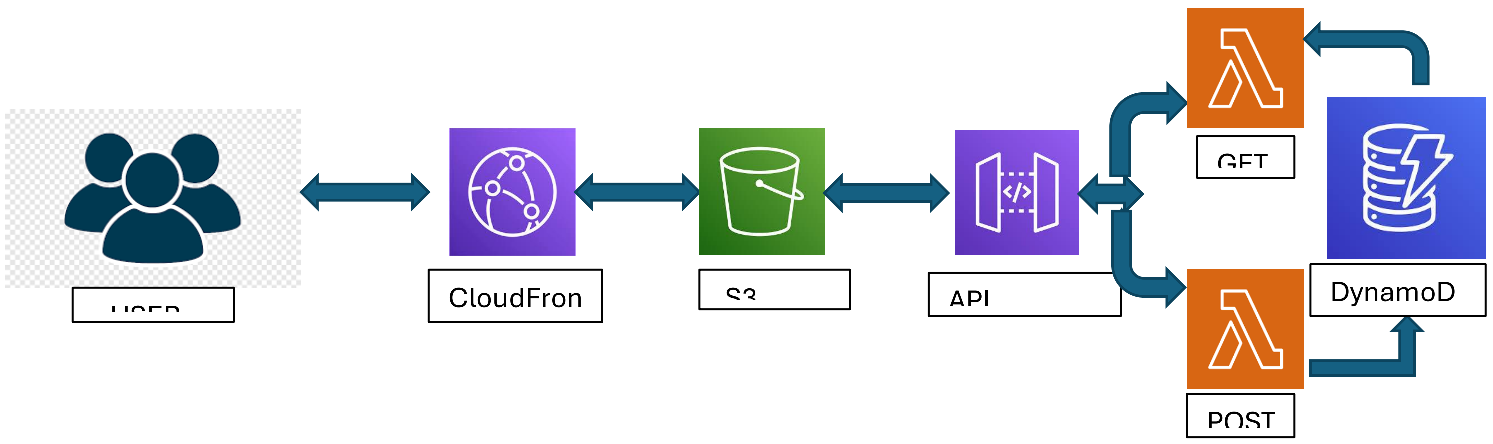
SERVERLESS WEB APPLICATION

SERVICES USED

1. S3
2. CloudFront
3. API Gateway
4. DynamoDB
5. Lambda

ARCHITECTURAL DIAGRAM



PRE – REQUISITES

1. AWS Console
2. Make sure all the name of the Services to be as the name of the Source code.

DIVIDED AS THREE PARTS :

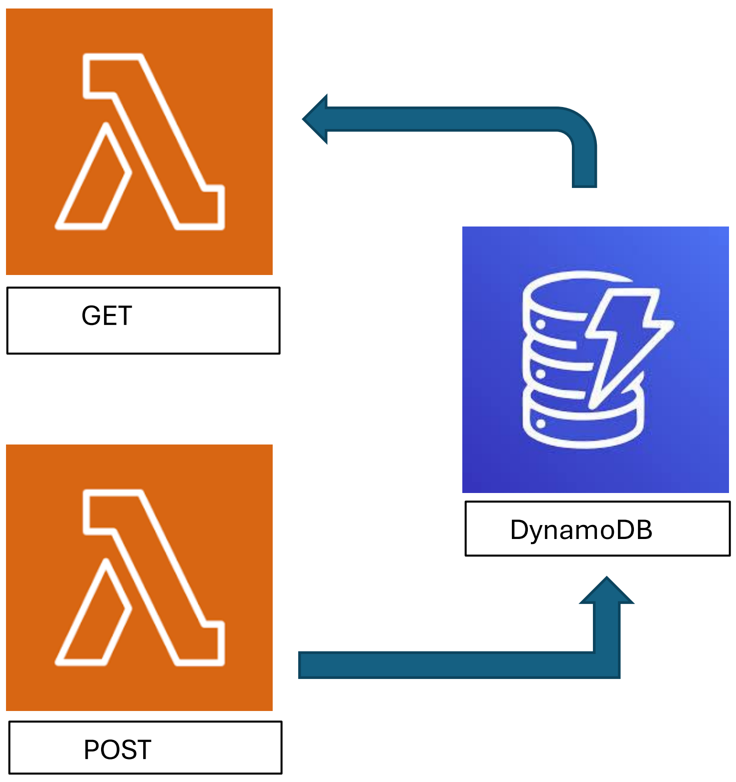
PART 1 : LAMBDA & DYNAMODB

PART 2 : S3 – Static Website & API Gateway

PART 3 :

PART – 1

We will be creating Lambda Function to Connect with DynamoDB



Step – 1 : CREATE DynamoDB TABLE

1. Create a table named "**StudentData**" under Table.
2. Create a Partition Key as "**Studentid**". **PARTITION KEY -** It is a primary key component, used to efficiently retrieve items based on a specific attribute value.
3. Leave everything as default settings.
4. Click **- Create Table.**

Step – 2 : CREATE LAMBDA Function

1. Create a function under the Lambda Function.
2. Select **Author from Scratch.**
3. Function name – **GetStudent.**
4. Runtime – **Python 3.13,** Architecture – **86 _64.**
5. Execution Role – **New role with Basic Lambda.**
6. Click **- Create Function.**

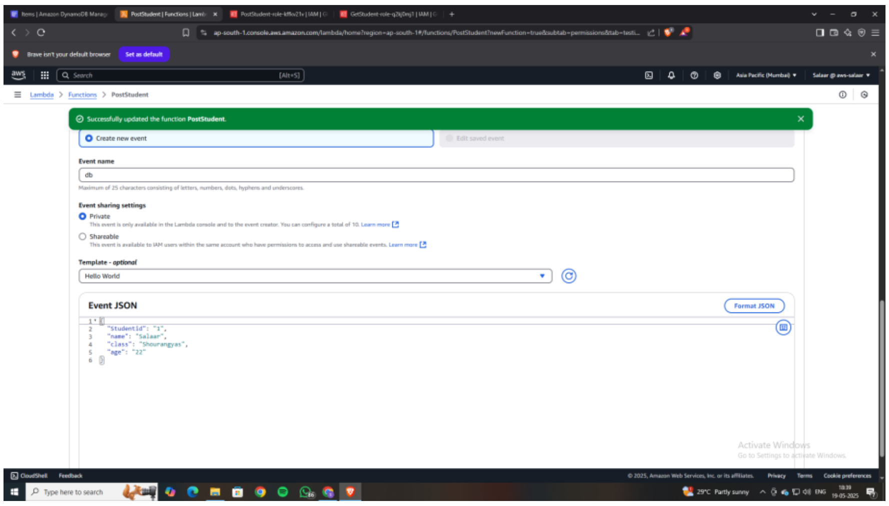
Step – 3 : GetStudent Function

1. Under code paste the source code given – **GetStudents.py** and click **DEPLOY.**
2. Under **Configuration** click on **Permission** and click the **Role Name – GetStudent-..**
3. It will forward you to **IAM Permission** for the **GetStudentLambda Function.**
4. Click on **Add Permission** and **Attach Policies.**
5. Select **DynamoDB Full Access** and attach the policy**.**
6. **This function is used to see or retrieve the student details.**

Step – 4 : PostStudent Function

1. Copy the same process but just name it as **PostStudent.**
2. **This function is used to post the student detail in DynamoDB.**

Step – 5 : Test PostStudent Function


Successfully updated the function **PostStudent**.

Create new event

Event name: db

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

Event sharing settings

[✓ Private] This event is only available in the Lambda console and to the event creator. You can configure a total of 10.

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

Template - optional: Hello World

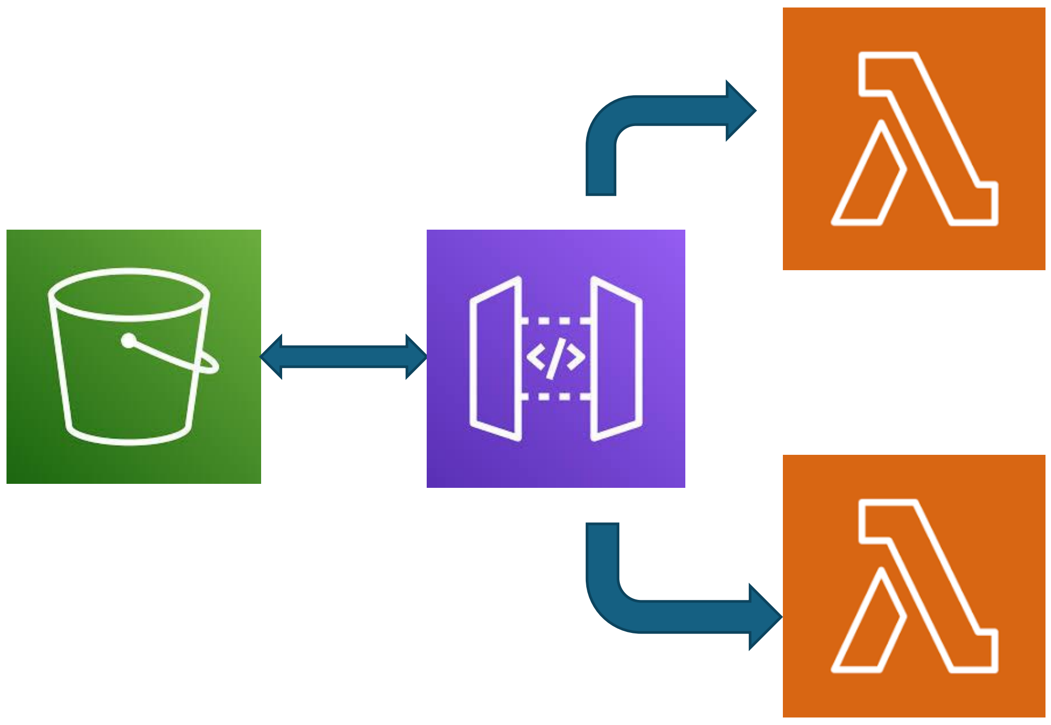
Event JSON

```
{
  "StudentId": "1",
  "name": "Salaar",
  "class": "Shourangyas",
  "age": "22"
}
```

1. The Data will be added to DynamoDB.
2. Check under Explore items in DynamoDB.

PART - 2

Creating S3to host static website and API Gateway to connect S3 and Lambda



Step - 1 : Create API GATEWAY

1. Under **API Gateway** click on create **API** and select **REST API** and **Build.**
2. Create **New API** and name it **"Students".**
3. Select **API endpoint type as "Edge - optimized".** Because, **This option allows users not only from our region and also allow users from all over the world.**

Step – 2: Configure Lambda Functions

1. Under **Students API** click on **Resources.**
2. Under Resources click on **Create Methods.**
3. Create 2 methods **GET and POST.**
4. **Method type – GET, POST.**
5. **Integration type – Lambda Function.**
6. Click on **respective ARN for both Lambda Functions. Region** your preferred region.

Step – 3 : DEPLOY API

1. Click on **Deploy API.**
2. Stage – **New Stage.**
3. Stage name – Name it as prod and click on Deploy**.**
4. The **API is** connected with the **Lambda Functions.**

5. If you check on your both **GET** and **POST Lambda Functions** an action will be triggered like **Lambda function is connected to the Students API Gateway.**
6. Under **Resource details** select **Enable CORS.**
7. Just select the **GET** and **POST** under **Access-Control-Allow-Methods** and **SAVE.**

Step – 4 : Change in Script.js

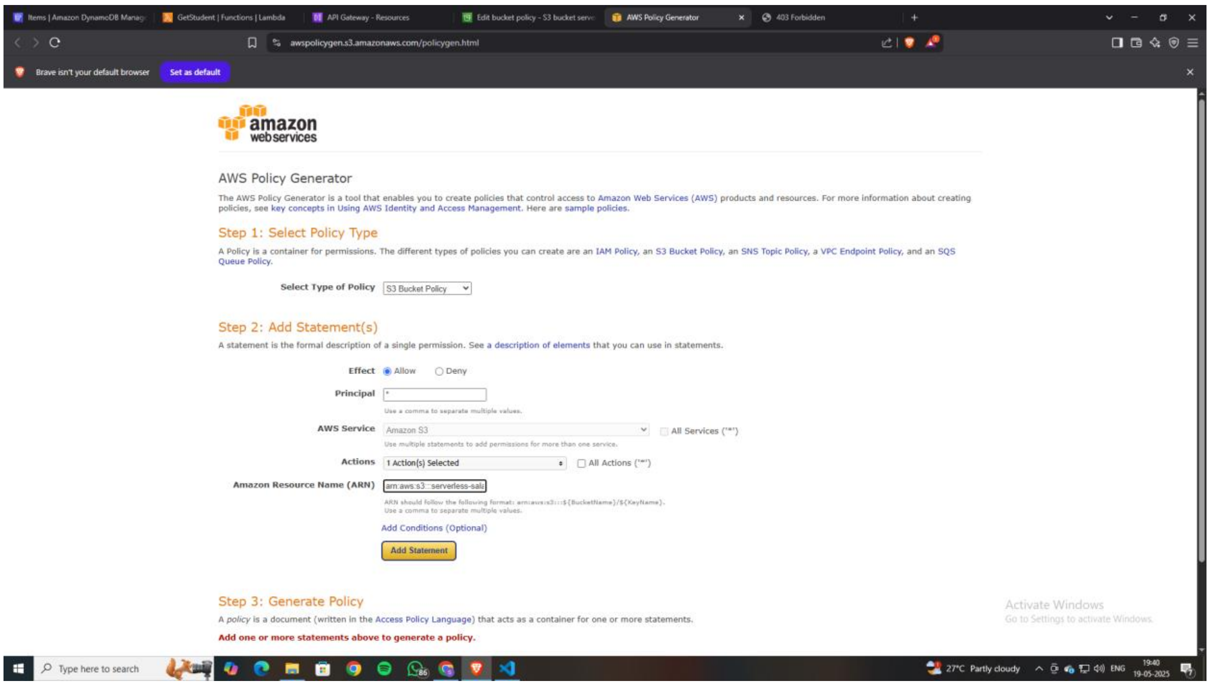
1. From **API Gateway** select on **Stages** > Select Deploy name **prod.**
2. Under **prod** copy the **Invoke URL.**
3. Replace it on **API_END_POINT > Your Invoke URL on Script.js** file before using it on **S3.**

Step – 5 :Create S3 Bucket

1. **Create Bucket by giving Bucket name then leave all the default settings as it is.**
2. Upload the file **index.html** and **script.js on your Bucket.**
3. Under **Properties of Bucket** scroll to last and select **Static website hosting.**
4. **Enable Static Website hosting.**
5. Index document **> index.html** and **Save Changes.**
6. A **URL** will be created to the website. But is you click on it won't work**. Because we defaulted to Block all Public Access to this Website.**

Step - 6 : Enabling Public Access on S3 and connecting with Students API

1. Select **Permissions** under **S3 Bucket > Block all public access.**
2. Click **edit** and **uncheck the Block all box** and **Save Changes.**
3. Under **Edit Bucket Policy** click on **Policy Generator to create new policy or** if you have an policy already paste it on**.**
4. Select policy type **> S3 Bucket policy.**
5. Effect **> Allow,** Principal > *****
6. Actions **> GetObject**
7. ARN **> Select and Paste the Bucket ARN.**
8. Click on **Generate Policy.**
9. **Paste** and make a change on Resources > add { /* } after your ARN and **Save Changes.**
10. Now the **Website will be accessed Publicly. Now Ready to Use.**


AWS Policy Generator

The AWS Policy Generator is a tool that enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see key concepts in Using AWS Identity and Access Management. Here are sample policies.

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select 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.

AWS Service: Amazon S3 — All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions: 1 Action(s) Selected — All Actions ('*')

Amazon Resource Name (ARN): arn:aws:s3:::serverless-sala

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

Add Conditions (Optional)

Add Statement

Step 3: Generate Policy

A *policy* is a document (written in the Access Policy Language) that acts as a container for one or more statements.

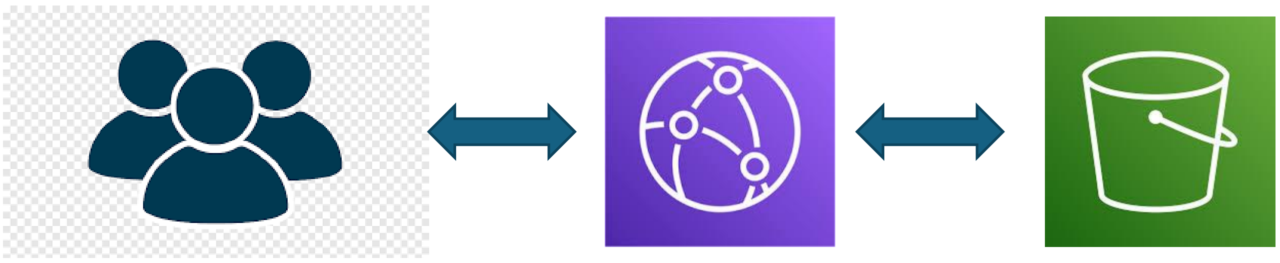
Add one or more statements above to generate a policy.

PART 3

Connecting our S3 Bucket to CloudFront

WHY ?

- **CloudFront is used in our Serverless Website hosting. Because, the Website is not Secure and the Contents in the Website is publicly accessed.**
- **The Reason is our Server runs on HTTP. If we connect it ti CloudFront the CloudFront provides a DNS (DOMAIN NAME SERVER) and it will run on HTTPS which acts as a Secure Server.**



Step – 1 : Create CloudFront

1. **Select Single website or app**
2. Origin in Domain > **Automatically there will your S3 Bucket name.**
3. Origin Access > **Origin access control settings.**
4. Create new OAC > **Create.**
5. Default Root Object > **index.html**
6. Web Application Firewall (WAF) > **Do not enable security protections.**
   IMPORTANT > Enable WAF will Cost.
7. **Generate Distribution** and **Copy Policy which is generated.**

Step – 2 : Connect CloudFront to S3 Bucket

1. **Replace the OLD BUCKET POLICY with the New Bucket policy copied from CloudFront.**
2. **Afterwards, Block all Public Access and Save Changes.**
3. **Now go to CloudFront under General copy > Distribution domain name.**
4. **On Browser https:// and add Distribution domain name.**