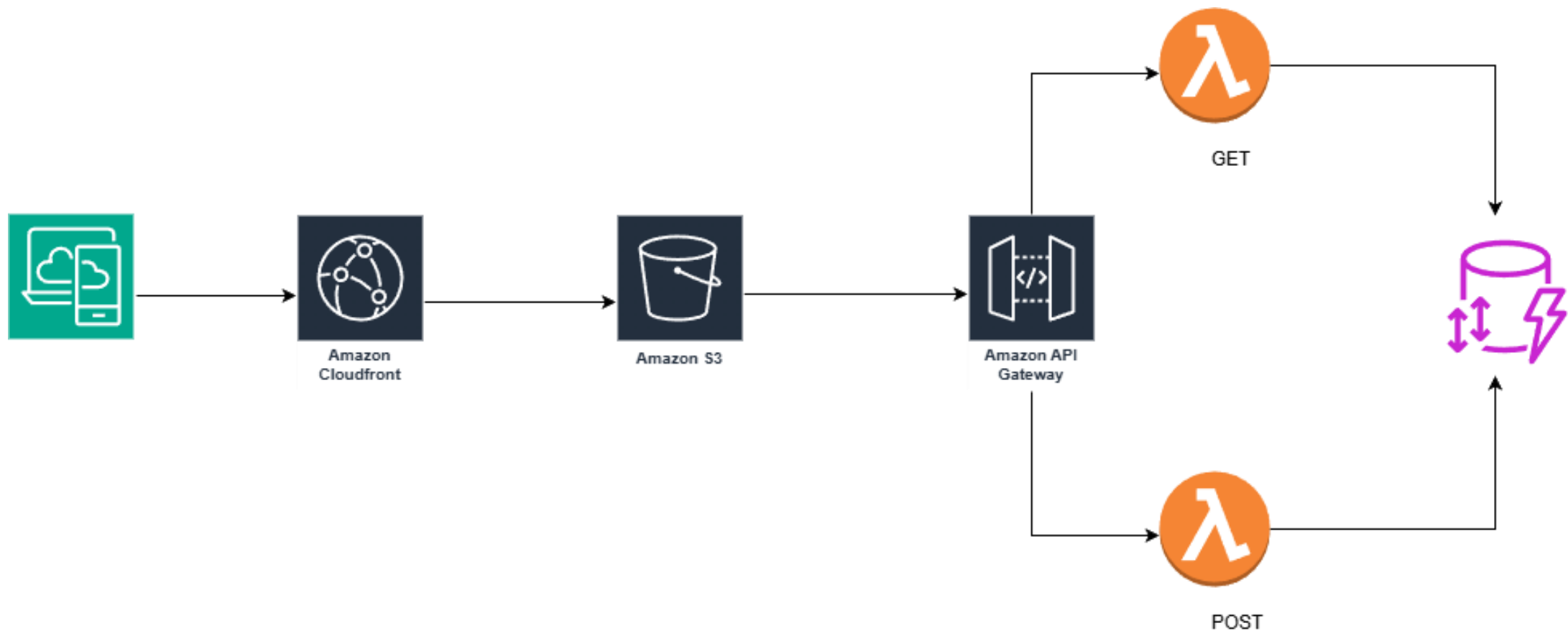# Deploying a Serverless Web Application on AWS Using S3, CloudFront, API Gateway, Lambda, DynamoDB, and Custom Domain Integration

AGENDA

- Deploy Lambda Functions for GET and POST

- Set Up DynamoDB Table

- Configure API Gateway

- Host Frontend on S3

- Set Up CloudFront CDN

- Troubleshooting & Common Issues

Amazon Cloudfront

Amazon S3

Amazon API Gateway

GET

POST

Set Up DynamoDB Table

- Open AWS Console → search "DynamoDB"
- Click "Create table"
- Enter Table Name
- Set Partition Key (e.g., EmployeeID – String)
- Leave Sort Key OFF (optional)
- In Table settings, keep Default settings
- Keep Table class as "DynamoDB Standard"
- Click "Create table"

Set Up DynamoDB Table

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.

```
EmployeeData
```

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

```
EmployeeID
```
| String ▼ |

1 to 255 characters and case sensitive.

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.

Set Up DynamoDB Table

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
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	AWS owned key	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Set Up DynamoDB Table

Deploy Lambda Functions

- Go to AWS Console → Open Lambda
- Click Create function
- Select Author from scratch
- Enter function name
- Choose runtime (Python/Node.js, etc.)
- Select or create an execution role
- Click Create function
- Add code → Deploy → Test

Deploy Lambda Functions for GET and POST

Creating a Lambda Function – getEmployeeData

- **Function Name:** getEmployeeData

- **Runtime:** Python 3.9

- **Architecture:** x86_64

- **Execution Role:** Custom IAM Role

  - Create role in IAM console
  - Attach the role to Lamda
  - Ensure the role has permissions for CloudWatch logging and DynamoDB access

- **Click on Create Functions**

Deploy Lambda Functions for GET and POST

Basic information

Function name
Enter a name that describes the purpose of your function.

getEmployeeData

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 3.9 ▼   ↻

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.

LambdaDynamoDBRole ▼   ↻

View the LambdaDynamoDBRole role ↗ on the IAM console.

Deploy Lambda Functions for GET and POST

Deploying Python Code on AWS Lamda (Serverless Deployment)

Deploy Lambda Functions for GET and POST

Lamda Test Execution
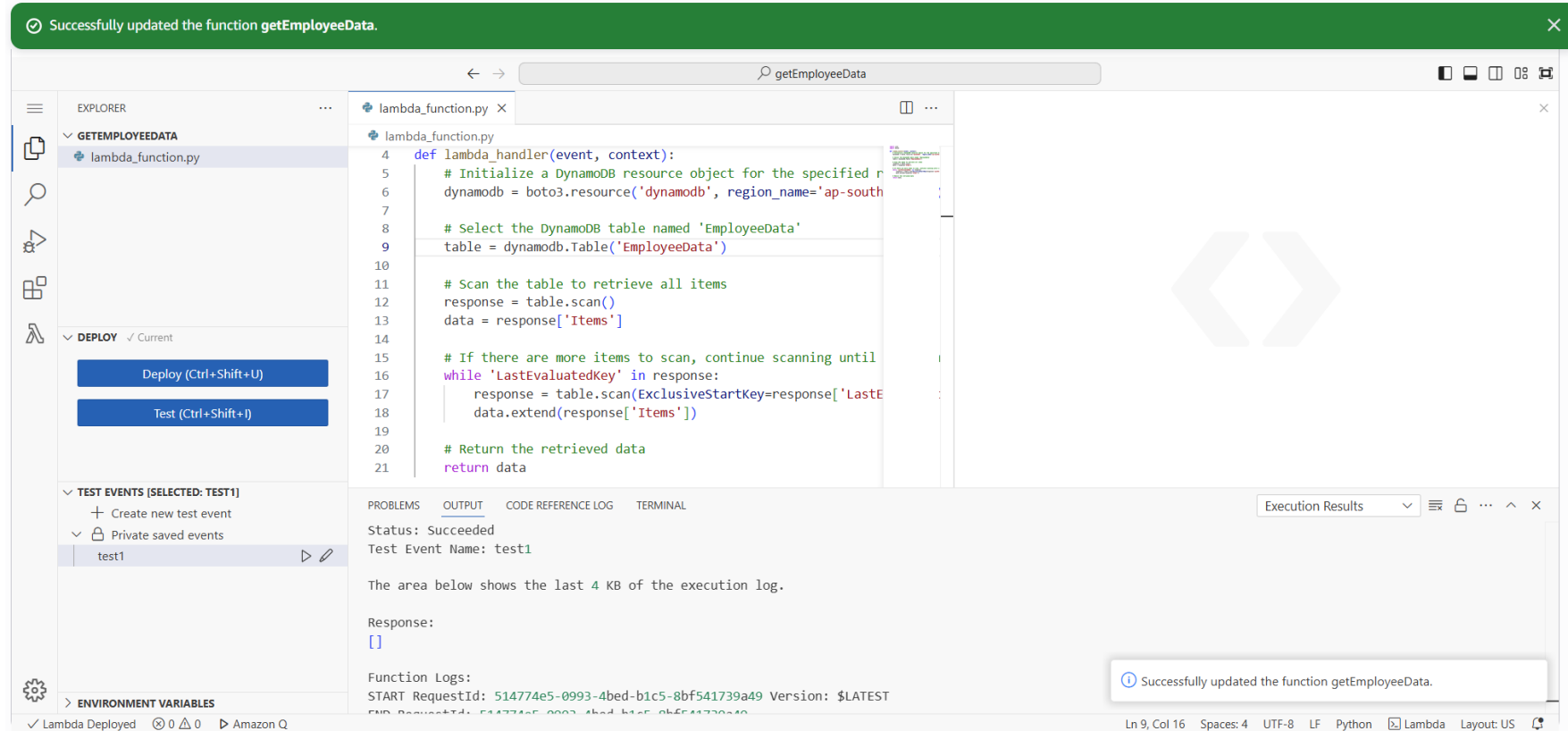
Deploy Lambda Functions for GET and POST

Creating a Lambda Function – insertEmployeeData

Remark: *Follow the steps on pages 8–11 to create the insertEmployeeData Lambda function*
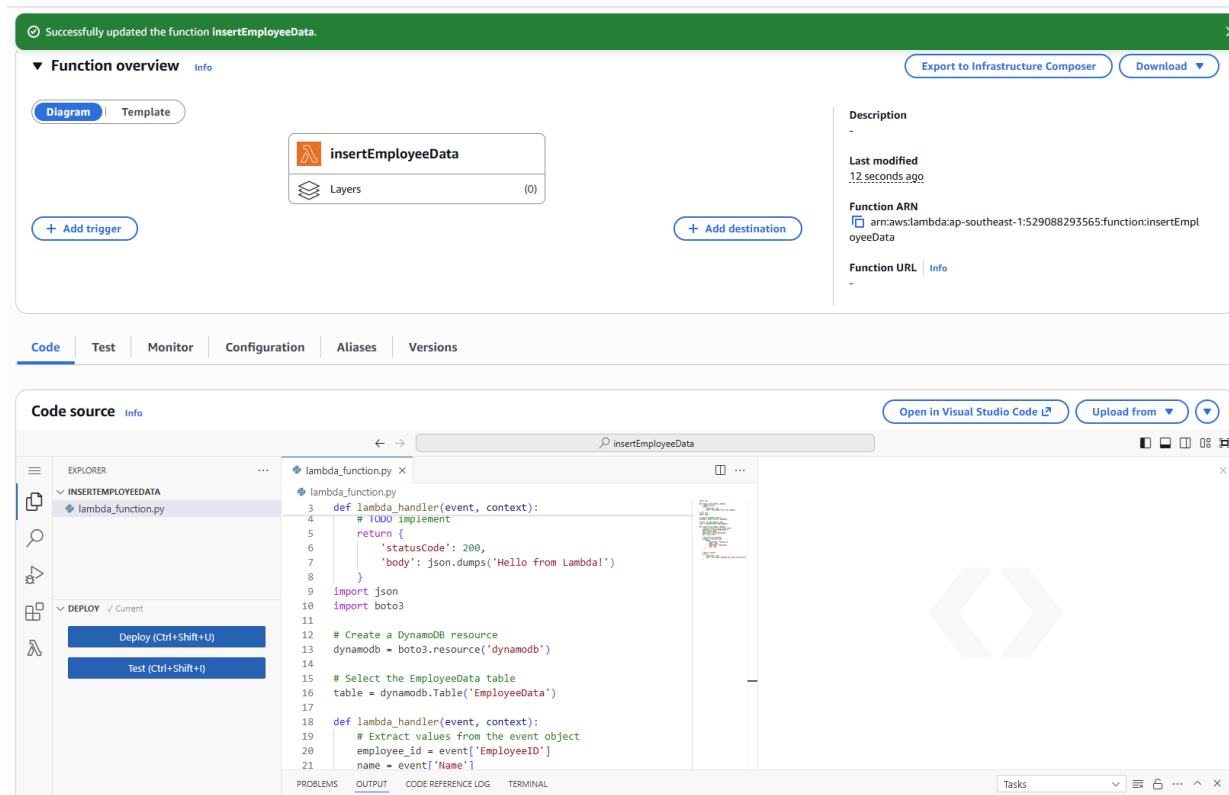
Configure API Gateway

- Open AWS Console → search "API Gateway"
- Click "Create API"
- Choose "REST API (Build)"
- Select "New API" → Give API Name
- Click "Create API"
- Create a Resource
- Create a Method (GET or POST)
- Choose Integration Type: Lambda Function
- Select your Lambda → Click "Save"
- Deploy API → Create a new Stage (e.g., prod)
- Copy the Invoke URL and test your REST API

Configure API Gateway

- Create an API and select REST API

Configure API Gateway

- Create a GET method

Configure API Gateway

- Create a PUT method

Configure API Gateway

- Deploy the API to a stage
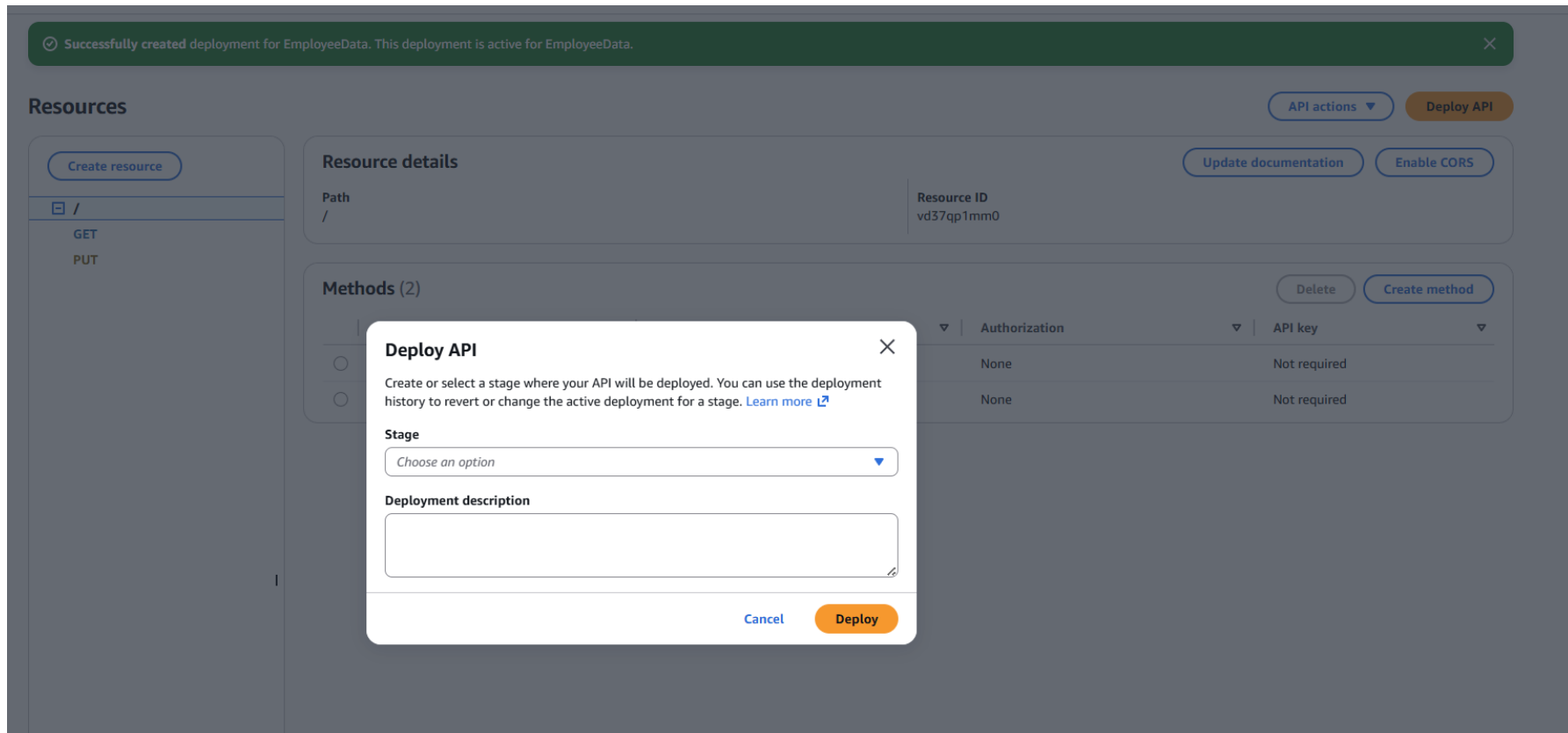
Configure API Gateway

- API Deployed

Host Frontend on S3

- Open AWS Console → search "S3"

- Create bucket → Uncheck "Block all public access"

- Upload your build files (index.html, JS)

- Go to Permissions → Add Bucket Policy to allow public read

- Go to Properties → Enable "Static website hosting"

- Set Index document = index.html

- Save changes

- Copy the S3 Website Endpoint URL and access your site

Host Frontend on S3

- Create an S3 bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

Asia Pacific (Singapore) ap-southeast-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 specialized low-latency use cases supported by AWS Availability Zones or data residency use cases supported by AWS Local Zones.

Bucket name | Info

```
employeedata-529088293565
```

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.

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 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 this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more ⎘

- ☑ **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

  - ☑ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Host Frontend on S3

- Upload the index.html and frontend JavaScript files to the S3 bucket

Host Frontend on S3

- Enable Static website hosting

Host Frontend on S3

- Edit the bucket policy under Permissions

Set Up CloudFront CDN

- Go to CloudFront → Create Distribution
- Choose Distribution Type: Single Page Web Application (or Web)
- Select your S3 bucket as the Origin
- Set Origin Access = OAC to secure S3
- Click Create OAC → Copy policy → Paste into S3 bucket policy
- Set Default Root Object = index.html
- Leave other settings as default and Create Distribution
- Wait for deployment, then copy the CloudFront Domain Name

Set Up CloudFront CDN

- Create a CloudFront distribution.

Set Up CloudFront CDN

- Selected S3 bucket as the *origin* for CloudFront

Set Up CloudFront CDN

- Create a new CloudFront Origin Access Control (OAC) to allow CloudFront to securely access my S3 bucket.

Set Up CloudFront CDN

- Copy the generated OAC bucket policy and paste it into the S3 bucket policy to allow CloudFront access

Bucket policy   Edit   Delete

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": "2008-10-17",
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Sid": "AllowCloudFrontServicePrincipal",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::employeedata-529088293565/*",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceArn": "arn:aws:cloudfront::529088293565:distribution/E2T3VRRQ39QHMS"
                }
            }
        }
    ]
}
```

Copy

Set Up CloudFront CDN

- Set Default Root Object = index.html

Set Up CloudFront CDN

- CloudFront is deployed. Copy the domain name and open it in your browser.

Set Up CloudFront CDN

- CloudFront is deployed. Copy the domain name and open it in your browser.

API stores employee data in DynamoDB and fetches all records on "View All Employees."

Save and View Employee Data

Employee ID:

Name:

Department:

Age:

Save Employee Data

View all Employees

Employee ID	Name	Department	Age

Troubleshooting & Common Issues

❑ 403 Forbidden

- OAC not configured correctly
- S3 bucket policy misconfigured
- Lambda IAM role missing required permissions
- API CORS Misconfiguration

❑ 404 Not Found

- Default Root Object was not configured in CDN
- API called with wrong resource path → Endpoint not found → 404 error.