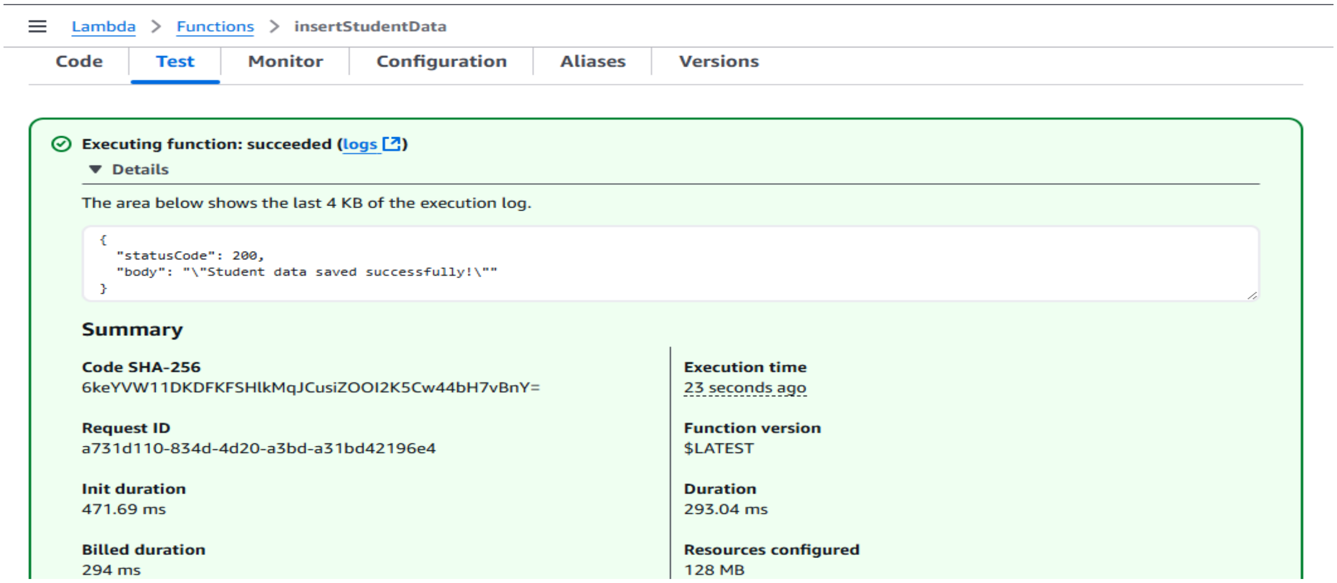
hello-world

Event JSON

```
{
   "studentid": "1",
  "name": "Ekom",
  "class": "A",
  "age": "22"
}
```

- Save and test

You will see a response “**student data saved successfully**!”.



Executing function: succeeded (logs)

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "\"Student data saved successfully!\""
}
```

Summary

Code SHA-256
6keYVW11DKDFKFSHlkMqJCusiZOOI2K5Cw44bH7vBnY=

Execution time
23 seconds ago

Request ID
a731d110-834d-4d20-a3bd-a31bd42196e4

Function version
$LATEST

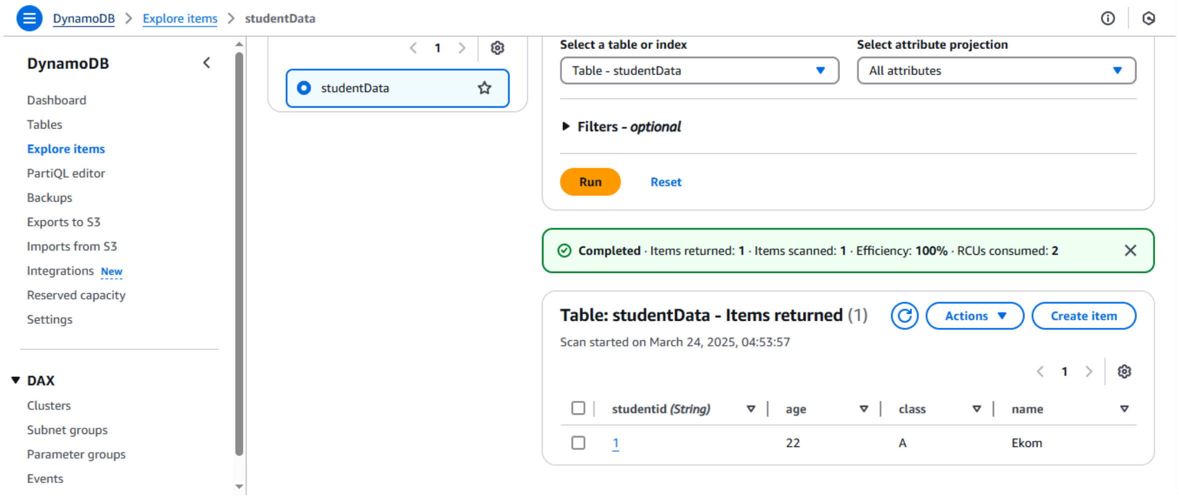
Init duration
471.69 ms

Duration
293.04 ms

Billed duration
294 ms

Resources configured
128 MB

If you go to DynamoDB table and refresh, you will find the student data you had tested on Lambda.



Select a table or index: Table - studentData

Select attribute projection: All attributes

Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 2

Table: studentData - Items returned (1)

Scan started on March 24, 2025, 04:53:57

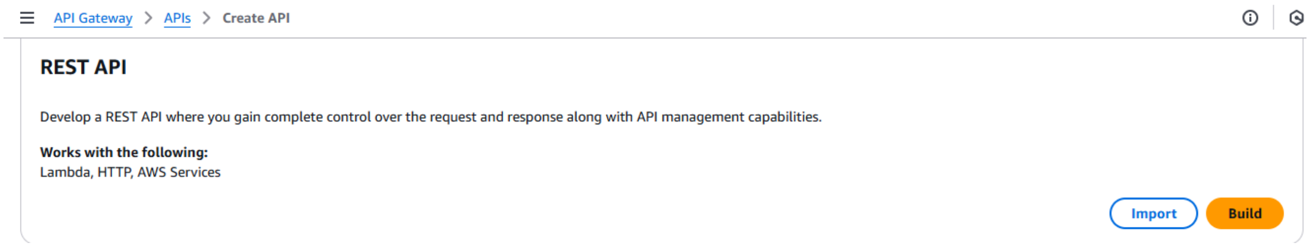
studentid (String)	age	class	name
1	22	A	Ekom

STEP 3: DEPLOY API GATEWAY (FOR PUBLIC ACCESS)

- Create API Gateway (For public access)

We need an API to connect to Lambda

- Go to API Console
- Open AWS API Gateway
- Click Create API
- Set up API
- Select **Rest API**
- Click Build

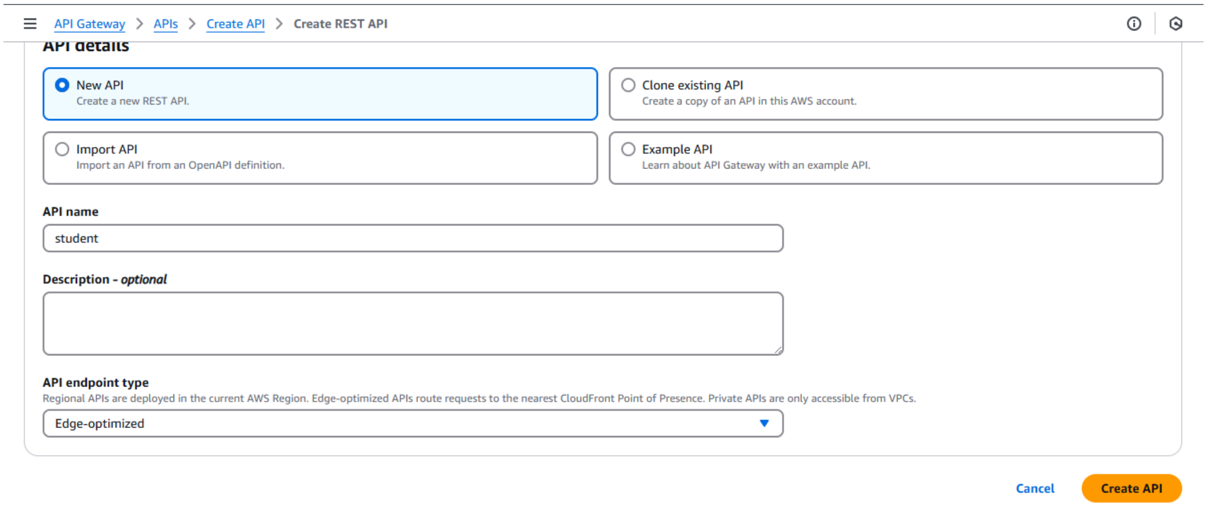


REST API

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

- Input the API details
- Choose New API
- Input API Name as **student**
- Endpoint: Edge-optimized
  Edge-optimized allows users from around the world to access the website anywhere.



☰ API Gateway > APIs > Create API > Create REST API

API details

● New API
Create a new REST API.

○ Clone existing API
Create a copy of an API in this AWS account.

○ Import API
Import an API from an OpenAPI definition.

○ Example API
Learn about API Gateway with an example API.

API name

student

Description - *optional*

API endpoint type

Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

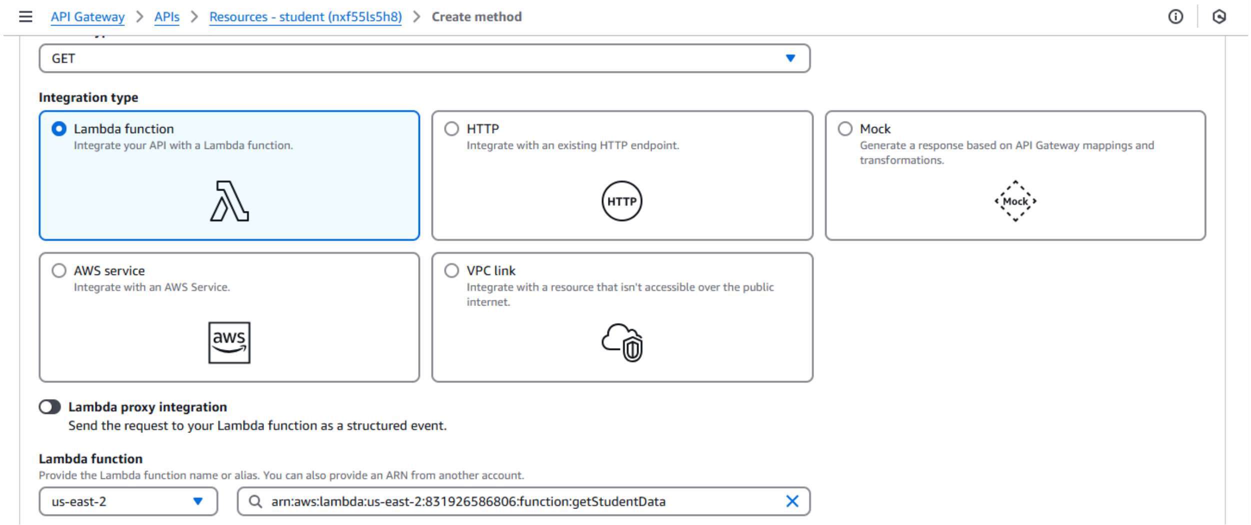
Edge-optimized ▼

Cancel | **Create API**

- Create API

➢ Create API resources and methods

- Click **create methods**
- Select “**GET**” as type
- Select **Lambda function** as the integration type.
- Select “**getStudentData**” function to connect to your Lambda function.



☰ API Gateway > APIs > Resources - student (nxf55ls5h8) > Create method

GET ▼

Integration type

● Lambda function
Integrate your API with a Lambda function.

○ HTTP
Integrate with an existing HTTP endpoint.

○ Mock
Generate a response based on API Gateway mappings and transformations.

○ AWS service
Integrate with an AWS Service.

○ VPC link
Integrate with a resource that isn't accessible over the public internet.

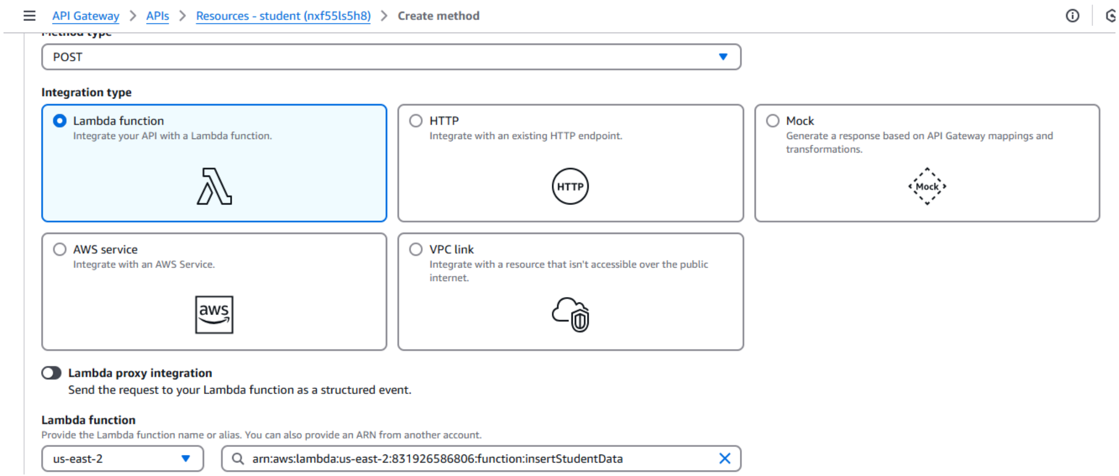
Lambda proxy integration
Send the request to your Lambda function as a structured event.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-2 ▼ | arn:aws:lambda:us-east-2:831926586806:function:getStudentData ✕

- Click on **Create method**

  We have to create another method
- Select “**POST**” as type
- Select “**insertStudentData**” function to connect to your Lambda function



API Gateway > APIs > Resources - student (nxf55ls5h8) > Create method

Method type

POST

Integration type

● Lambda function
Integrate your API with a Lambda function.

○ HTTP
Integrate with an existing HTTP endpoint.

○ Mock
Generate a response based on API Gateway mappings and transformations.

○ AWS service
Integrate with an AWS Service.

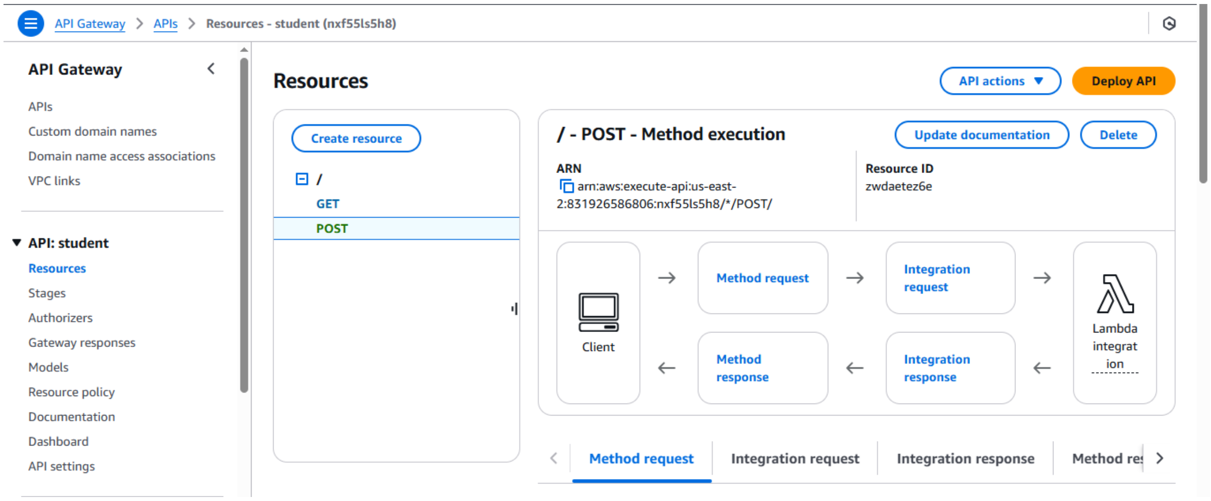
○ VPC link
Integrate with a resource that isn't accessible over the public internet.

Lambda proxy integration
Send the request to your Lambda function as a structured event.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-2 | arn:aws:lambda:us-east-2:831926586806:function:insertStudentData

- Click **Deploy API**



API Gateway > APIs > Resources - student (nxf55ls5h8)

Resources

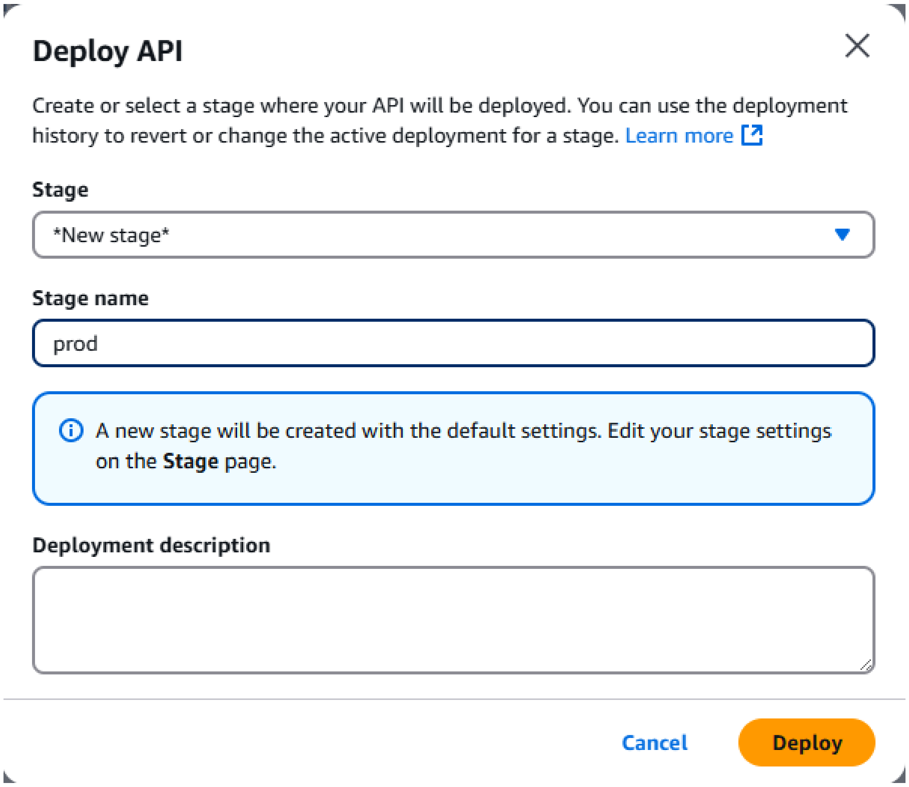
/
GET
POST

/ - POST - Method execution

ARN
arn:aws:execute-api:us-east-2:831926586806:nxf55ls5h8/*/POST/

Resource ID
zwdaetez6e

- Select new stage name and type in “**prod”** as the stage name



Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. Learn more

Stage

New stage

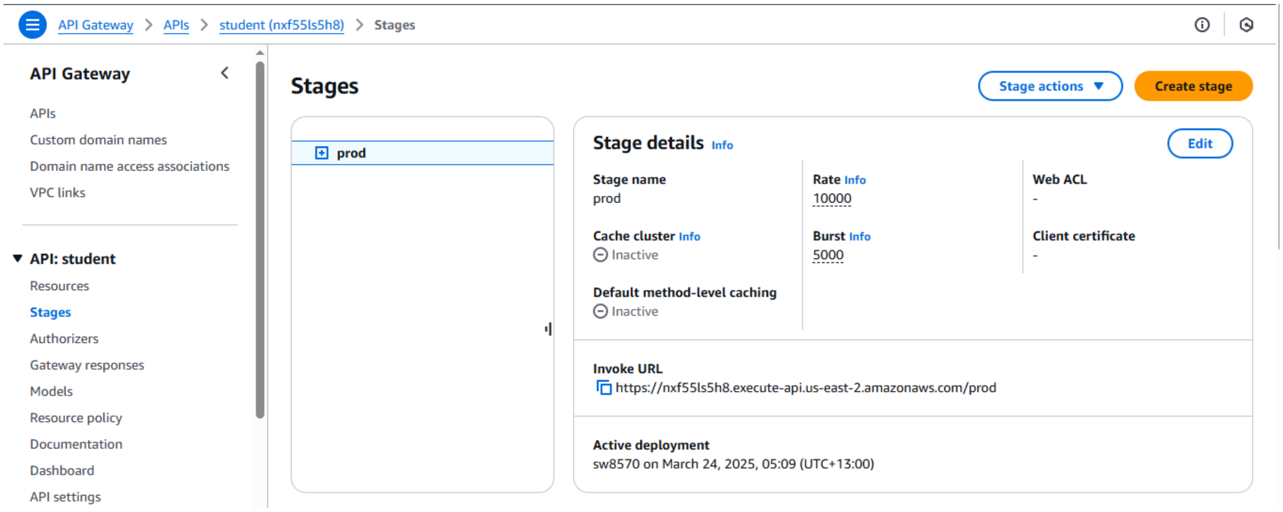
Stage name

prod

A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

Deployment description

- Click **Deploy**



Stages

Stage details Info

Stage name: prod

Rate Info: 10000

Web ACL: -

Cache cluster Info: Inactive

Burst Info: 5000

Client certificate: -

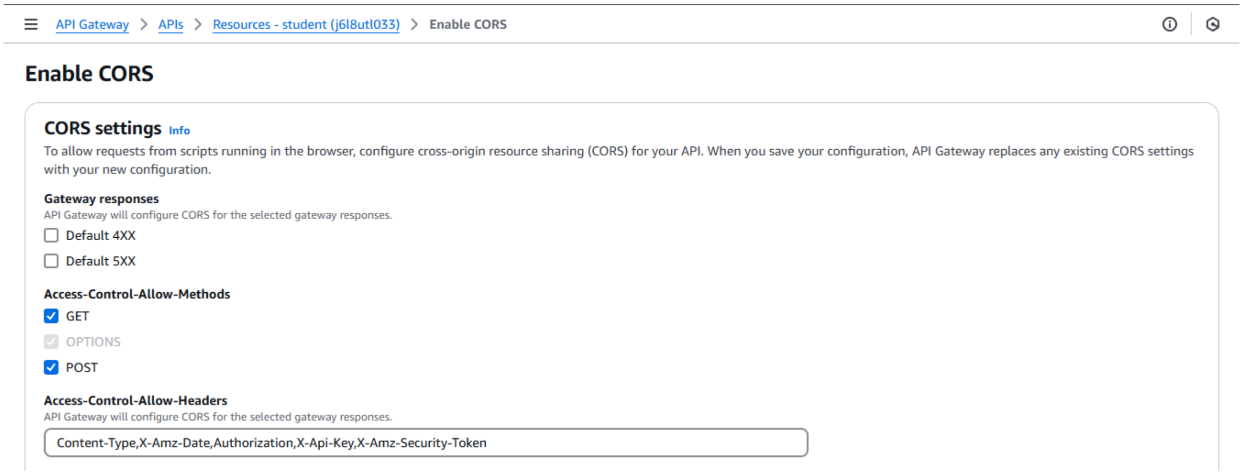
Default method-level caching: Inactive

Invoke URL
https://nxf55ls5h8.execute-api.us-east-2.amazonaws.com/prod

Active deployment
sw8570 on March 24, 2025, 05:09 (UTC+13:00)

You will find your invoke URL. This URL will invoke our Lambda function.

- Go to resources and enable CORS by enabling **POST** and **GET** in the CORS settings.



API Gateway > APIs > Resources - student (j6l8utl033) > Enable CORS

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses
API Gateway will configure CORS for the selected gateway responses.

☐ Default 4XX
☐ Default 5XX

Access-Control-Allow-Methods

☑ GET
☑ OPTIONS
☑ POST

Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

Now your API is live.

- Note the API Gateway URL.
- Copy the invoked URL
- Open the script.js and paste it where you find the endpoint

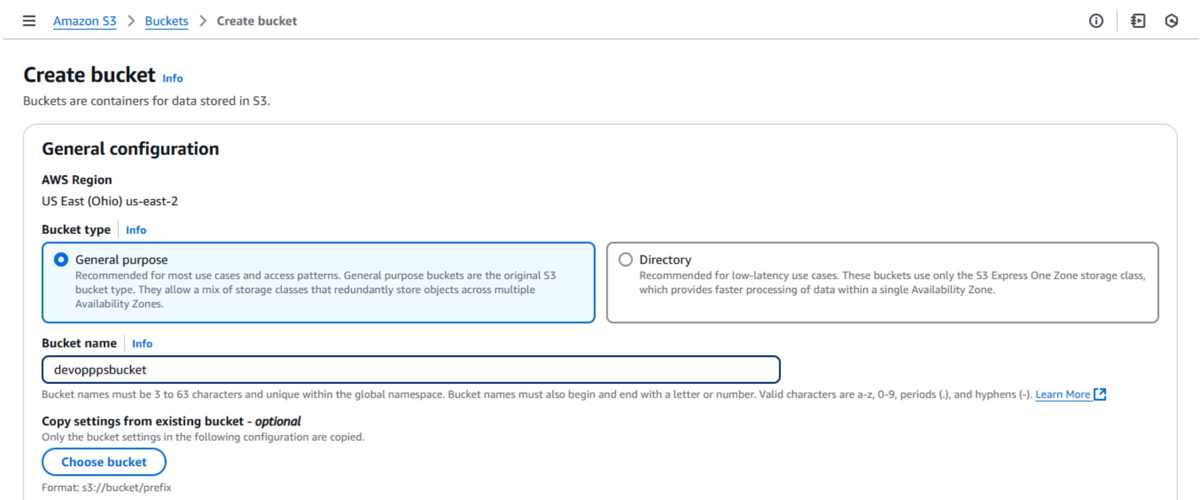
```
   // Add your API endpoint here
 var API_ENDPOINT = "https://j6l8utl033.execute-api.us-east-2.amazonaws.com/prod";
```

When you click on “**getStudentData**”, the URL pasted on the .js script will be invoked and it will go to get Lambda function which will be triggered from the DynamoDB table.

STEP 4: DEPLOY THE FRONTEND USING S3 AND CLOUDFRONT

➢ Go to AWS S3 Console

- Click **Create Bucket**
- Enter a unique Bucket Name **devopppsbucket**



Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (Ohio) us-east-2

Bucket type | Info

● General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

○ Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name | Info

devopppsbucket

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). Learn More

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Format: s3://bucket/prefix

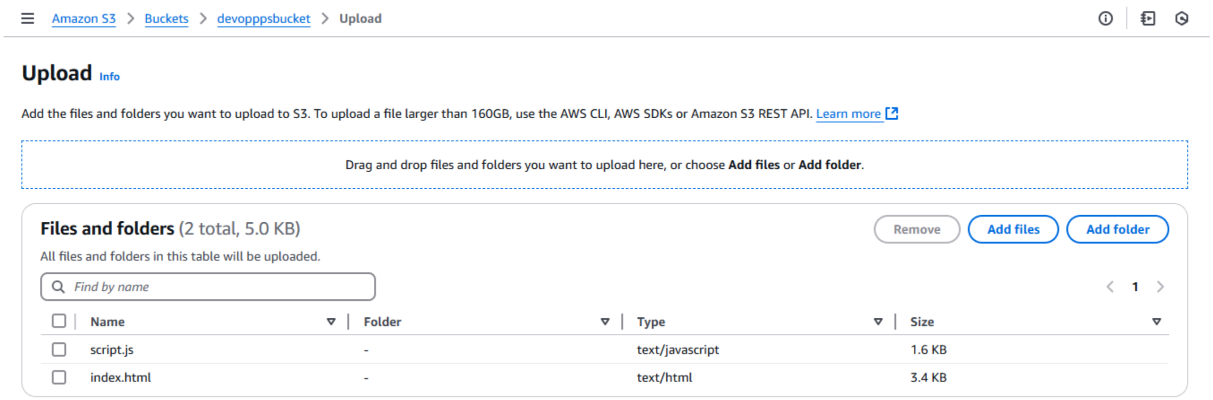
- Uncheck "**Block all public access**" settings
- Click **create bucket** to finalize the setup

➢ Upload Files to the Bucket

- Open the javascript file and replace "Endpoint" with the API Gateway invoke URL
- Click on the Bucket name to open the Bucket.
- Click Upload to select files from your local machine and click **upload.**
- Upload the websites files (an html and javascript frontend for users to enter URLs).

The html is the main web page and the javascript is used to call the API Gateway.

- Upload files to the S3 bucket.



Upload Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDKs or Amazon S3 REST API. Learn more

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (2 total, 5.0 KB)

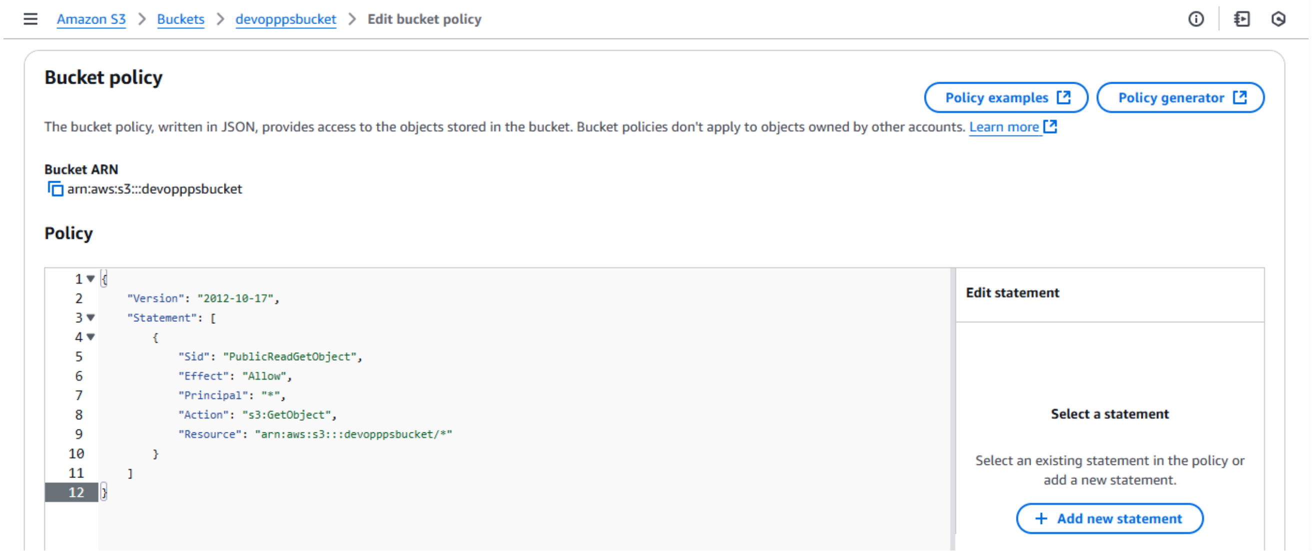
All files and folders in this table will be uploaded.

□	Name	Folder	Type	Size
□	script.js	-	text/javascript	1.6 KB
□	index.html	-	text/html	3.4 KB

➢ Configure Bucket Permissions (for public access)

- Go to the **Permission** tab of your Bucket.
- Scroll down to the **Bucket Policy** Section and click on Edit to add a policy. Use the Bucket policy.

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



Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Bucket ARN
arn:aws:s3:::devopppsbucket

Policy

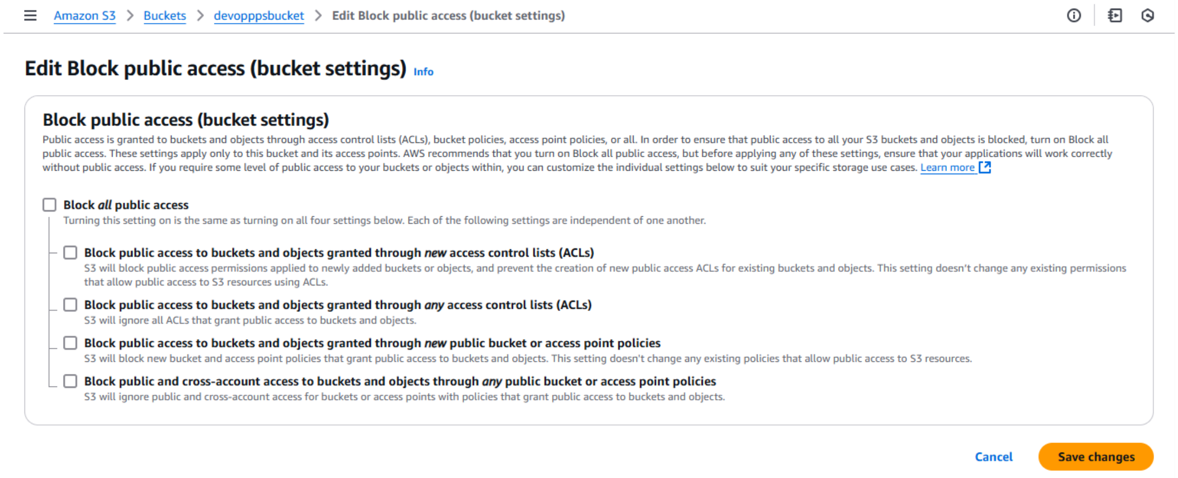
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::devopppsbucket/*"
        }
    ]
}
```

Select an existing statement in the policy or add a new statement.

  - Click **Save changes** to apply the policy.

➢ Configure Permissions for your Bucket

  - Navigate to **Permissions** tab of your Bucket
  - Uncheck “**Block all public access**”.



Amazon S3 > Buckets > devopppsbucket > Edit Block public access (bucket settings)

Edit Block public access (bucket settings) Info

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

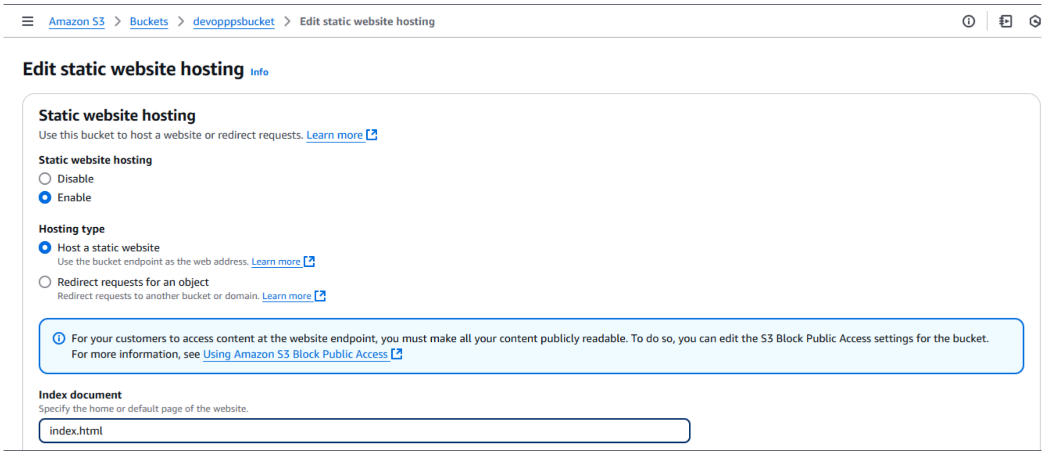
□ **Block *all* public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- □ **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.

- Click on **Save changes.**

➢ Enable Static Website Hosting

- Go to Properties
- Scroll down to **Static website hosting** and click **Edit**
- Select **Enable**
- Choose **Host a static website.**
- Set index document as **index.html**



Amazon S3 > Buckets > devopppsbucket > Edit static website hosting

Edit static website hosting Info

Static website hosting

Use this bucket to host a website or redirect requests. Learn more

Static website hosting
○ Disable
● Enable

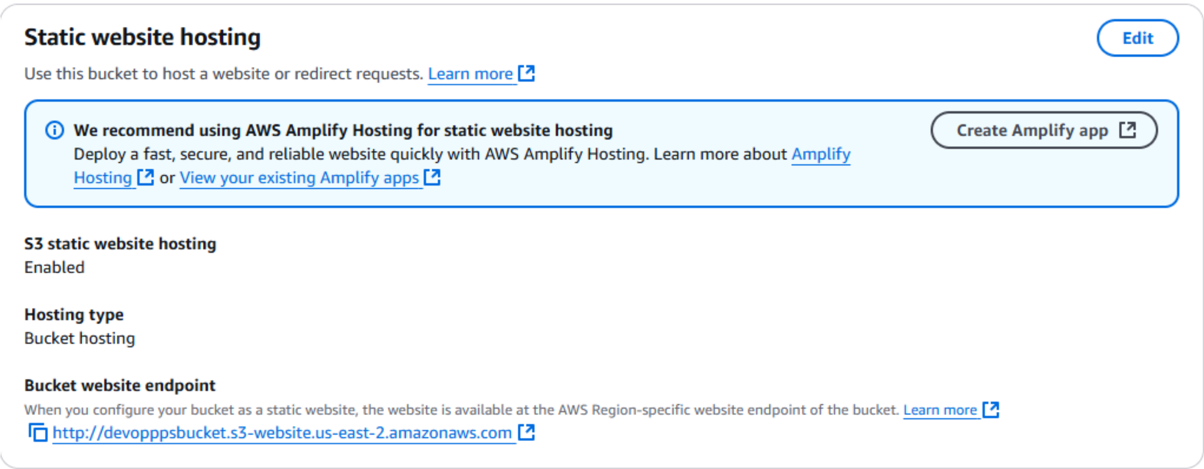
Hosting type
● Host a static website
Use the bucket endpoint as the web address. Learn more
○ Redirect requests for an object
Redirect requests to another bucket or domain. Learn more

For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see Using Amazon S3 Block Public Access

Index document
Specify the home or default page of the website.
index.html

- Click **Save changes**

Your website endpoint will be displayed



Static website hosting

Use this bucket to host a website or redirect requests. Learn more

We recommend using AWS Amplify Hosting for static website hosting

Deploy a fast, secure, and reliable website quickly with AWS Amplify Hosting. Learn more about Amplify Hosting or View your existing Amplify apps

S3 static website hosting

Enabled

Hosting type

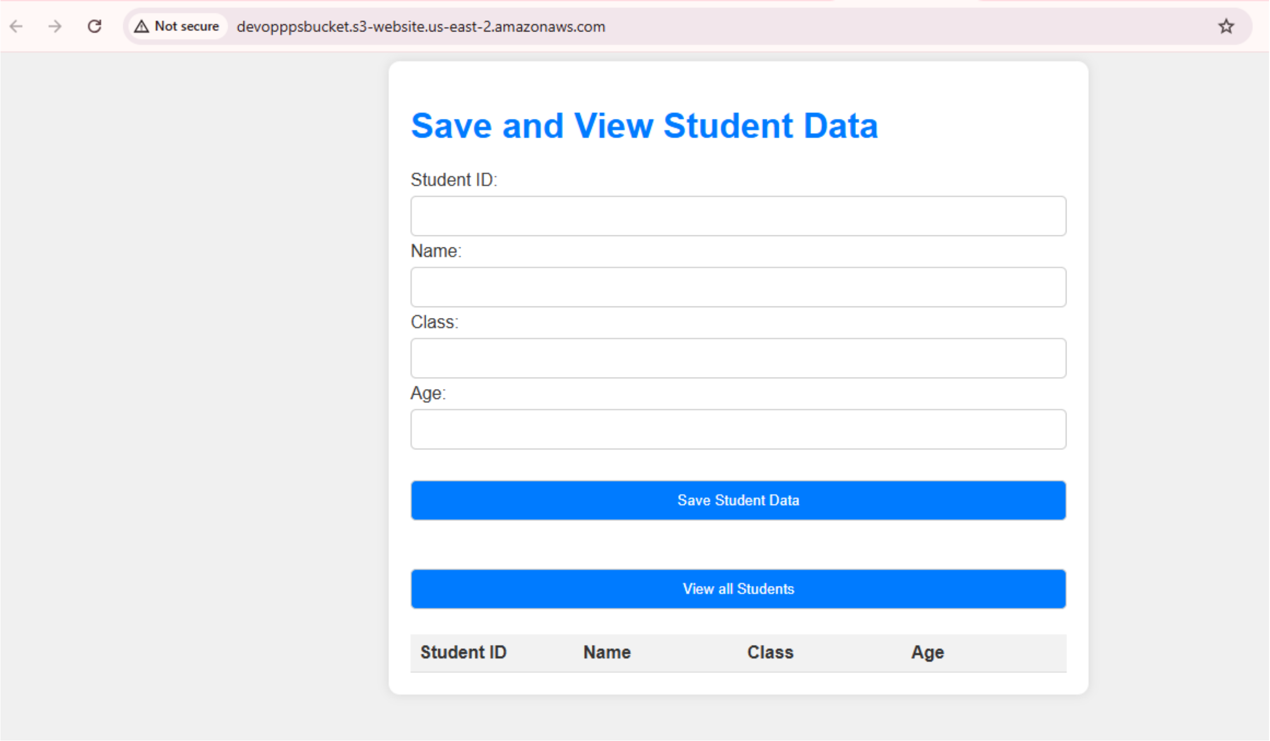
Bucket hosting

Bucket website endpoint

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. Learn more

http://devopppsbucket.s3-website.us-east-2.amazonaws.com

Your application is deployed.



Not secure devopppsbucket.s3-website.us-east-2.amazonaws.com

Save and View Student Data

Student ID:

Name:

Class:

Age:

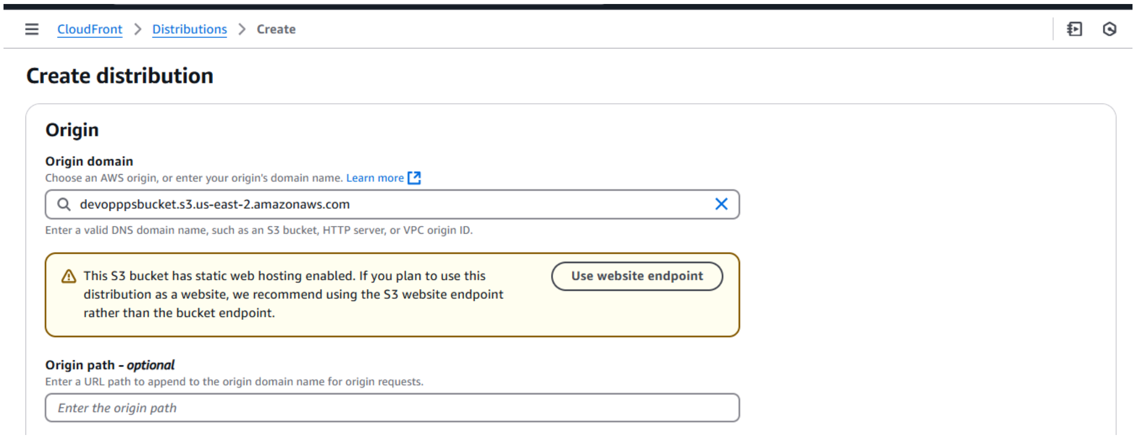
Student ID	Name	Class	Age

CloudFront would be placed in front of S3 bucket because if you check the website, it is not secure. Its only http which makes our bucket exposed to the public. We need CloudFront to make it secure.

STEP 5: SET UP CLOUDFRONT FOR SHORT URLS

CloudFront will serve as a CDN layer for the URL shortener therefore improving performance.To make the short URL load faster globally and secured, use a custom domain.

- ➢ Create a Distribution
    - Go to **CloudFront**
    - Click **Create Distribution**.
    - Select your S3 Bucket as the origin domain



Create distribution

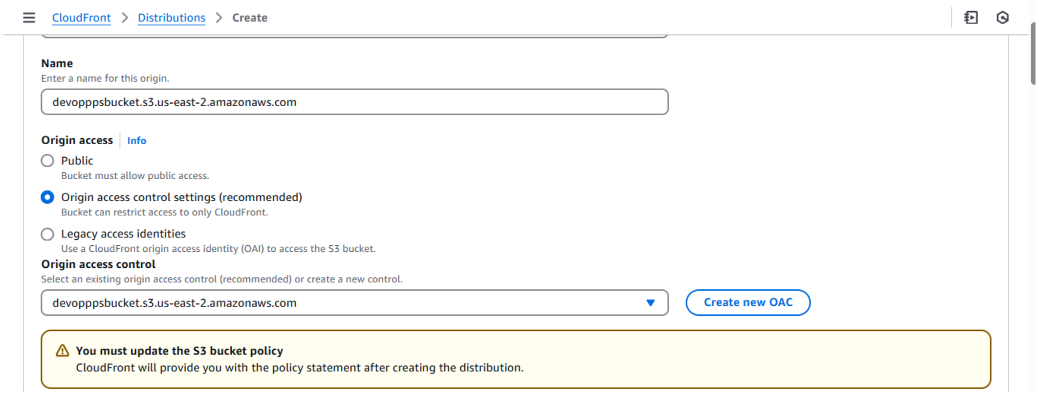
Origin

Origin domain
Choose an AWS origin, or enter your origin's domain name. Learn more
devopppsbucket.s3.us-east-2.amazonaws.com
Enter a valid DNS domain name, such as an S3 bucket, HTTP server, or VPC origin ID.

This S3 bucket has static web hosting enabled. If you plan to use this distribution as a website, we recommend using the S3 website endpoint rather than the bucket endpoint.

Origin path - *optional*
Enter a URL path to append to the origin domain name for origin requests.
Enter the origin path

- Select origin access control.



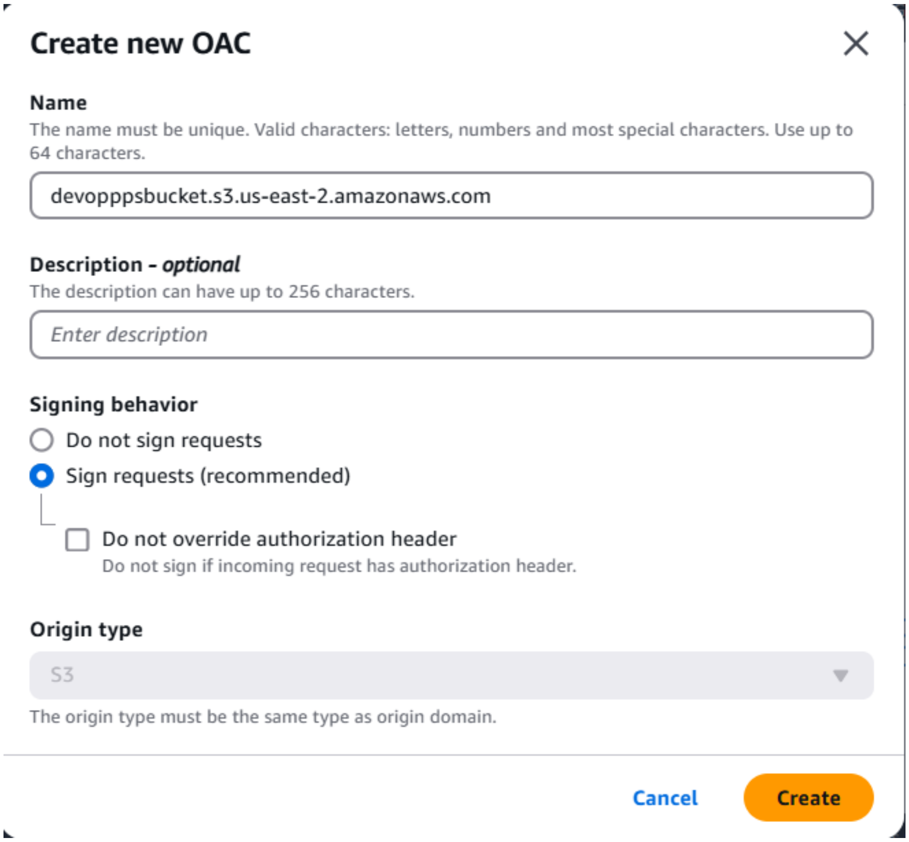
Name
Enter a name for this origin.
devopppsbucket.s3.us-east-2.amazonaws.com

Origin access
○ Public
Bucket must allow public access.
● Origin access control settings (recommended)
Bucket can restrict access to only CloudFront.
○ Legacy access identities
Use a CloudFront origin access identity (OAI) to access the S3 bucket.

Origin access control
Select an existing origin access control (recommended) or create a new control.
devopppsbucket.s3.us-east-2.amazonaws.com

You must update the S3 bucket policy
CloudFront will provide you with the policy statement after creating the distribution.

- Set a new OAC using your S3website endpoint.
- Click **Create new OAC** to create your OAC.



Create new OAC

Name

The name must be unique. Valid characters: letters, numbers and most special characters. Use up to 64 characters.

devopppsbucket.s3.us-east-2.amazonaws.com

Description - *optional*

The description can have up to 256 characters.

Enter description

Signing behavior

○ Do not sign requests

● Sign requests (recommended)

□ Do not override authorization header

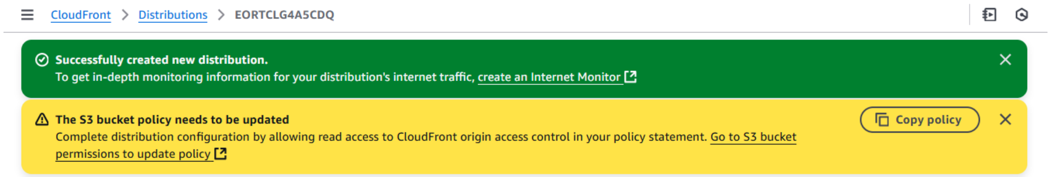
Do not sign if incoming request has authorization header.

Origin type

S3

The origin type must be the same type as origin domain.

- Scroll down to default root objectand type in **index.html**
- Check "**Do not enable security protections**"
- Click on **Create Distribution**
- Update S3 Bucket policy by copying the policy to S3

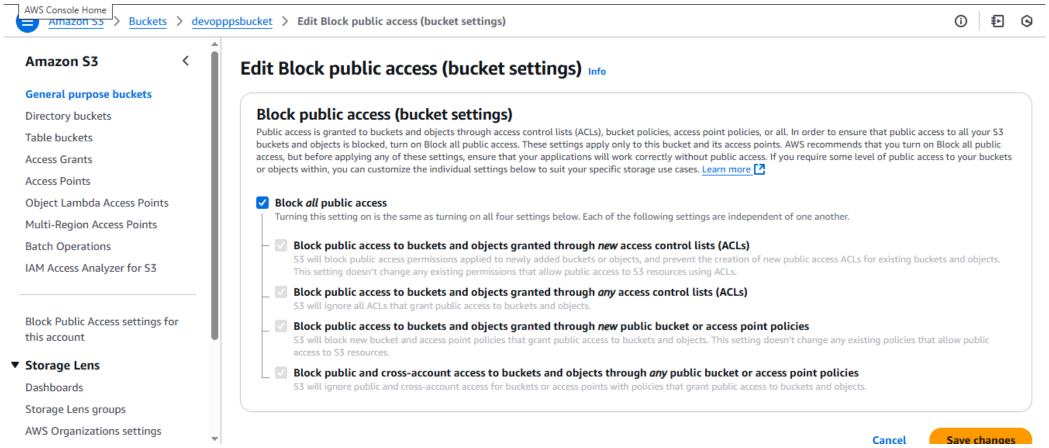


CloudFront > Distributions > EORTCLG4A5CDQ

Successfully created new distribution.
To get in-depth monitoring information for your distribution's internet traffic, create an Internet Monitor

The S3 bucket policy needs to be updated
Complete distribution configuration by allowing read access to CloudFront origin access control in your policy statement. Go to S3 bucket permissions to update policy

- Navigate to **Permissions**
- Check "**Block all public access**" to make the S3 bucket private.



Edit Block public access (bucket settings)

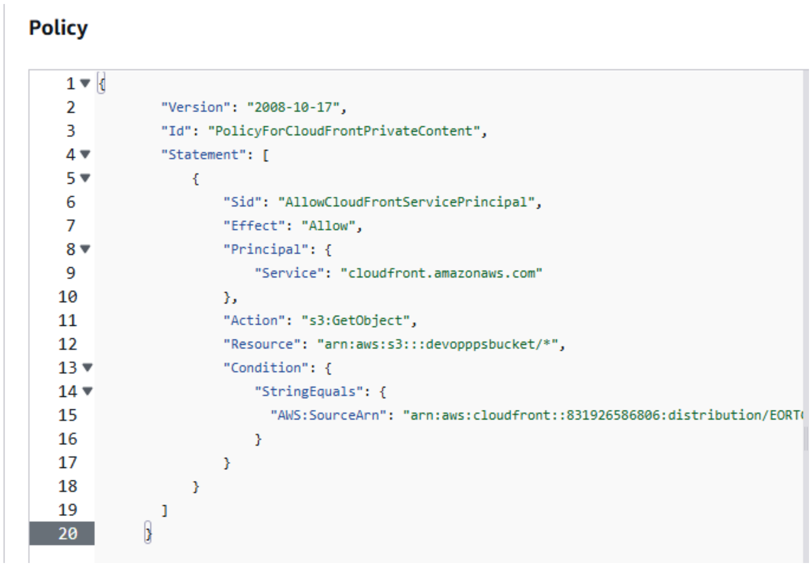
Block public access (bucket settings)

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.

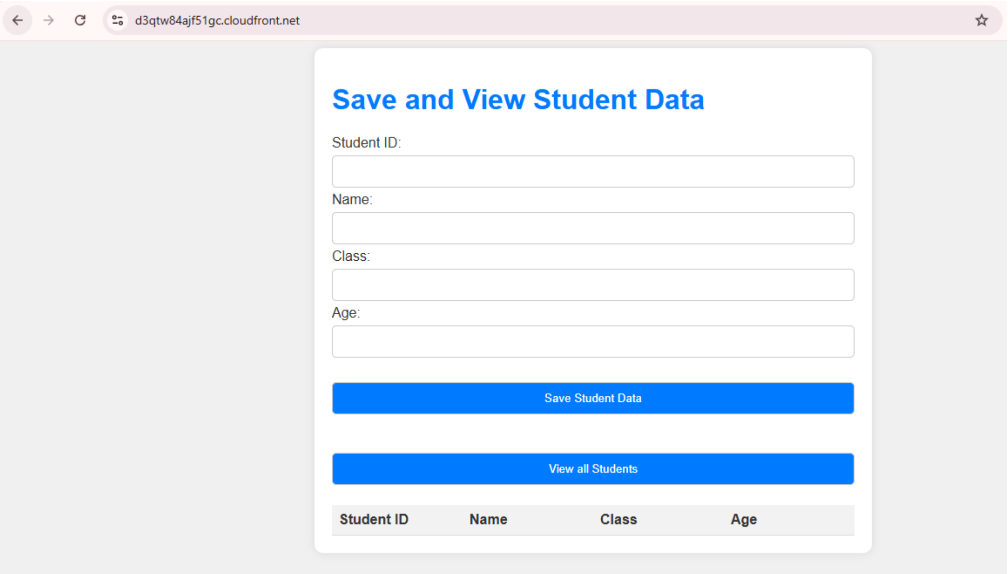
- Save changes
- Click on **Edit** policy
- Paste the policy you copied from CloudFront to access S3 Bucket.



Policy

```
{
        "Version": "2008-10-17",
        "Id": "PolicyForCloudFrontPrivateContent",
        "Statement": [
            {
                "Sid": "AllowCloudFrontServicePrincipal",
                "Effect": "Allow",
                "Principal": {
                    "Service": "cloudfront.amazonaws.com"
                },
                "Action": "s3:GetObject",
                "Resource": "arn:aws:s3:::devopppsbucket/*",
                "Condition": {
                    "StringEquals": {
                      "AWS:SourceArn": "arn:aws:cloudfront::831926586806:distribution/EORT(
                    }
                }
            }
        ]
}
```

- Save changes.
- Navigate to CloudFront and scroll down to details.
- Copy the Distribution Domain and paste on a new tab
- Refresh the page and you can access your website



STEP 6: DELETE RESOURCES

- Delete API Gateway Resources
  - Go to Amazon API Gateway in the AWS Console
  - Select your API
  - Delete the API by clicking on Actions
  - Delete API
  - Confirm the deletion
- Delete AWS Lambda Function
  - Navigate to AWS Lambda
  - Select your functions used for the URL Shortener
  - Click Actions
  - Delete and Confirm
- Delete DynamoDB Table
  - Go to DynamoDB in the AWS Console
  - Find the table used for storing short URLs
  - Click Delete Table and Confirm
- Delete Amazon S3 Bucket
  - Open Amazon S3
  - Find the Bucket you created for storing the frontend
  - Empty the Bucket first by deleting the objects.
  - Then delete the Bucket

- Remove CloudFront Distribution
    - Go to Amazon CloudFront
    - Select the distribution associated with the URL Shortener
    - Disable it first and then Delete it
    - Wait for the status to change to Deleted