Serverless Web Application Deployment for Student Data Management

Krisha 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

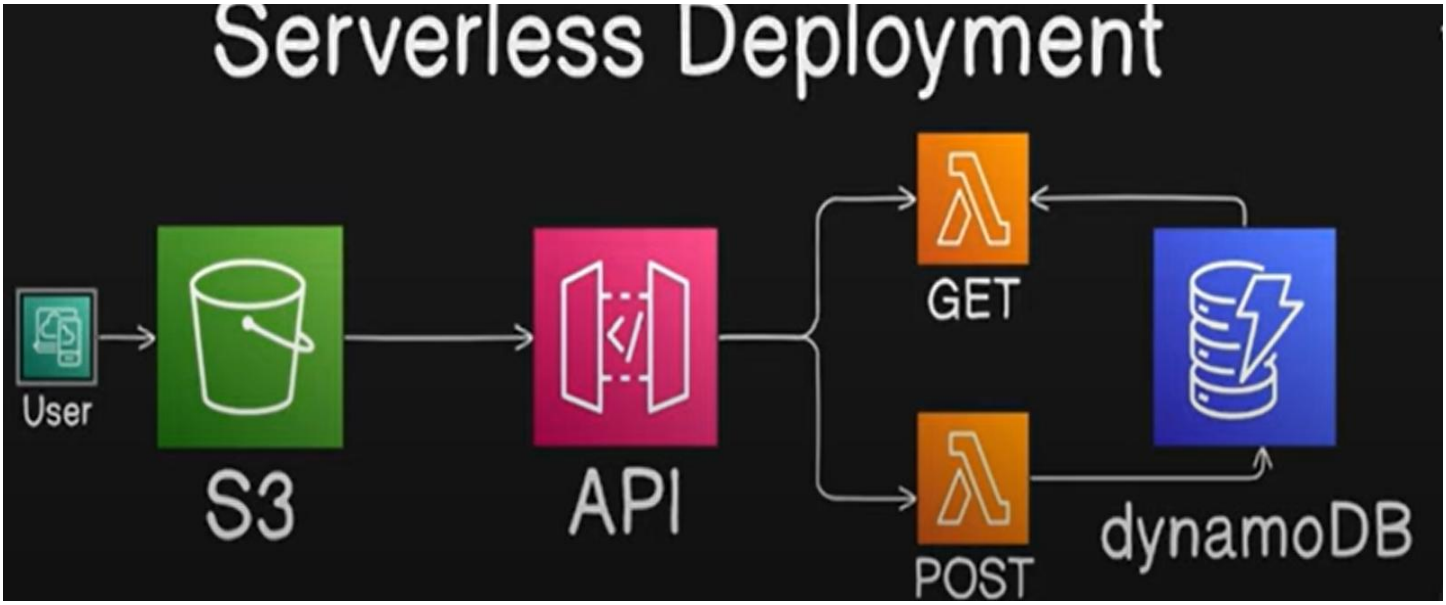
The core objective of this project was two-fold:

1. **Technical Goal:** To deploy a full-stack, real-time application capable of creating and retrieving student records.
2. **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.



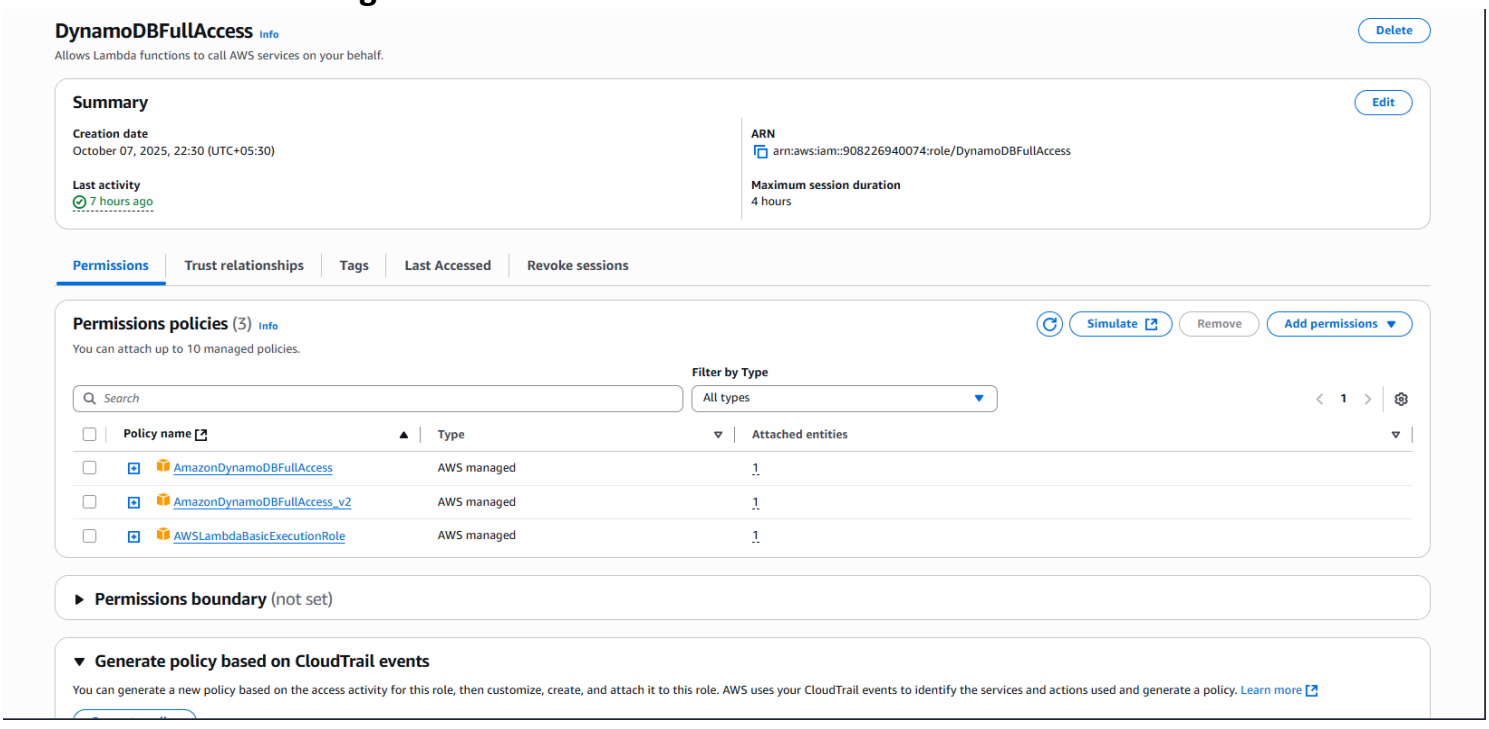
4. Methodology (Deployment and Implementation)

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

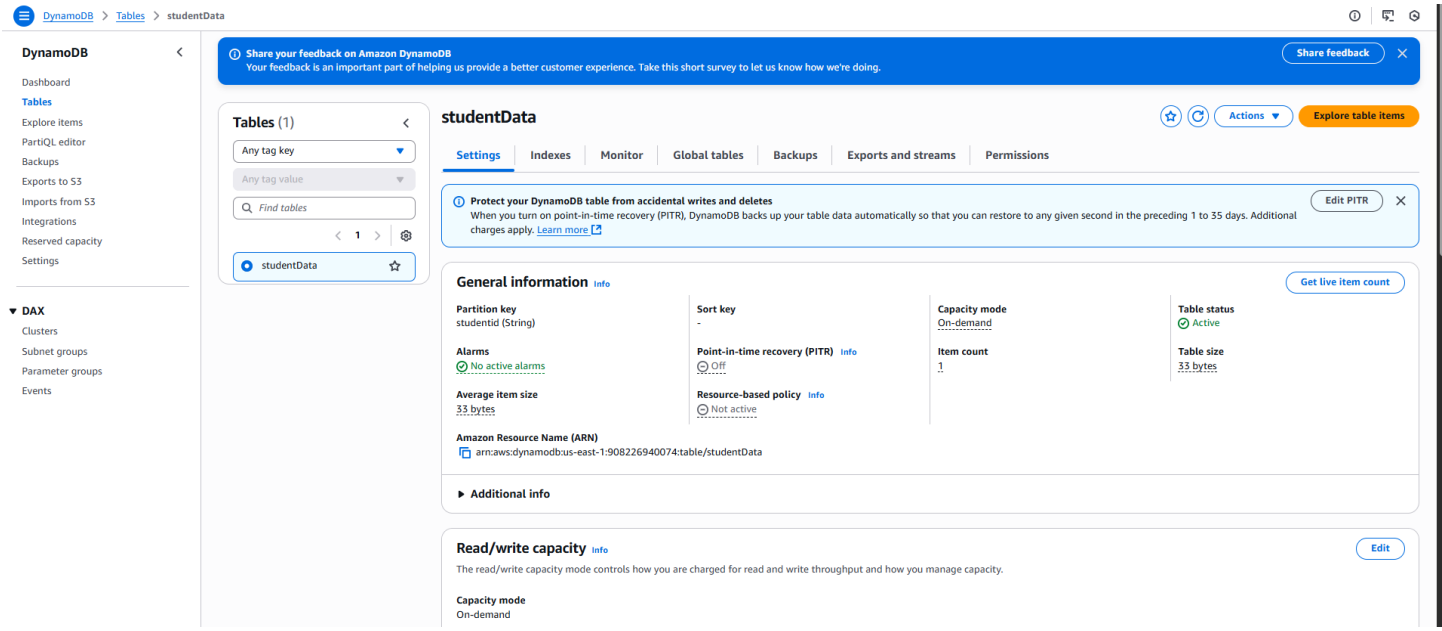
4.1. AWS Account Setup & Service Selection

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



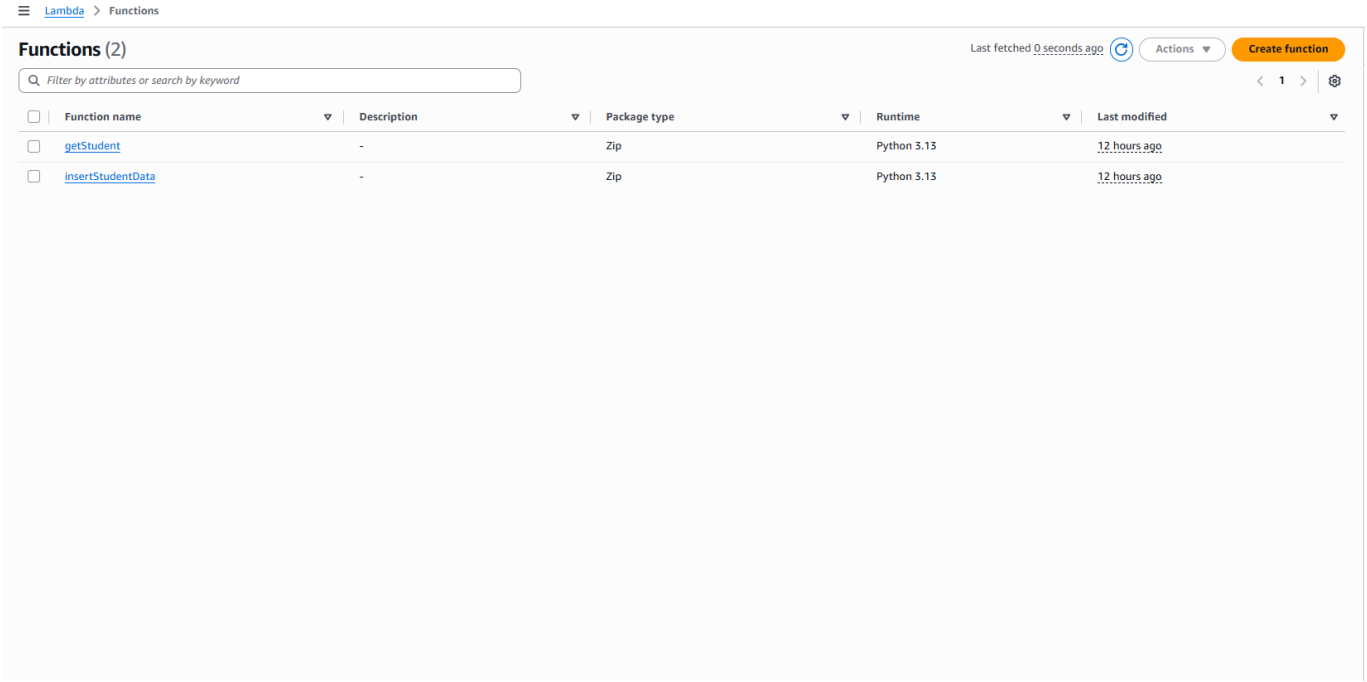
- **Database Setup:** A DynamoDB table named student data was created with a partition key defined for unique student identification.



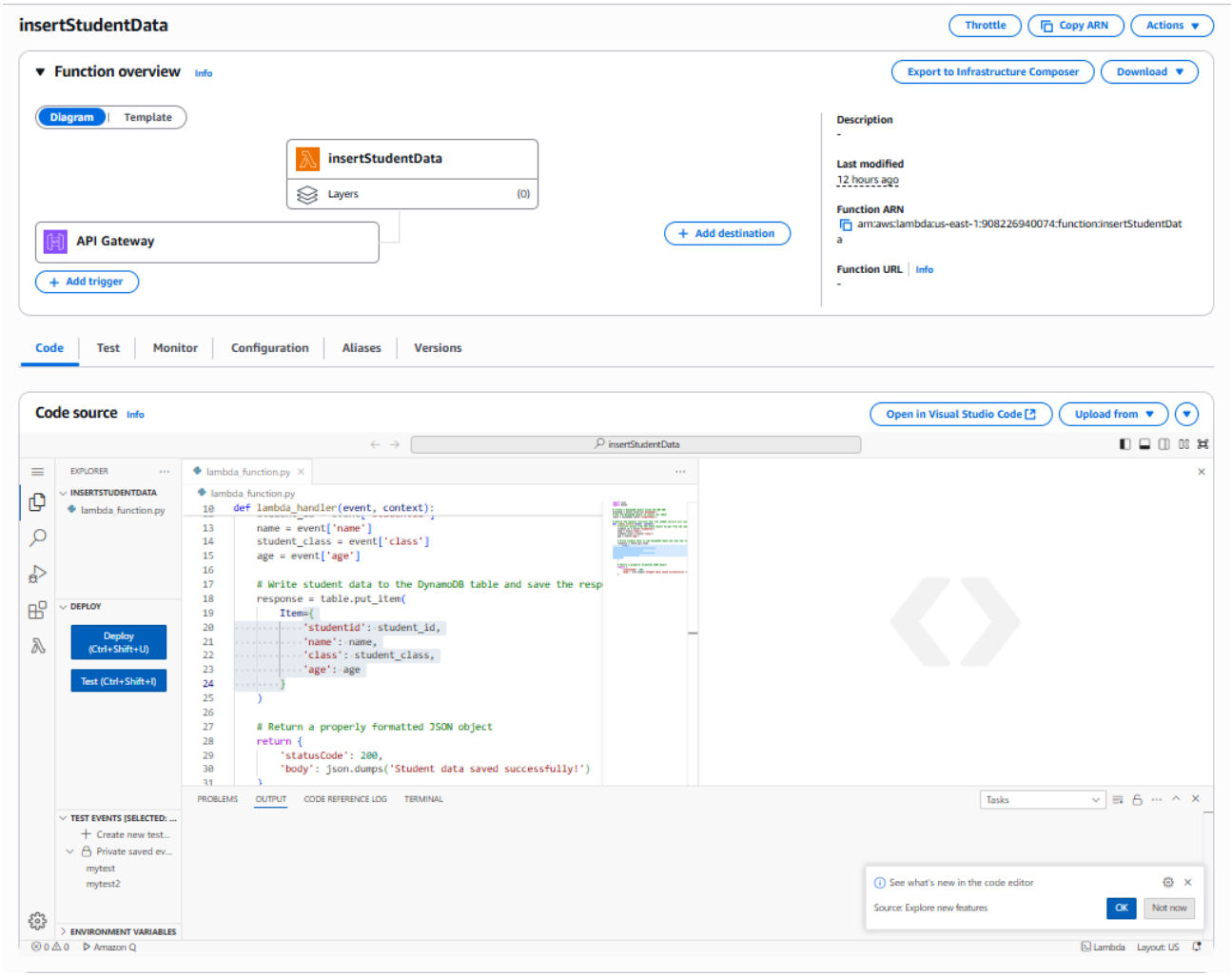
- **IAM Role Configuration:** This was critical for security. We configured a specific **IAM Execution Role** and attached it to the Lambda functions. This role granted *only* the necessary permissions (dynamodb:PutItem, dynamodb:Scan) limited strictly to the student data table, following the principle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

4.3. Code Deployment

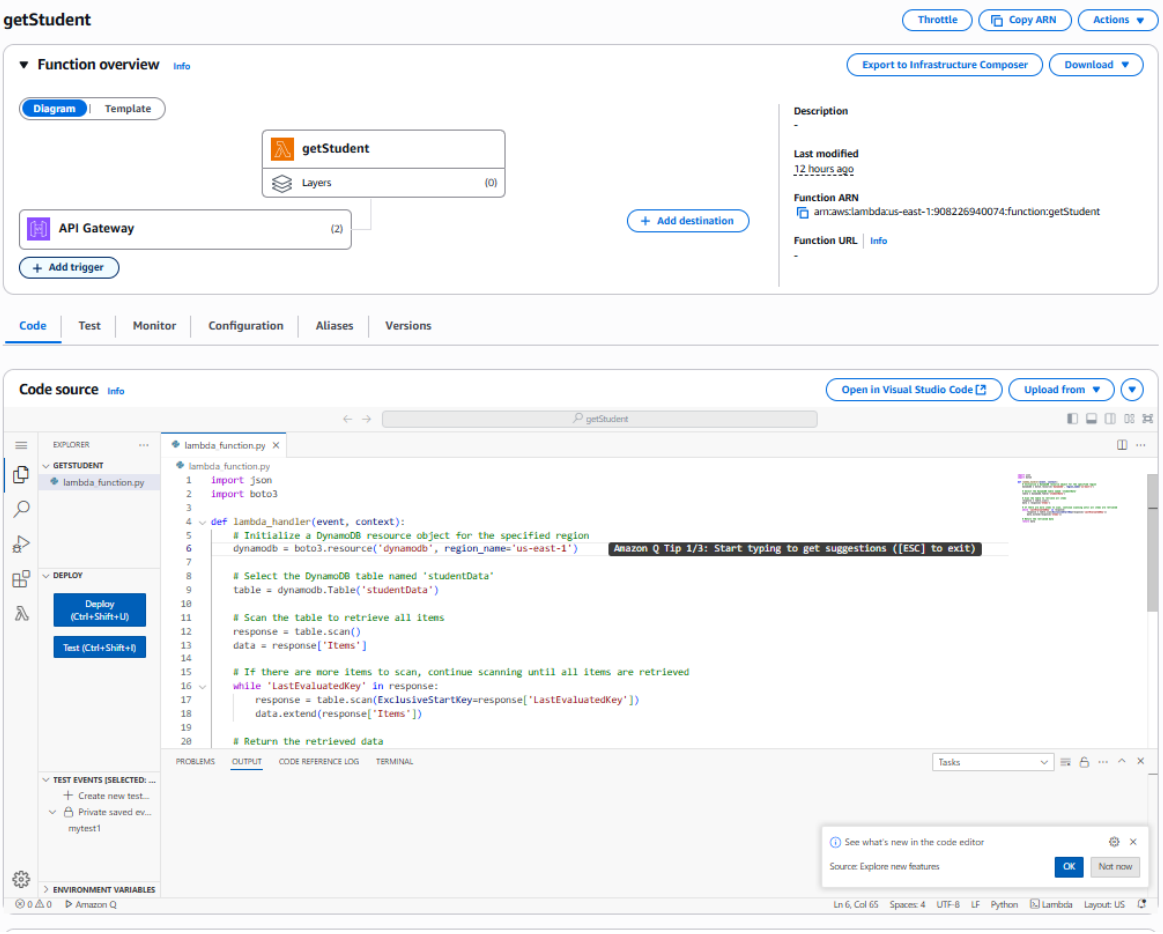
- **Backend Deployment:** I created two separate Lambda functions using **Python 3.13**:



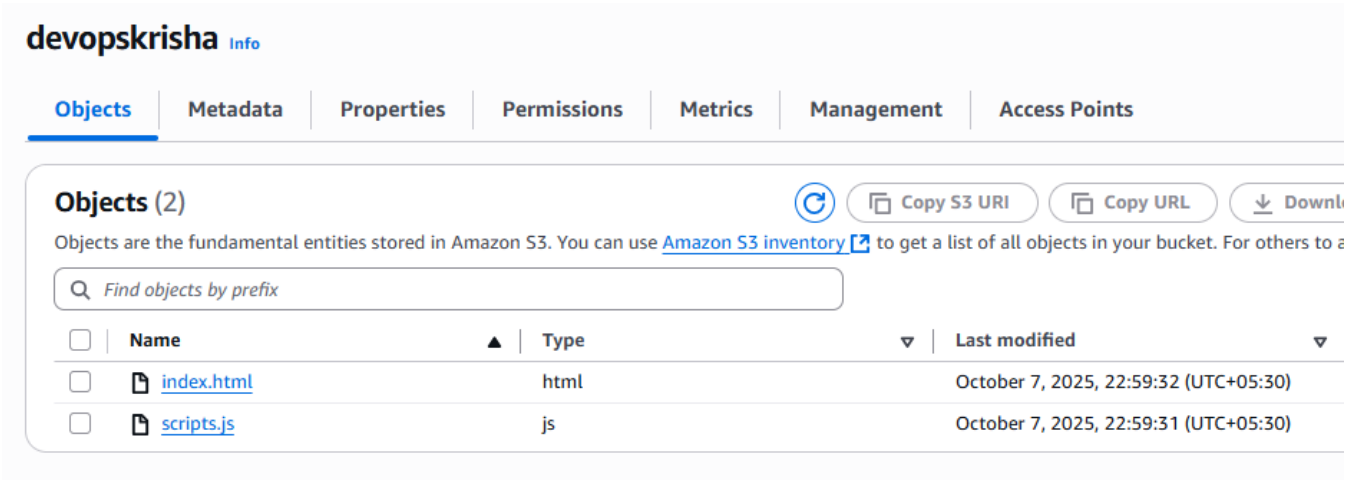
1. **insert_student_data** (linked to the API POST method).



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



- **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.



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)

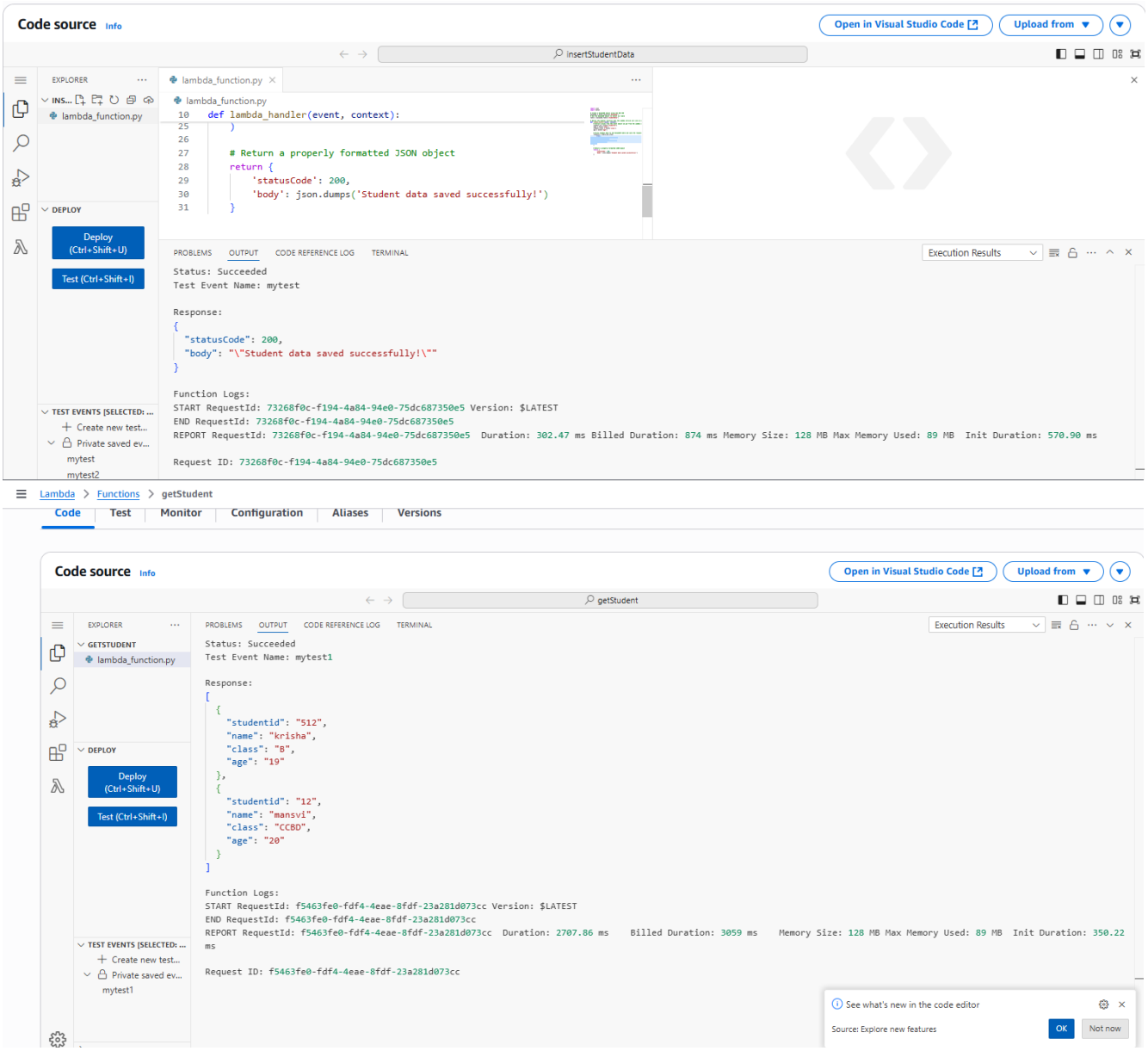
The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. Learn more

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

4.4. Testing & Validation

Validation was a continuous process:

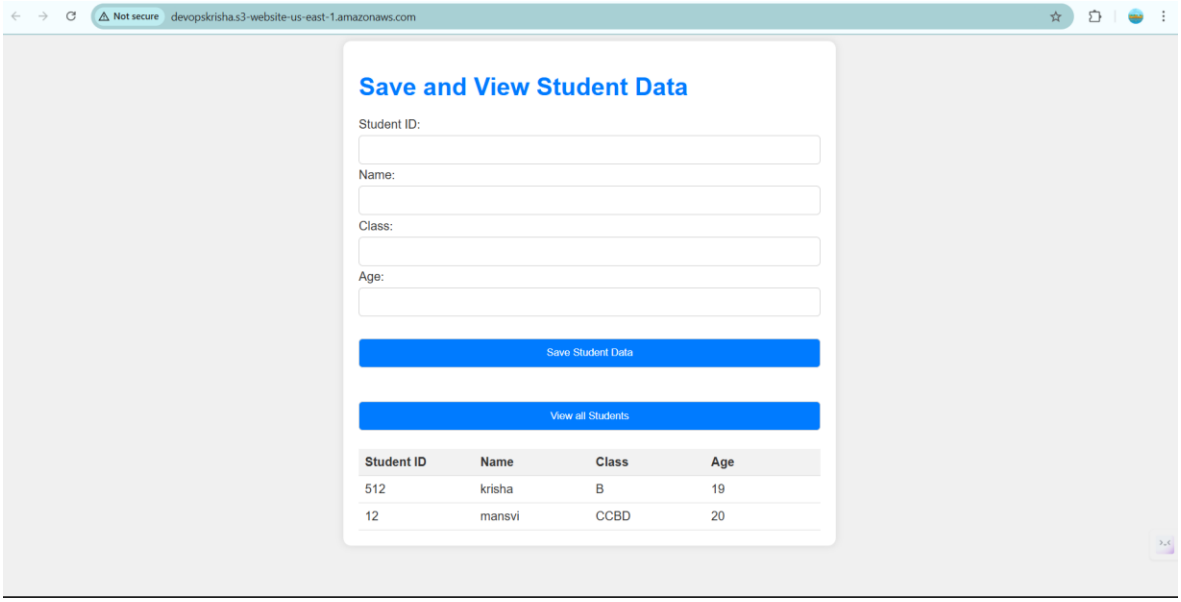
1. **Unit Validation:** We first tested the Lambda functions directly in the console using mock JSON events to verify successful data reads and writes to DynamoDB.



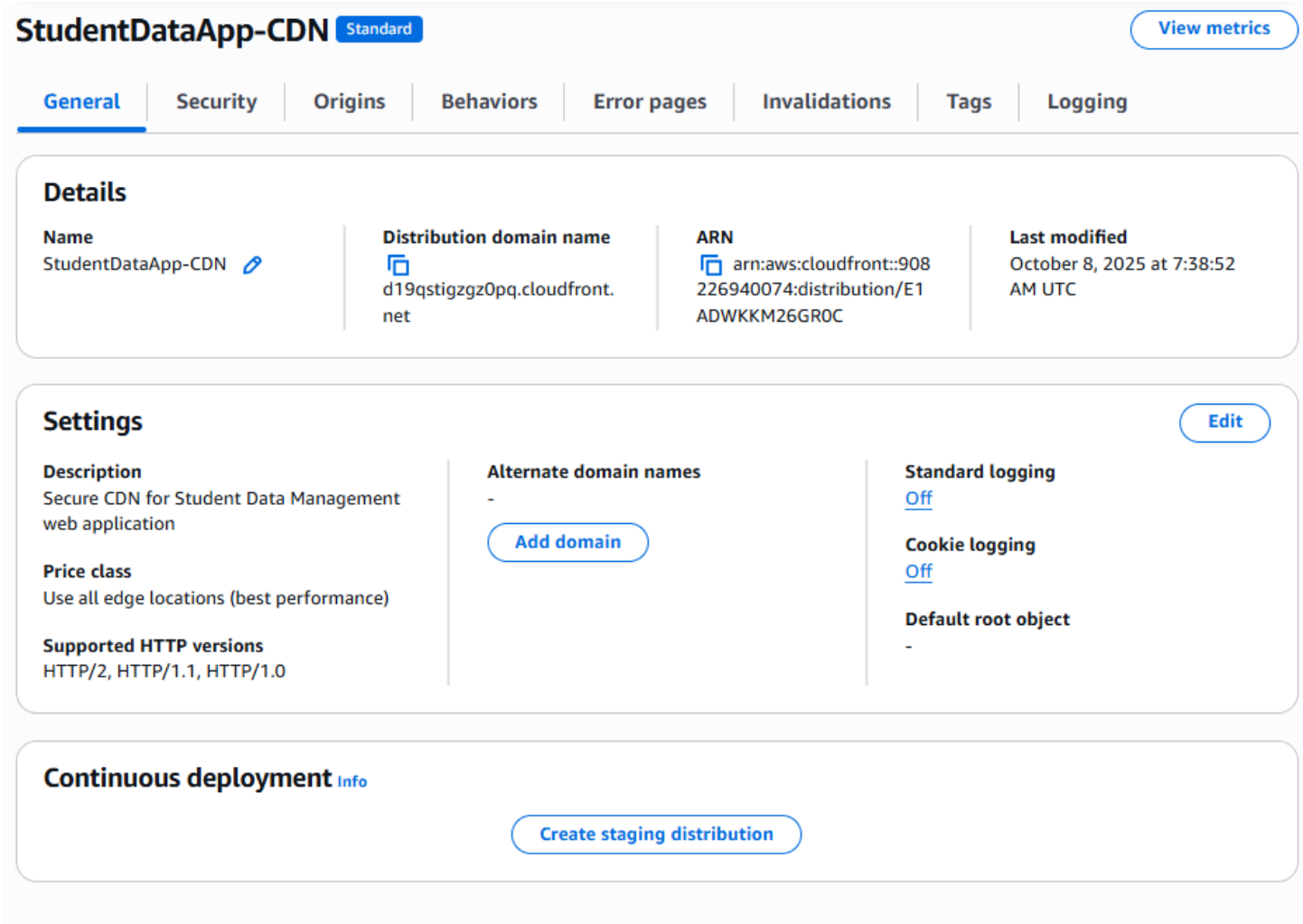
2. **Integration Validation:** We then tested the API Gateway routes to confirm they were correctly integrated, successfully triggering the Lambdas and returning the expected responses.

```
// 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.



4.5. Monitoring & Security Configuration



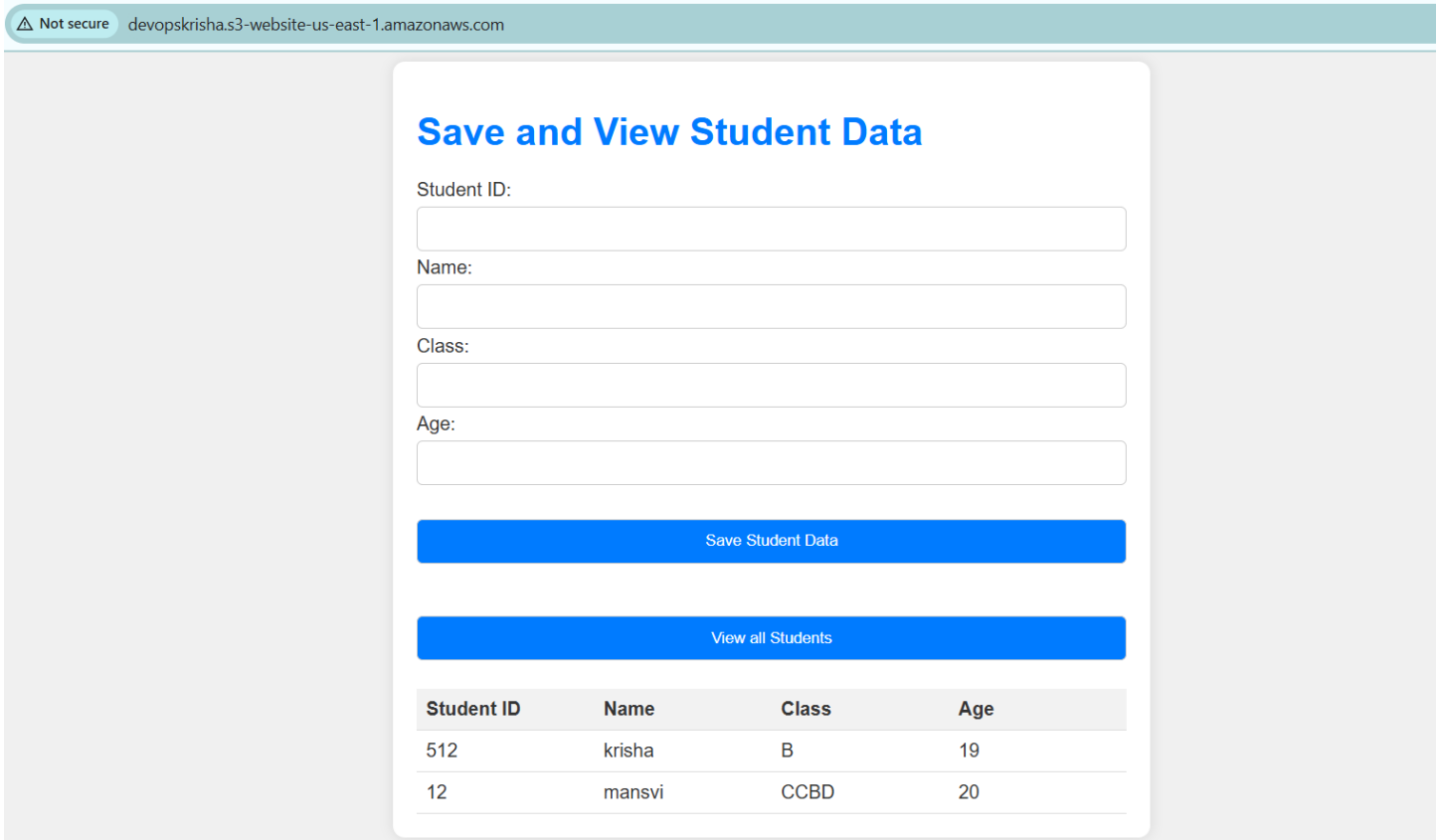
Security was hardened after initial functionality was confirmed:

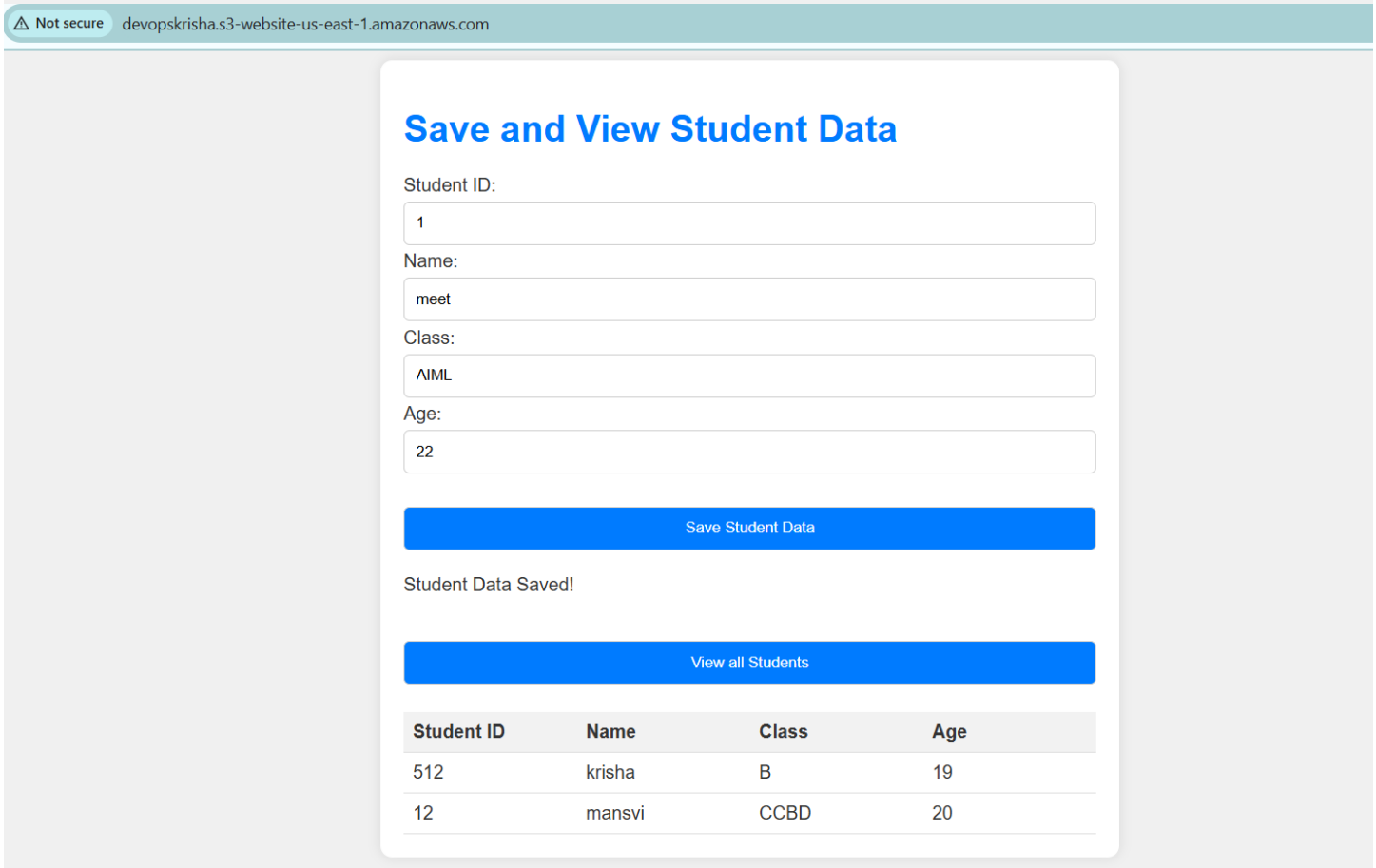
- **HTTPS Enforcement:** We introduced **CloudFront** as the primary access point for the public. This service automatically provides a secure **HTTPS** connection, encrypting all user traffic.
- **S3 Access Restriction:** We applied a strict security measure: the S3 bucket's public access was **blocked**, 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.


Not secure
devopskrisha.s3-website-us-east-1.amazonaws.com
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


Not secure
devopskrisha.s3-website-us-east-1.amazonaws.com
Save and View Student Data
Student ID:
1
Name:
meet
Class:
AIML
Age:
22
Save Student Data
Student Data Saved!
View all Students
Student ID
Name
Class
Age
512
krisha
B
19
12
mansvi
CCBD
20