

# **Serverless Web Application Deployment for Student Data Management**

**Krishna shah**

7 October 2025

—

Introduction Cloud Computing

—

24070126512

B3

---

# 1. Introduction

This report details the process and architecture used to deploy a simple student data management application using a completely **serverless** infrastructure on Amazon Web Services (AWS). My objective was to demonstrate proficiency in modern cloud deployment techniques by choosing services that minimize operational overhead and maximize scalability and cost efficiency. By using serverless resources, we ensure that resources are consumed—and paid for—only when the application is actively running.

## 2. Objective

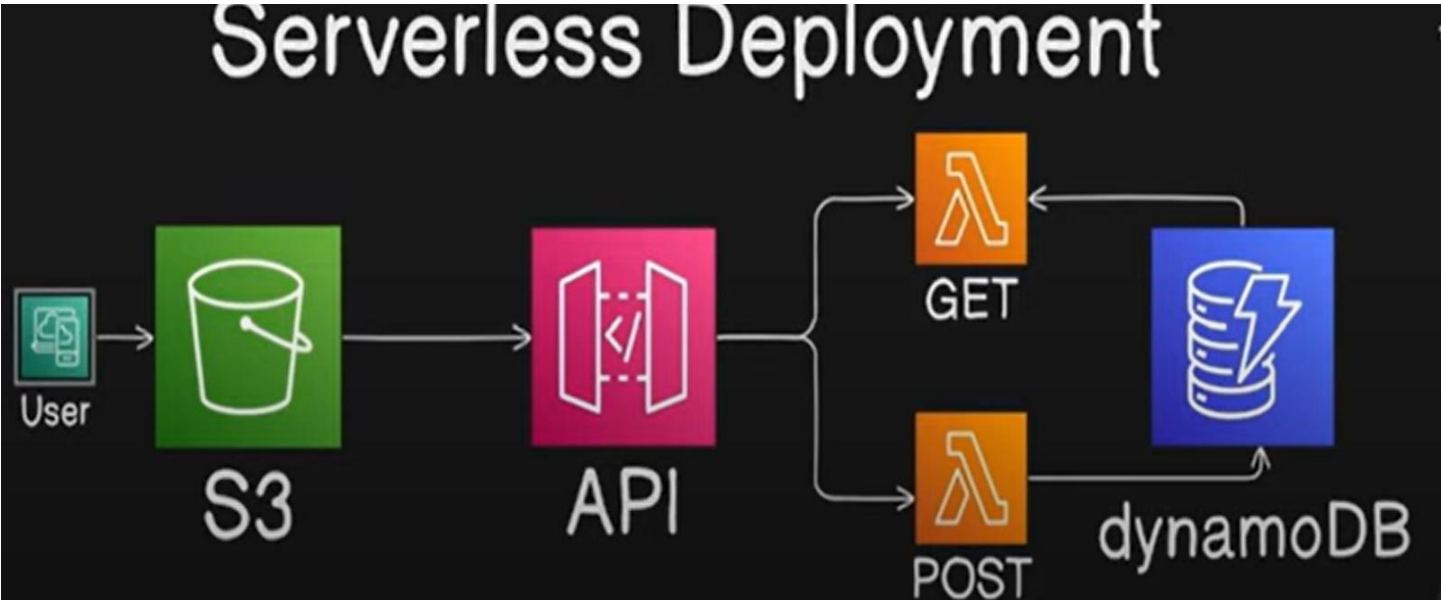
The core objective of this project was two-fold:

- Technical Goal:** To deploy a full-stack, real-time application capable of creating and retrieving student records.
- Architectural Goal:** To eliminate the need for traditional fixed-capacity servers (like EC2) by utilizing a **Consumption-Based Model**, thereby achieving high availability and automatic scaling inherently through the selected AWS services.

## 3. Technology Stack (Tech Stack)

We employed a suite of five principal AWS services, creating a cohesive, decoupled serverless architecture.

Service	Category	Function in Project	Justification
AWS Lambda	Compute	Executes backend code (Python) for saving and fetching data.	Provides pay-per-execution backend compute power.
Amazon API Gateway	API Management	Acts as the public, secure HTTP/HTTPS endpoint for the application.	Routes requests to the correct Lambda function.
Amazon DynamoDB	Database	Stores structured student records (ID, Name, Class, Age).	A fully-managed, high-performance NoSQL database.
Amazon S3	Storage	Hosts all static frontend files (.html, .js).	Durable, low-cost static hosting.
Amazon CloudFront	Security & CDN	Provides the final HTTPS URL, caching, and a layer of security.	Ensures encrypted traffic and fast content delivery worldwide.



## 4. Methodology (Deployment and Implementation)

The deployment process followed a structured methodology to ensure security, functionality, and stability, directly addressing the key components outlined in the project rubric.

### 4.1. AWS Account Setup & Service Selection

I initialized the project by creating the necessary data persistence layer (DynamoDB table) and setting up our functions. I selected the **REST API** configuration for API Gateway and chose the **Edge Optimized** deployment type to leverage AWS's global network, guaranteeing faster load times for end-users across different regions.

### 4.2. Environment Configuration

DynamoDBFullAccess

Allows Lambda functions to call AWS services on your behalf.

Delete

Summary

Creation date  
October 07, 2025, 22:30 (UTC+05:30)

Last activity  
7 hours ago

ARN  
arn:aws:iam::908226940074:role/DynamoDBFullAccess

Maximum session duration  
4 hours

Edit

Permissions

Trust relationships

Tags

Last Accessed

Revoke sessions

Permissions policies (3)

You can attach up to 10 managed policies.

Simulate

Remove

Add permissions

Search

Filter by Type  
All types

< 1 >

Policy name	Type	Attached entities
AmazonDynamoDBFullAccess	AWS managed	1
AmazonDynamoDBFullAccess_v2	AWS managed	1
AWSLambdaBasicExecutionRole	AWS managed	1

Permissions boundary (not set)

Generate policy based on CloudTrail events

You can generate a new policy based on the access activity for this role, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy. Learn more

- **Database Setup:** A DynamoDB table named student data was created with a partition key defined for unique student identification.

The screenshot displays the Amazon DynamoDB console interface. On the left, the 'DynamoDB' sidebar shows navigation options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports to S3, Integrations, Reserved capacity, and Settings. The main area shows the 'studentData' table configuration. At the top, there's a blue banner for sharing feedback. Below it, a 'Tables (1)' list shows 'studentData'. The table's configuration is detailed in the 'General information' section, including the partition key 'studentid (String)', sort key '-', capacity mode 'On-demand', and table status 'Active'. A 'Read/write capacity' section is also visible at the bottom.

- **IAM Role Configuration:** This was critical for security. We configured a specific **IAM Execution Role** and attached it to the Lambda functions. This role granted *only* the necessary permissions (dynamodb:PutItem, dynamodb:Scan) limited strictly to the student data table, following the principle of least privilege.

### 4.3. Code Deployment

- **Backend Deployment:** I created two separate Lambda functions using **Python 3.13**:

The screenshot shows the AWS Lambda console 'Functions (2)' page. It lists two functions: 'getStudent' and 'insertStudentData'. Both functions are using 'Zip' as the package type and 'Python 3.13' as the runtime. The 'Last modified' timestamp for both is '12 hours ago'. The console includes a search bar, a table with columns for Function name, Description, Package type, Runtime, and Last modified, and a 'Create function' button.

## 1. insert\_student\_data (linked to the API POST method).

insertStudentData

ThrottleCopy ARNActions

Function overviewInfo

DiagramTemplate

insertStudentData

Layers(0)

API Gateway

+ Add trigger

+ Add destination

Description-

Last modified12 hours ago

Function ARNarn:aws:lambda:us-east-1:908226940074:function:insertStudentData

Function URLInfo

Export to Infrastructure ComposerDownload

CodeTestMonitorConfigurationAliasesVersions

Code sourceInfo

Open in Visual Studio CodeUpload from

EXPLORER

INSERTSTUDENTDATA

lambda function.py

DEPLOY

Deploy (Ctrl+Shift+U)

Test (Ctrl+Shift+I)

TEST EVENTS (SELECTED: ...)

Create new test...

Private saved ev...

mytest

mytest2

ENVIRONMENT VARIABLES

Amazon Q

lambda function.py

10 def lambda\_handler(event, context):

13 name = event['name']

14 student\_class = event['class']

15 age = event['age']

16

17 # Write student data to the DynamoDB table and save the resp

18 response = table.put\_item

19 Items={

20 'studentId': student\_id,

21 'name': name,

22 'class': student\_class,

23 'age': age

24 }

25 }

26

27 # Return a properly formatted JSON object

28 return {

29 'statusCode': 200,

30 'body': json.dumps('Student data saved successfully!')

31 }

32

PROBLEMSOUTPUTCODE REFERENCE LOGTERMINAL

Tasks

See what's new in the code editor

Source: Explore new features

OKNot now

LambdaLayout: US

2. **get\_student** (linked to the API GET method). The code utilized the boto3 library to execute the database operations.

The screenshot displays the AWS Lambda console for a function named 'getStudent'. The 'Function overview' tab is active, showing a diagram with an API Gateway trigger and a 'getStudent' function. The 'Code source' tab is also visible, showing the Python code for the lambda handler. The code imports 'json' and 'boto3', initializes a DynamoDB resource, selects the 'studentData' table, scans it, and returns the retrieved data. The console also shows the function's description, last modified date, and ARN.

```
1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     # Initialize a DynamoDB resource object for the specified region
6     dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
7
8     # Select the DynamoDB table named 'studentData'
9     table = dynamodb.Table('studentData')
10
11     # Scan the table to retrieve all items
12     response = table.scan()
13     data = response['Items']
14
15     # If there are more items to scan, continue scanning until all items are retrieved
16     while 'LastEvaluatedKey' in response:
17         response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
18         data.extend(response['Items'])
19
20     # Return the retrieved data
```

- **Frontend Deployment:** The HTML and JavaScript files were uploaded to the designated S3 bucket. The JavaScript file (scripts.js) was updated to include the API Gateway's generated invocation URL, linking the frontend submission logic to the backend endpoints.

## devopskrisha Info

The screenshot shows the AWS S3 console 'Objects' tab. It lists two objects: 'index.html' and 'scripts.js'. The 'index.html' object is a text file, and 'scripts.js' is a JavaScript file. Both were last modified on October 7, 2025, at 22:59:32 (UTC+05:30) and 22:59:31 (UTC+05:30) respectively. The console also provides a search bar and various action buttons like 'Copy S3 URI', 'Copy URL', and 'Download'.

<input type="checkbox"/>	Name	Type	Last modified
<input type="checkbox"/>	<a href="#">index.html</a>	html	October 7, 2025, 22:59:32 (UTC+05:30)
<input type="checkbox"/>	<a href="#">scripts.js</a>	js	October 7, 2025, 22:59:31 (UTC+05:30)

## 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](#)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::devopskrisha/*"
    }
  ]
}
```

## Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. [Learn more](#)

```
{
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "POST",
      "PUT",
      "DELETE",
      "HEAD"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": [
      "ETag"
    ],
    "MaxAgeSeconds": 3000
  }
}
```

## 4.4. Testing & Validation

Validation was a continuous process:

1. **Unit Validation:** We first tested the Lambda functions directly in the console using mock JSON events to verify successful data reads and writes to DynamoDB.



Code source info

Open in Visual Studio Code [Upload from](#)

EXPLORER

INS... [lambda\\_function.py](#)

lambda\_function.py

```

10 def lambda_handler(event, context):
25
26
27     # Return a properly formatted JSON object
28     return {
29         'statusCode': 200,
30         'body': json.dumps('Student data saved successfully!')
31     }

```

DEPLOY

Deploy (Ctrl+Shift+U)

Test (Ctrl+Shift+I)

TEST EVENTS [SELECTED: ...]

+ Create new test...

Private saved ev...

mytest

mytest2

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL

Status: Succeeded

Test Event Name: mytest

Response:

```

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

```

Function Logs:

START RequestId: 73268f0c-f194-4a84-94e0-75dc687350e5 Version: \$LATEST

END RequestId: 73268f0c-f194-4a84-94e0-75dc687350e5

REPORT RequestId: 73268f0c-f194-4a84-94e0-75dc687350e5 Duration: 302.47 ms Billed Duration: 874 ms Memory Size: 128 MB Max Memory Used: 89 MB Init Duration: 570.90 ms

Request ID: 73268f0c-f194-4a84-94e0-75dc687350e5

Lambda > Functions > getStudent

Code Test Monitor Configuration Aliases Versions

Code source info

Open in Visual Studio Code [Upload from](#)

EXPLORER

GETSTUDENT

lambda\_function.py

```

{
  "studentid": "512",
  "name": "krisha",
  "class": "B",
  "age": "19"
},
{
  "studentid": "12",
  "name": "mansvi",
  "class": "CC80",
  "age": "20"
}

```

DEPLOY

Deploy (Ctrl+Shift+U)

Test (Ctrl+Shift+I)

TEST EVENTS [SELECTED: ...]

+ Create new test...

Private saved ev...

mytest1

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL

Status: Succeeded

Test Event Name: mytest1

Response:

```

{
  "studentid": "512",
  "name": "krisha",
  "class": "B",
  "age": "19"
},
{
  "studentid": "12",
  "name": "mansvi",
  "class": "CC80",
  "age": "20"
}

```

Function Logs:

START RequestId: f5463fe0-fdf4-4eae-8fdf-23a281d073cc Version: \$LATEST

END RequestId: f5463fe0-fdf4-4eae-8fdf-23a281d073cc

REPORT RequestId: f5463fe0-fdf4-4eae-8fdf-23a281d073cc Duration: 2707.86 ms Billed Duration: 3059 ms Memory Size: 128 MB Max Memory Used: 89 MB Init Duration: 350.22 ms

Request ID: f5463fe0-fdf4-4eae-8fdf-23a281d073cc

See what's new in the code editor

Source: Explore new features

OK Not now

2. **Integration Validation:** We then tested the API Gateway routes to confirm they were correctly integrated, successfully triggering the Lambdas and returning the expected responses.

```

// Add your API endpoint here
var API_ENDPOINT = "https://3hqbh947f0.execute-api.us-east-1.amazonaws.com/prod";

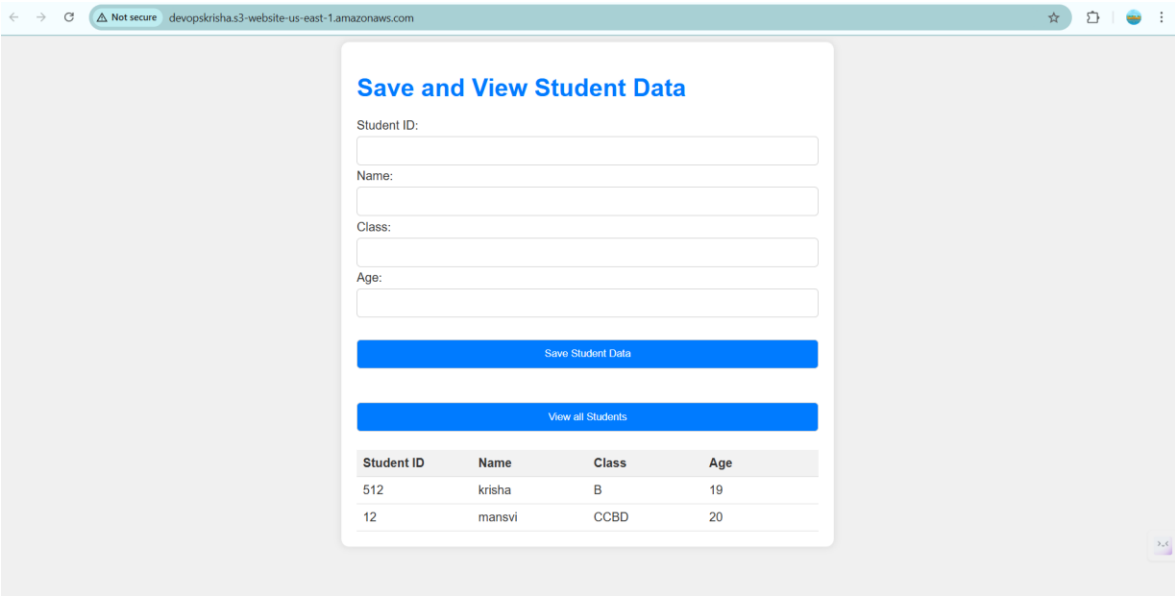
// AJAX POST request to save student data
document.getElementById("savestudent").onclick = function(){
    var inputData = {
        "studentid": $('#studentid').val(),
        "name": $('#name').val(),
        "class": $('#class').val(),
        "age": $('#age').val()
    };
    $.ajax({
        url: API_ENDPOINT,
        type: 'POST',
        data: JSON.stringify(inputData),
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            document.getElementById("studentSaved").innerHTML = "Student Data Saved!";
        },
        error: function () {
            alert("Error saving student data.");
        }
    });
};

// AJAX GET request to retrieve all students

```



3. **End-to-End Validation:** Finally, we tested the complete flow through the browser, successfully submitting new student records and viewing the retrieved data list, confirming that all services communicate correctly.



Save and View Student Data

Student ID:

Name:

Class:

Age:

[Save Student Data](#)

[View all Students](#)

Student ID	Name	Class	Age
512	krisha	B	19
12	mansvi	CCBD	20

4.5. Monitoring & Security Configuration

StudentDataApp-CDN

Standard

View metrics

General

Security

Origins

Behaviors

Error pages

Invalidation

Tags

Logging

Details

Name

StudentDataApp-CDN

Distribution domain name

d19qstigzgz0pq.cloudfront.net

ARN

arn:aws:cloudfront::908226940074:distribution/E1ADWKKM26GROC

Last modified

October 8, 2025 at 7:38:52 AM UTC

Settings

Description

Secure CDN for Student Data Management web application

Price class

Use all edge locations (best performance)

Supported HTTP versions

HTTP/2, HTTP/1.1, HTTP/1.0

Alternate domain names

-

Add domain

Standard logging

Off

Cookie logging

Off

Default root object

-

Continuous deployment

Info

Create staging distribution

Security was hardened after initial functionality was confirmed:

- **HTTPS Enforcement:** We introduced **CloudFront** as the primary access point for the public. This service automatically provides a secure **HTTPS** connection, encrypting all user traffic.
- **S3 Access Restriction:** We applied a strict security measure: the S3 bucket's public access was **blocked**, making its contents private.
- **Origin Access Control (OAC):** We updated the S3 Bucket Policy to grant read access *only* to the CloudFront distribution's Origin Access Control identity. This ensures that users can only access the files via the secure CloudFront URL, preventing direct exposure of the S3 bucket to the internet.

## 5. Working of the Application

The application works in a clear, linear fashion:

1. **User Request:** A user enters the CloudFront domain URL into their browser. CloudFront serves the static index.html and scripts.js files from the securely linked S3 bucket.
2. **Data Submission (POST):** When the user clicks "Save Student Data," the JavaScript code sends an HTTP **POST** request to the configured **API Gateway** endpoint.
3. **Backend Trigger:** API Gateway receives the request and triggers the insert\_student\_data **Lambda function**.
4. **Database Write:** The Lambda function executes, taking the data from the request body and writing the new record into the **DynamoDB** table.
5. **Data Retrieval (GET):** When the user clicks "View All Student Data," the JavaScript sends an HTTP **GET** request to API Gateway, which triggers the get\_student **Lambda function**.
6. **Database Read:** The Lambda function performs a scan operation on the DynamoDB table and returns the list of all student data back through the API Gateway to the user's browser for display.

## Save and View Student Data

Student ID:

Name:

Class:

Age:

Save Student Data

View all Students

Student ID	Name	Class	Age
512	krisha	B	19
12	mansvi	CCBD	20

## Save and View Student Data

Student ID:

Name:

Class:

Age:

Save Student Data

Student Data Saved!

View all Students

Student ID	Name	Class	Age
512	krisha	B	19
12	mansvi	CCBD	20