# Serverless Web Application Deployment using AWS

**TEAM MEMBERS:**

P.Prudhvi Sai-23BIT0429

B.K.Pranavi -23BIT0314

**TITLE:** Serverless Web Application Deployment

## DESCRIPTION:

Managing student information is a common requirement for educational institutions. Traditional web applications often rely on dedicated servers, which require continuous management, scaling, and maintenance, leading to higher operational costs and complexity. This project addresses the need for a modern, cost-effective, and scalable solution for managing student data. By leveraging a serverless three-tier architecture on AWS, we can build a system that is resilient, scales automatically, and operates on a pay-as-you-go model, minimizing idle costs.

## OUTCOMES:

- The web application successfully performs data storage and retrieval operations.
- The frontend is hosted on AWS S3 and accessible via a CloudFront HTTPS URL.
- The application scales automatically without any manual server management.
- Demonstrates cost-efficient and secure deployment using AWS's serverless ecosystem.

## PROBLEM STATEMENT:

Traditional web applications rely on dedicated servers for hosting and backend processing, which require continuous management, scaling, and maintenance.
This leads to higher operational costs and increased complexity.

There is a need for a modern, cost-effective, and scalable solution that allows developers to focus on functionality rather than infrastructure.

This project addresses that challenge by implementing a **Serverless Web Application Deployment using AWS**, where backend logic, data storage, and hosting are fully managed by AWS services. The aim is to develop a **student data management system** that scales automatically, minimizes overhead, and operates on a pay-as-you-use model.

## KEY TECHNOLOGIES USED:

| AWS Service | Purpose in this Project |
|---|---|
| Amazon S3 | Hosts the static frontend website (HTML, CSS, JavaScript) and serves as the origin for CloudFront. |
| Amazon DynamoDB | Fully managed NoSQL database used to store the studentData items. |
| AWS Lambda | Provides the serverless compute logic (Python/boto3) for the getStudent and insertStudentData functions. |
| Amazon API Gateway | Creates and manages the RESTful API endpoints (GET, POST) that trigger the Lambda functions. |
| Amazon CloudFront | (Enhancement) Acts as a CDN to securely and quickly deliver the S3 website to users worldwide. |
| IAM | Manages permissions and roles, specifically the LambdaDynamoDb-Role which allows Lambda to access DynamoDB. |
| Programming Language | Python (used for developing AWS Lambda functions) |
| Frontend Technologies | HTML, CSS, and JavaScript (used to design and control the web interface) |
| Data Format | JSON (used for sending and receiving data between frontend, API Gateway, and Lambda functions) |

| | |
|---|---|
| Web Browser | Google Chrome (used to run and test the deployed web application) |

## OVERVIEW:

- The project focuses on deploying a **serverless web application** using **Amazon Web Services (AWS)**.
- It demonstrates how to build and host a **student data management system** without managing servers.
- The application allows users to **add** and **view** student details such as ID, name, class, and age.
- **AWS Lambda** handles backend logic for inserting and retrieving data.
- **Amazon DynamoDB** stores all student records in a NoSQL format.
- **Amazon API Gateway** connects the frontend to backend Lambda functions.
- **Amazon S3** hosts the static frontend files (HTML, CSS, JavaScript).
- **Amazon CloudFront** provides global HTTPS access and improved security.
- The system ensures **automatic scalability**, **cost-efficiency**, and **simplified deployment**.
- The project demonstrates a complete **end-to-end serverless architecture** for modern web development.

## OBJECTIVES:

- To design and deploy a **serverless web application** using Amazon Web Services (AWS).
- To develop a **student data management system** that allows adding and viewing student details.
- To implement **AWS Lambda functions** for backend processing without using traditional servers.
- To integrate **Amazon API Gateway** with Lambda for handling HTTP GET and POST requests.
- To use **Amazon DynamoDB** as a NoSQL database for storing student information.
- To host the frontend (HTML, CSS, JavaScript) on **Amazon S3** as a static website.
- To enhance application **security and performance** using **Amazon CloudFront** for HTTPS access.
- To demonstrate **cost optimization, scalability, and reliability** through a serverless architecture.

## Architecture Design :

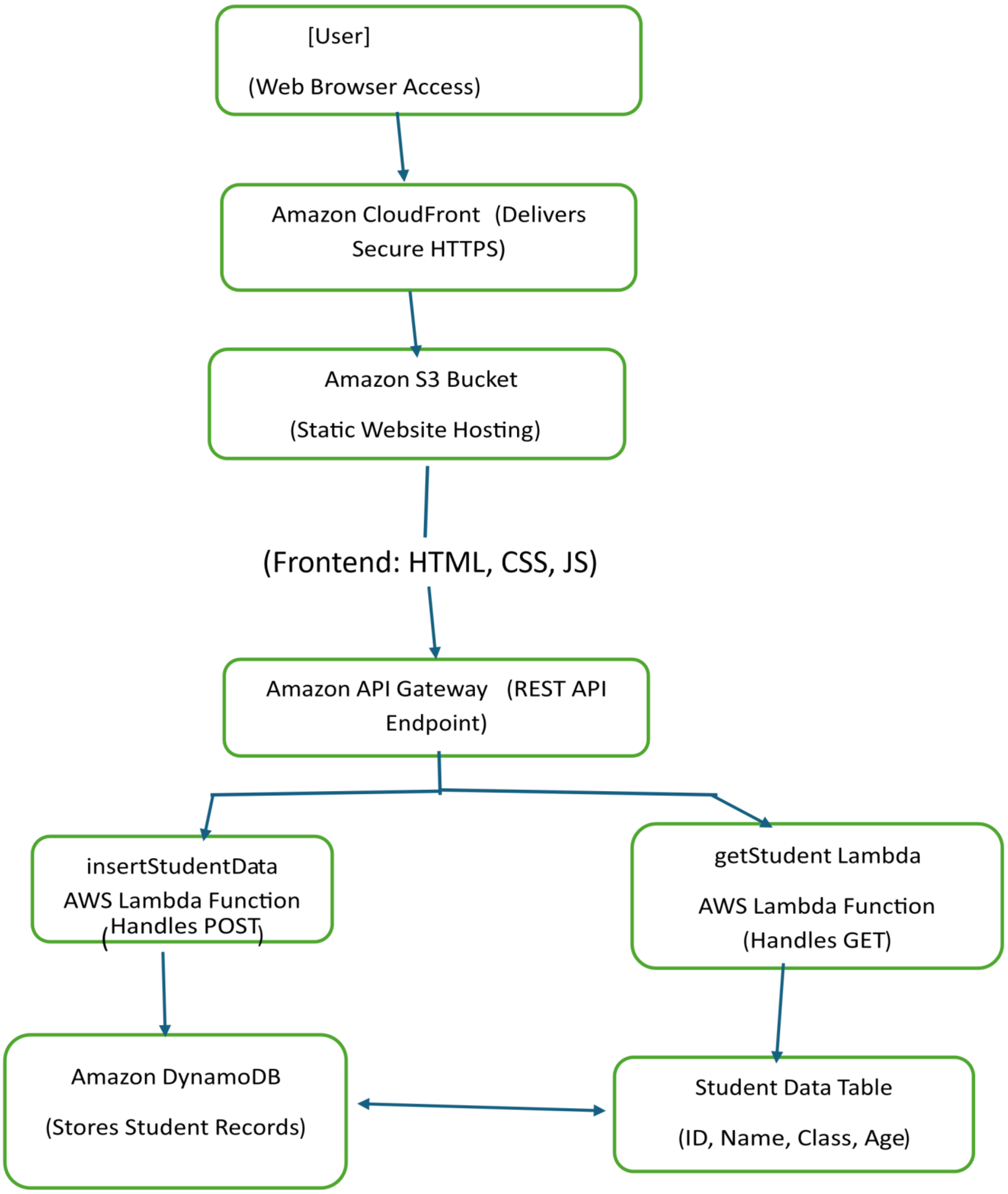
The system follows a classic three-tier architecture, implemented with serverless components:

1. **Presentation Tier (Frontend):**
    - **Amazon S3**: Used for static website hosting. It stores the index.html, scripts.js, and any CSS files.
    - **Amazon CloudFront**: Acts as the Content Delivery Network (CDN). It provides a secure, public-facing URL (https://...cloudfront.net) and caches the website at edge locations, improving performance. It also allows the S3 bucket itself to remain private, enhancing security.
2. **Application Tier (Logic):**
    - **Amazon API Gateway:** Provides the RESTful API "front door." It exposes HTTP endpoints (e.g., POST / and GET /) that the frontend JavaScript can call.
    - **AWS Lambda:** Contains the serverless business logic (written in Python). Two functions are used:

      ->insertStudentData: Triggered by the POST request. It parses the incoming student data and saves it to DynamoDB.

      ->getStudent: Triggered by the GET request. It scans the DynamoDB table and returns a list of all students.
3. **Data Tier (Database):**
    - **Amazon DynamoDB**: A fully managed NoSQL database. A table named studentData is used to store student records, with studentId as the partition key.

**WORKFLOW:**


[User]
(Web Browser Access)
Amazon CloudFront (Delivers Secure HTTPS)
Amazon S3 Bucket
(Static Website Hosting)
(Frontend: HTML, CSS, JS)
Amazon API Gateway (REST API Endpoint)
insertStudentData
AWS Lambda Function
(Handles POST)
getStudent Lambda
AWS Lambda Function
(Handles GET)
Amazon DynamoDB
(Stores Student Records)
Student Data Table
(ID, Name, Class, Age)

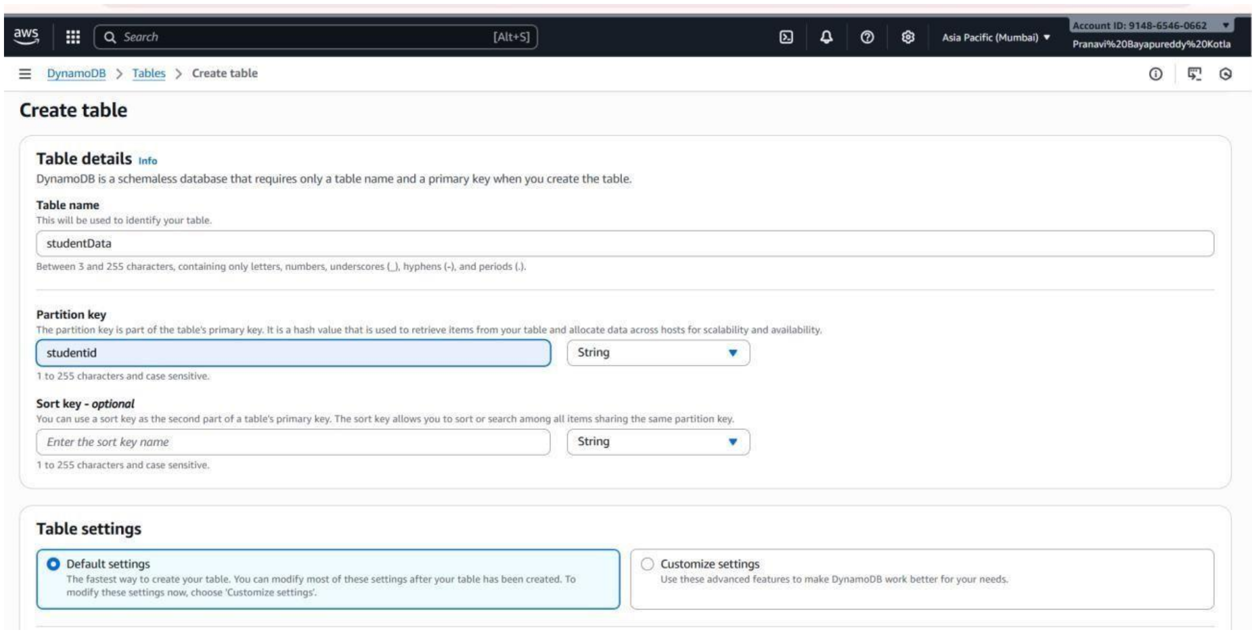
## Implementation Steps & Screenshots:

This section details the step-by-step implementation, as captured in the project screenshots.

**Part 1: Data & Logic Tiers (DynamoDB & Lambda):-**

**1.Create DynamoDB Table**

- Open the AWS Management Console and navigate to DynamoDB.
- Click Create Table and name it studentData.
- Set the Partition Key as Studentid (String).
- Leave other settings as default and create the table.



DynamoDB > Tables > Create table

Create table

Table details

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

Table name: studentData

Partition key: studentid — String

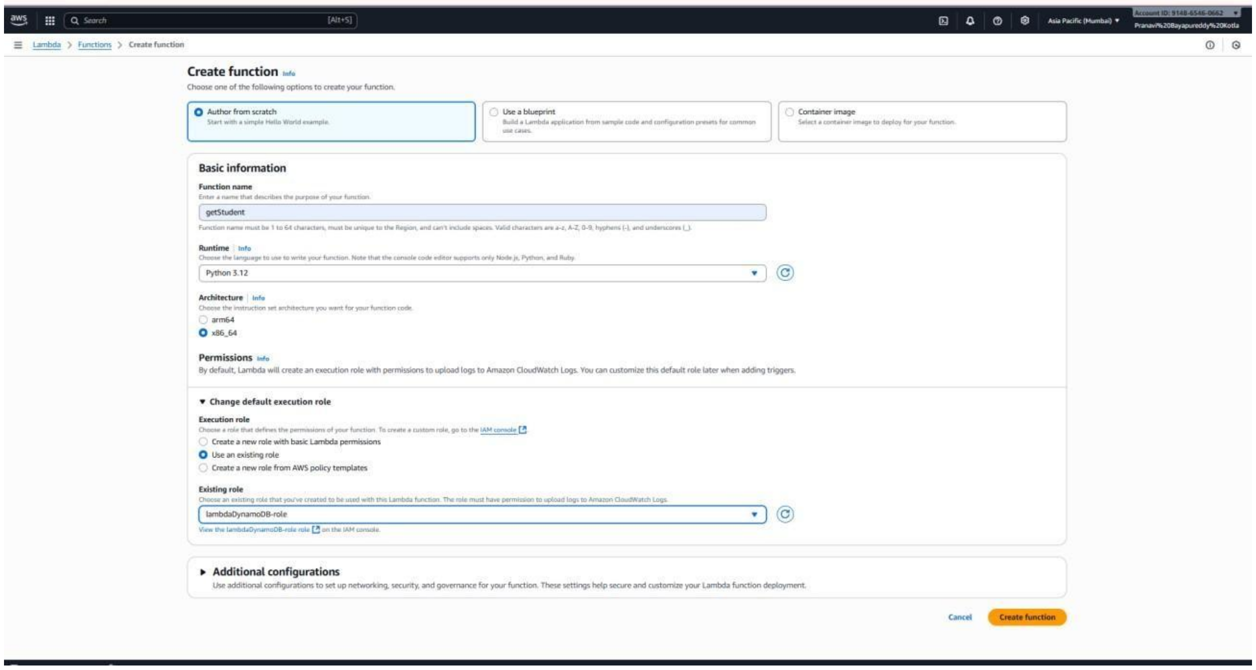
Sort key - optional: Enter the sort key name — String

Table settings: [✓ Default settings] Customize settings

**2.Create Lambda Function for Retrieving Data (GET) ·**

Go to AWS Lambda → Create Function → From Scratch.

- Function name: getStudent.
- Runtime: Python 3.12.
- Assign an existing IAM role with DynamoDB read permissions.



Create function

Choose one of the following options to create your function.

[✓ Author from scratch] Use a blueprint   Container image

Basic information

Function name: getStudent

Runtime: Python 3.12

Architecture: arm64 [✓ x86_64]

Permissions

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

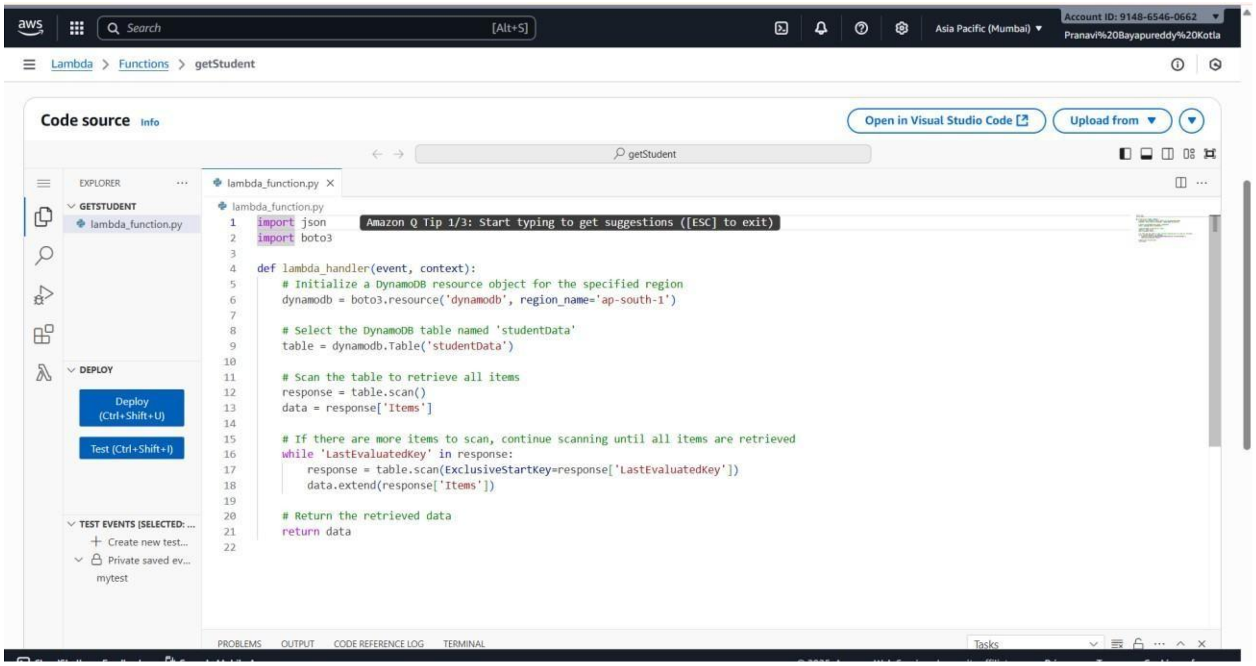
Execution role: Create a new role with basic Lambda permissions [✓ Use an existing role] Create a new role from AWS policy templates

Existing role: lambdaDynamoDB-role

Additional configurations

- Add Python code using boto3 to scan and return all items from the DynamoDB table.

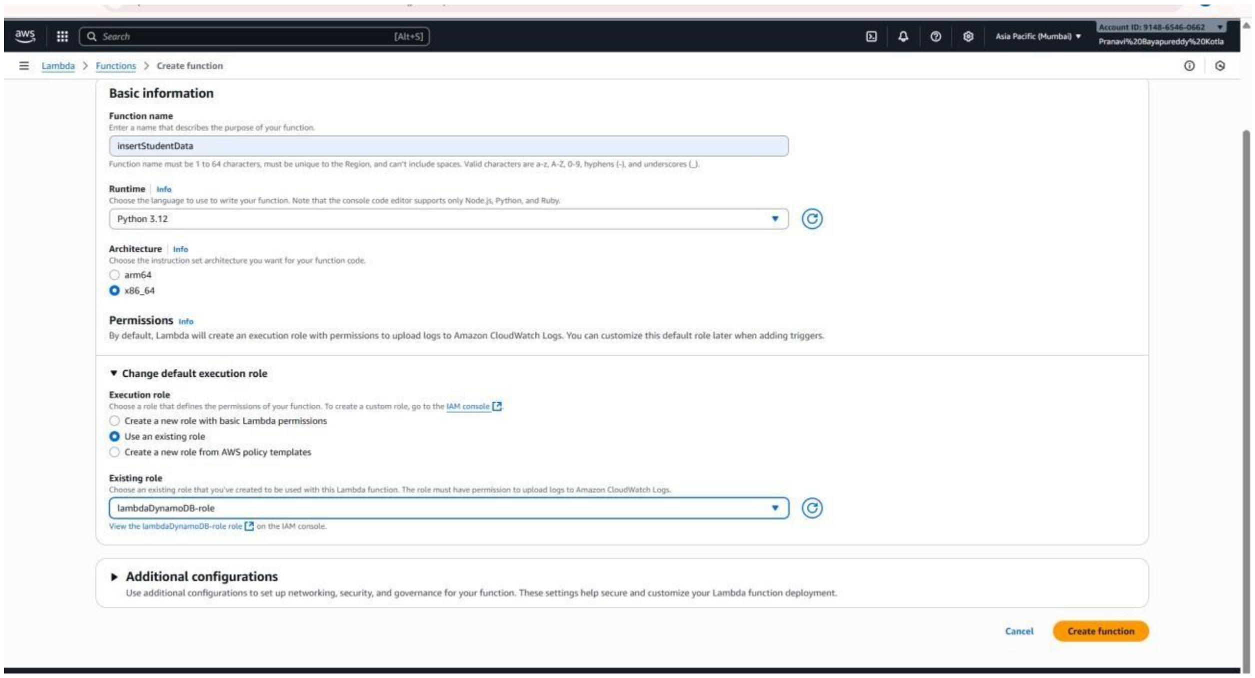
- Click Deploy and test using a sample event (empty payload).



```python
import json
import boto3

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified region
    dynamodb = boto3.resource('dynamodb', region_name='ap-south-1')

    # 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
```

## 3.Create Lambda Function for Inserting Data (POST)

- Create another function named insertStudentData.
- Use the same runtime (**Python 3.12**) and role as before.


aws
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
Lambda > Functions > Create 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 | Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.12

Architecture | Info
Choose the instruction set architecture you want for your function code.
arm64 [✓ x86_64]

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.

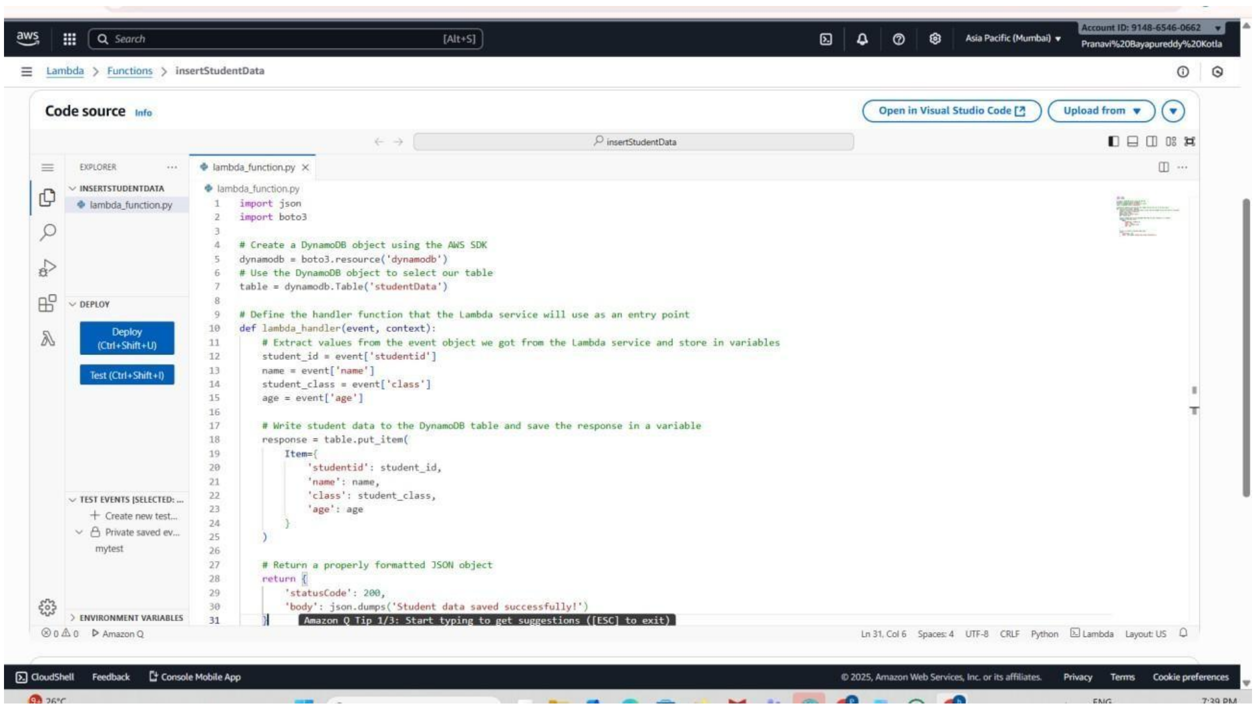
▼ Change default execution role

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.
lambdaDynamoDB-role
View the lambdaDynamoDB-role role on the IAM console.

▶ Additional configurations
Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

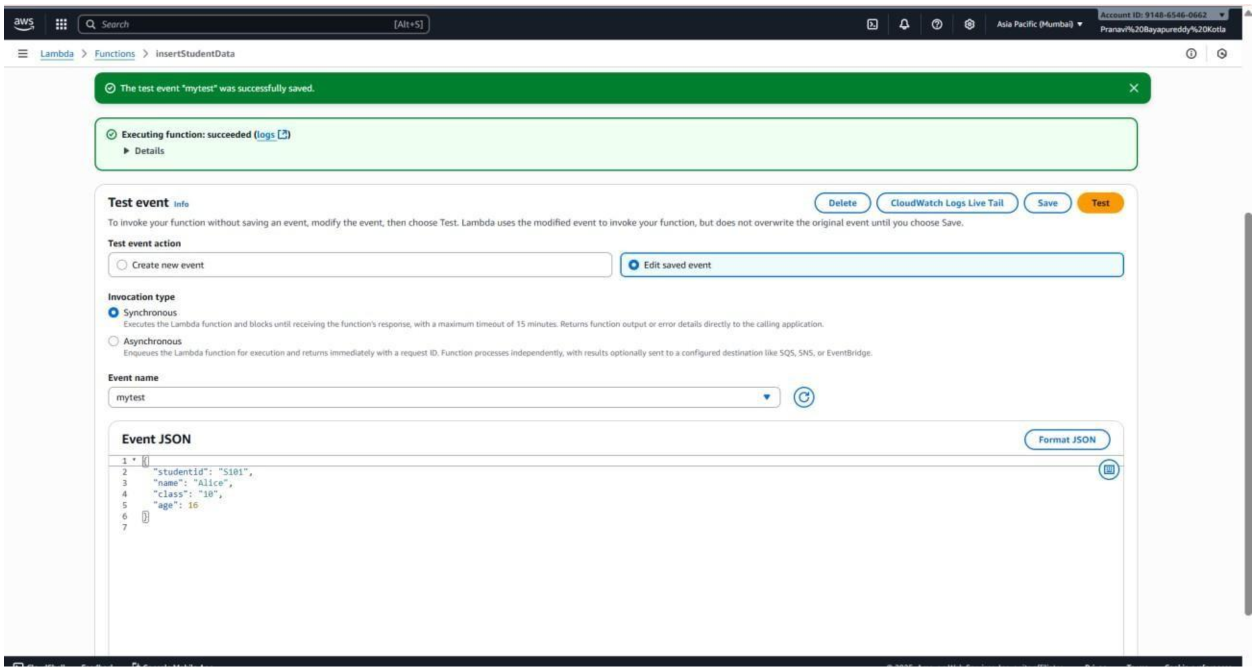
- Write Python code to parse the input JSON (Studentid, Name, Class, Age) and insert it into DynamoDB using put_item().
- Deploy and test using a JSON test event to verify that data is stored correctly
- Validate the record in DynamoDB using **Explore Table Items**



Lambda > Functions > insertStudentData
Code source Info
Open in Visual Studio Code
Upload 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
ENVIRONMENT VARIABLES
Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
Ln 31, Col 6 Spaces: 4 UTF-8 CRLF Python Lambda Layout: US

```python
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!')
    }
```

4. **Test Lambda Function:**
    - The insertStudentData function was tested directly in the Lambda console.
    - A test event named mytest was configured with sample JSON data for a student.


aws
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
Lambda > Functions > insertStudentData

The test event "mytest" was successfully saved.

Executing function: succeeded (logs)

▶ Details

Delete
CloudWatch Logs Live Tail
Save
Test
Format JSON

**Test event** Info

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save.

**Test event action**

Create new event [✓ Edit saved event]

**Invocation type**

[✓ Synchronous]
Executes the Lambda function and blocks until receiving the function's response, with a maximum timeout of 15 minutes. Returns function output or error details directly to the calling application.

Asynchronous
Enqueues the Lambda function for execution and returns immediately with a request ID. Function processes independently, with results optionally sent to a configured destination like SQS, SNS, or EventBridge.

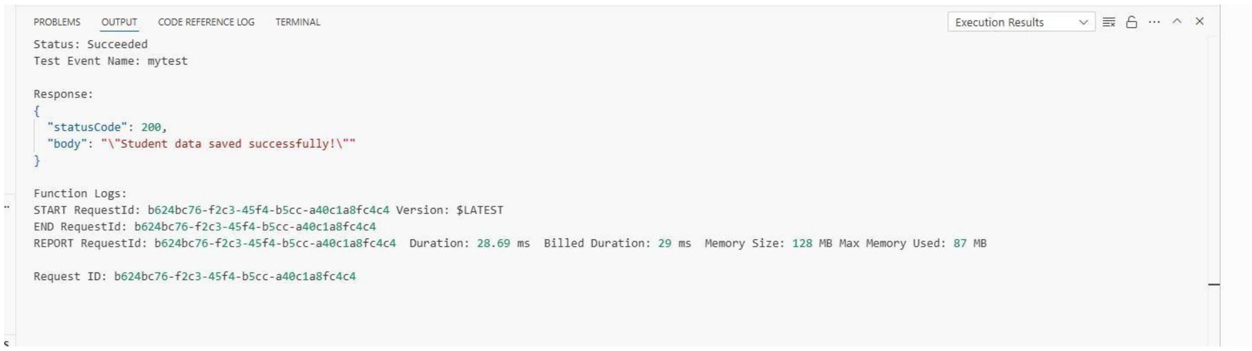
**Event name**

mytest

**Event JSON**

```
{
  "studentid": "S101",
  "name": "Alice",
  "class": "10",
  "age": 16
}
```

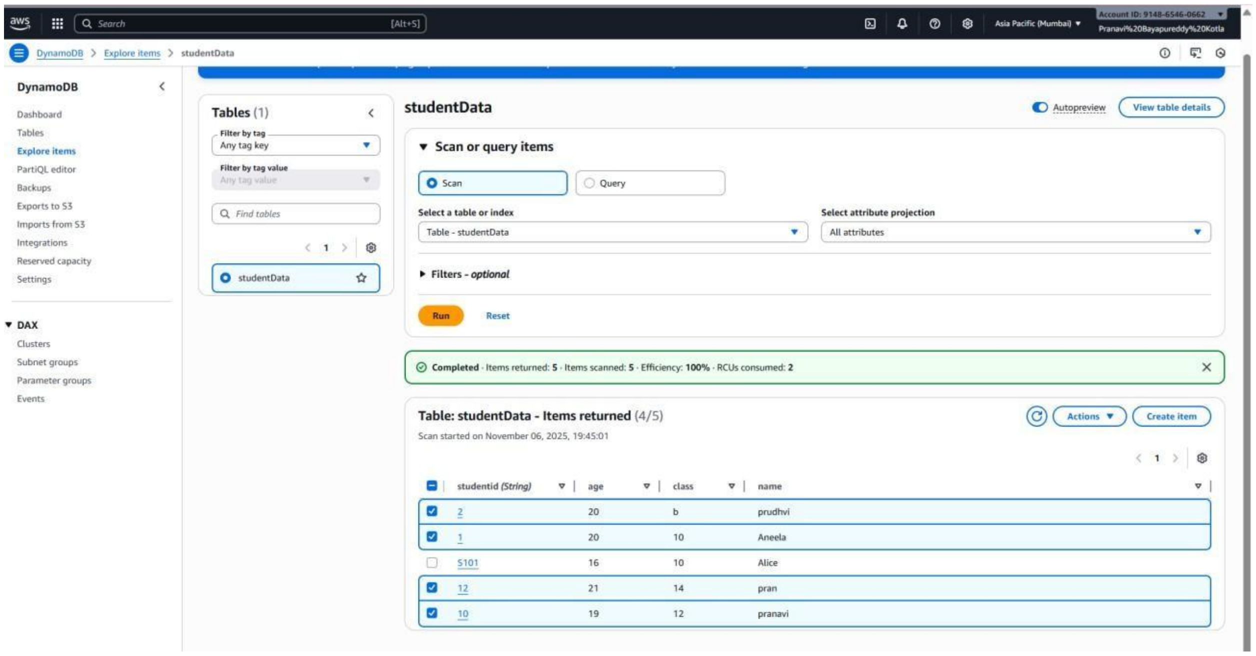
- The test executed successfully, returning a statusCode: 200 and a success message.



```
Status: Succeeded
Test Event Name: mytest

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

Function Logs:
START RequestId: b624bc76-f2c3-45f4-b5cc-a40c1a8fc4c4 Version: $LATEST
END RequestId: b624bc76-f2c3-45f4-b5cc-a40c1a8fc4c4
REPORT RequestId: b624bc76-f2c3-45f4-b5cc-a40c1a8fc4c4  Duration: 28.69 ms  Billed Duration: 29 ms  Memory Size: 128 MB Max Memory Used: 87 MB

Request ID: b624bc76-f2c3-45f4-b5cc-a40c1a8fc4c4
```

- The successful insertion was verified by going to DynamoDB > Explore items

> studentData. The new record  was visible in the table



studentData

Select a table or index: Table - studentData

Select attribute projection: All attributes

Completed · Items returned: 5 · Items scanned: 5 · Efficiency: 100% · RCUs consumed: 2

Table: studentData - Items returned (4/5)

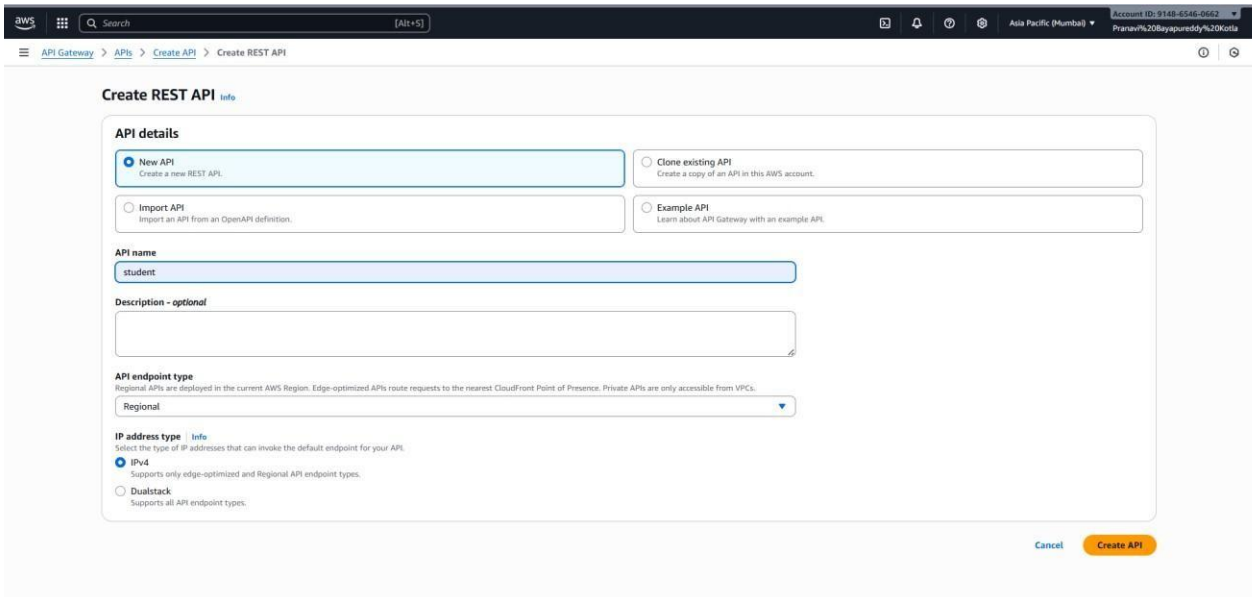
Scan started on November 06, 2025, 19:45:01

| | studentid (String) | age | class | name |
|---|---|---|---|---|
| ✓ | 2 | 20 | b | prudhvi |
| ✓ | 1 | 20 | 10 | Aneela |
| ☐ | S101 | 16 | 10 | Alice |
| ✓ | 12 | 21 | 14 | pran |
| ✓ | 10 | 19 | 12 | pranavi |

## Part 2: Application and Presentation Tiers (API Gateway & S3):-

### 1.Create API Gateway

- Navigate to API Gateway in the AWS Console.
- Click Create API → REST API → New API.
- Name it student and choose Edge-Optimized endpoint type for global access.


aws
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
API Gateway > APIs > Create API > Create REST API
Cancel
Create API

Create REST API Info

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

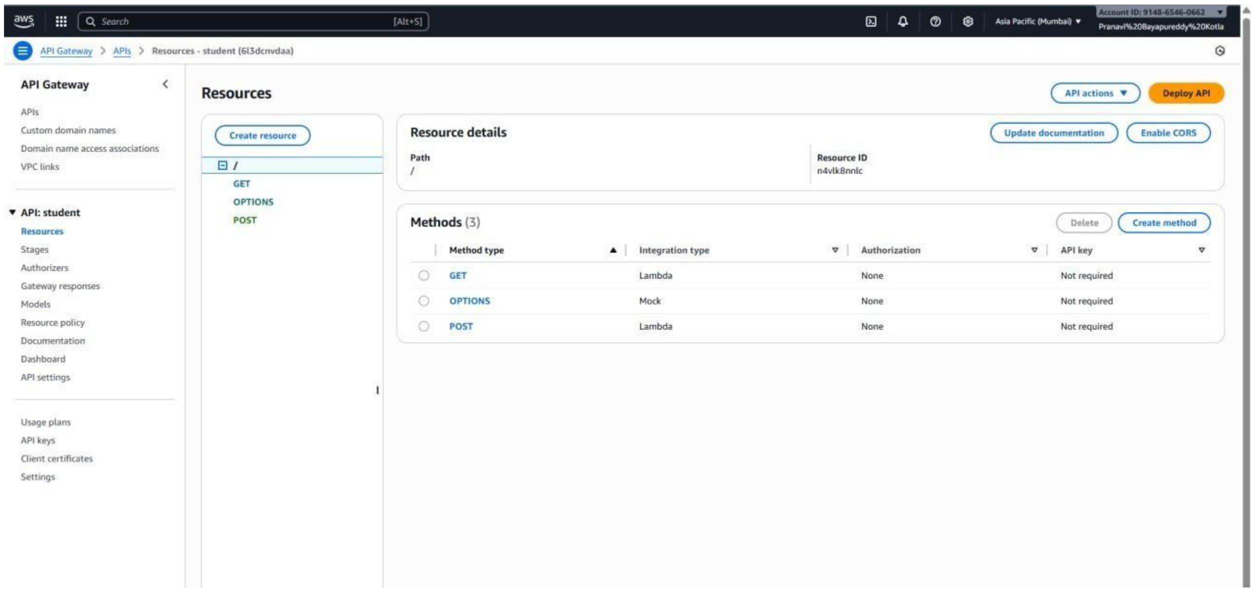
Description - *optional*

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.
Regional

IP address type Info
Select the type of IP addresses that can invoke the default endpoint for your API.
[✓ IPv4] Supports only edge-optimized and Regional API endpoint types.
Dualstack — Supports all API endpoint types.

## 2.Create API Methods

- Add a **GET** method and integrate it with the getStudent Lambda function.
- Add a **POST** method and integrate it with the insertStudentData Lambda function.
- Test both methods using the **Test** option in API Gateway to confirm successful responses.



Resource details

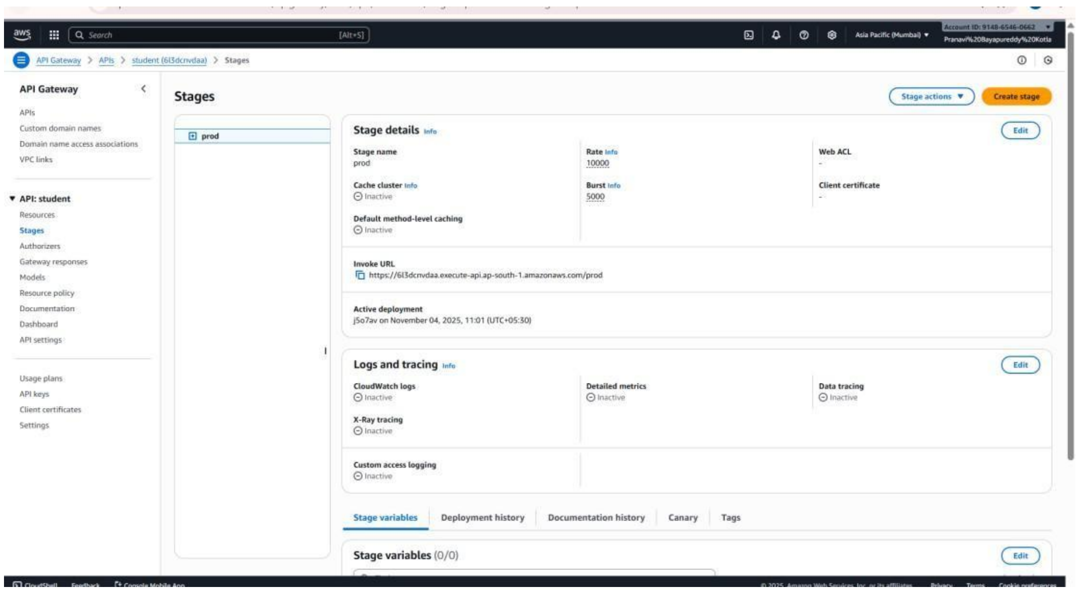
Path: /
Resource ID: n4vlk8nnlc

Methods (3)

| Method type | Integration type | Authorization | API key |
|---|---|---|---|
| GET | Lambda | None | Not required |
| OPTIONS | Mock | None | Not required |
| POST | Lambda | None | Not required |

## 3. Deploy API & Configure CORS

- The API was deployed to a New Stage named prod.
- This generated the public Invoke URL(https://6l3dcnvdaa.executeapi.apsouth-1.amazonaws.com/prod)



Stage details

Stage name: prod
Rate: 10000
Web ACL: -
Cache cluster: Inactive
Burst: 5000
Client certificate: -
Default method-level caching: Inactive

Invoke URL: https://6l3dcnvdaa.execute-api.ap-south-1.amazonaws.com/prod

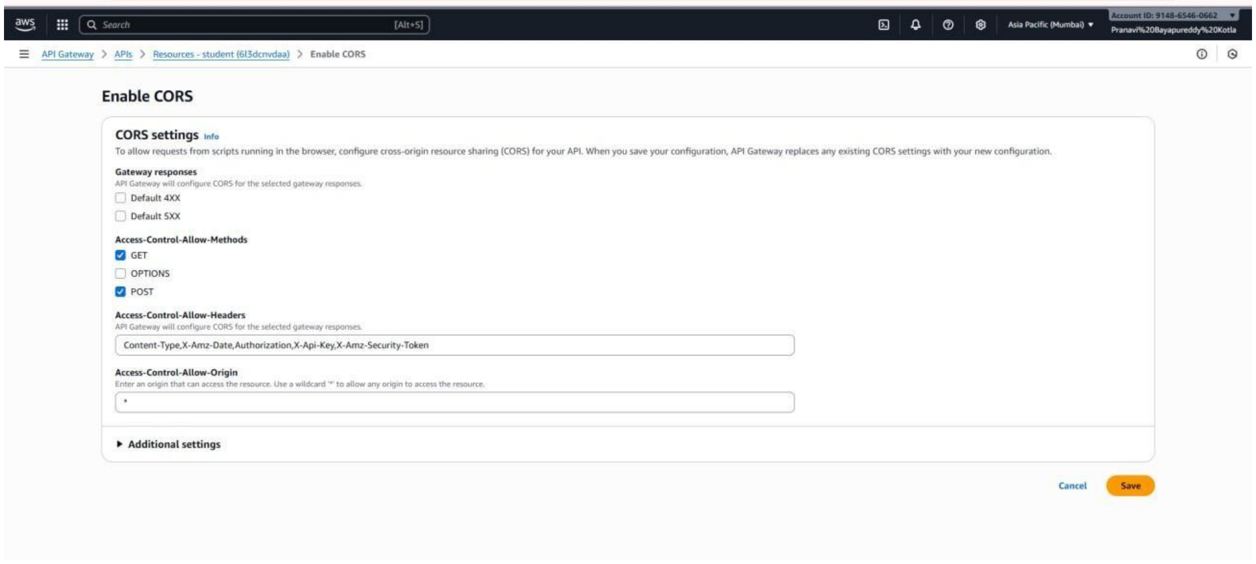
Active deployment: j5o7av on November 04, 2025, 11:01 (UTC+05:30)

Logs and tracing

CloudWatch logs: Inactive
Detailed metrics: Inactive
Data tracing: Inactive
X-Ray tracing: Inactive
Custom access logging: Inactive

Stage variables (0/0)

**CORS (Cross-Origin Resource Sharing)** was enabled for the API to allow the S3 website (on a different domain) to make requests. This automatically set up an OPTIONS method and added the necessary Access-Control-AllowOrigin: * headers.


Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
API Gateway > APIs > Resources - student (6l3dcnvdaa) > 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

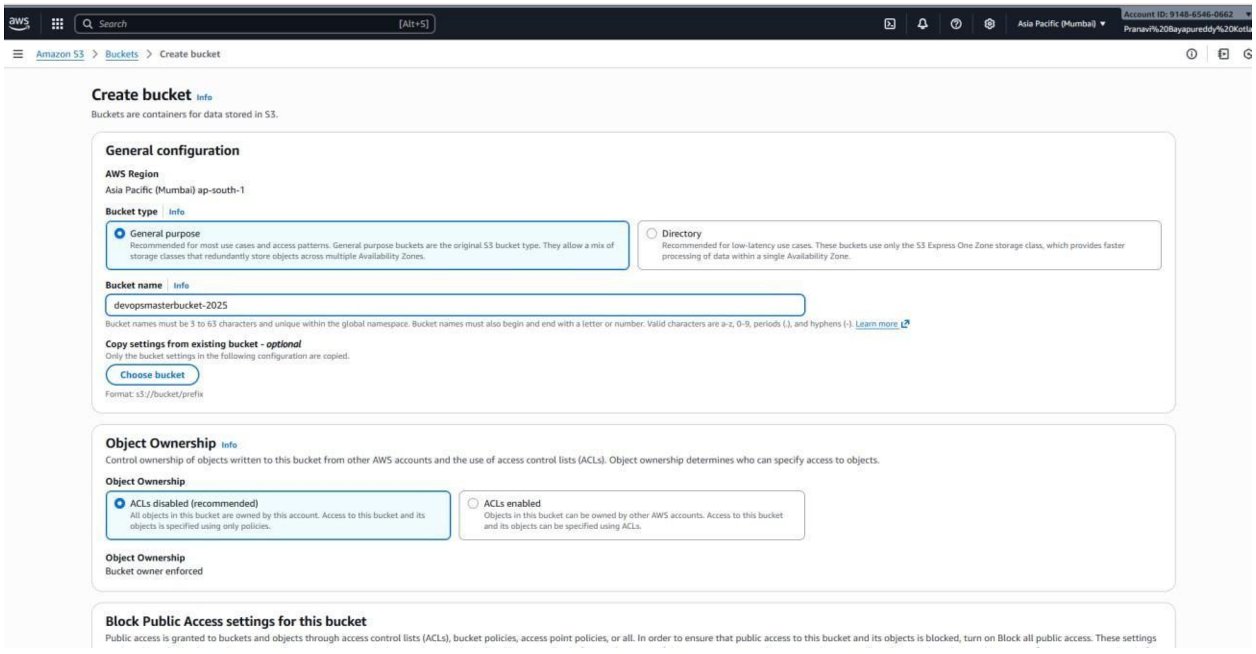
Access-Control-Allow-Methods
- [✓] GET
- [ ] OPTIONS
- [✓] POST

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

Access-Control-Allow-Origin
Enter an origin that can access the resource. Use a wildcard '*' to allow any origin to access the resource.
*

## 4.Configure S3 Bucket for Website Hosting

- An S3 bucket named devopsmasterbucket-2025 was created in the apsouth-1 region.


Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region
Asia Pacific (Mumbai) ap-south-1

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
devopsmasterbucket-2025
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

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

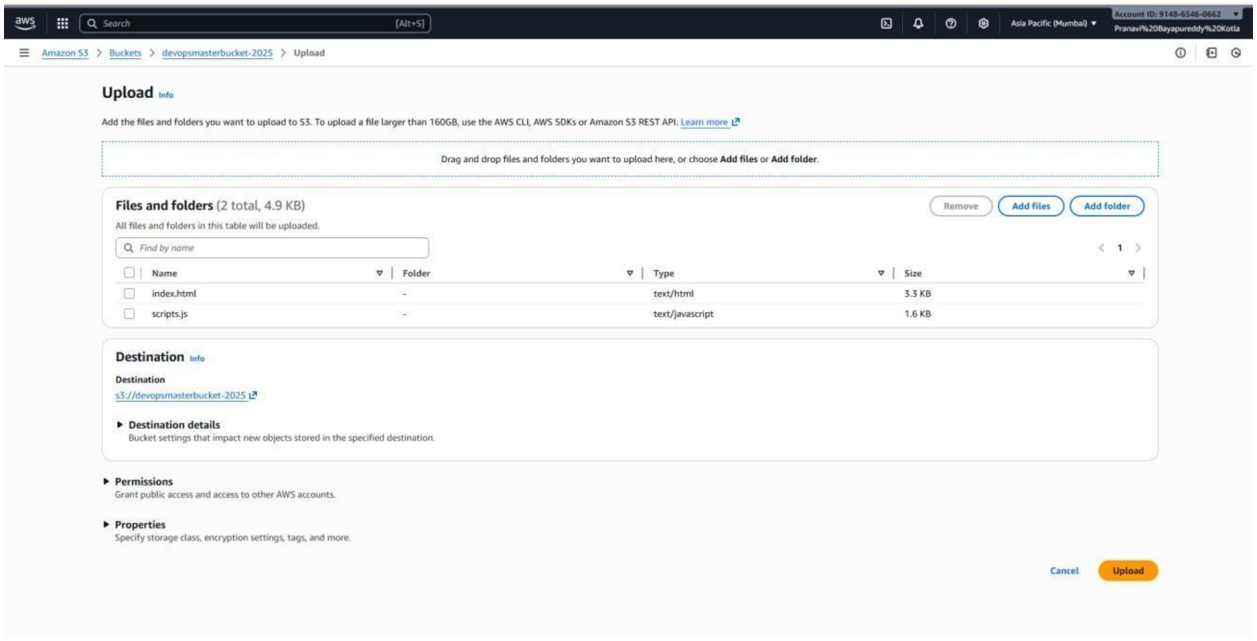
Object Ownership
[✓ ACLs disabled (recommended)] All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
ACLs enabled: Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

Block Public Access settings for this bucket

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 this bucket and its objects is blocked, turn on Block all public access. These settings

- The frontend files (scripts.js, index.html) were uploaded to the bucket.



Amazon S3 > Buckets > devopsmasterbucket-2025 > Upload

**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, 4.9 KB)

All files and folders in this table will be uploaded.

| Name | Folder | Type | Size |
|---|---|---|---|
| index.html | - | text/html | 3.3 KB |
| scripts.js | - | text/javascript | 1.6 KB |

**Destination** Info

Destination

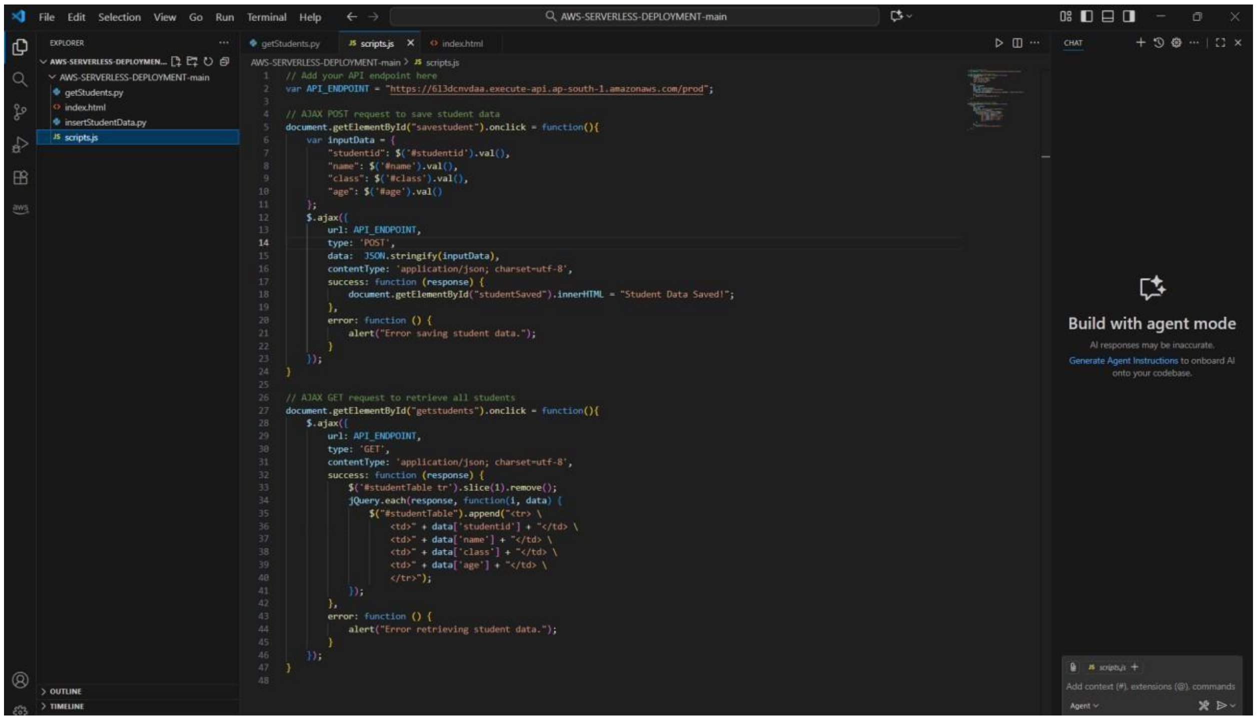
s3://devopsmasterbucket-2025

**Destination details**
Bucket settings that impact new objects stored in the specified destination.

**Permissions**
Grant public access and access to other AWS accounts.

**Properties**
Specify storage class, encryption settings, tags, and more.

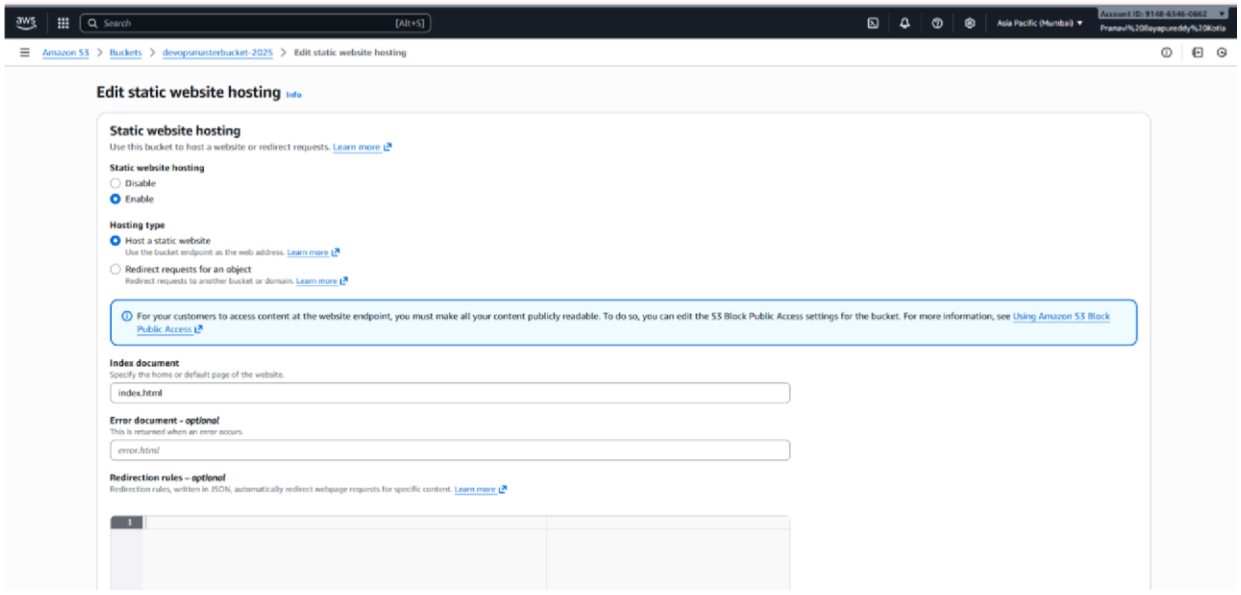
- The scripts.js file was edited to include the API Gateway Invoke URL in the API_ENDPOINT variable.



```javascript
// Add your API endpoint here
var API_ENDPOINT = "https://613dcnvdaa.execute-api.ap-south-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
document.getElementById("getstudents").onclick = function(){
    $.ajax({
        url: API_ENDPOINT,
        type: 'GET',
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            $('#studentTable tr').slice(1).remove();
            jQuery.each(response, function(i, data) {
                $("#studentTable").append("<tr> \
                    <td>" + data['studentid'] + "</td> \
                    <td>" + data['name'] + "</td> \
                    <td>" + data['class'] + "</td> \
                    <td>" + data['age'] + "</td> \
                    </tr>");
            });
        },
        error: function () {
            alert("Error retrieving student data.");
        }
    });
}
```

File
Edit
Selection
View
Go
Run
Terminal
Help
AWS-SERVERLESS-DEPLOYMENT-main
EXPLORER
AWS-SERVERLESS-DEPLOYMEN...
AWS-SERVERLESS-DEPLOYMENT-main
getStudents.py
index.html
insertStudentData.py
scripts.js
getStudents.py
scripts.js
index.html
AWS-SERVERLESS-DEPLOYMENT-main > scripts.js
CHAT
Build with agent mode
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Add context (#), extensions (@), commands
Agent
OUTLINE
TIMELINE

- Static website hosting was enabled on the bucket, with index.html as the index document.


aws
Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
Amazon S3 > Buckets > devopsmasterbucket-2025 > 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

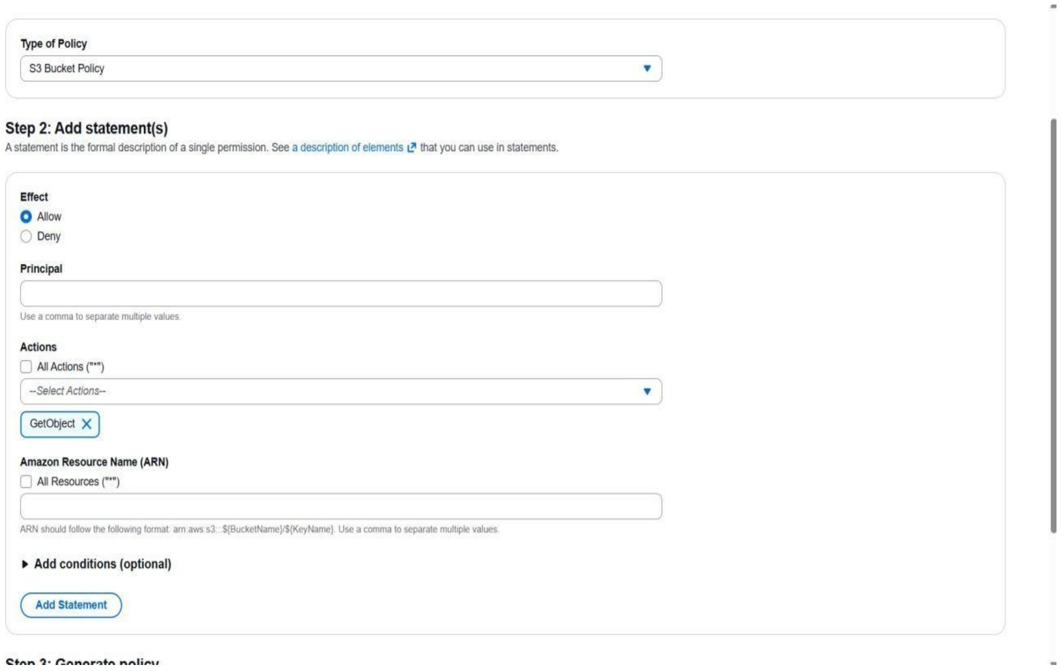
For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see Using Amazon S3 Block Public Access

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

Error document - *optional*
This is returned when an error occurs.
*error.html*

Redirection rules – *optional*
Redirection rules, written in JSON, automatically redirect webpage requests for specific content. Learn more

- A public bucket policy was generated (using the AWS Policy Generator) and applied to the bucket to allow s3:GetObject actions from any principal ("Principal": "*").



Type of Policy
S3 Bucket Policy

Step 2: Add statement(s)
A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect
[✓ Allow] Deny

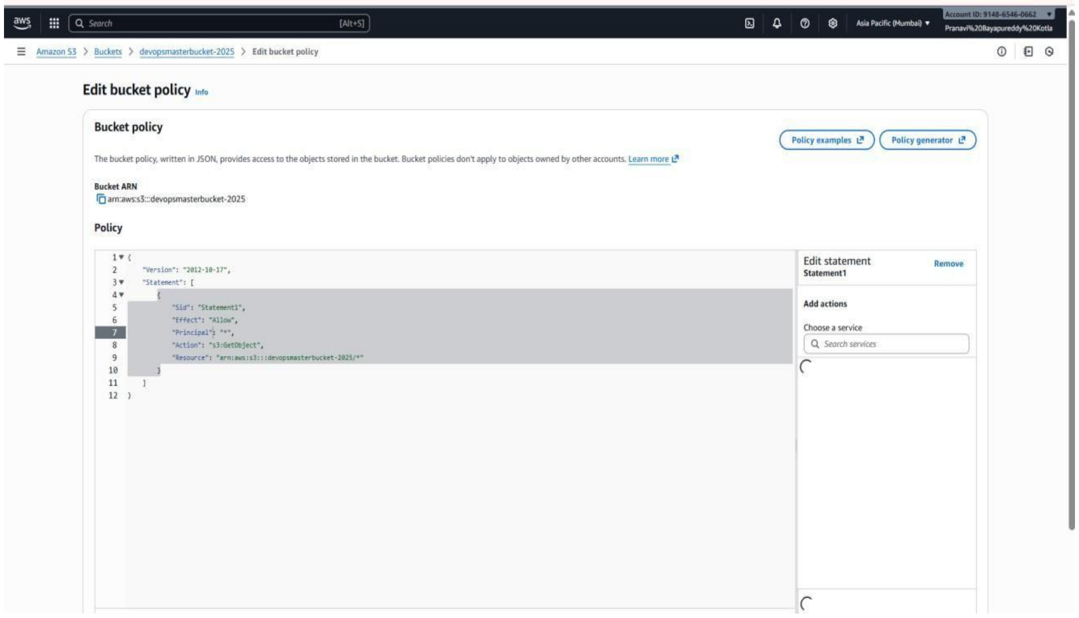
Principal

Use a comma to separate multiple values.

Actions
All Actions ("*")
--Select Actions--
GetObject

Amazon Resource Name (ARN)
All Resources ("*")

ARN should follow the following format: arn:aws:s3:::${BucketName}/${KeyName}. Use a comma to separate multiple values.


Search [Alt+S]
Asia Pacific (Mumbai)
Account ID: 9148-6546-0662
Pranavi%20Bayapureddy%20Kotla
Amazon S3 > Buckets > devopsmasterbucket-2025 > Edit bucket policy

Edit bucket policy Info

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

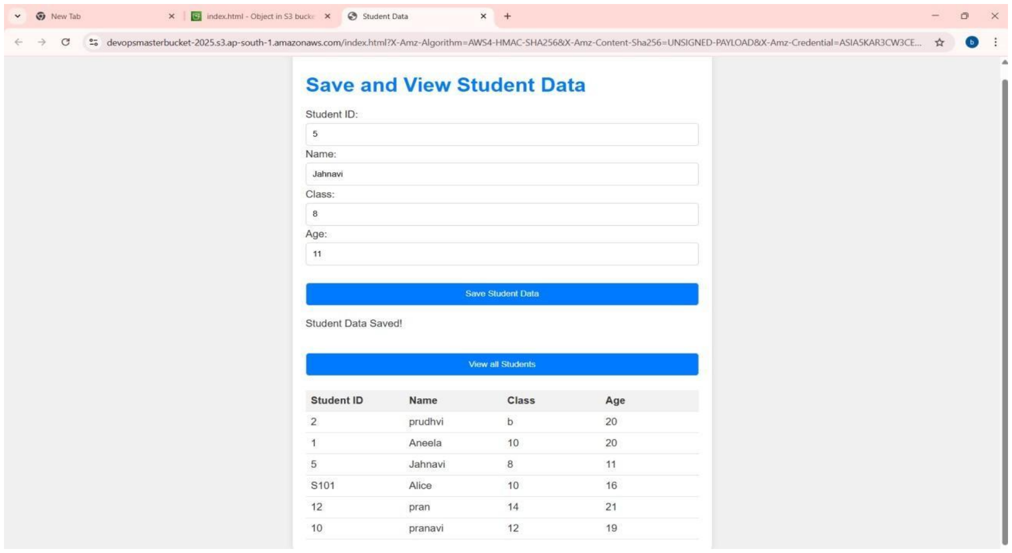
arn:aws:s3:::devopsmasterbucket-2025

Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::devopsmasterbucket-2025/*"
        }
    ]
}
```

## 5. Test Final Website (S3 URL):

- The S3 website endpoint URL was opened in a browser.
- The "View all Students" button successfully fetched and displayed the initial test data.
- A new student ("Jahnavi") was added via the form, which successfully saved and updated the table on the page.



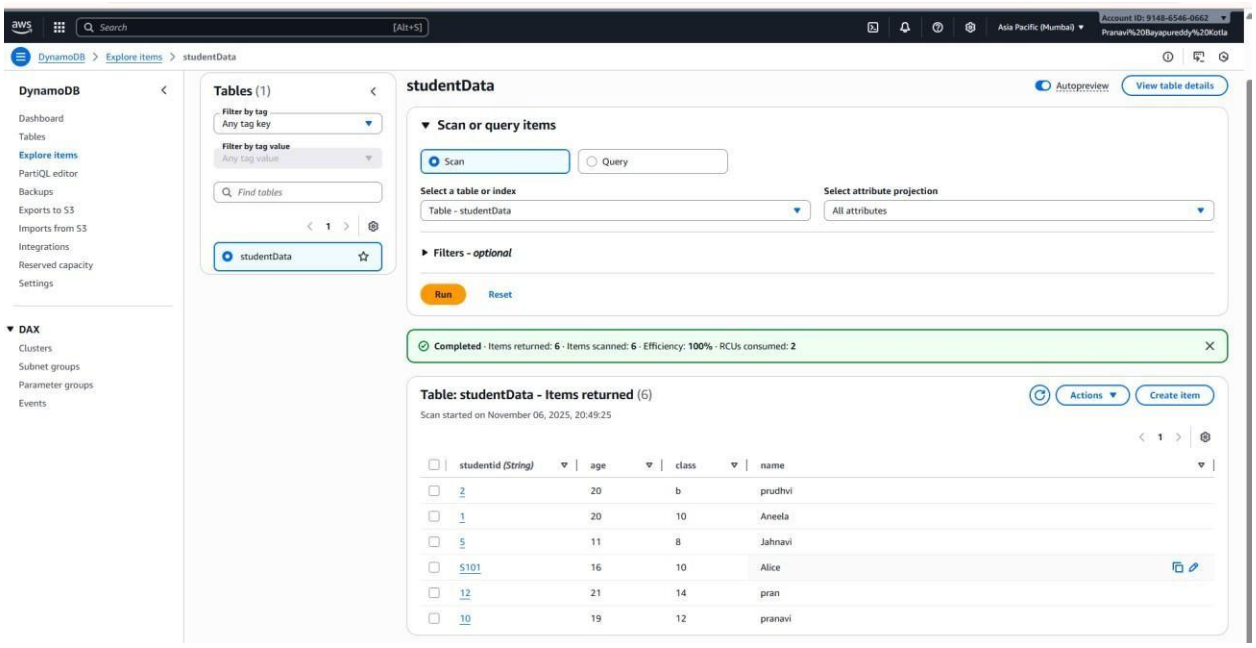
Save and View Student Data

| | |
|---|---|
| Student ID: | 5 |
| Name: | Jahnavi |
| Class: | 8 |
| Age: | 11 |

Student Data Saved!

| Student ID | Name | Class | Age |
|---|---|---|---|
| 2 | prudhvi | b | 20 |
| 1 | Aneela | 10 | 20 |
| 5 | Jahnavi | 8 | 11 |
| S101 | Alice | 10 | 16 |
| 12 | pran | 14 | 21 |
| 10 | pranavi | 12 | 19 |

## DB Table Result:



studentData

Select a table or index: Table - studentData

Select attribute projection: All attributes

Completed · Items returned: 6 · Items scanned: 6 · Efficiency: 100% · RCUs consumed: 2

Table: studentData - Items returned (6)

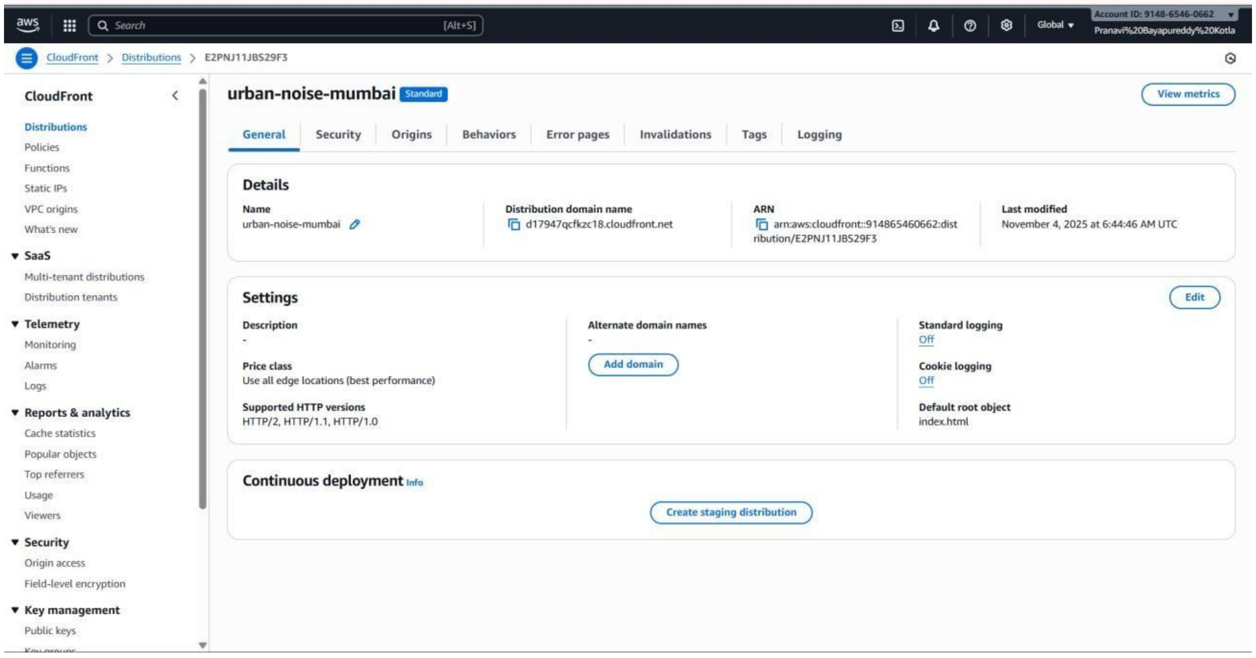
Scan started on November 06, 2025, 20:49:25

| studentid (String) | age | class | name |
|---|---|---|---|
| 2 | 20 | b | prudhvi |
| 1 | 20 | 10 | Aneela |
| 5 | 11 | 8 | Jahnavi |
| S101 | 16 | 10 | Alice |
| 12 | 21 | 14 | pran |
| 10 | 19 | 12 | pranavi |

## Part 3: (Enhancement) Securing with CloudFront

### 1.Create CloudFront Distribution:

- A CloudFront distribution (aws-three-tier) was created.
- The Origin domain was set to the S3 bucket's static website endpoint (http://devopsmasterbucket-2025.s3-website.ap-south-1.amazonaws.com).



urban-noise-mumbai Standard

Details

Name: urban-noise-mumbai
Distribution domain name: d17947qcfkzc18.cloudfront.net
ARN: arn:aws:cloudfront::914865460662:distribution/E2PNJ11JBS29F3
Last modified: November 4, 2025 at 6:44:46 AM UTC

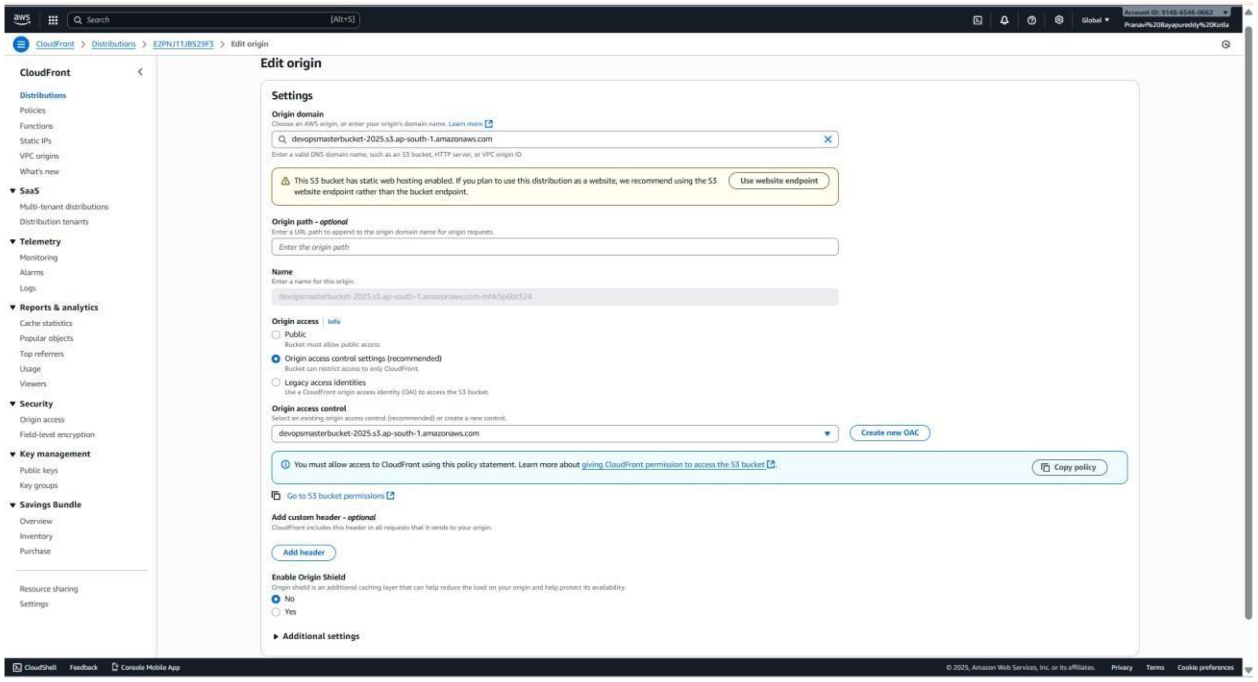
Settings

Description: -
Price class: Use all edge locations (best performance)
Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0
Alternate domain names: -
Standard logging: Off
Cookie logging: Off
Default root object: index.html

Continuous deployment

### 2. Secure S3 Bucket:

- To ensure users only access the site via CloudFront, Origin access control settings (OAC) was configured.
- This generated a new S3 bucket policy that only allows s3:GetObject from the CloudFront service principal.
- The previous public bucket policy was replaced with this new, more secure policy. o Block public access (bucket settings) was then turned On for the S3 bucket.


CloudFront > Distributions > E2PNJ11JBS29F3 > Edit origin

**Edit origin**

**Settings**

**Origin domain**
Choose an AWS origin, or enter your origin's domain name. Learn more
devopsmasterbucket-2025.s3.ap-south-1.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.

**Name**
Enter a name for this origin.
devopsmasterbucket-2025.s3.ap-south-1.amazonaws.com-mhk5p6bt324

**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.

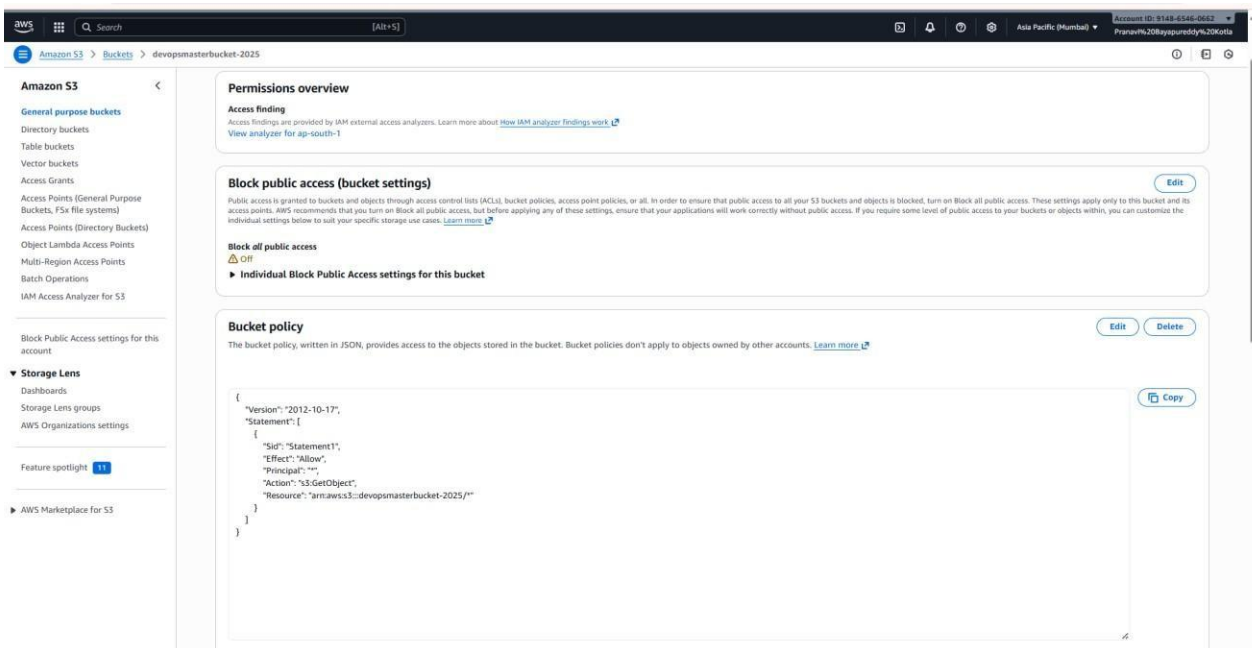
**Origin access control**
Select an existing origin access control (recommended) or create a new control.
devopsmasterbucket-2025.s3.ap-south-1.amazonaws.com

You must allow access to CloudFront using this policy statement. Learn more about giving CloudFront permission to access the S3 bucket.

**Add custom header - *optional***
CloudFront includes this header in all requests that it sends to your origin.

**Enable Origin Shield**
Origin shield is an additional caching layer that can help reduce the load on your origin and help protect its availability.
[✓ No] Yes

## Block public access and Update new bucket policy:


Amazon S3 > Buckets > devopsmasterbucket-2025

**Permissions overview**

**Access finding**
Access findings are provided by IAM external access analyzers. Learn more about How IAM analyzer findings work
View analyzer for ap-south-1

**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**
Off

**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:::devopsmasterbucket-2025/*"
    }
  ]
}
```

**3. Final Secure URL:**

The final, secure, and globally distributed URL for the application is https://devopsmasterbucket-2025.s3.ap-south-1.amazonaws.com/index.html

## ADVANTAGES:

- No need to manage servers — AWS handles everything automatically.
- Very cost-efficient because you only pay for what you use.
- Automatically scales when more users access the application.
- Easy to deploy and update since all components are serverless.

## LIMITATIONS:

- Lambda functions may have a short delay during the first run (cold start).
- Limited execution time for AWS Lambda functions.
- Requires an active internet connection to access AWS services.
- Debugging and error tracking can be more difficult than traditional servers.

## FUTURE ENHANCEMENTS:

- Add user login and authentication using **Amazon Cognito**.
- Include update and delete options for complete data management.
- Integrate **AWS CloudWatch** for monitoring and performance tracking. Use **AWS Amplify** for automated deployment and hosting.

## CONCLUSION:

This project successfully demonstrates the development and deployment of a **serverless web application** using various **AWS cloud services** such as **Lambda, API Gateway, DynamoDB, S3, and CloudFront**. The application manages student data efficiently and allows users to add and view information through a simple web interface.By using a **serverless architecture**, the project removes the need for setting up or maintaining any physical servers. All infrastructure tasks such as scaling, security, and resource management are handled automatically by AWS. This results in **lower costs, faster performance, and easier maintenance**.

The use of **AWS Lambda** for backend logic and **DynamoDB** for data storage ensures that the system can handle multiple requests simultaneously without any downtime. **API Gateway** helps connect the frontend and backend securely, while **S3** and **CloudFront** provide a fast and safe way to host and access the website globally.Overall, this project proves that **serverless computing** is an ideal solution for modern applications. It provides **flexibility, scalability, and cost efficiency**, making it suitable for real-world use in both small and large-scale systems. This project also gives practical experience in integrating multiple AWS services to build a complete, fully functional web-based system.