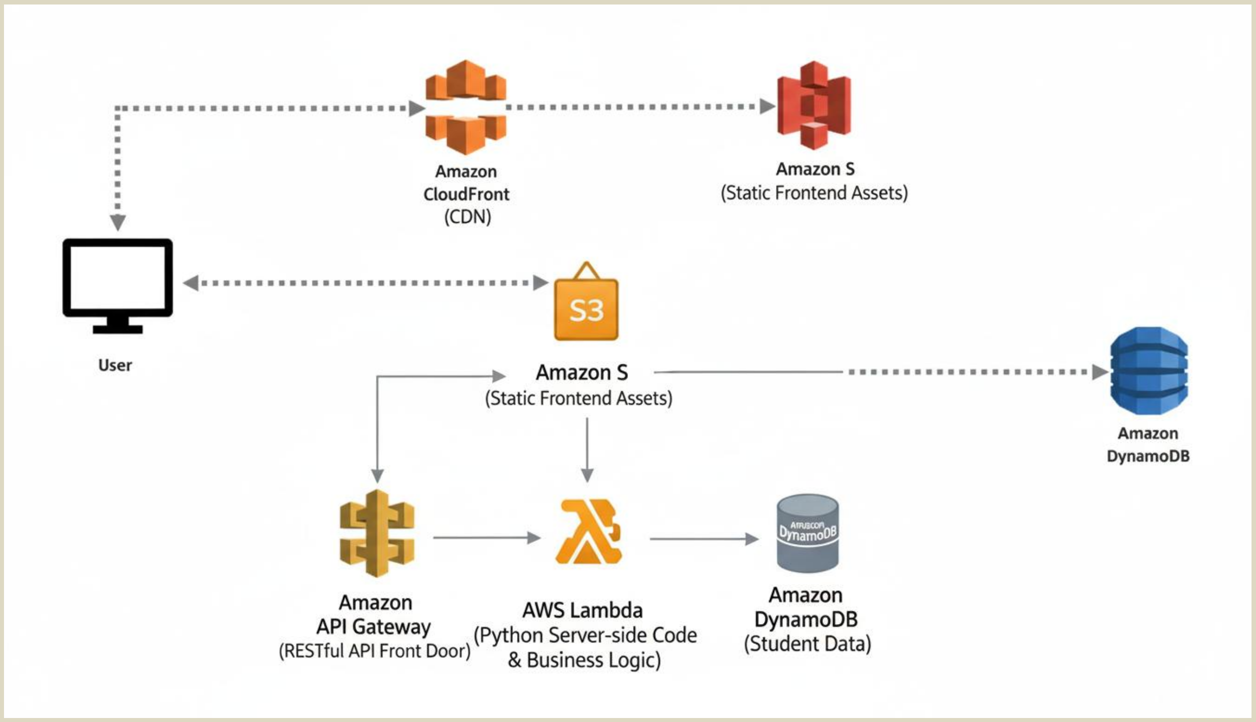
Building a Serverless Web Application on AWS Services with Python: A Comprehensive Guide for Student Data Management

This document provides a detailed, step-by-step guide for constructing a modern serverless web application on Amazon Web Services (AWS) using Python for the backend logic. This architecture emphasizes scalability, cost-effectiveness, and operational efficiency, allowing developers to concentrate on application features rather than infrastructure management.

Project Overview

This project outlines the creation of a serverless web application designed to manage student data. By leveraging AWS's fully managed serverless offerings, we aim to demonstrate a robust and scalable solution for data storage, retrieval, and presentation without the overhead of traditional server provisioning. The choice of Python for AWS Lambda functions ensures a widely adopted and versatile language for backend processing.



Amazon CloudFront (CDN)
Amazon S (Static Frontend Assets)
User
S3
Amazon S (Static Frontend Assets)
Amazon DynamoDB
Amazon API Gateway (RESTful API Front Door)
AWS Lambda (Python Server-side Code & Business Logic)
Amazon DynamoDB (Student Data)

Figure 1Cloud Architecture

Architecture Components

Our serverless architecture adheres to a well-established pattern, integrating various AWS services to achieve a highly available, performant, and secure application:

- **Amazon S3 (Simple Storage Service)**: Serves as the origin for hosting all static frontend assets, including HTML, CSS, JavaScript, and images. S3 offers high durability, availability, and scalability for static content.
- **Amazon CloudFront**: Acts as a global Content Delivery Network (CDN), caching content at edge locations worldwide. This reduces latency, improves content delivery speed, and enhances security by providing HTTPS connections and acting as the primary entry point for web traffic.
- **Amazon API Gateway**: Provides a fully managed service for creating, publishing, maintaining, monitoring, and securing RESTful APIs. It acts as the "front door" for applications to access backend services, handling request routing, authorization, and throttling.
- **AWS Lambda (with Python)**: A serverless compute service that executes code in response to events without requiring server provisioning or management. Our Python-based Lambda functions will encapsulate the application's business logic for CRUD (Create, Read, Update, Delete) operations on student data.
- **Amazon DynamoDB**: A fast, flexible NoSQL database service offering single-digit millisecond performance at any scale. It is utilized for storing student data, providing a highly available and durable data store.

Step-by-Step Implementation Guide

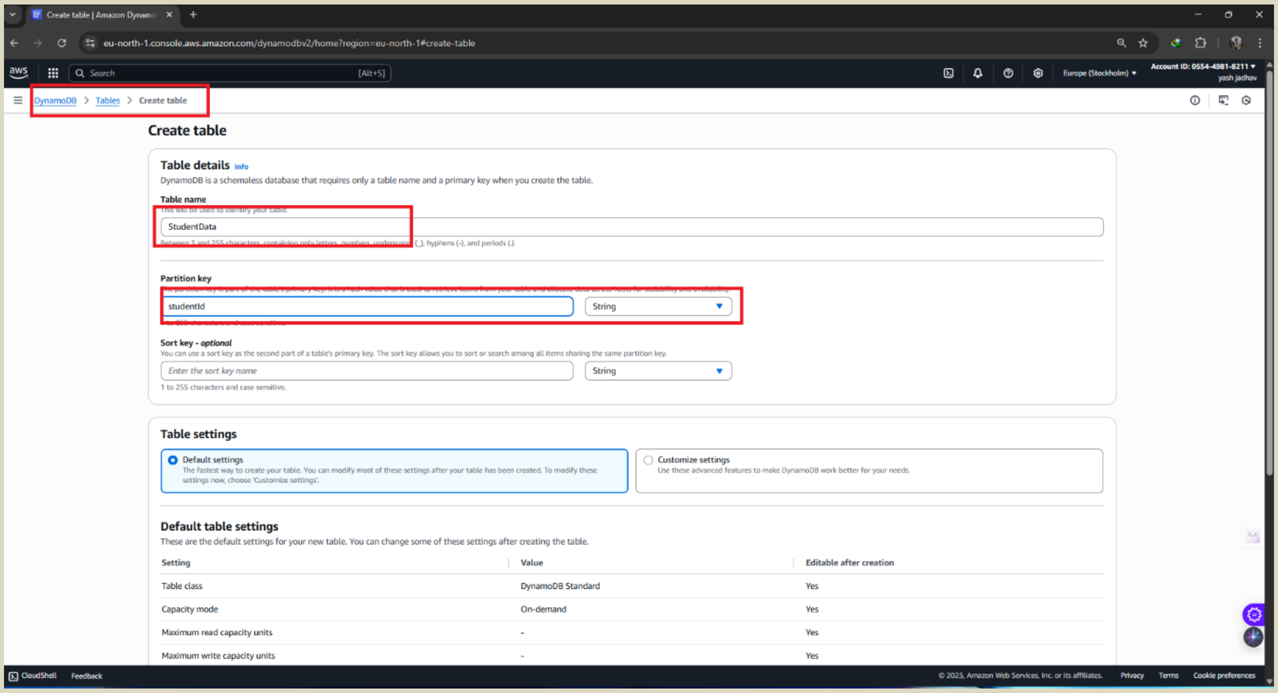
Step 1: Backend Setup (Amazon DynamoDB and AWS Lambda with Python)

This section details the configuration of the core backend components responsible for data storage and business logic.

1.1. Create an Amazon DynamoDB Table

The DynamoDB table will serve as the persistent storage for student records.

1. **Navigate to DynamoDB**: Access the AWS Management Console and open the DynamoDB service.
2. **Initiate Table Creation**: Click "**Create table**".---(Figure 2)


Create table | Amazon Dynamo
eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#create-table
Search [Alt+S]
Europe (Stockholm)
Account ID: 0554-4981-8211
yash jadhav
DynamoDB > Tables > Create table
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

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

studentId | String

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.

Table settings

● Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.

○ Customize settings
Use these advanced features to make DynamoDB work better for your needs.

Default table settings

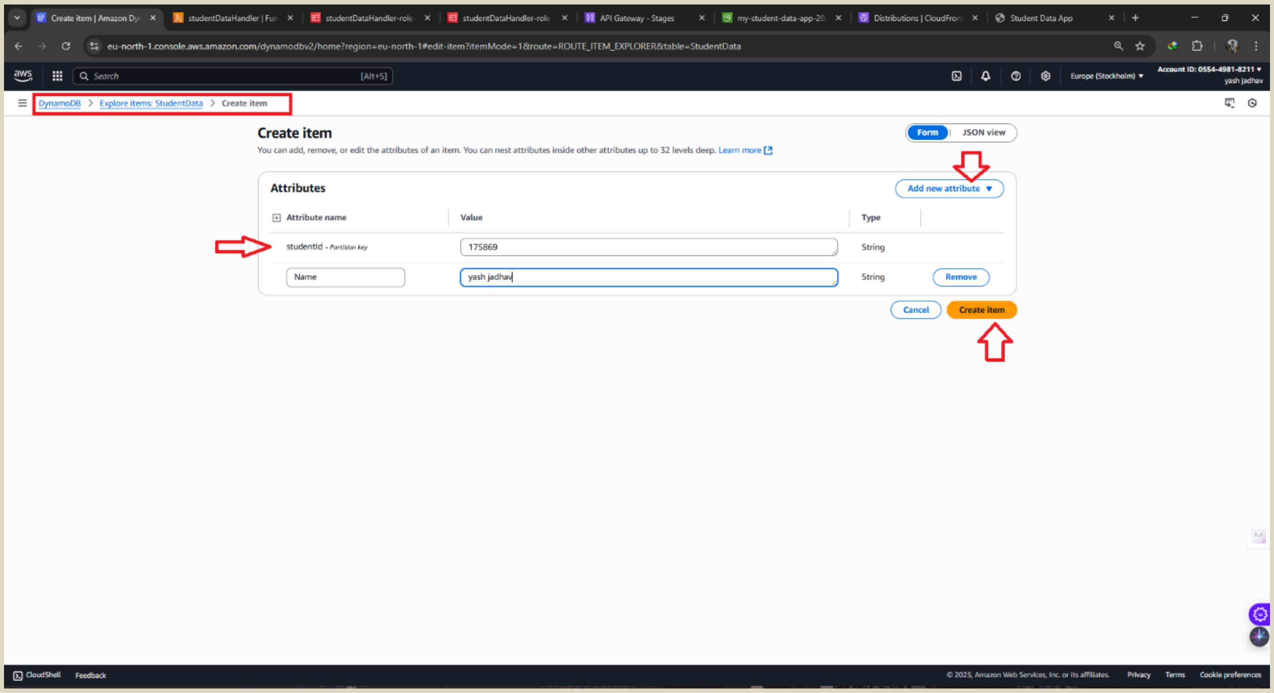
These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Table class	DynamoDB Standard	Yes
Capacity mode	On-demand	Yes
Maximum read capacity units	-	Yes
Maximum write capacity units	-	Yes

Figure 2 Create DynamoDB

3. **Define Table Properties**:
   - **Table name**: Enter a descriptive name, such as StudentData.
   - **Partition key**: Specify studentId (String type) as the primary key. This uniquely identifies each student record. ---(Figure 3)
   - **Sort key (Optional)**: For more complex queries, a Sort Key can be added (e.g., courseId if students can enroll in multiple courses, and you want to retrieve courses for a specific student efficiently). For this guide, it's optional.
   - **Default settings**: Retain the default settings for initial setup.

4. **Finalize Table Creation**: Click "**Create table**".


eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#edit-item?itemMode=1&route=ROUTE_ITEM_EXPLORER&table=StudentData

DynamoDB > Explore items: StudentData > Create item

Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. Learn more

Attributes

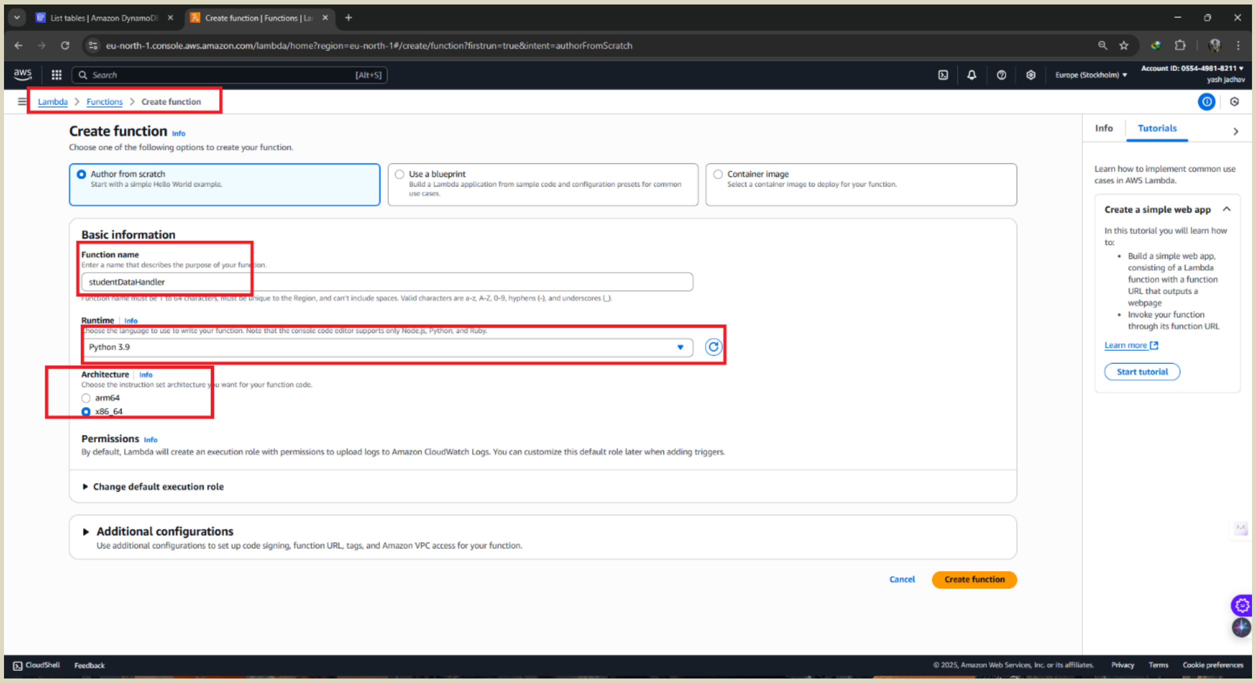
Attribute name	Value	Type
studentId - *Partition key*	175869	String
Name	yash jadhav	String

Figure 3 Add table and Create item in DynamoDB

1.2. Create an AWS Lambda Function

The Lambda function will execute the Python code for interacting with DynamoDB.

1. **Navigate to Lambda**: Access the AWS Management Console and open the Lambda service.
2. **Initiate Function Creation**: Click "**Create function**".
3. **Configure Basic Information**:
   - **Authoring option**: Select "**Author from scratch**".
   - **Function name**: Provide a clear name, e.g., studentDataHandler.
   - **Runtime**: Choose "**Python 3.9**" (or a newer supported version) to execute your Python code.
   - **Architecture**: Select x86_64 for general compatibility.
   - **Execution role**: Under "Change default execution role", you can create a new role or select an existing one. A new role typically gets Basic Lambda permissions which includes CloudWatch Logs permissions.
4. **Finalize Function Creation**: Click "**Create function**". ---(Figure 4)



Basic information

Function name: studentDataHandler

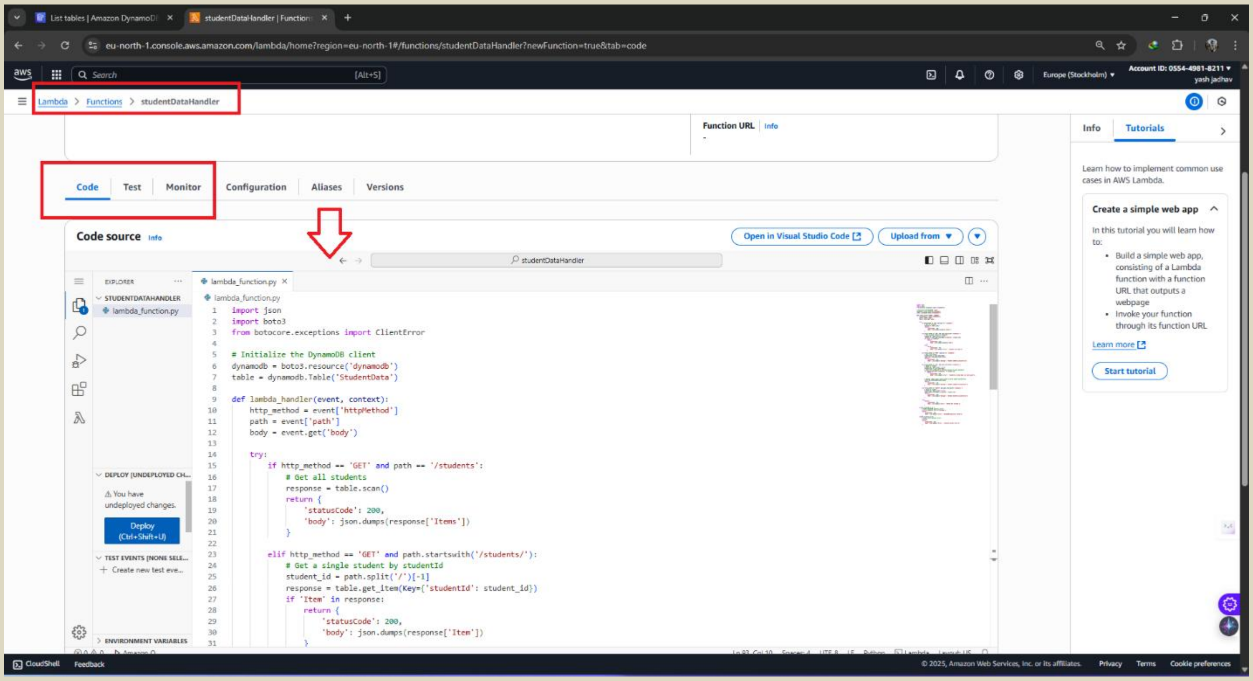
Runtime: Python 3.9

Architecture: x86_64

Figure 4 Create a Lambda Function

1.3. Develop Python Lambda Code for CRUD Operations

Replace the default lambda handler code in the Lambda console's code editor with the following Python code. This function processes HTTP requests from API Gateway to perform CRUD operations on the StudentData DynamoDB table.Copy the code in source code of lambda function ---(Figure below)



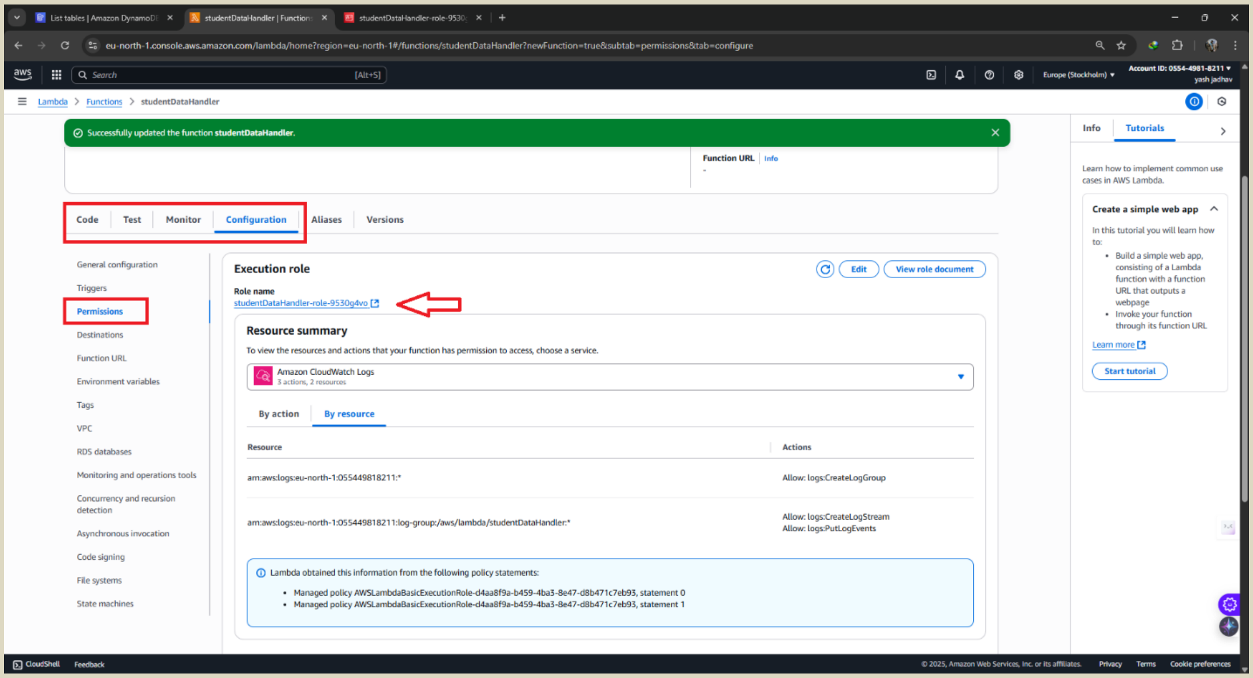
```python
import json
import boto3
from botocore.exceptions import ClientError

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('StudentData')

def lambda_handler(event, context):
    http_method = event['httpMethod']
    path = event['path']
    body = event.get('body')

    try:
        if http_method == 'GET' and path == '/students':
            # Get all students
            response = table.scan()
            return {
                'statusCode': 200,
                'body': json.dumps(response['Items'])
            }

        elif http_method == 'GET' and path.startswith('/students/'):
            # Get a single student by studentId
            student_id = path.split('/')[-1]
            response = table.get_item(Key={'studentId': student_id})
            if 'Item' in response:
                return {
                    'statusCode': 200,
                    'body': json.dumps(response['Item'])
                }
```

```
import json
import boto3
from botocore.exceptions import ClientError

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('StudentData')

def lambda_handler(event, context):
    http_method = event['httpMethod']
    path = event['path']
    body = event.get('body')

    # Define a standard response dictionary with headers
    response_headers = {
        'Content-Type': 'application/json',
        'Access-Control-Allow-Origin': '*'
    }

    try:
        if http_method == 'GET' and path == '/students':
            # Get all students
            response = table.scan()
            return {
                'statusCode': 200,
                'headers': response_headers,
                'body': json.dumps(response['Items'])
            }

        elif http_method == 'GET' and path.startswith('/students/'):
            # Get a single student by studentId
            student_id = path.split('/')[-1]
            response = table.get_item(Key={'studentId': student_id})
            if 'Item' in response:
                return {
                    'statusCode': 200,
                    'headers': response_headers,
                    'body': json.dumps(response['Item'])
                }
            else:
                return {
                    'statusCode': 404,
                    'headers': response_headers,
                    'body': json.dumps({'error': 'Student not found'})
                }

        elif http_method == 'POST' and path == '/students':
            # Add a new student
            student_data = json.loads(body)
            table.put_item(Item=student_data)
            return {
                'statusCode': 201,
                'headers': response_headers,
                'body': json.dumps({'message': 'Student added successfully'})
            }

        elif http_method == 'PUT' and path.startswith('/students/'):
            # Update a student
            student_id = path.split('/')[-1]
            student_data = json.loads(body)
            if student_data.get('studentId') != student_id:
                return {
                    'statusCode': 400,
                    'headers': response_headers,
                    'body': json.dumps({'error': 'studentId in body does not match path'})
                }

            table.put_item(Item=student_data)
            return {
                'statusCode': 200,
                'headers': response_headers,
                'body': json.dumps({'message': 'Student updated successfully'})
            }

        elif http_method == 'DELETE' and path.startswith('/students/'):
            # Delete a student
            student_id = path.split('/')[-1]
            table.delete_item(Key={'studentId': student_id})
            return {
                'statusCode': 200,
                'headers': response_headers,
                'body': json.dumps({'message': 'Student deleted successfully'})
            }

        else:
            return {
                'statusCode': 405,
                'headers': response_headers,
                'body': json.dumps({'error': 'Method Not Allowed'})
            }

    except ClientError as e:
        print(e.response['Error']['Message'])
        return {
            'statusCode': 500,
            'headers': response_headers,
            'body': json.dumps({'error': 'DynamoDB operation failed'})
        }
    except Exception as e:
        print(e)
        return {
            'statusCode': 500,
            'headers': response_headers,
            'body': json.dumps({'error': 'Internal server error'})
        }
```

1.4. Configure IAM Permissions for AWS Lambda

The Lambda function requires appropriate permissions to interact with DynamoDB and log to CloudWatch.

1. **Access Lambda Function Permissions**: Navigate to the studentDataHandler function in the Lambda console, then click the "**Configuration**" tab and select "**Permissions**".—(Figure 5)
2. **Edit Execution Role**: Click on the **Role name** (e.g., studentDataHandler-role-xxxxxx) to open the IAM console for that role.



eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#/functions/studentDataHandler?newFunction=true&subtab=permissions&tab=configure

Lambda > Functions > studentDataHandler

Successfully updated the function **studentDataHandler**.

Execution role

Role name
studentDataHandler-role-9530g4vo

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs
3 actions, 2 resources

Resource	Actions
arn:aws:logs:eu-north-1:055449818211:*	Allow: logs:CreateLogGroup
arn:aws:logs:eu-north-1:055449818211:log-group:/aws/lambda/studentDataHandler:*	Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Lambda obtained this information from the following policy statements:

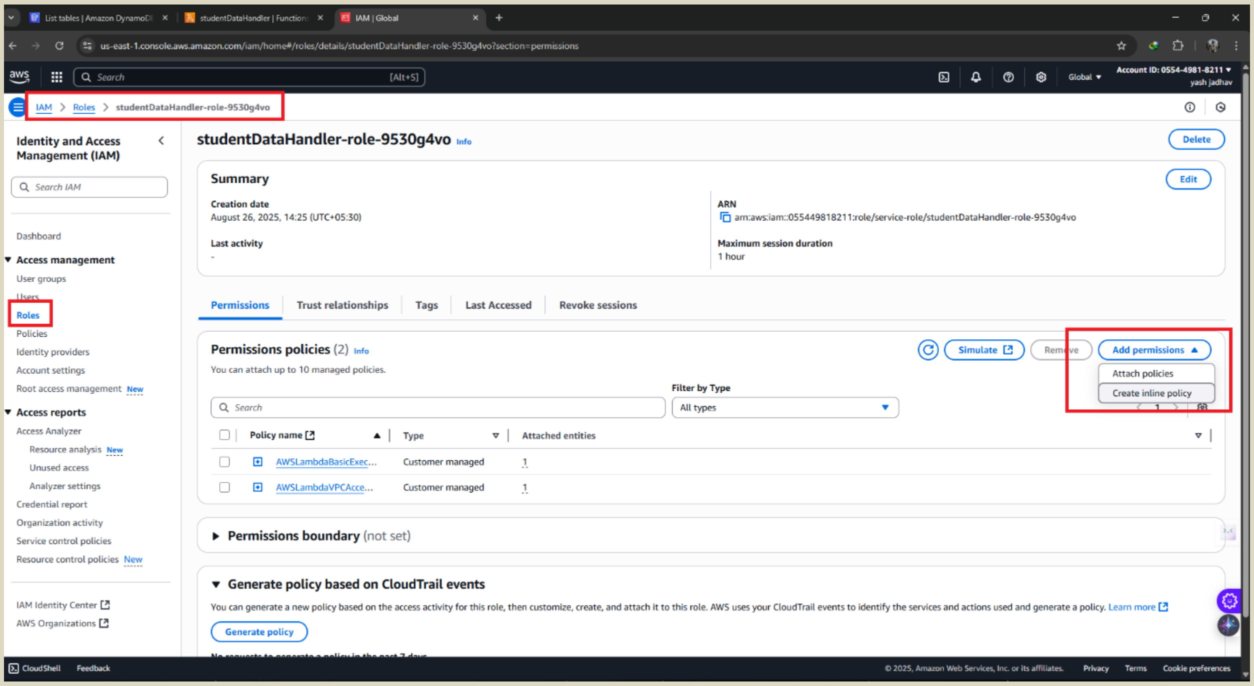
- Managed policy AWSLambdaBasicExecutionRole-d4aa8f9a-b459-4ba3-8e47-d8b471c7eb93, statement 0
- Managed policy AWSLambdaBasicExecutionRole-d4aa8f9a-b459-4ba3-8e47-d8b471c7eb93, statement 1

Figure 5 Lambda Function Permissions

3. **Add Inline Policy**:---(Figure 6)
    - In the IAM role, click "**Add permissions**" -> "**Create inline policy**".
    - **Service**: Choose DynamoDB.
    - **Actions**: Under "Access level", expand Write, Read, and Permissions management.

Select the following actions:

- GetItem
- PutItem
- DeleteItem
- Scan
- UpdateItem (although put_item is used in the example for updates, UpdateItem is more granular for partial updates and good to include for future expansion).


IAM > Roles > studentDataHandler-role-9530g4vo

Identity and Access Management (IAM)

Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report
Organization activity
Service control policies
Resource control policies New
IAM Identity Center
AWS Organizations

studentDataHandler-role-9530g4vo Info

Summary

Creation date
August 26, 2025, 14:25 (UTC+05:30)

ARN
arn:aws:iam::055449818211:role/service-role/studentDataHandler-role-9530g4vo

Last activity
-

Maximum session duration
1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (2) Info

You can attach up to 10 managed policies.

Add permissions: Attach policies / Create inline policy

Policy name	Type	Attached entities
AWSLambdaBasicExec...	Customer managed	1
AWSLambdaVPCAcce...	Customer managed	1

Permissions boundary (not set)

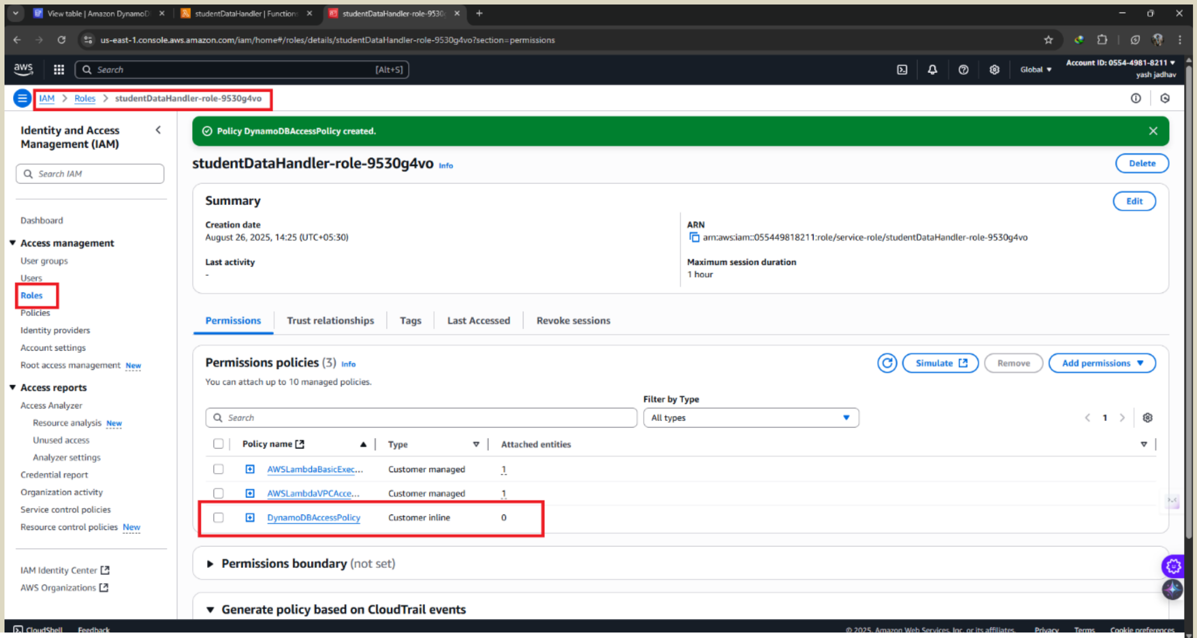
Generate policy based on CloudTrail events

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

Generate policy

© 2025, Amazon Web Services, Inc. or its affiliates.

Figure 7 Create inline policy


IAM > Roles > studentDataHandler-role-9530g4vo

Policy DynamoDBAccessPolicy created.

studentDataHandler-role-9530g4vo Info

Summary

Creation date
August 26, 2025, 14:25 (UTC+05:30)

ARN
arn:aws:iam::055449818211:role/service-role/studentDataHandler-role-9530g4vo

Last activity
-

Maximum session duration
1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

Permissions policies (3) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AWSLambdaBasicExec...	Customer managed	1
AWSLambdaVPCAcce...	Customer managed	1
DynamoDBAccessPolicy	Customer inline	0

Permissions boundary (not set)

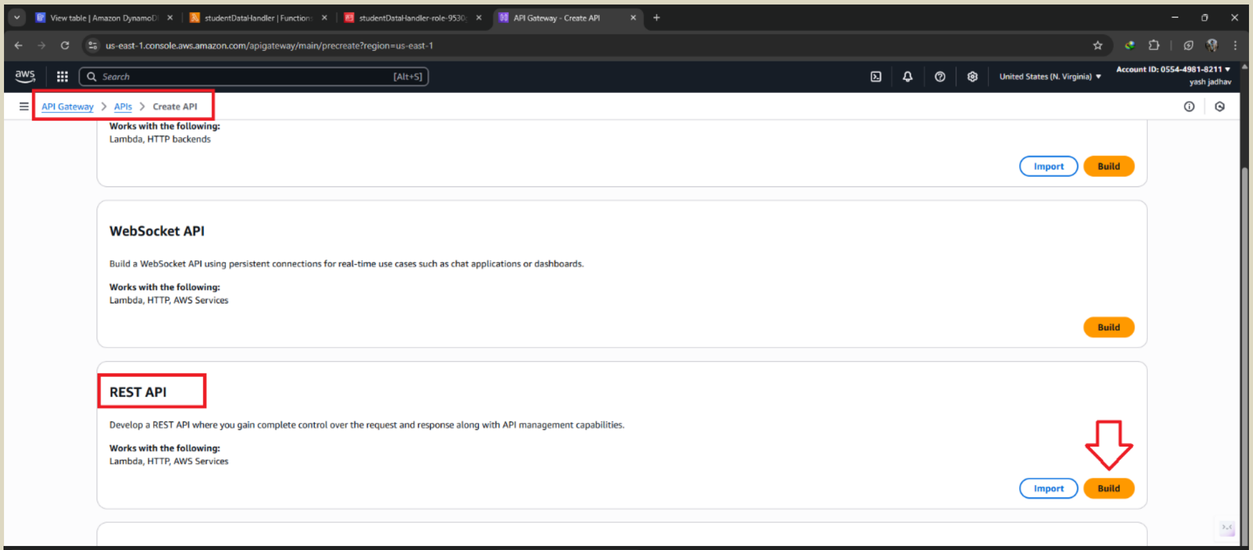
Generate policy based on CloudTrail events

Figure 6 created a policy

Step 2: Set up Amazon API Gateway

API Gateway provides the public HTTP endpoint for clients to interact with the Lambda backend.

1. **Navigate to API Gateway**: Access the AWS Management Console and open the API Gateway service.
2. **Create a New REST API**:
   - Click "**Build**" under "**REST API**".
   - Choose "**New API**".---(Figure 8)
   - **API name**: Enter a name, e.g., StudentDataAPI.
   - **Endpoint Type**: Select Regional.
   - Click "**Create API**".



us-east-1.console.aws.amazon.com/apigateway/main/precreate?region=us-east-1

API Gateway > APIs > Create API

Works with the following:
Lambda, HTTP backends

WebSocket API

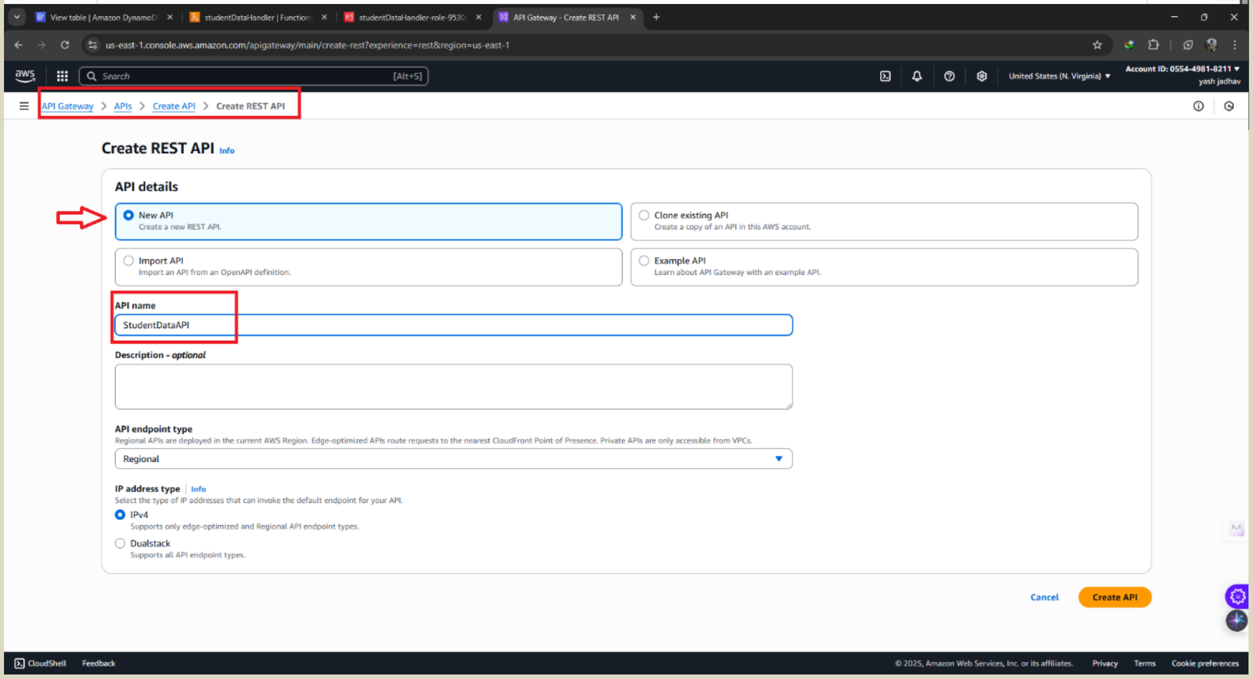
Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services

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



us-east-1.console.aws.amazon.com/apigateway/main/create-rest?experience=rest&region=us-east-1

API Gateway > APIs > Create API > Create REST 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: StudentDataAPI

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
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.

2.1. Create Resources

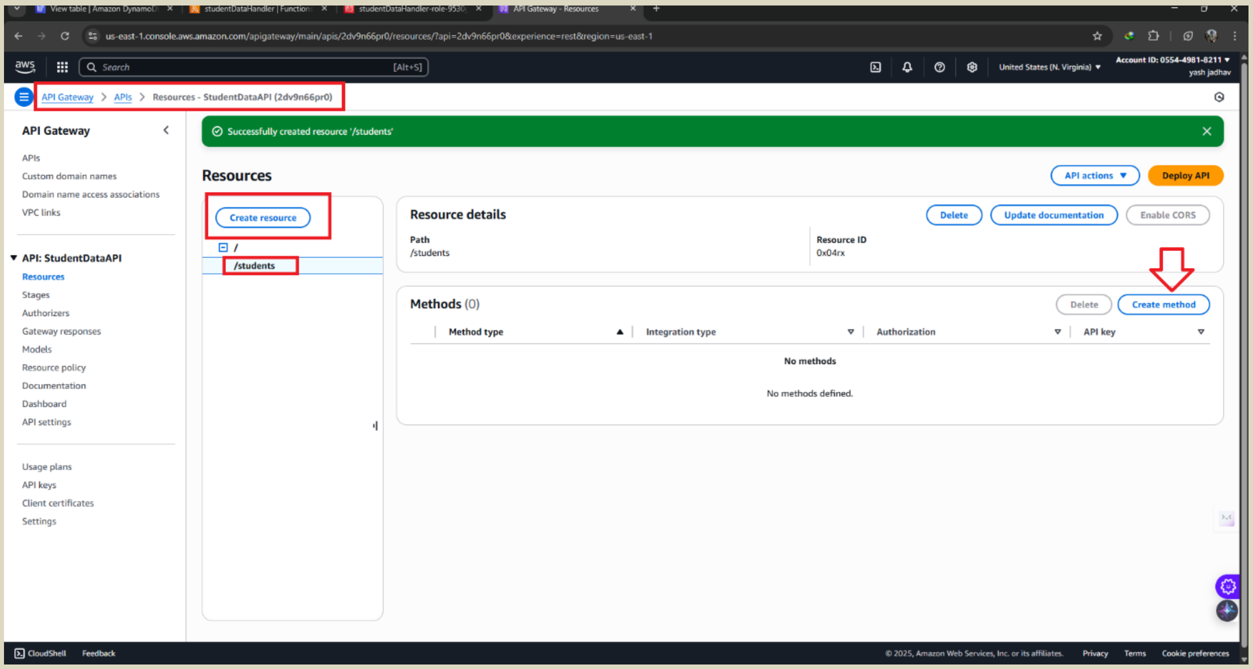
Resources define the paths for your API endpoints.

1. **Create /students Resource**:
   - Select the root (/) resource.
   - Click "**Actions**" -> "**Create Resource**".
   - **Resource Name**: Type students.
   - **Resource Path**: It will auto-populate as /students.
   - Click "**Create Resource**".
2. **Create /students/{studentId} Resource**: This path will be used for operations on individual student records.
   - Select the /students resource you just created.
   - Click "**Actions**" -> "**Create Resource**".
   - **Resource Name**: Type studentId.
   - **Resource Path**: It will auto-populate as /{studentId}.
   - Check "**Use Lambda Proxy integration**".
   - Click "**Create Resource**".

2.2. Create Methods for Resources

Methods (GET, POST, PUT, DELETE) define the HTTP operations for your resources. We will use Lambda Proxy Integration for simplicity and flexibility.

1. **For /students Resource**:
   - Select the /students resource.
   - Click "**Actions**" -> "**Create Method**".
   - Choose GET.
   - **Integration type**: Select Lambda Function.
   - Check "**Use Lambda Proxy integration**".
   - **Lambda Region**: Select the region where your Lambda function resides.
   - **Lambda Function**: Enter studentDataHandler and click the checkmark.
   - A pop-up will appear asking to add permission to Lambda function. Click "**OK**".
   - Repeat the process for POST method on /students resource, integrating it with studentDataHandler.
2. **For /students/{studentId} Resource**:
   - Select the /students/{studentId} resource.
   - Click "**Actions**" -> "**Create Method**".
   - Choose GET. Configure with studentDataHandler using Lambda Proxy Integration.
   - Repeat for PUT and DELETE methods, also integrating with studentDataHandler using Lambda Proxy Integration.---(Figure 8)



Successfully created resource '/students'

Path
/students

Resource ID
0x04rx

No methods

No methods defined.

Figure 8 API RESOURSE

2.3. Enable CORS for the API

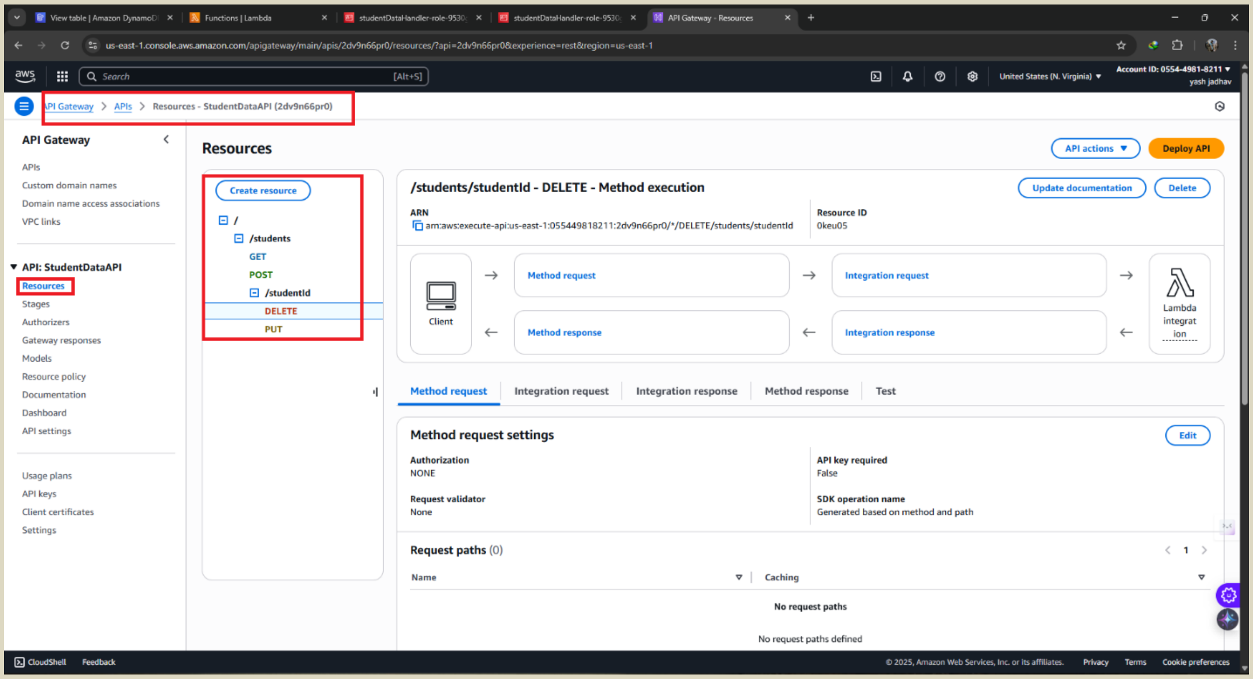
For a web application hosted on a different domain (S3/CloudFront), Cross-Origin Resource Sharing (CORS) must be enabled.

1. Select the /students resource (and if needed, the root / resource and /students/{studentId} resource).
2. Click "**Actions**" -> "**Enable CORS**".
3. For a simple setup, you can keep the default settings (allowing GET,HEAD,POST,PUT,DELETE,OPTIONS from * origin). In a production environment, restrict the Access-Control-Allow-Origin to your CloudFront domain.
4. Click "**Enable CORS and replace existing CORS headers**".

2.4. Deploy the API

Deployment makes your API accessible via a public URL.

1. Click "**Actions**" -> "**Deploy API**".
2. **Deployment stage**: Select "[New Stage]".
3. **Stage name**: Give it a name, e.g., prod.
4. Click "**Deploy**".
5. After deployment, note the **Invoke URL**. This URL, combined with your resource paths (e.g., https://xxxxxx.execute-api.region.amazonaws.com/prod/students), will be used by your frontend.---(Figure 9)



Resources

/students/studentId - DELETE - Method execution

ARN
arn:aws:execute-api:us-east-1:055449818211:2dv9n66pr0/*/DELETE/students/studentId

Resource ID
0keu05

Method request settings

Authorization
NONE

API key required
False

Request validator
None

SDK operation name
Generated based on method and path

Request paths (0)

No request paths

No request paths defined

Figure 9 Methods and Deploy

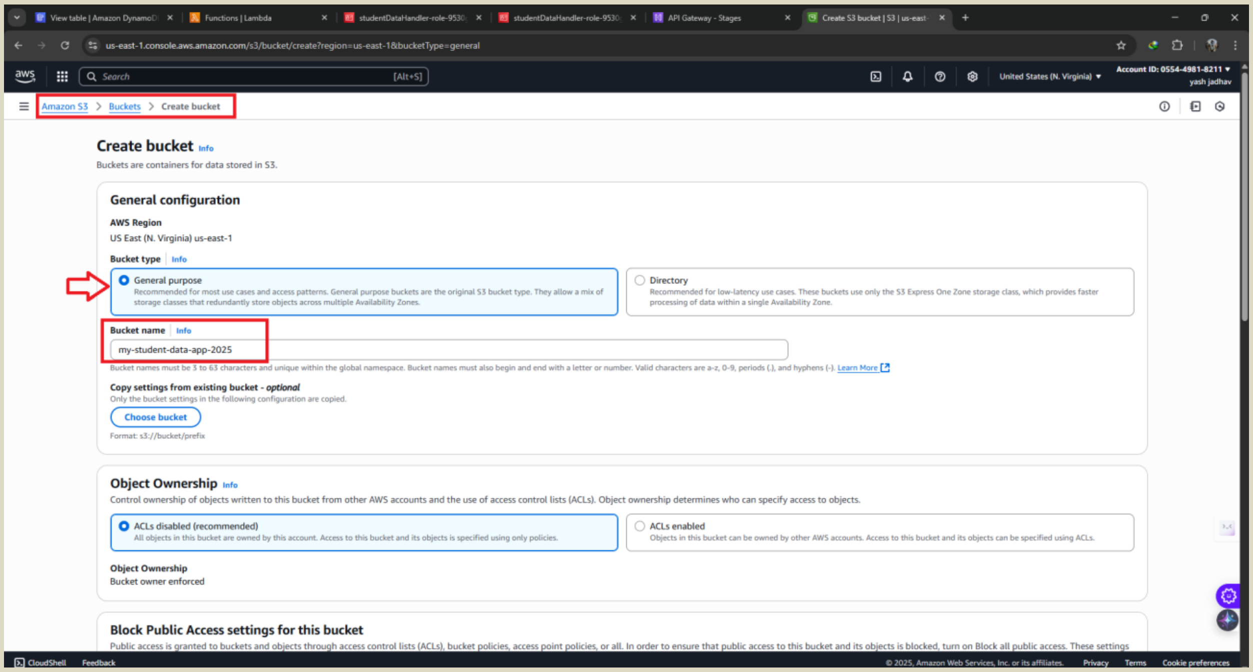
Step 3: Frontend Setup (Amazon S3 and Amazon CloudFront)

This section covers hosting the static frontend assets and distributing them globally.

3.1. Create an Amazon S3 Bucket for Static Website Hosting

The S3 bucket will store your HTML, CSS, and JavaScript files.

1. **Navigate to S3**: Access the AWS Management Console and open the S3 service.
2. **Initiate Bucket Creation**: Click "**Create bucket**".
3. **Configure Bucket Properties**:
   - **Bucket name**: Choose a globally unique name, e.g., my-student-data-app-2025-frontend.
   - **AWS Region**: Select a region geographically close to your users or primary backend services.
   - **Object Ownership**: Keep ACLs disabled (recommended).
   - **Block Public Access settings for this bucket**: **Uncheck** "Block all public access" and acknowledge the warning. This is crucial for enabling public access required by static website hosting. *Note: For a more secure approach, especially when using CloudFront, you would typically keep S3 private and use an Origin Access Control (OAC) with CloudFront to restrict direct S3 access.*
4. **Finalize Bucket Creation**: Click "**Create bucket**".—(Figure 10)


us-east-1.console.aws.amazon.com/s3/bucket/create?region=us-east-1&bucketType=general

Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region
US East (N. Virginia) us-east-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
my-student-data-app-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.
Choose bucket
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.

[✓ 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

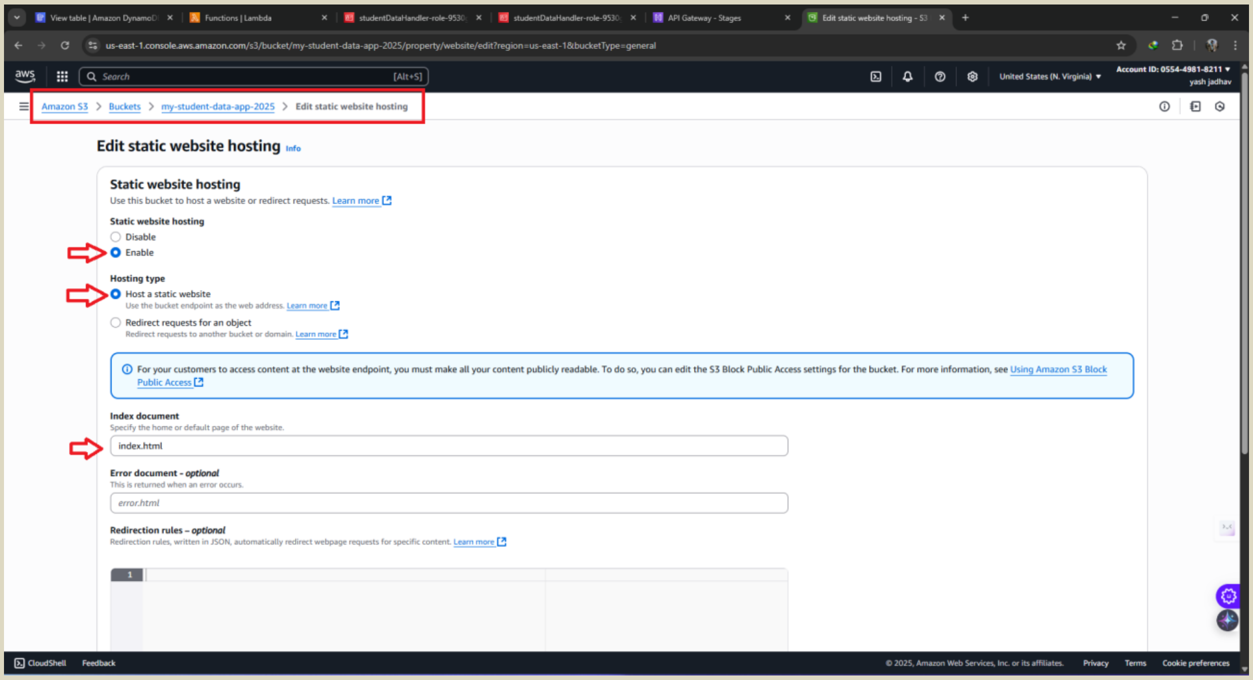
Block Public Access settings for this bucket

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
View table | Amazon DynamoD
Functions | Lambda
studentDataHandler-role-9530
studentDataHandler-role-9530
API Gateway - Stages
Create S3 bucket | S3 | us-east-
Search [Alt+S]
United States (N. Virginia)
Account ID: 0554-4981-8211
yash jadhav
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Figure 10 S3 bucket

3.2. Enable Static Website Hosting on the S3 Bucket

1. **Access Bucket Properties**: Select your newly created S3 bucket and go to the "**Properties**" tab.
2. **Configure Static Website Hosting**:
   - Scroll down to "**Static website hosting**" and click "**Edit**".
   - Select "**Enable**".
   - **Index document**: Enter index.html.—(Figure 11)
   - **Error document (Optional)**: Enter error.html for custom error pages.
3. **Save Changes**: Click "**Save changes**". Note the **Bucket website endpoint** from this section (e.g., http://my-student-data-app-2025-frontend.s3-website.region.amazonaws.com).


View table | Amazon DynamoD
Functions | Lambda
studentDataHandler-role-9530
studentDataHandler-role-9530
API Gateway - Stages
Edit static website hosting - S3
us-east-1.console.aws.amazon.com/s3/bucket/my-student-data-app-2025/property/website/edit?region=us-east-1&bucketType=general
Search [Alt+S]
United States (N. Virginia)
Account ID: 0554-4981-8211
yash jadhav
Amazon S3 > Buckets > my-student-data-app-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

Figure 11 Enable Static Website Hosting on the S3 Bucket

3.3. Create and Upload Frontend Assets

Create your static web files (index.html, styles.css, script.js if separate) and upload them to the S3 bucket.

***index.html (Example)**:

```
<!DOCTYPE html>
<link rel="stylesheet" href="style.css">
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Data App</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Student Data Management</h1>
  <div id="student-list"></div>
  <script>
     const API_URL = 'https://bsf6c7ib6g.execute-api.eu-north-
1.amazonaws.com/prod/students';{YOUR API}

    async function fetchStudents() {
      const response = await fetch(API_URL);
      const students = await response.json();
      const listElement = document.getElementById('student-list');
      listElement.innerHTML = ''; // Clear previous data
      students.forEach(student => {
        const studentDiv = document.createElement('div');
        studentDiv.textContent = `ID: ${student.studentId}, Name: ${student.name}, Percentage:
${student.percentage}`;
        listElement.appendChild(studentDiv);
      });
    }

    fetchStudents();
  </script>
</body>
</html>
```

***styles.css (Example)**:

```
/* General body and typography */
body {
    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Helvetica, Arial, sans-
serif;
    background-color: #f4f7f6;
    color: #333;
    line-height: 1.6;
    margin: 0;
    padding: 20px;
    display: flex;
    flex-direction: column;
    align-items: center;
}

/* Header styling */
h1 {
    color: #2c3e50;
    text-align: center;
    margin-bottom: 30px;
    font-weight: 600;
    letter-spacing: 1px;
    border-bottom: 2px solid #3498db;
    padding-bottom: 10px;
}

/* Student list container */
#student-list {
    width: 100%;
    max-width: 600px;
    background-color: #ffffff;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);
    padding: 20px;
}

/* Individual student item styling */
#student-list div {
    padding: 15px;
    border-bottom: 1px solid #e9ecef;
    transition: background-color 0.3s ease;
    font-size: 1rem;
    color: #555;
}

#student-list div:last-child {
    border-bottom: none;
}

#student-list div:hover {
    background-color: #e8f4f8;
}

/* Responsive design */
@media (max-width: 768px) {
    body {
        padding: 10px;
    }

    h1 {
        font-size: 1.5rem;
    }

    #student-list {
        padding: 10px;
    }
}
```

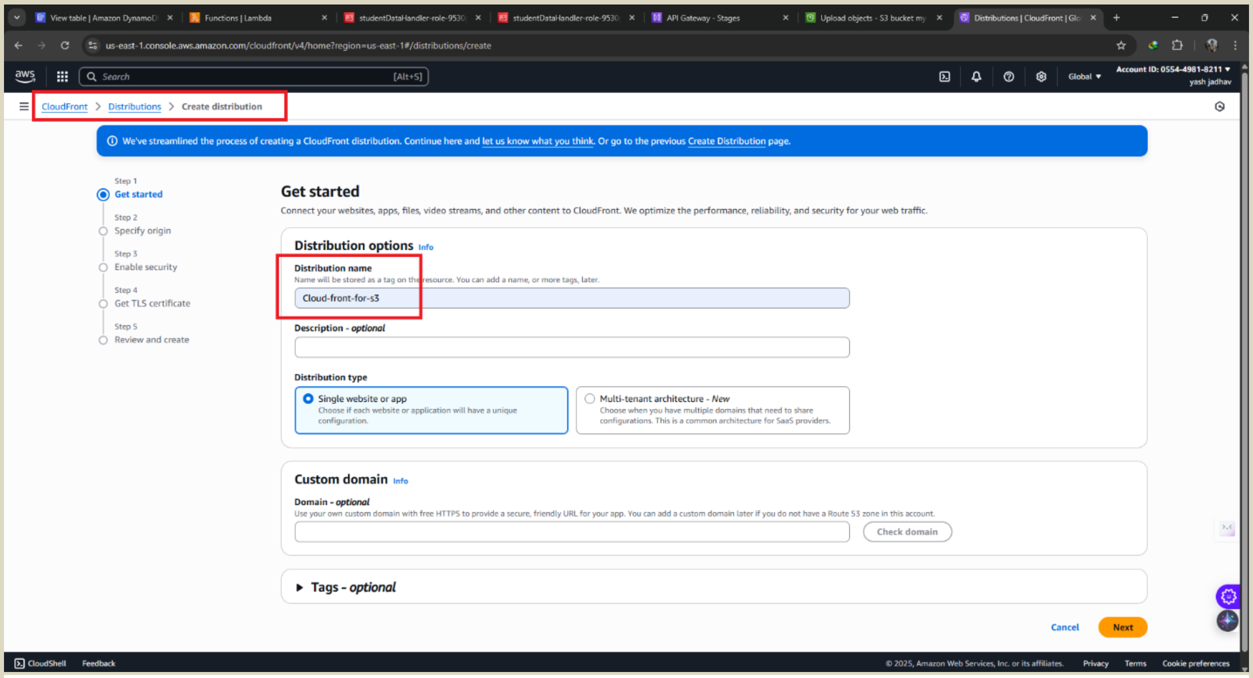
1. **Upload Assets**: Upload index.html and styles.css to your S3 bucket. Ensure the API_BASE_URL in index.html is updated with your API Gateway Invoke URL.

3.4. Create an Amazon CloudFront Distribution

CloudFront will serve your static content securely and efficiently.

1. **Navigate to CloudFront**: Access the AWS Management Console and open the CloudFront service.
2. **Initiate Distribution Creation**: Click "**Create Distribution**".
3. **Configure Origin Settings**:
    - **Origin domain**: Select your S3 bucket's **Bucket website endpoint** from the dropdown list (e.g., http://my-student-data-app-2025-frontend.s3-website.region.amazonaws.com). *Do NOT select the S3 REST API endpoint that CloudFront automatically suggests if you want S3 static website hosting features like index documents.*
    - CloudFront will automatically populate the **Origin ID**.—(Figure 13)
4. **Default Cache Behavior Settings**:
    - **Viewer protocol policy**: Select Redirect HTTP to HTTPS for secure access.
    - **Allowed HTTP methods**: Ensure GET, HEAD, OPTIONS are allowed (or more if your frontend uses other methods for static assets).
    - **Cache policy**: Use CachingOptimized or create a custom policy.
5. **Distribution Settings**:
    - **Default root object**: Enter index.html. This ensures that when users access your domain, index.html is served by default.—(Figure 14)
    - **Price class**: Choose based on your geographic reach and cost preferences.
    - **Web Application Firewall (WAF) (Optional)**: Consider enabling AWS WAF for additional protection against common web exploits.

6. **Finalize Distribution Creation**: Click "**Create Distribution**". This process can take several minutes to complete.
7. **Distribution Domain Name**: Once deployed, note the **Distribution domain name** (e.g., d1234abcd.cloudfront.net). This is the secure URL for your web application.—(Figure 14)



us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/create

CloudFront > Distributions > Create distribution

We've streamlined the process of creating a CloudFront distribution. Continue here and let us know what you think. Or go to the previous Create Distribution page.

Step 1 Get started
Step 2 Specify origin
Step 3 Enable security
Step 4 Get TLS certificate
Step 5 Review and create

Get started

Connect your websites, apps, files, video streams, and other content to CloudFront. We optimize the performance, reliability, and security for your web traffic.

Distribution options Info

Distribution name
Name will be stored as a tag on the resource. You can add a name, or more tags, later.
Cloud-front-for-s3

Description - *optional*

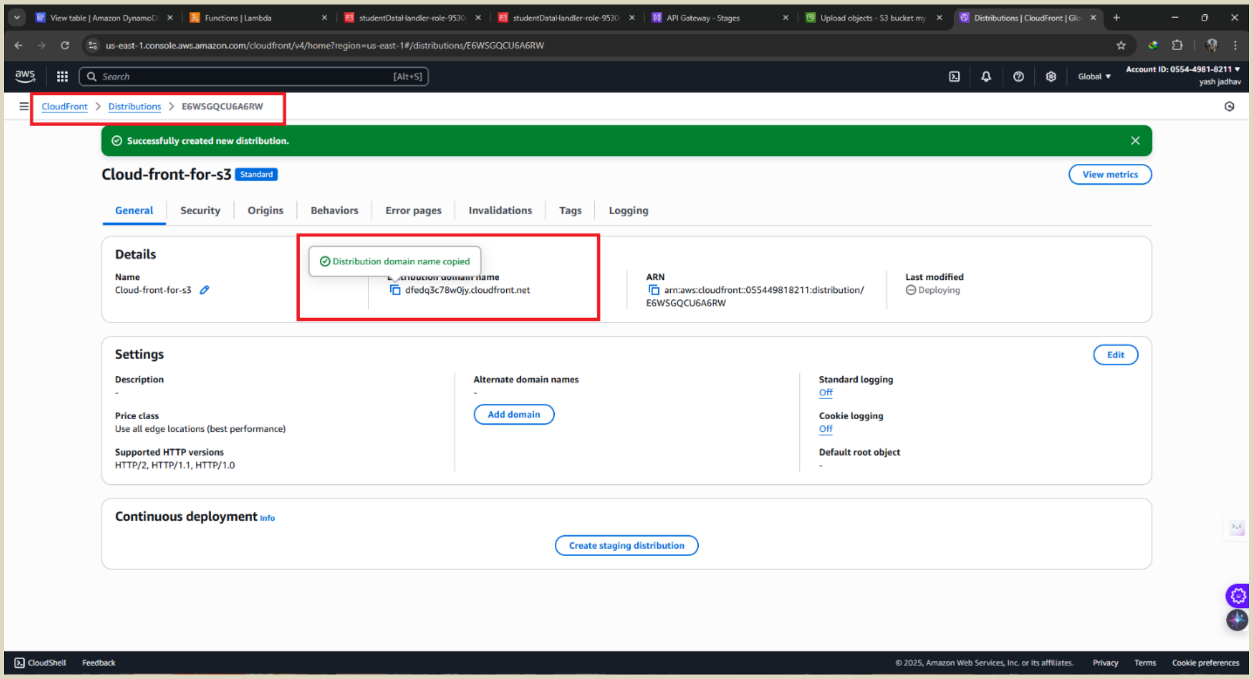
Distribution type
Single website or app — Choose if each website or application will have a unique configuration. [✓]
Multi-tenant architecture - *New* — Choose when you have multiple domains that need to share configurations. This is a common architecture for SaaS providers.

Custom domain Info

Domain - *optional*
Use your own custom domain with free HTTPS to provide a secure, friendly URL for your app. You can add a custom domain later if you do not have a Route 53 zone in this account.

Tags - *optional*

Figure 14 Add s3 and root object as index.html



us-east-1.console.aws.amazon.com/cloudfront/v4/home?region=us-east-1#/distributions/E6WSGQCU6A6RW

CloudFront > Distributions > E6WSGQCU6A6RW

Successfully created new distribution.

Cloud-front-for-s3 Standard

General | Security | Origins | Behaviors | Error pages | Invalidations | Tags | Logging

Details

Name: Cloud-front-for-s3
Distribution domain name copied
Distribution domain name: dfedq3c78w0jy.cloudfront.net
ARN: arn:aws:cloudfront::055449818211:distribution/E6WSGQCU6A6RW
Last modified: Deploying

Settings

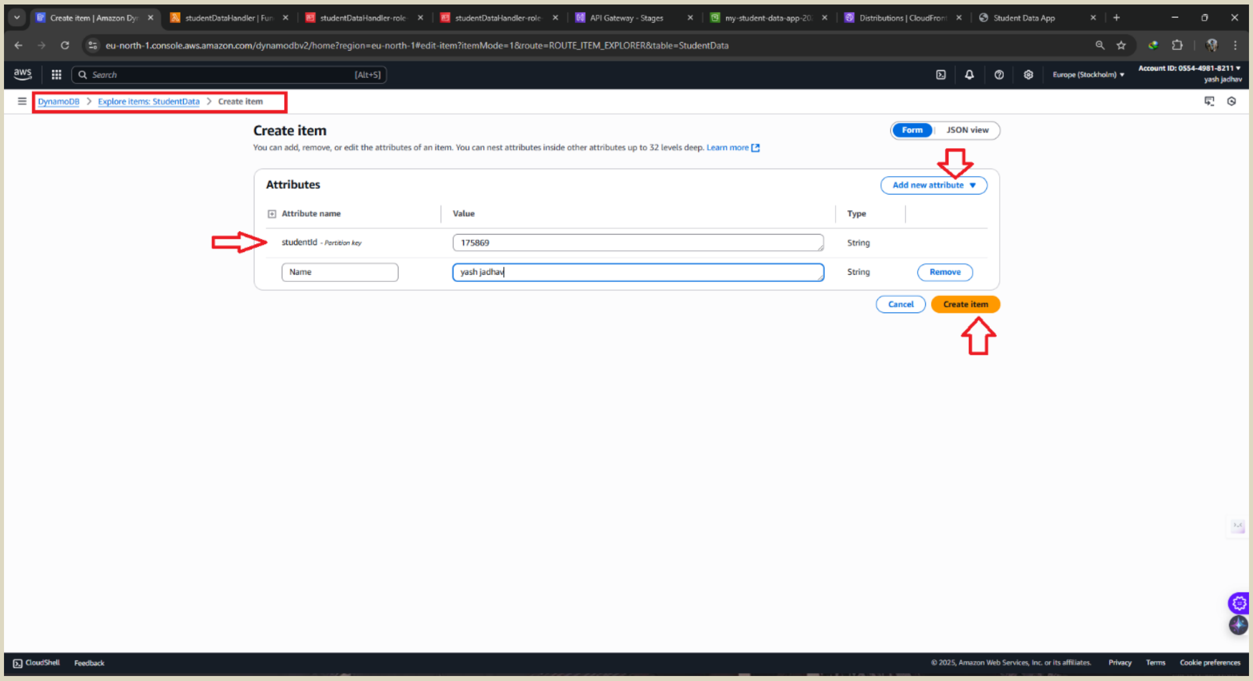
Description: -
Price class: Use all edge locations (best performance)
Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0
Alternate domain names: -
Standard logging: Off
Cookie logging: Off
Default root object: -

Continuous deployment Info

View metrics
Edit
Add domain
Create staging distribution
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Figure 13 Add your S3 ,API in cloud front and change root object too index.html

3.5. Add Data to your Dynamodb and Paste your ARN in Browers



eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#edit-item?itemMode=1&route=ROUTE_ITEM_EXPLORER&table=StudentData

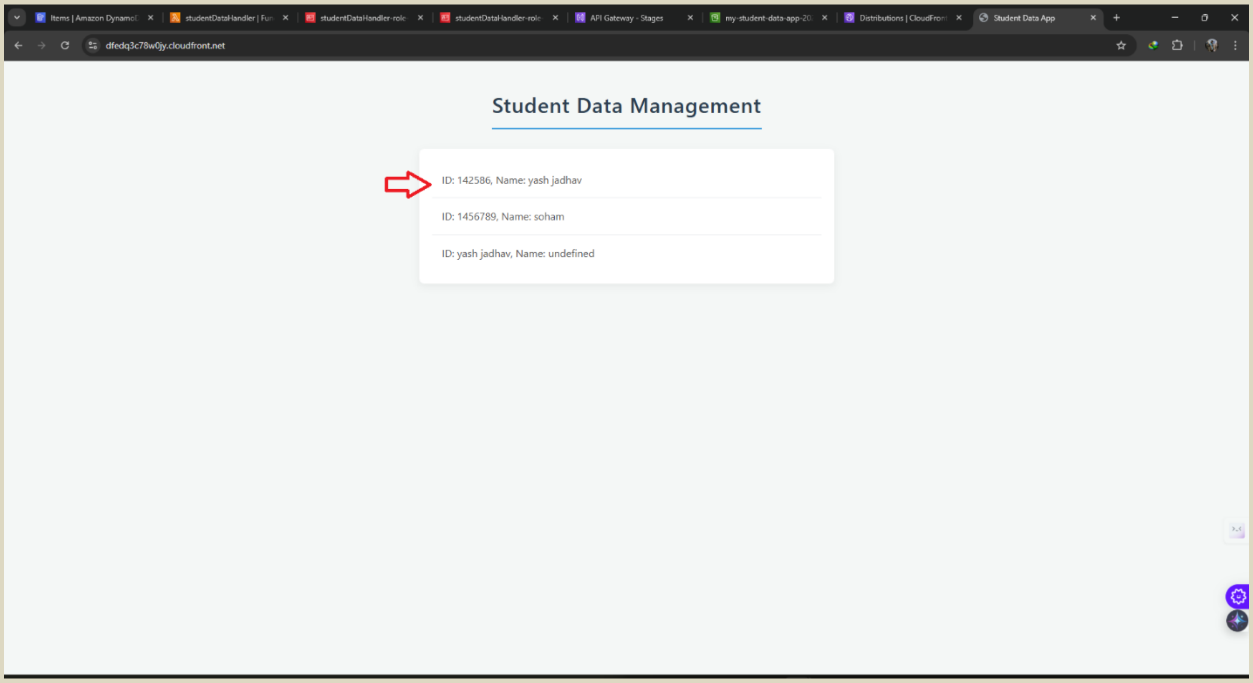
DynamoDB > Explore items: StudentData > Create item

Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. Learn more

Attributes

Attribute name	Value	Type
studentId - Partition key	175869	String
Name	yash jadhav	String



dfedq3c78w0jy.cloudfront.net

Student Data Management

ID: 142586, Name: yash jadhav

ID: 1456789, Name: soham

ID: yash jadhav, Name: undefined

Enhanced Security and Best Practices

Monitoring and Logging

- **AWS CloudWatch**: All AWS services used (Lambda, API Gateway, DynamoDB) integrate with CloudWatch for logging and metrics.
    - **Lambda Logs**: Lambda automatically sends function logs to CloudWatch Logs. Configure custom log groups and metrics for advanced analysis.
    - **API Gateway Logs**: Enable CloudWatch Logs for API Gateway to capture detailed request and response information, including errors and access logs.
    - **CloudWatch Alarms**: Set up alarms on critical metrics (e.g., Lambda errors, API Gateway 5xx errors, DynamoDB throttled events) to notify administrators of operational issues.
- **AWS X-Ray (Distributed Tracing)**: Integrate AWS X-Ray with Lambda and API Gateway to trace requests end-to-end. This helps visualize application components, identify performance bottlenecks, and debug issues across services.

Cost Optimization

- **Serverless Pricing Model**: Pay only for the compute time and resources consumed. Lambda charges per invocation and duration, DynamoDB charges for read/write capacity units (RCUs/WCUs) or on-demand, and API Gateway charges per million API calls.
- **DynamoDB On-Demand vs. Provisioned**: For unpredictable workloads, DynamoDB On-Demand capacity can be more cost-effective. For stable or predictable workloads, Provisioned capacity with Auto Scaling can optimize costs.
- **Lambda Memory Allocation**: Optimize Lambda function memory. Higher memory often correlates with more CPU, so finding the right balance is crucial for performance and cost.
- **CloudFront Cache Policy**: Optimize caching policies to minimize requests to your S3 origin, reducing data transfer costs.

Deployment and CI/CD

- **Infrastructure as Code (IaC)**: Utilize IaC tools like AWS Serverless Application Model (SAM) or AWS CloudFormation (or Terraform) to define and manage your entire serverless application stack. This enables version control, automated deployments, and consistent environments.
- **CI/CD Pipeline**: Implement a Continuous Integration/Continuous Deployment (CI/CD) pipeline using services like AWS CodePipeline, CodeBuild, and CodeDeploy. This automates testing, building, and deploying your application changes, enabling faster and more reliable releases.

Frontend Enhancements

The provided index.html is a basic starting point. For a production-ready application, consider:

- **Form for Adding/Updating**: Implement a dedicated form for submitting student data, including validation.

- **Dynamic UI Updates**: Use JavaScript to dynamically update the student list without full page reloads.
- **Error Handling in UI**: Display user-friendly error messages for API failures.
- **Loading States**: Show loading indicators during API calls.
- **More Student Attributes**: Expand student data (e.g., age, grades, contact information) in both frontend and backend.
- **Styling Libraries**: Integrate modern CSS frameworks (e.g., Tailwind CSS, Bootstrap) for a polished look and responsive design.

This comprehensive guide provides a foundation for building a robust and scalable serverless web application on AWS using Python. By following these steps and incorporating best practices, you can develop efficient and secure student data management solutions.

This video provides an overview of how to securely host a static website with S3 and CloudFront. Host a Static Website Securely with S3 & CloudFront Step-by-step