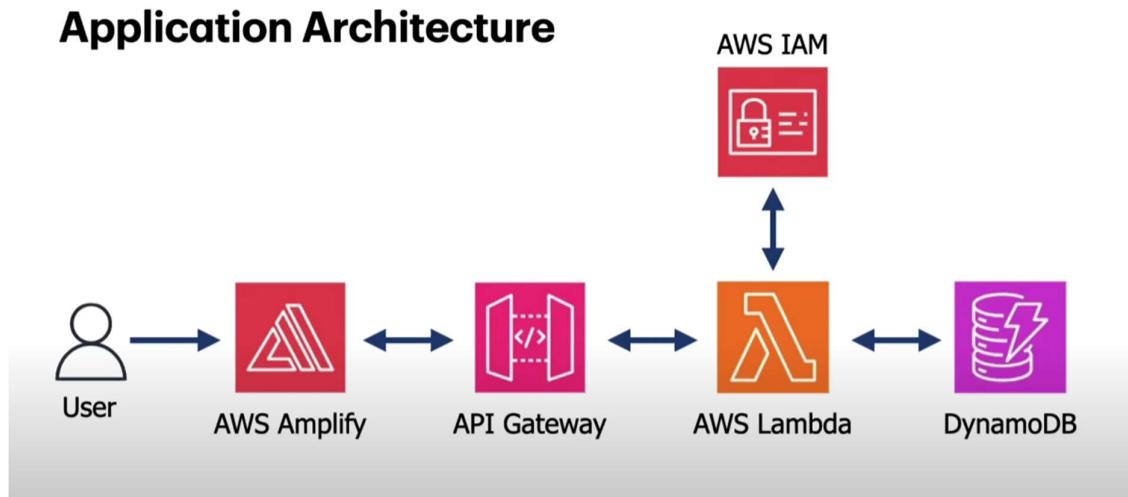


M. Sai Saranya

22BAI1471

Lab-05

Creating and deploying a serverless application using DynamoDB



### 1. DynamoDB (Database):

- **Purpose:** Stores plant data (e.g., PlantID, Name, Price, etc.).
- **Link:** The table is accessed by the Lambda function for reading (Scan) and writing (PutItem) operations.

### 2. Lambda (Serverless Backend):

- **Purpose:** Contains the logic to interact with DynamoDB (add plants, fetch plant details).
- **Link:** Lambda reads from and writes to the DynamoDB table. It is invoked by API Gateway to perform these operations when triggered by HTTP requests.

### 3. API Gateway (REST API):

- **Purpose:** Provides a REST API to access Lambda functions through HTTP methods (POST, GET).
- **Link:** Acts as an interface between the front-end (HTML/JavaScript) and the Lambda function, sending user requests to Lambda.

### 4. IAM (Access Control):

- **Purpose:** Manages permissions and access control for AWS resources.
- **Link:** IAM roles and policies grant the Lambda function permissions to read and write data in DynamoDB.

### 5. AWS Amplify (Web Hosting):

- **Purpose:** Hosts and deploys the front-end (HTML/JavaScript) of the botanist shop app.
- **Link:** Amplify serves the front-end which interacts with API Gateway to make requests that trigger the Lambda function.

These services work together to create a full-stack application where users can interact with the front-end (Amplify), make requests via API Gateway, which trigger Lambda functions to interact with DynamoDB, all controlled by IAM permissions.

- 1) Create a table in DynamoDB and give a partition key that serves as the primary key for accessing the table items.

The screenshot shows the 'Create table' interface in the AWS DynamoDB console. The top navigation bar includes the AWS logo, Services dropdown, search bar, and user information (Stockholm, Davinci005). The main title is 'Create table'. The first section, 'Table details', contains fields for 'Table name' (set to 'FloralBliss') and 'Partition key' (set to 'PlantID' of type 'String'). The second section, 'Table settings', offers two options: 'Default settings' (selected) and 'Customize settings'. At the bottom, there are links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences, along with a copyright notice: '© 2024, Amazon Web Services, Inc. or its affiliates.'

Create table | Amazon DynamoDB

eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#create-table

aws Services Search [Alt+S] Stockholm Davinci005

**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	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

**Tags**

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel Create table

CloudShell Feedback Privacy Terms Cookie preferences © 2024, Amazon Web Services, Inc. or its affiliates.

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Default table settings' section displays various configuration options like table class, capacity mode, and encryption. The 'Tags' section shows no existing tags and provides a button to 'Add new tag'. At the bottom, there are 'Cancel' and 'Create table' buttons, along with standard AWS footer links.

- 2) Create a lambda function by choosing the desired language and writing functionalities of how the app should work.

The screenshot shows the AWS Lambda console homepage. At the top, there are tabs for "List tables | Amazon DynamoDB" and "AWS Lambda | Lambda". The main title "AWS Lambda" is prominently displayed with the tagline "lets you run code without thinking about servers." Below this, a paragraph explains the cost model: "You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration." A "Get started" section contains a "Create a function" button. Below this, a "How it works" section shows a code snippet for Node.js:

```
1 * exports.handler = async (event) => {
2     console.log(event);
3     return 'Hello from Lambda!';
4 };
5
```

Navigation links for ".NET", "Java", "Python", "Ruby", and "Custom runtime" are also visible.

List tables | Amazon DynamoDB   Create function | Functions | Lambda   +

eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#/create/function?firstrun=true

AWS Services Search [Alt+S] Stockholm Davinci005

Lambda > Functions > Create function

## Create function Info

Choose one of the following options to create your function.

- Author from scratch  
Start with a simple Hello World example.
- Use a blueprint  
Build a Lambda application from sample code and configuration presets for common use cases.
- Container image  
Select a container image to deploy for your function.
- Browse serverless app repository  
Deploy a sample Lambda application from the AWS Serverless Application Repository.

### Basic information

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

Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime Info**  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture Info**  
Choose the instruction set architecture you want for your function code.  
 x86\_64  
 arm64

**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

CloudShell Feedback Privacy Terms Cookie preferences © 2024, Amazon Web Services, Inc. or its affiliates.

The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'List tables | Amazon DynamoDB' and 'Botanic\_22BAI1471 | Functions'. The main area is titled 'Botanic\_22BAI1471 | Functions'. Below this, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab is selected. The code editor displays a Python file named 'lambda\_function.py' with the following content:

```
1 import json
2 import boto3
3 import time
4 dynamodb = boto3.resource('dynamodb')
5 # Replace 'Plant' with your actual table name
6 table = dynamodb.Table('FloralBliss')
7
8 def lambda_handler(event, context):
9     if event['operation'] == 'addPlant':
10         return savePlant(event)
11     else:
12         return getPlants()
13
14 def savePlant(event):
15     gmt_time = time.gmtime()
16     now = time.strftime('%a, %d %b %Y %H:%M:%S', gmt_time)
17     table.put_item(
18         Item={
19             'plantID': event['plantID'],
20             'plantName': event['plantName'],
21             'cost': event['cost'],
22             'createdAt': now
23         }
24     )
25     return {
26         'statusCode': 200,
27         'body': json.dumps('Plant with PlantID: ' + event['plantID'] + ' added')
28     }
29
30 def getPlants():
31     response = table.scan()
32     items = response['Items']
33     print(items)
34
35
36
```

The code uses the boto3 library to interact with a DynamoDB table named 'FloralBliss'. It defines two functions: 'lambda\_handler' which handles events and calls either 'savePlant' or 'getPlants' based on the operation type; and 'savePlant' which adds a new item to the table with fields like plantID, plantName, cost, and createdAt.

- 3) But this function is not ready to use, since we didn't give access to it like for iam user for execution role
- 4) Go to the permissions and create an inline policy where we can select the service, we can use like DynamoDB and also the operations that we need to perform mainly two-> to read (Scan) and to write (PutItem)

List tables | Amazon DynamoDB   Botanic\_22BAI1471 | Functions | +

eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#functions/Botanic\_22BAI1471?ne...

AWS Services Search [Alt+S] Stockholm Davinci005

Successfully updated the function Botanic\_22BAI1471.

Code Test Monitor Configuration Aliases Versions

General configuration Triggers Permissions Destinations Function URL Environment variables Tags VPC RDS databases Monitoring and operations tools Concurrency and recursion detection Asynchronous invocation

Execution role [Edit](#) [View role document](#)

Role name Botanic\_22BAI1471-role-mu0qv237

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

By action By resource

Resource

arn:aws:logs:eu-north-1:009160060743:  
arn:aws:logs:eu-north-1:009160060743:log-group:/aws/lambda/Botanic\_22BAI1471:\*

Lambda obtained this information from the following policy statements:

CloudShell Feedback Privacy Terms Cookie preferences

[https://eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#functions/Botanic\\_22BAI1471?tab=configure](https://eu-north-1.console.aws.amazon.com/lambda/home?region=eu-north-1#functions/Botanic_22BAI1471?tab=configure)

Screenshot of the AWS IAM Roles page for the role "Botanic\_22BAI1471-role-mu0qv237".

**Summary:**

Creation date	ARN
October 02, 2024, 13:16 (UTC+05:30)	<code>arn:aws:iam::009160060743:role/service-role/Botanic_22BAI1471-role-mu0qv237</code>
Last activity	Maximum session duration
-	1 hour

**Permissions:** (1) Info

You can attach up to 10 managed policies.

Attach policies

Filter Create inline policy

Search All types

Policy name Type Attached to

Custom permissions

Screenshot of the "Create policy" wizard for the role "Botanic\_22BAI1471-role-mu0qv237".

**Step 1: Specify permissions** Info

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

**Policy editor:** Visual JSON Actions

**DynamoDB** Allow 2 Actions

Specify what actions can be performed on specific resources in [DynamoDB](#).

**Actions allowed:**

Specify actions from the service to be allowed.

Filter Actions

Effect: Allow Deny

Manual actions | Add actions

All DynamoDB actions (`dynamodb:*`)

Access level: List (6)

Read (Selected 1/27)

All read actions

- BatchGetItem [Info](#)
- DescribeContinuousBackups [Info](#)
- DescribeExport [Info](#)
- DescribeImport [Info](#)
- DescribeReservedCapacity [Info](#)
- DescribeTable [Info](#)
- ConditionCheckItem [Info](#)
- DescribeContributorInsights [Info](#)
- DescribeGlobalTable [Info](#)
- DescribeKinesisStreamingDestination [Info](#)
- DescribeReservedCapacityOfferings [Info](#)
- DescribeTableReplicaAutoScaling [Info](#)
- DescribeBackup [Info](#)
- DescribeEndpoints [Info](#)
- DescribeGlobalTableSettings [Info](#)
- DescribeLimits [Info](#)
- DescribeStream [Info](#)
- DescribeTimeToLive [Info](#)

Expand all | Collapse all

The screenshot shows the 'Create policy' page in the AWS IAM console. The policy document is being edited to grant permissions for the 'Botanic\_22BAI1471' role. The policy includes the following actions:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "dynamodb:DescribeReservedCapacity",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DescribeTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:GetItem",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:GetShardIterator",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PartiQLSelect",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DescribeReservedCapacityOfferings",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DescribeTableReplicaAutoScaling",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:ListStreams",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:Query",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DescribeStream",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DescribeTimeToLive",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:GetResourcePolicy",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>ListTagsOfResource",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:Scan",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:BatchWriteItem",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb CreateTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>DeleteItem",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:DisableKinesisStreamingDestination",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:ImportTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PartiQLUpdate",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:RestoreTableFromAwsBackup",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:StartAwsBackupJob",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateGlobalTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateItem",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateTableReplicaAutoScaling",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>CreateBackup",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>CreateTableReplica",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>DeleteTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:EnableKinesisStreamingDestination",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PartiQLDelete",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:RestoreTableFromBackup",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateContinuousBackups",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateGlobalTableSettings",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateKinesisStreamingDestination",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateTimeToLive",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>CreateGlobalTable",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>DeleteBackup",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb>DeleteTableReplica",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:ExportTableToPointInTime",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PartiQLInsert",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:PutItem",
            "Effect": "Allow" (Selected)
        },
        {
            "Action": "dynamodb:RestoreTableToPointInTime",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateContributorInsights",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateGlobalTableVersion",
            "Effect": "Allow"
        },
        {
            "Action": "dynamodb:UpdateTable",
            "Effect": "Allow"
        }
    ]
}

```

**Resources**: Specify resource ARNs for these actions.

All  Specific

The screenshot shows the 'View table' page for the 'FloralBliss' table in the eu-north-1 region. The table details are as follows:

- Active alarms**: No active alarms.
- Resource-based policy**: Not active.
- Additional info**:
  - Table class: DynamoDB Standard
  - Indexes: 0 globals, 0 locals
  - DynamoDB stream: Off
  - Time to Live (TTL): Off
  - Replication Regions: 0 Regions
  - Encryption: Owned by Amazon
  - Date created: October 2, 2024, 13:15:18 (UTC+05:30)
  - Deletion protection: Off
  - Integrations: 0
- Amazon Resource Name (ARN)**: arn:aws:dynamodb:eu-north-1:009160060743:table/FloralBliss
- Items summary**: Item count: 0, Table size: 0 bytes.

- 5) Copy the ARN and paste it into the option named add resources in permissions of the user role that we need to select resource region, table name, and ARN

Screenshot of the AWS Lambda function configuration page showing the 'Permissions' tab.

**Available permissions**

- RestoreTableFromAwsBackup
- RestoreTableToPointInTime
- UpdateContinuousBackups
- UpdateGlobalTable
- UpdateGlobalTableVersion
- UpdateKinesisStreamingDestination
- RestoreTableFromBackup
- StartAwsBackupJob
- UpdateContributorInsights
- UpdateGlobalTableSettings
- UpdateItem
- UpdateTable

**Specify ARN(s)**

**Visual** **Text**

Resource in  This account  Any account  Other account

Resource region   Any region

Resource table name   Any table name

Resource ARN

**Add ARNs**

Actions on resources are allowed or denied only when three conditions are met:

**Add more permissions**

Security: 0 Errors: 0 Warnings: 0 Suggestions: 2

**Create policy | IAM | Global**

**us-east-1.console.aws.amazon.com/iam/home#/roles/details/Botanic\_22BAI1471-role-mu0qv237/createPolicy**

**IAM > Roles > Botanic\_22BAI1471-role-mu0qv237 > Create policy**

**Review and create**

**Step 1 Specify permissions**

**Step 2 Review and create**

**Policy details**

Policy name  Enter a meaningful name to identify this policy. Maximum 128 characters. Use alphanumeric and '+-=\_,@-' characters.

**Permissions defined in this policy** Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Service	Access level	Resource
DynamoDB	Limited: Read, Write	region  string like [eu-north-1, TableName  string like  FloralBliss]

**Create policy**

Successfully updated the function Botanic\_22BAI1471.

Code | Test | Monitor | Configuration | Aliases | Versions

General configuration

Triggers

**Permissions**

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

Monitoring and operations tools

Concurrency and recursion detection

Asynchronous invocation

Code signing

File systems

State machines

**Execution role**

Role name: Botanic\_22BAI1471-role-mu0qv237

**Resource summary**

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

Amazon DynamoDB

2 actions, 1 resource

By action | **By resource**

Resource	Actions
arn:aws:dynamodb:eu-north-1:009160060743:table/FloralBliss	Allow: dynamodb:PutItem Allow: dynamodb:Scan

**Info** Lambda obtained this information from the following policy statements:

- Inline policy LambdaDynamoDBRole, statement VisualEditor0

CloudShell | Feedback | © 2024, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences

- 6) Now we can test the function using a test event by providing a Json format of data which can be inserted into the table

Successfully updated the function Botanic\_22BAI1471.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

Create new event    Edit saved event

Event name

TestEvent

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. [Learn more](#)

Shareable  
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

Format JSON

```
1 * {  
2   "operation": "addPlant",  
3   "plantID": "1",  
4   "plantName": "Aloevera",  
5   "cost": "12.55"  
6 }
```

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Code source Info Upload from

File Edit Find View Go Tools Window Test Deploy

Environment

Go to Anything (Ctrl-P)

lambda\_function Environment Var Execution result

Status: Succeeded | Max memory used: 78 MB | Time: 573.79 ms

Execution results

Test Event Name (unsaved) test event

Response

```
{  
  "statusCode": 200,  
  "body": "\\"Plant with PlantID: 1 created at Wed, 02 Oct 2024 07:59:49\\\""  
}
```

Function Logs

```
START RequestId: ab5feb1-a724-4be4-b7ea-3201a88352fd Version: $LATEST  
END RequestId: ab5feb1-a724-4be4-b7ea-3201a88352fd  
REPORT RequestId: ab5feb1-a724-4be4-b7ea-3201a88352fd Duration: 573.79 ms Billed Duration: 574 ms Memory Size: 128 MB Ma  
Request ID  
ab5feb1-a724-4be4-b7ea-3201a88352fd
```

DynamoDB > Explore items > FloralBliss

Tables (1) X

Any tag key  
Any tag value

Find tables Q

< 1 > ⟳

FloralBliss ●

FloralBliss Autopreview  View table details (i)

Scan or query items Expand to query or scan items.

Completed. Read capacity units consumed: 0.5 X

Items returned (1) ⟳ Actions ▾ Create item

< 1 > ⟳ (i) X

	PlantID (String)	cost	createdAt	plantName
<input type="checkbox"/>	<a href="#">1</a>	12.55	Wed, 02 Oc...	Aloevera

7) Now we need to create a rest API to access the table via function

Items | Amazon DynamoDB Botanic\_22BAI1471 | Functions | API Gateway - Create API

eu-north-1.console.aws.amazon.com/apigateway/main/apis?region=eu-north-1

Services Search [Alt+S] Stockholm Davinci005

## 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

Build

## 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

Import Build

## REST API Private

Create a REST API that is only accessible from within a VPC.

Works with the following:  
Lambda, HTTP, AWS Services

Import Build

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Items | Amazon DynamoDB Botanic\_22BAI1471 | Functions | API Gateway - Create REST API

eu-north-1.console.aws.amazon.com/apigateway/main/create-rest?experience=private&region=eu-north-1

Services Search [Alt+S] Stockholm Davinci005

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

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.

Cancel Create API

- 8) Create a resource named getData and add post and get methods to the api

The screenshot shows the AWS API Gateway Resources page. On the left, there's a sidebar with a 'Create resource' button. The main area displays a resource with the path '/'. The 'Resource details' section shows the Path as '/' and the Resource ID as 't1wrnkllx1'. Below it, the 'Methods (0)' section indicates 'No methods' defined. A green success message at the top of the browser window says 'Successfully created REST API 'Floral\_API (ytu5zfhuqe)'.'

**API Gateway > APIs > Resources - Floral\_API (ytu5zfhuqe)**

**Resources**

**Create resource**

**Resource details**

Path: / Resource ID: t1wrnkllx1

**Methods (0)**

No methods

No methods defined.

Successfully created REST API 'Floral\_API (ytu5zfhuqe)'

**Create resource**

**Resource details**

**Proxy resource** Info  
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path: / Resource name: getData

**CORS (Cross Origin Resource Sharing)** Info  
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Cancel **Create resource**

**Methods (0)**

**Delete** **Create method**

**Method type** ▲ **Integration type** ▼ **Authorization** ▼

**No methods**

No methods defined.

**Create method**

**Method details**

**Method type**: POST

**Integration type**:

- Lambda function**  
Integrate your API with a Lambda function.  

- HTTP**  
Integrate with an existing HTTP endpoint.  

- Mock**  
Generate a response based on API Gateway mappings and transformations.  


**AWS service**  
Integrate with an AWS Service.  


**VPC link**  
Integrate with a resource that isn't accessible over the public internet.  


**Lambda proxy integration**  
Send the request to your Lambda function as a structured event.

**Lambda function**  
Provide the Lambda function name or alias. You can also provide an ARN from another account.  
eu-north-1 ▾  X

**Grant API Gateway permission to invoke your Lambda function.** To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

9) We can test whether the api is working or not using Request body like that of test event we did earlier

AWS Services Search [Alt+S] Stockholm Davinci005

### API Gateway

APIs Custom domain names VPC links

▼ API: Floral\_API

#### Resources

- Stages
- Authorizers
- Gateway responses
- Models
- Resource policy
- Documentation
- Dashboard
- API settings

Usage plans API keys Client certificates Settings

Create resource

/ /getData POST

**Headers**  
Enter a header name and value separated by a colon (:). Use a new line for each header.  
header1:value1  
header2:value2

**Client certificate**  
No client certificates have been generated.

**Request body**

```
1 * {  
2     "operation": "addPlant",  
3     "plantID": "2",  
4     "plantName": "Rose",  
5     "cost": "14.45"  
6 }
```

Test

This screenshot shows the AWS API Gateway test interface. On the left, the navigation sidebar is visible with sections like APIs, Custom domain names, VPC links, and the expanded API: Floral\_API section which includes Resources, Stages, Authorizers, etc. The main area shows a resource path /getData with a POST method selected. The Headers section contains two entries: header1:value1 and header2:value2. The Client certificate section indicates no certificates are generated. The Request body section displays a JSON payload with six numbered lines. Lines 1 through 4 represent the object structure with keys operation, plantID, plantName, and cost respectively. Line 5 contains the value "14.45". Line 6 is a closing brace. A large orange Test button is located at the bottom of the request body panel.

Create resource

/

/getData

POST

Integration response

Method response

Test

### Test method

Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method.

#### Query strings

```
param1=value1&param2=value2
```

#### Headers

Enter a header name and value separated by a colon (:). Use a new line for each header.

```
header1:value1
header2:value2
```

#### Client certificate

No client certificates have been generated.

#### Request body

```
1 ▼ {
  2   "operation": "addPlant",
  3   "plantID": "2",
  4   "plantName": "Rose",
  5   "cost": "14.45"
  6 }
```



### /getData - POST method test results

Request

/getData

Latency ms

543

Status

200

Response body

```
{"statusCode": 200, "body": "\"Plant with  
PlantID: 2 created at Wed, 02 Oct 2024  
08:05:26\""}}
```

Response headers

```
{  
    "Content-Type": "application/json",  
    "X-Amzn-Trace-Id": "Root=1-66fcfec6-  
10b4dc60687fdacfc4575e43;Parent=382291edcb1e301d;  
Sampled=0;Lineage=1:5abc95e2:0"  
}
```

Logs

```
Execution log for request 8e5b3ec5-e1a2-4bde-  
be7f-4622672fea6b  
Wed Oct 02 08:05:26 UTC 2024 : Starting execution  
for request: 8e5b3ec5-e1a2-4bde-be7f-4622672fea6b  
Wed Oct 02 08:05:26 UTC 2024 : HTTP Method: POST,  
Resource Path: /getData  
Wed Oct 02 08:05:26 UTC 2024 : Method request
```

## FloralBliss

Autopreview

[View table details](#)

### Scan or query items

Scan

Query

Select a table or index

Table - FloralBliss

Select attribute projection

All attributes

#### Filters

[Run](#)

[Reset](#)

Completed. Read capacity units consumed: 0.5

X

#### Items returned (2)

[C](#) Actions ▾ Create item

< 1 >

	PlantID (String)	cost	createdAt	plantName
<input type="checkbox"/>	2	14.45	Wed, 02 Oc...	Rose
<input type="checkbox"/>	1	12.55	Wed, 02 Oc...	Aloeverta

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

#### Test event action

Create new event

Edit saved event

#### Event name

Event2

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. [Learn more](#)

Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

#### Template - optional

[Format JSON](#)

```
1 * {  
2   "operation": "getPlants"  
3 }
```

[Cancel](#)

[Invoke](#)

[Save](#)

lambda\_function × Environment Var × Execution result ×

Execution results

Test Event Name: Event2

Response

```
{
  "statusCode": 200,
  "body": "[{"createdAt": "Wed, 02 Oct 2024 08:05:26", "cost": "14.45", "PlantID": "2", "plantName": "Rose"}, {"createdAt": "Wed, 02 Oct 2024 07:59:49", "cost": "12.55", "PlantID": "1", "plantName": "Aloevera"}]",
  "headers": {
    "Content-Type": "application/json"
  }
}
```

Function Logs

```
START RequestId: cb929195-e635-4b2a-8c32-0b10d48737b7 Version: $LATEST
[{"createdAt": "Wed, 02 Oct 2024 08:05:26", "cost": "14.45", "PlantID": "2", "plantName": "Rose"}, {"createdAt": "Wed, 02 Oct 2024 07:59:49", "cost": "12.55", "PlantID": "1", "plantName": "Aloevera"}]
END RequestId: cb929195-e635-4b2a-8c32-0b10d48737b7
REPORT RequestId: cb929195-e635-4b2a-8c32-0b10d48737b7 Duration: 489.53 ms Billed Duration: 490 ms Memory Size: 128 MB Max Memory Used: 79 MB
```

Request ID: cb929195-e635-4b2a-8c32-0b10d48737b7

## Resources

API actions ▾ Deploy API

[Create resource](#)

  /

[/getData](#)

POST

**Resource details**

[Delete](#)

[Update documentation](#)

[Enable CORS](#)

Path: /getData

Resource ID: bu6trl

10) Now we need to enable CORS where post method is given access

API Gateway > APIs > Resources - Floral\_API (ytu5zfhue) > 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**

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

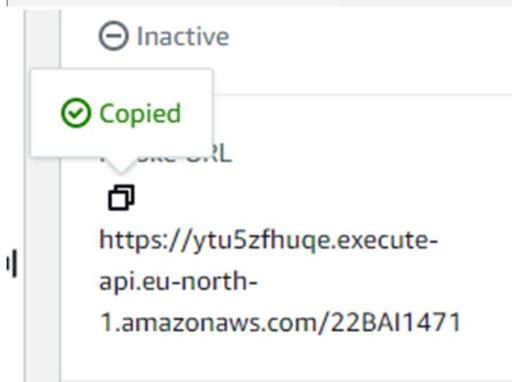
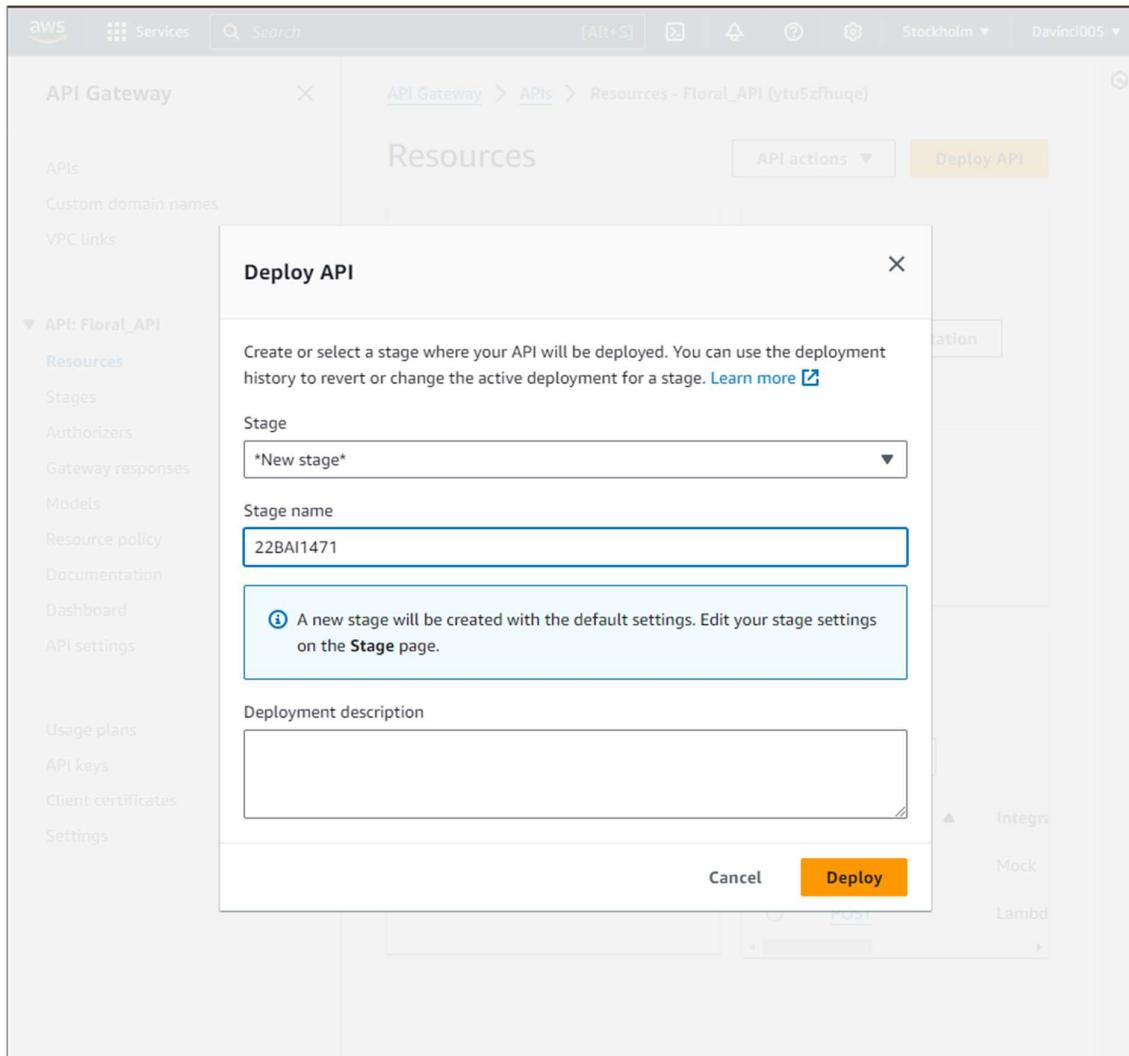
**Access-Control-Allow-Origin**  
Enter an origin that can access the resource. Use a wildcard '\*' to allow any origin to access the resource.

\*

▶ Additional settings

Cancel Save

11) Now deploy the API



- 12) Now create a html file and make it as zip file that has overall website design and JavaScript for backend operations and upload the zip file into the aws amplify (deploy without git option) and hence the app URL will be available

AWS Services Search [Alt+S] Stockholm Davinci005

## AWS Amplify All apps / Create new app

Support Docs

Choose create method

Start a manual deployment

### Start building with Amplify

Amplify provides a fully-managed web hosting experience and a backend building service to build fullstack apps. If you need a starter project, please visit the [docs](#).

#### Deploy your app

To deploy an app from a Git provider, select one of the options below:

GitHub  BitBucket  CodeCommit  GitLab

Amplify requires read-only access to your repository.

To deploy an app manually, select "Deploy without Git"

**Deploy without Git**

▶ Start with a template

Cancel Previous Next

Looking to build an app with our Gen 1 tools (Amplify Studio/Amplify CLI)? [Create an app with Gen 1](#)

All apps / Create new app

Support Docs

Choose create method

Start a manual deployment

### Start a manual deployment

Manually upload objects to deploy your app. You can choose to drag and drop the artifacts directly, pull a zip from an existing S3 bucket or any other URL.

App name: FloralBliss Branch name: staging

**⚠ Zip the contents of your build output, not the top level folder**  
Make sure you zip the contents of your build output and not the top level folder. For example, if your build output generates a folder named "build" or "public", first navigate into that folder, select all of the contents, and zip it from there. [Read more](#)

Method:  Drag and drop  Amazon S3  Any URL

index.zip Uploaded Remove

Cancel Previous Save and deploy

# Floral Bliss - A World of Plants

## Add a New Plant

Plant Name:

Cost:

Add Plant

## Available Plants

Plant Name

Cost

Date Added

API Gateway > APIs > Resources - Floral\_API (ytu5zfhue)

### Resources

API actions ▾ Deploy API

OPTIONS

POST

#### Resource details

Delete

Update documentation

Enable CORS

Path

Resource ID

bu6trt

#### Methods (2)

Delete

Create method

Method type

Integration type

Authorization

API key

OPTIONS

Mock

None

Not required

POST

Lambda

None

Not required

/getData

GET

OPTIONS

POST

Inactive

Default method-level caching

Inactive

Copied

https://ytu5zfhue.execute-api.eu-north-1.amazonaws.com/22BAI1471

Active deployment

3viszx on October 02, 2024, 13:59 (UTC+05:30)

## Stages

Stage actions ▾ Create stage

**Stage details** [Info](#)

Stage name: 22BAI1471

Cache cluster [Info](#): Inactive

Default method-level caching: Inactive

Active deployment: 3yiszx on October 02, 2024, 13:59 (UTC+05:30)

**Logs and tracing** [Info](#)

CloudWatch logs: Inactive

X-Ray tracing: Inactive

Custom access logging: Inactive

Floral Bliss - A World of Plants

### Add Plant

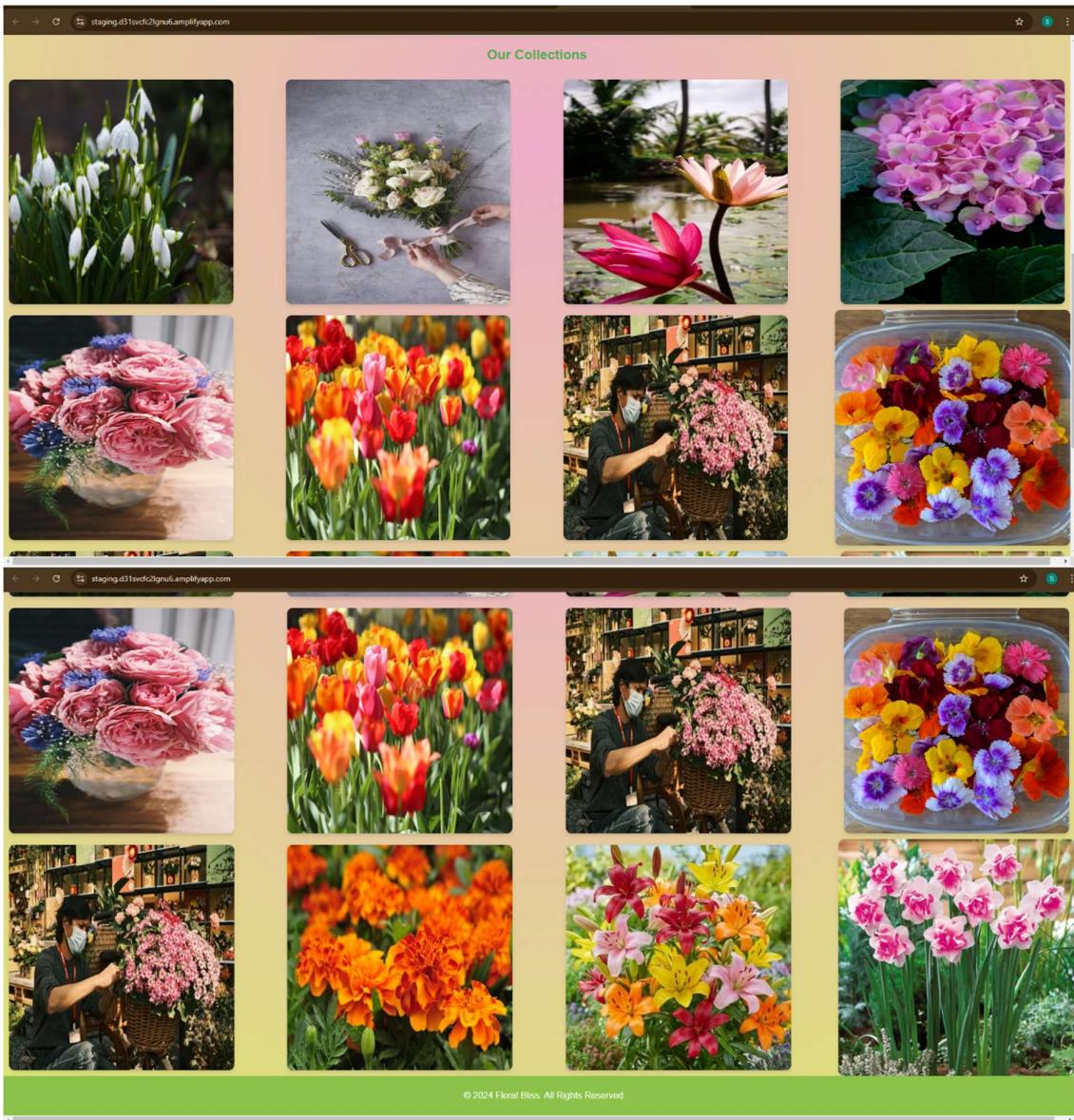
Plant ID:

Plant Name:

Cost:

**Add Plant**

Plant ID	Plant Name	Cost	Created At
2	Rose	14.45	Wed, 02 Oct 2024 08:05:26
1	Lavenders	1230	Wed, 02 Oct 2024 09:19:07
1727858135333	China Rose	12	Wed, 02 Oct 2024 08:35:35
1727858078018	China Rose	12345	Wed, 02 Oct 2024 08:47:58
23	lotus	12	Wed, 02 Oct 2024 09:13:19
456	Sunflowers	24	Wed, 02 Oct 2024 09:45:51



Overall steps

### Step 1: Create a DynamoDB Table

1. **Go to DynamoDB Console:**
  - o Navigate to the [DynamoDB Console](#).
2. **Create a Table:**
  - o Click on **Create Table**.
  - o Define a **Table Name** (e.g., `FloralBliss`).
  - o Choose a **Partition Key** (Primary Key) like `PlantID` (`String`). This serves as the unique identifier for each plant.
  - o Choose **On-Demand** or **Provisioned** throughput as needed.
  - o Click **Create** to finish.

**Reason:** The table will store the plant details, with `PlantID` as the primary key to ensure each plant has a unique identifier.

## Step 2: Create a Lambda Function

1. **Navigate to the Lambda Console:**
  - Go to the [AWS Lambda Console](#).
2. **Create a New Function:**
  - Click **Create Function**.
  - Choose **Author from Scratch**.
  - Name your function (e.g., PlantHandlerFunction).
  - Choose your desired language (e.g., Python or Node.js).
  - For **Execution Role**, select **Create a New Role with Basic Lambda Permissions**.
3. **Write Lambda Code:**
  - Inside your Lambda function, write code to handle both reading (scanning) and writing (putting) data into DynamoDB. (Refer to the code provided earlier for get\_plants and add\_plant functionality.)

**Reason:** The Lambda function will act as the backend logic to handle plant data, performing read and write operations on the DynamoDB table.

---

## Step 3: Grant Lambda Function Access to DynamoDB

1. **Create an Inline Policy:**
  - In the **Lambda Console**, click on the **Permissions** tab.
  - Under **Execution Role**, click on the role associated with your function.
  - In the **IAM Console**, go to **Inline Policies** and select **Create Policy**.
2. **Configure Policy for DynamoDB:**
  - Choose **DynamoDB** as the service.
  - Select the actions **PutItem** (to write) and **Scan** (to read).
  - In the resources section, choose your specific **DynamoDB table** (e.g., **FloralBliss**).
  - Save the policy and attach it to your Lambda execution role.

**Reason:** The Lambda function requires explicit permissions to access DynamoDB, enabling it to read and write data securely.

---

## Step 4: Test Lambda Function

1. **Create a Test Event:**
  - In the Lambda console, click **Test**.
  - Provide a sample JSON test event, like the following:

json

Copy code

{

```
"operation": "addPlant",
"PlantID": "001",
"Name": "Rose",
"Category": "Outdoor",
"Price": 10,
"Description": "Beautiful red rose"
}
```

- Click **Test** to ensure your Lambda function works correctly and stores data in the DynamoDB table.

**Reason:** Testing verifies that the Lambda function can successfully write data to the table.

---

## Step 5: Create an API Gateway

### 1. Navigate to API Gateway Console:

- Go to the [API Gateway Console](#).

### 2. Create a New REST API:

- Click **Create API** and choose **REST API**.
- Name your API (e.g., `FloralBlissAPI`).
- Create a resource (e.g., `/getData`).

### 3. Add POST and GET Methods:

- Under the `/getData` resource, add **POST** and **GET** methods.
- Link both methods to your Lambda function (`PlantHandlerFunction`).
- Ensure both methods are set to use the **Lambda Proxy Integration**.

**Reason:** API Gateway provides a REST API interface to interact with your Lambda function and thus access the data in DynamoDB.

---

## Step 6: Enable CORS on the API

### 1. Enable CORS:

- In API Gateway, go to the **POST** and **GET** methods and click **Enable CORS**.
- This allows your front-end (HTML/JS) to make requests from the browser to your API.

**Reason:** CORS is necessary to allow the front-end to communicate with your API securely.

---

## Step 7: Deploy the API

### 1. Deploy API:

- In API Gateway, click on **Deploy API**.
- Create a new **stage** (e.g., `prod` or `dev`) and deploy it.

- Take note of the **API endpoint URL**.

**Reason:** Deployment makes the API publicly available, so it can be accessed by your front-end code.

---

## Step 8: Create Front-End (HTML/JavaScript)

### 1. Design Front-End:

- Create an HTML file with a form to add plant data and display existing plants.
- Use **JavaScript** to send **POST** requests to the API to add plants and **GET** requests to display plants (see code provided in earlier steps).

### 2. Zip Front-End Files:

- Once the front-end code is ready, zip all the files (index.html, scripts.js, etc.).

**Reason:** The front-end provides users with an interface to interact with the API (e.g., adding new plants and viewing the list of plants).

---

## Step 9: Deploy Front-End with AWS Amplify

### 1. Go to AWS Amplify Console:

- Navigate to the [AWS Amplify Console](#).

### 2. Deploy Without Git:

- In the Amplify Console, select **Deploy Without Git**.
- Upload the zip file containing your front-end code.

### 3. Configure and Deploy:

- Amplify will build and deploy your front end.
- Once the deployment is complete, you will get a **URL** where your app is accessible.

**Reason:** AWS Amplify is used to host and deploy your front-end code, making your app accessible to users via a web URL.

---

## Step 10: Test the Full Application

### 1. Go to Your App's URL:

- Use the URL provided by Amplify to access your app.
- Test adding a plant via the form and verify that it shows up in the list of available plants.
- Check if the back-end API is working correctly by interacting with the DynamoDB table.

**Reason:** This step ensures that all components (front-end, API, Lambda, DynamoDB) are working together as intended.