AWS PROJECT

Architect and Build an End-to-End AWS Web Application

Overview of AWS Services Used:

1. **AWS Amplify** - To host and deploy the frontend HTML page.
2. **AWS Lambda** - To implement backend logic for processing requests.
3. **Amazon API Gateway** - To provide a public API endpoint for invoking the backend Lambda function.
4. **Amazon DynamoDB** - To store and retrieve data.
5. **AWS Identity and Access Management (IAM)** - To configure permissions between services securely.

Steps to Complete the Project

1. **Host the Frontend Using AWS Amplify** - Deploy an HTML file as a static website.
2. **Create and Configure a Lambda Function** - Set up a serverless function to handle backend processing.
3. **Set Up API Gateway** - Establish a REST API to link the frontend with the Lambda backend.
4. **Set Up DynamoDB** - Create a database to store processed data.
5. **Update Permissions and Code** - Ensure that the Lambda function has the required permissions to access DynamoDB.
6. **Deploy and Update the Frontend** - Redeploy the HTML file with API endpoint integration.

Step 1: Host the Frontend Using AWS Amplify

1. Prepare the HTML File:

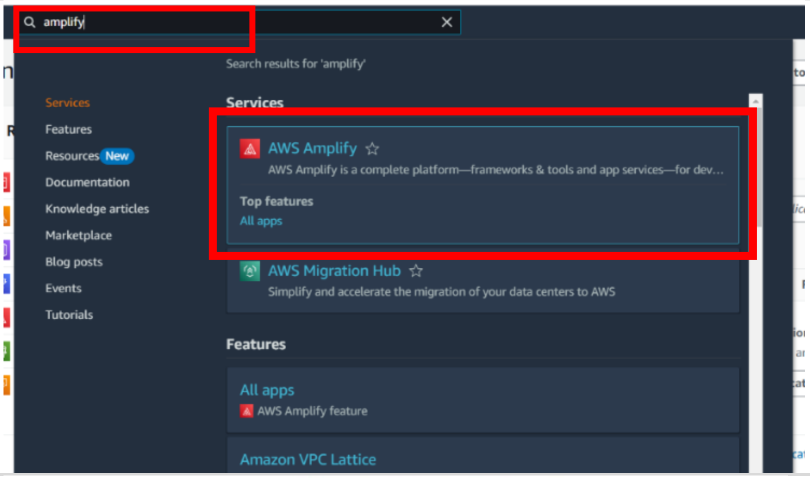
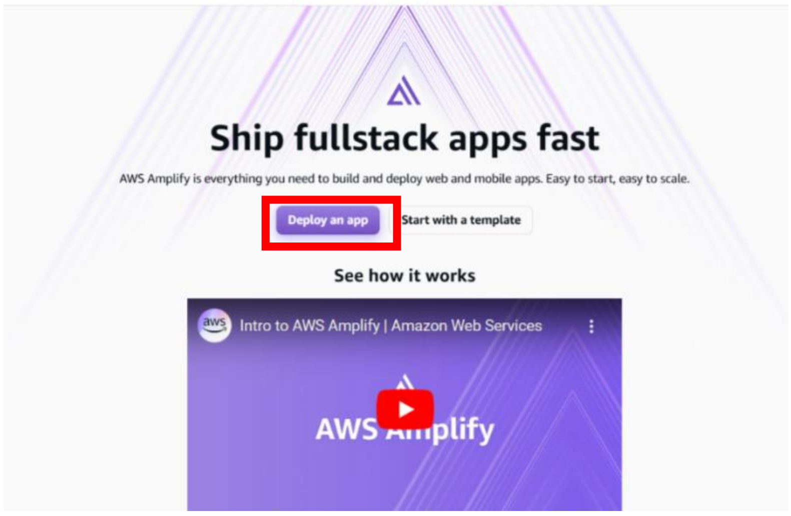
a) Create your index.html file with the necessary content.

Example index.html

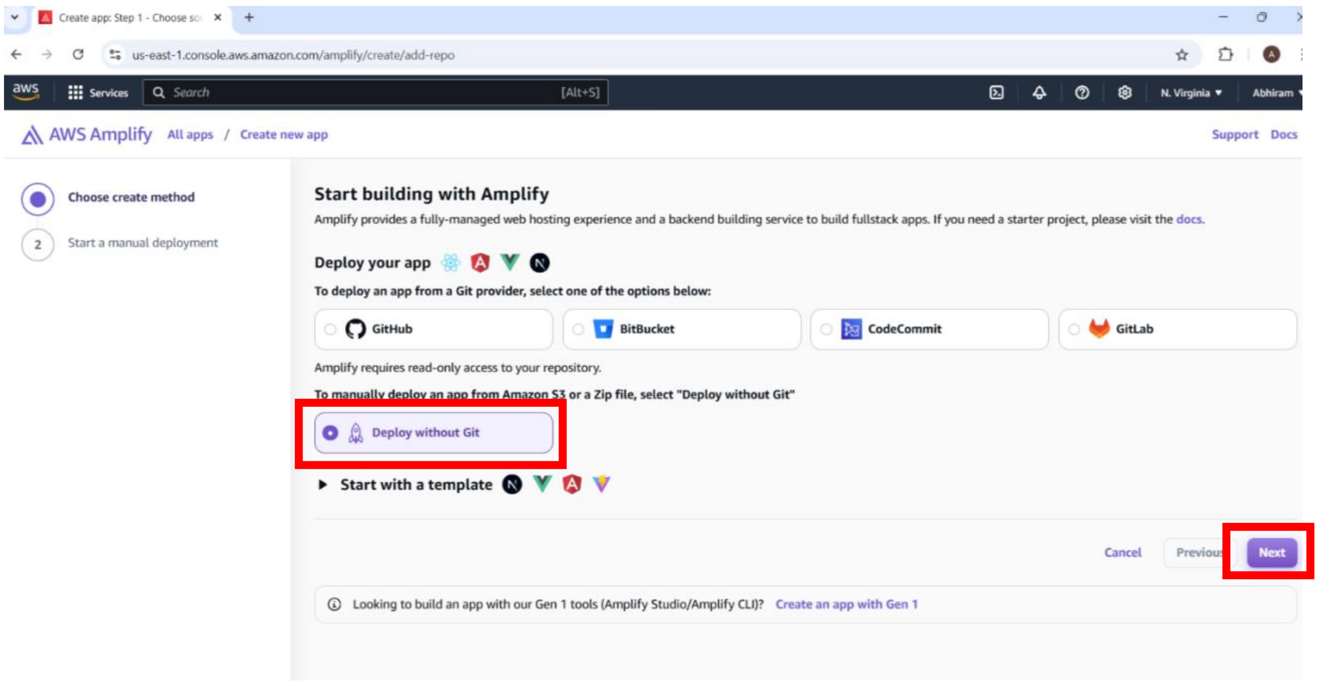
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Age Calculator</title>
</head>
<body>
    <h1>Age Calculator</h1>
    <form id="ageForm">
        Enter your birth year: <input type="number" id="birthYear"
required><br><br>
        <button type="button" onclick="calculateAge()">Calculate Age</button>
    </form>

    <script>
        function calculateAge() {
            let birthYear = document.getElementById("birthYear").value;
            fetch('<API_INVOKE_URL>', {
                method: 'POST',
                body: JSON.stringify({ birthYear: birthYear }),
                headers: {
                    'Content-Type': 'application/json'
                }
            })
            .then(response => response.json())
            .then(data => {
                alert('Your age is: ' + data.age);
            });
        }
    </script>
</body>
</html>
```

b) Compress the file into a .zip archive.

2. Deploy the Application Using AWS Amplify:

In the AWS Console, search for **Amplify.**	 Search results for 'amplify' AWS Amplify is a complete platform—frameworks & tools and app services—for dev… Simplify and accelerate the migration of your data centers to AWS
Click **Deploy an app**	 AWS Amplify is everything you need to build and deploy web and mobile apps. Easy to start, easy to scale.

Deploy without Git.
Click **Next**



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.

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

Amplify requires read-only access to your repository.

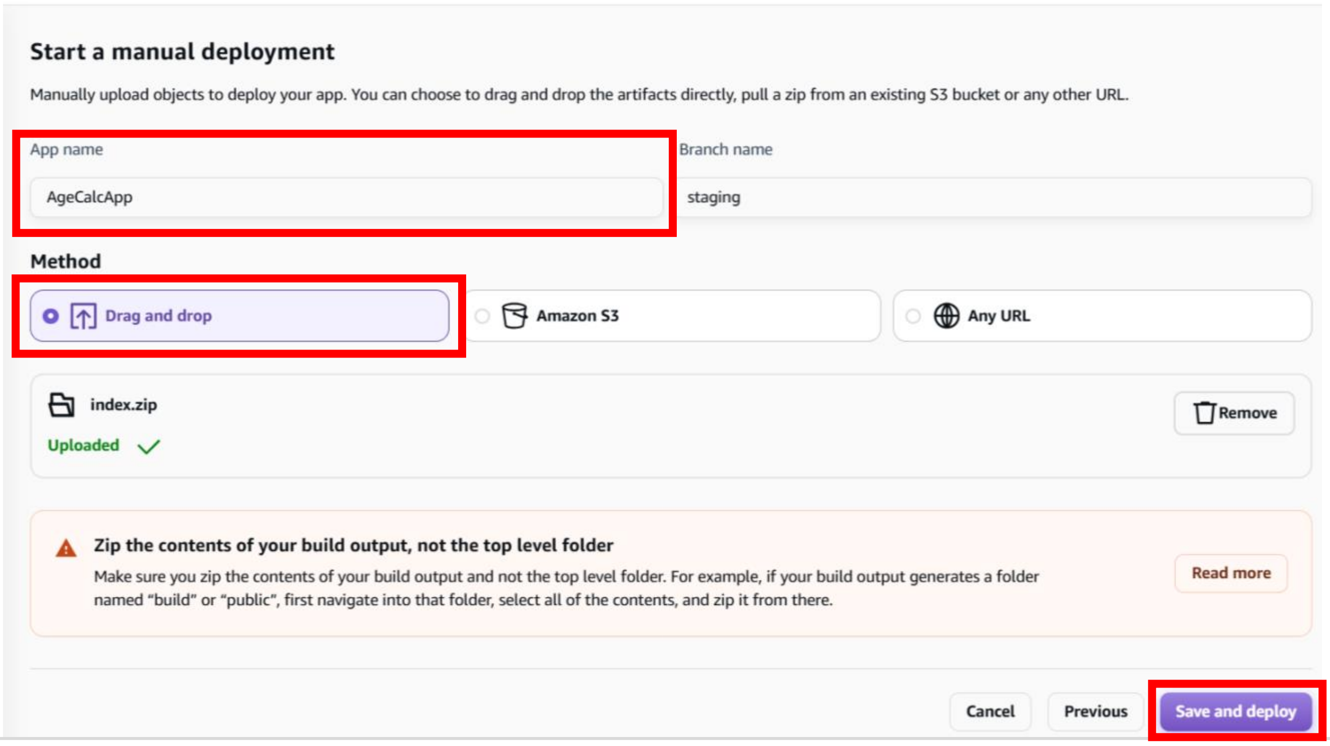
To manually deploy an app from Amazon S3 or a Zip file, select "Deploy without Git"

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

Give your App name.
`eg: AgeCalcApp`

Drag and drop the .zip file containing index.html into the Amplify console.

Click **Save and Deploy**.



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: AgeCalcApp

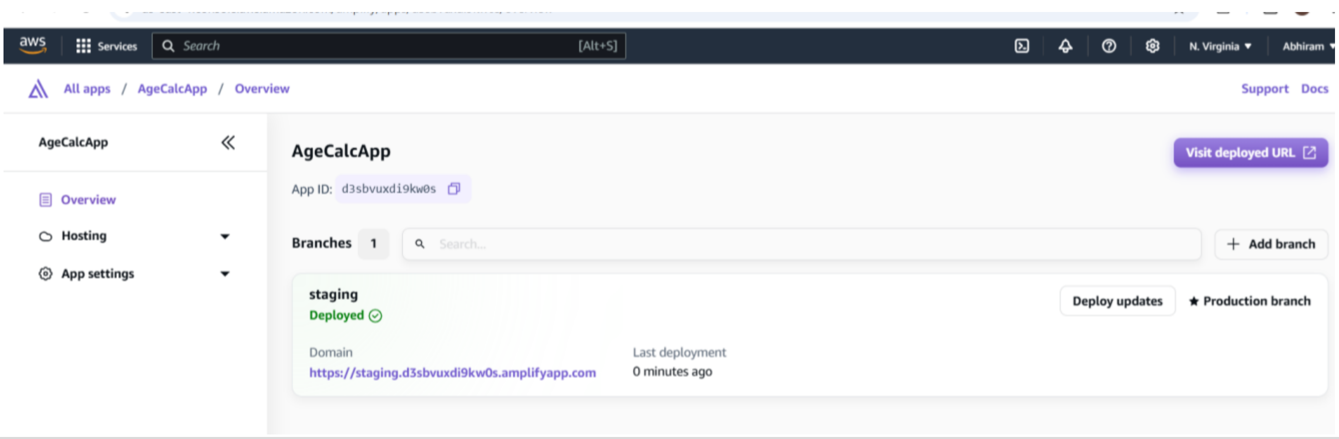
Branch name: staging

Method: [✓ Drag and drop] Amazon S3 Any URL

index.zip
Uploaded ✓

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.

Wait for the deployment to complete (**Deployed**✅), and click the domain link to view the live site.



All apps / AgeCalcApp / Overview

AgeCalcApp

App ID: `d3sbvuxdi9kw0s`

Branches 1

staging
Deployed ✓

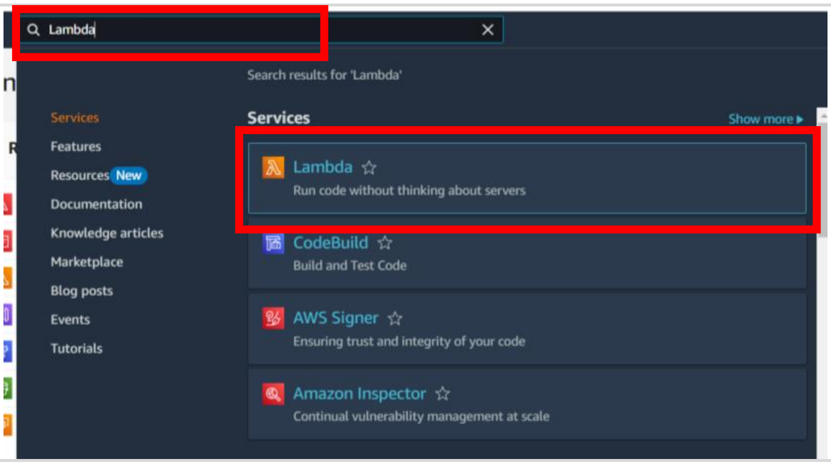
Domain: https://staging.d3sbvuxdi9kw0s.amplifyapp.com

Last deployment: 0 minutes ago

Step 2: Create a Lambda Function

1. Set Up Lambda:

Open a new tab and search for Lambda in the AWS Console.

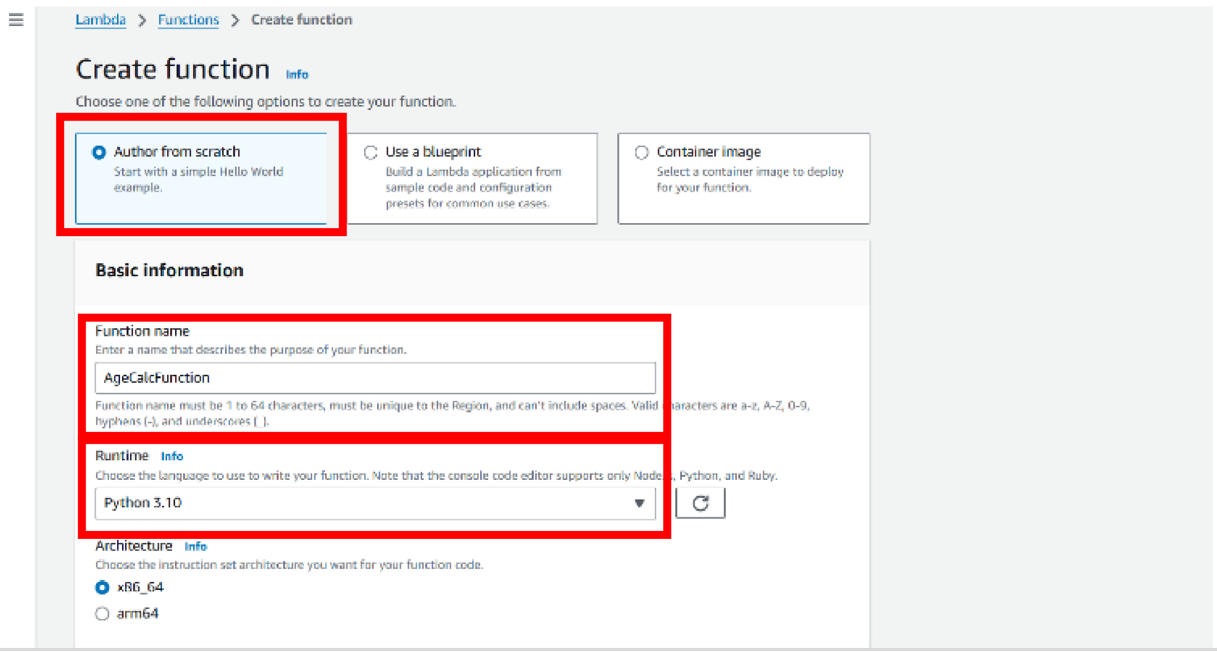


Click Create Function

Choose Author from scratch.

Provide a Function name. `eg: AgeCalcFunction`
select the latest version of Python as the runtime

Click Create function



Create function

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.

Basic information

Function name
Enter a name that describes the purpose of your function.
AgeCalcFunction
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 3.10

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

2. Add Code to Lambda:

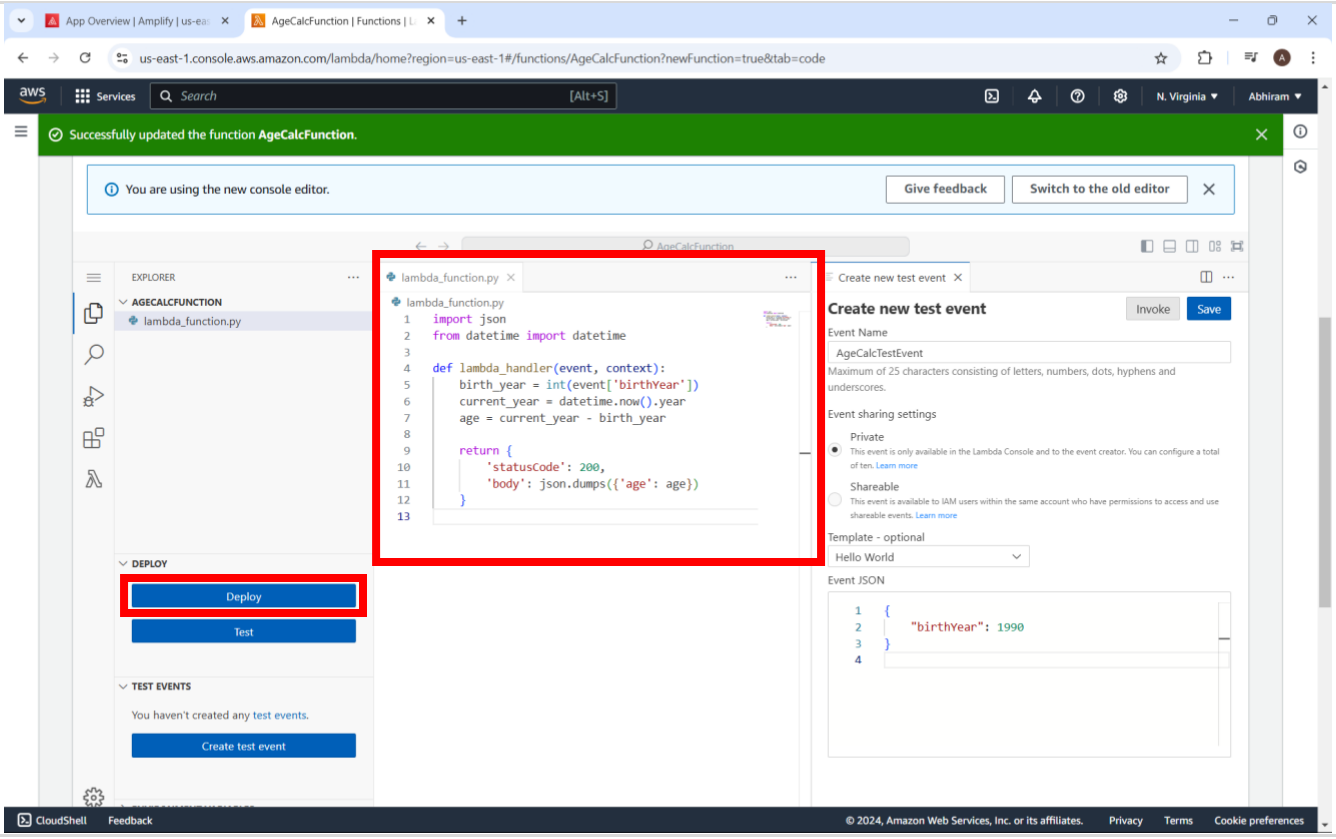
In the Lambda editor, replace the default code with your custom Lambda function code

Example Lambda code

```
import json
from datetime import datetime

def lambda_handler(event, context):
    birth_year = int(event['birthYear'])
    current_year = datetime.now().year
    age = current_year - birth_year

    return {
        'statusCode': 200,
        'body': json.dumps({'age': age})
    }
```

Save the code using Ctrl + S.

Click **Deploy** to apply the changes.


App Overview | Amplify | us-eas
AgeCalcFunction | Functions | L
us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/AgeCalcFunction?newFunction=true&tab=code
Services
Search [Alt+S]
N. Virginia
Abhiram
Give feedback
Switch to the old editor
EXPLORER
AGECALCFUNCTION
lambda_function.py
DEPLOY
Deploy
Test
TEST EVENTS
Create test event
Invoke
Save
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Successfully updated the function **AgeCalcFunction**.

You are using the new console editor.

You haven't created any test events.

```
import json
from datetime import datetime

def lambda_handler(event, context):
    birth_year = int(event['birthYear'])
    current_year = datetime.now().year
    age = current_year - birth_year

    return {
        'statusCode': 200,
        'body': json.dumps({'age': age})
    }
```

Create new test event

Event Name: AgeCalcTestEvent

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 ten. 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

```
{
    "birthYear": 1990
}
```

3. Test the Lambda Function:

Click **Create test event**.

Name the test event `eg: AgeCalcTestEvent`

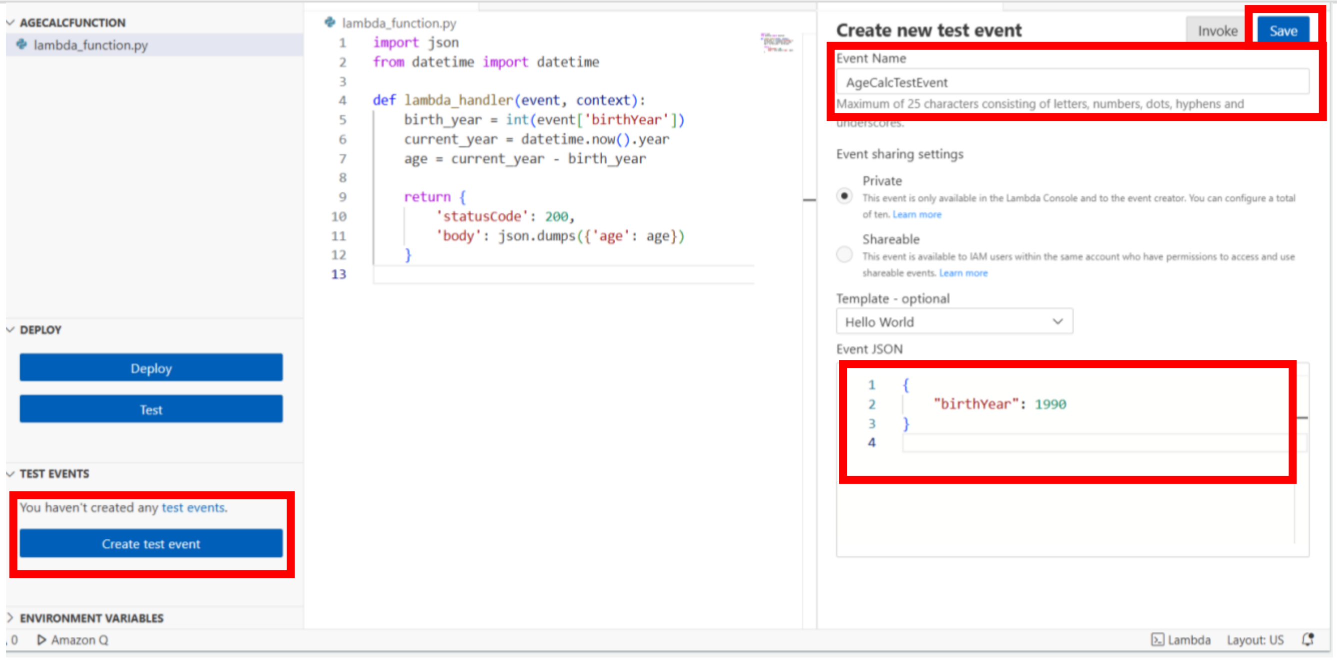
Enter your test data in the event box.

Example Test Event

```
{
    "birthYear": 1990
}
```

Click **Save**.

Click **Test** and Select the created test event to test. Verify the function's output.



```python
import json
from datetime import datetime

def lambda_handler(event, context):
    birth_year = int(event['birthYear'])
    current_year = datetime.now().year
    age = current_year - birth_year

    return {
        'statusCode': 200,
        'body': json.dumps({'age': age})
    }
```

Create new test event

Event Name: AgeCalcTestEvent

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 ten. Learn more

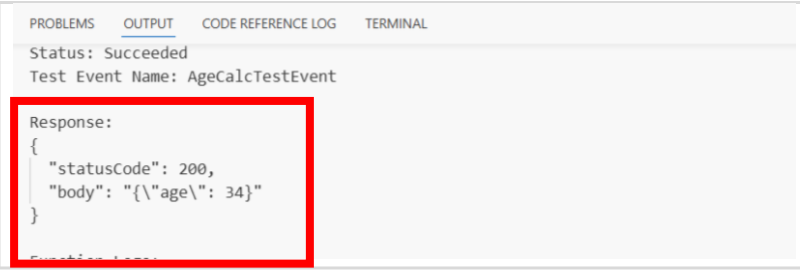
○ Shareable — This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - optional: Hello World

Event JSON

```
{
    "birthYear": 1990
}
```

Verify the function's output.

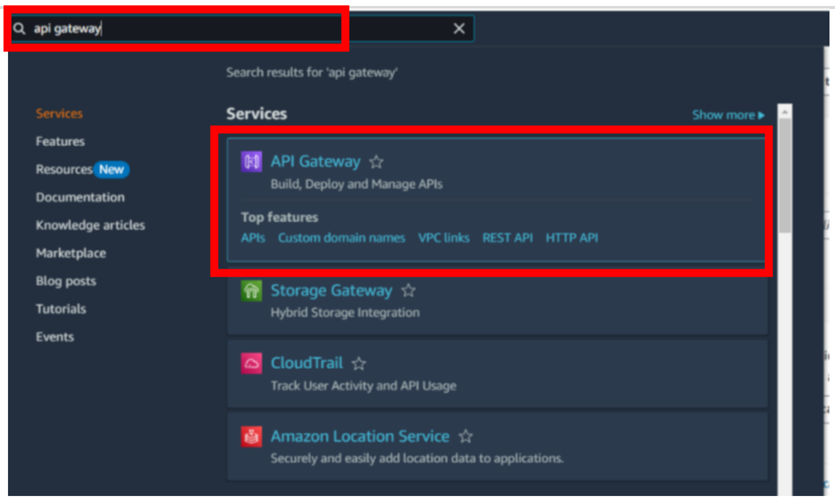


```
Status: Succeeded
Test Event Name: AgeCalcTestEvent

Response:
{
  "statusCode": 200,
  "body": "{\"age\": 34}"
}
```

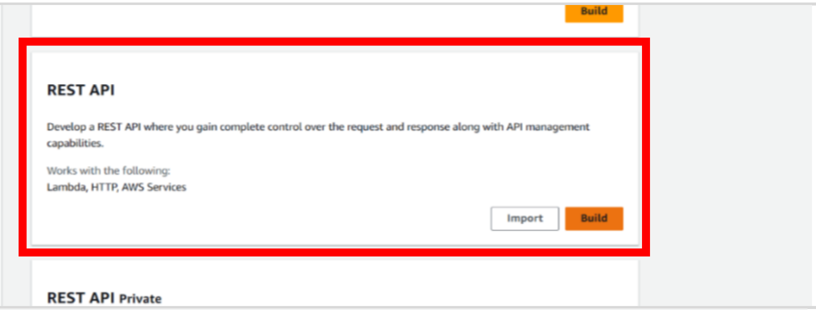
Step 3: Set Up API Gateway

1. Create a New API:

Open a new tab, search for **API Gateway**



Click **Build** in the **REST API** section



REST API

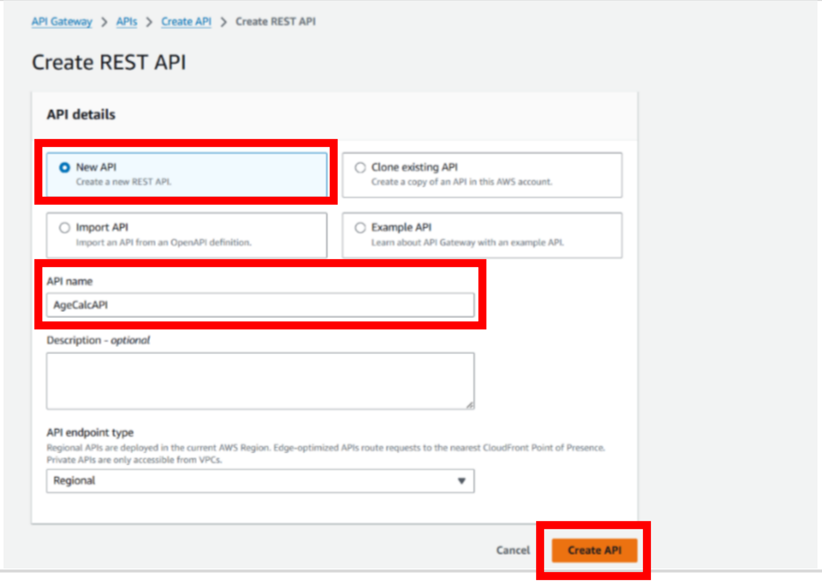
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

Choose **New API**

Give **API name**.
`eg: AgeCalcAPI`

Click **Create API**.


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.
Cancel
Create API

API name: AgeCalcAPI

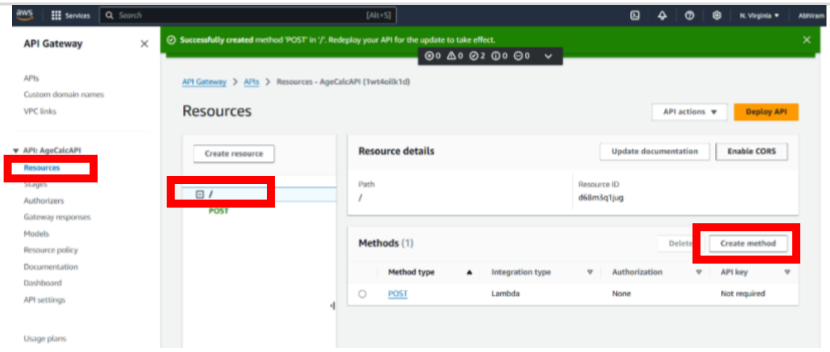
Description - *optional*

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

2. Configure API Gateway:

Ensure you're in **Resources** section, inside it **/** is selected

Click **Create Method**



Successfully created method 'POST' in '/'. Redeploy your API for the update to take effect.

API Gateway > APIs > Resources - AgeCalcAPI (1wt4oilk1d)

Resources

Resource details

Path: /

Resource ID: d68m3q1jug

Methods (1)

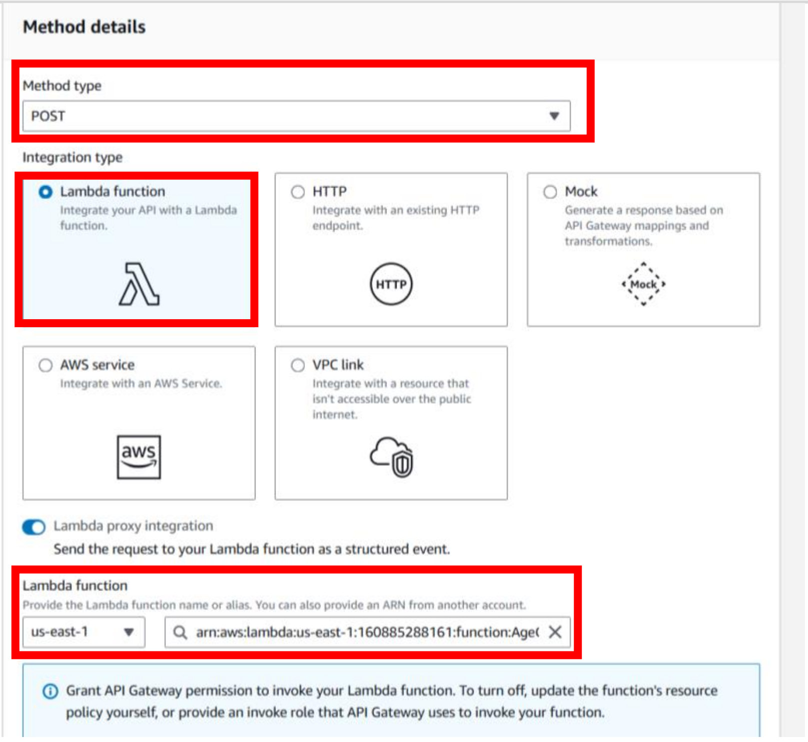
	Method type	Integration type	Authorization	API key
○	POST	Lambda	None	Not required

select **POST** as Method type

Choose **Lambda Function** as the **integration type**

Enable **Lambda proxy integration**

Select the **Lambda function** you created earlier.



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: on — Send the request to your Lambda function as a structured event.

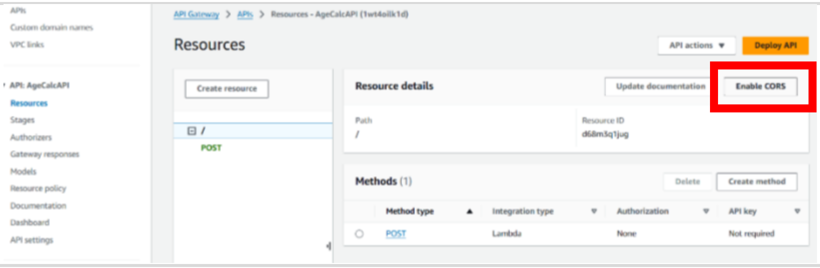
Lambda function — Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1 | arn:aws:lambda:us-east-1:160885288161:function:Age(

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.

3. Enable CORS:

Go back to / and click **Enable CORS**.



API Gateway > APIs > Resources - AgeCalcAPI (1wt4oilk1d)

Resources

Resource details

Path: /

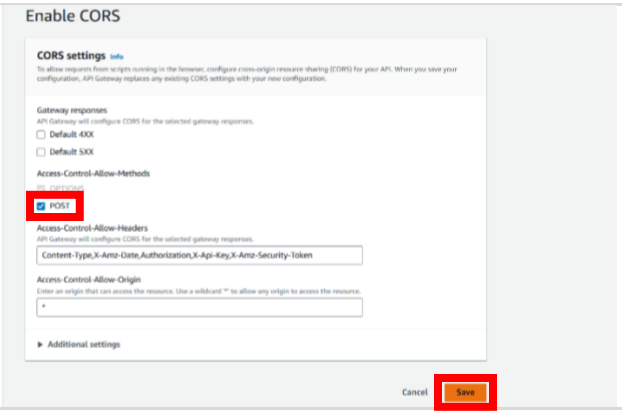
Resource ID: d68m3q1jug

Methods (1)

	Method type	Integration type	Authorization	API key
○	POST	Lambda	None	Not required

Check on POST in Access-Control-Allow-Method

Click Save



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

- [x] OPTIONS
- [x] POST

Access-Control-Allow-Headers

API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

Access-Control-Allow-Origin

Enter an origin that can access the resource. Use a wildcard '*' to allow any origin to access the resource.

*

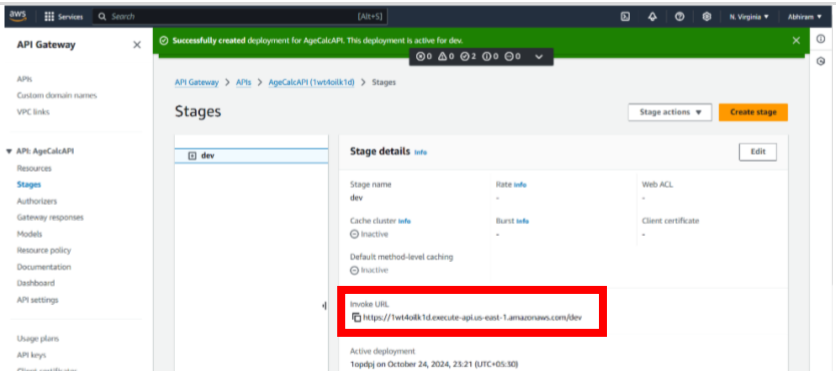
4. Deploy the API:

Click **Deploy API**

On stage, select ***New Stage***.

Give stage name as 'dev'

Copy and save the '**Invoke URL**' on notepad which we will use later



Successfully created deployment for AgeCalcAPI. This deployment is active for dev.

API Gateway > APIs > AgeCalcAPI (1wt4oilk1d) > Stages

Stages

dev

Stage details Info

Stage name: dev

Rate Info: -

Web ACL: -

Cache cluster Info: Inactive

Burst Info: -

Client certificate: -

Default method-level caching: Inactive

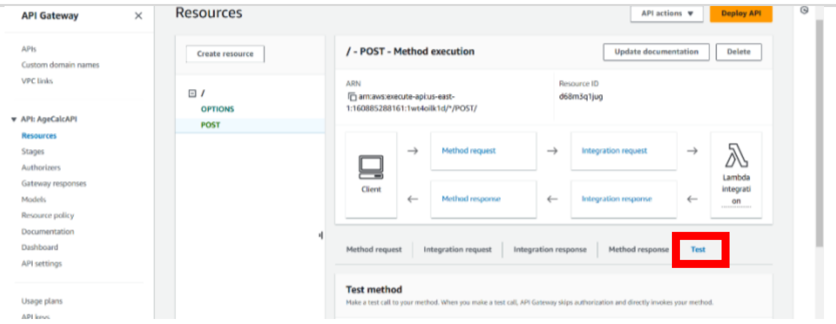
Invoke URL: https://1wt4oilk1d.execute-api.us-east-1.amazonaws.com/dev

Active deployment: 1opdpj on October 24, 2024, 23:21 (UTC+05:30)

5. Test the API:

In API Gateway, return to **Resources** then select POST.

Go to the **Test** section



Resources

/ - POST - Method execution

ARN: arn:aws:execute-api:us-east-1:160885288161:1wt4oilk1d/*/POST/

Resource ID: d68m3q1jug

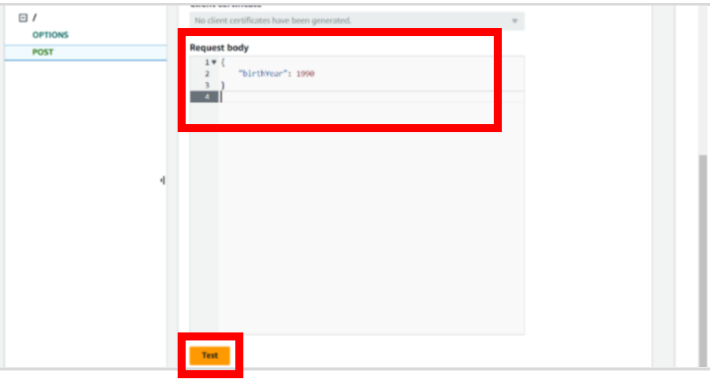
Test method

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

scroll down & input **test data** into the '**Request Body**'

Example Test Event

```
{
    "birthYear": 1990
}
```



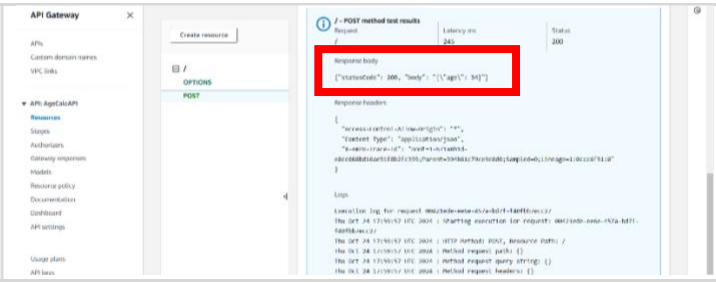
No client certificates have been generated.

Request body

```
{
    "birthYear": 1990
}
```

Click **Test**.

Confirm that the correct output is returned.



/ - POST method test results

Request: /   Latency ms: 245   Status: 200

Response body

```
{"statusCode": 200, "body": "{\"age\": 34}"}
```

Response headers

```
{
  "Access-Control-Allow-Origin": "*",
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-[illegible]
}
```

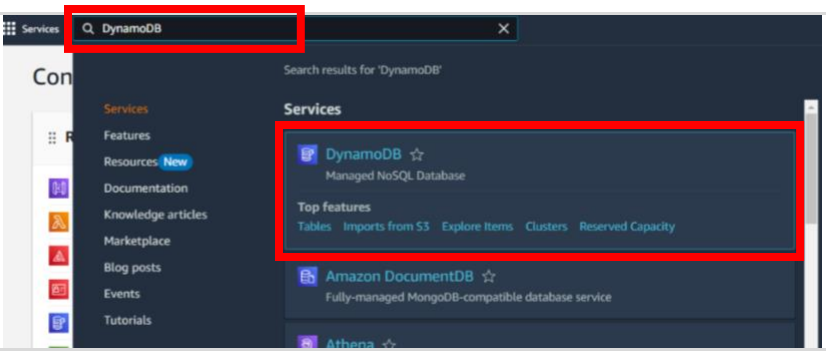
Logs

```
Execution log for request [illegible]
Thu Oct 24 17:59:57 UTC 2024 : Starting execution for request: [illegible]
Thu Oct 24 17:59:57 UTC 2024 : HTTP Method: POST, Resource Path: /
Thu Oct 24 17:59:57 UTC 2024 : Method request path: {}
Thu Oct 24 17:59:57 UTC 2024 : Method request query string: {}
Thu Oct 24 17:59:57 UTC 2024 : Method request headers: {}
```

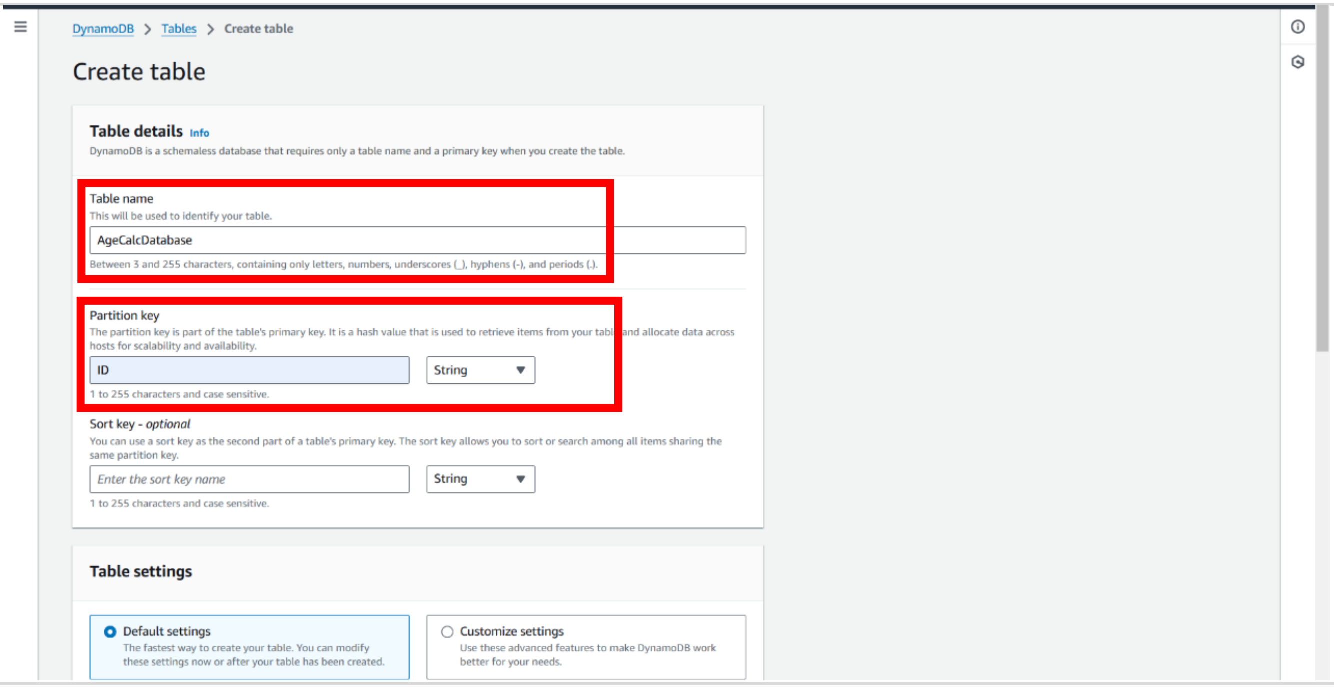
Step 4: Set Up DynamoDB

1. Create a DynamoDB Table:

Open a new tab, search for **DynamoDB**, and click **Create Table**.



Give a name for the table `eg: AgeCalcDatabase`
Set ID as the partition key.
Click **Create Table** from the bottom.



DynamoDB > Tables > Create 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.
AgeCalcDatabase
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.
ID   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.

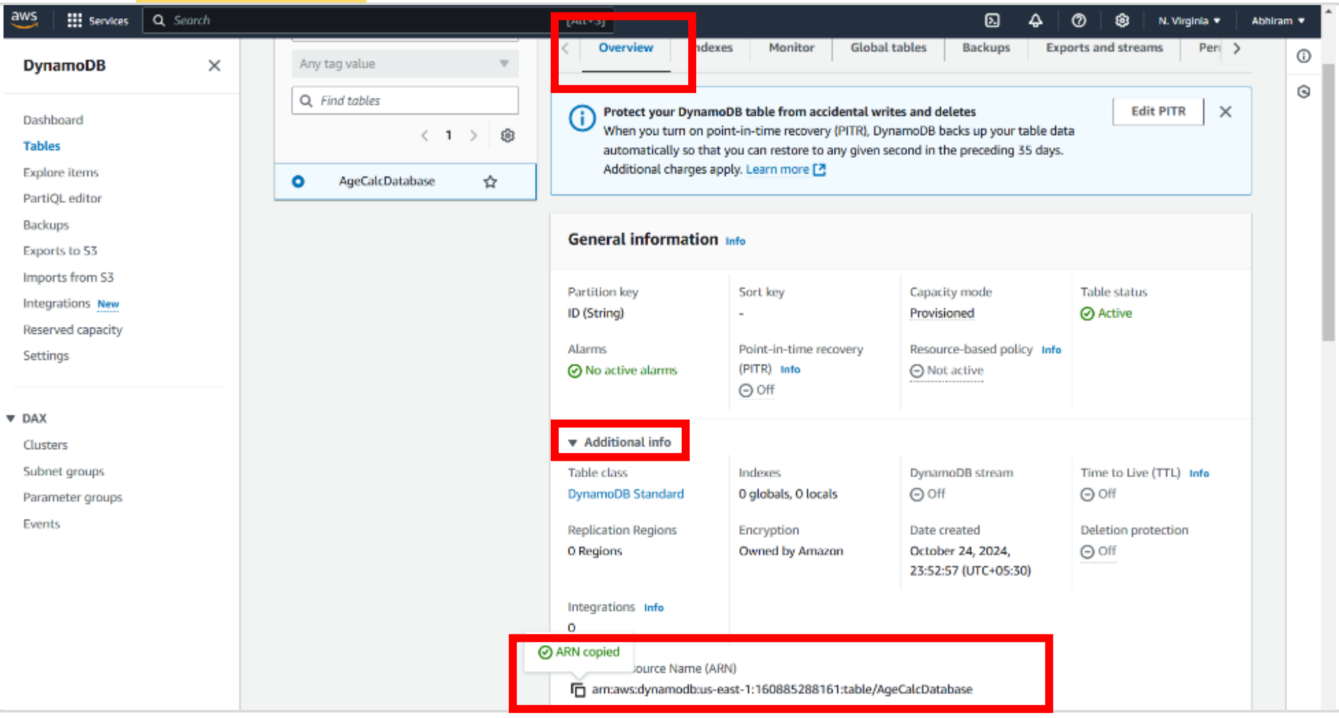
Table settings

[✓ Default settings] The fastest way to create your table. You can modify these settings now or after your table has been created.

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

2. Copy the Table ARN:

After the table is created, open it and go to the **Overview** section.
click on '**Additional info**' copy the **Amazon Resource Name (ARN)** and save it for later



Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. Learn more

General information

Partition key: ID (String)	Sort key: -	Capacity mode: Provisioned	Table status: Active
Alarms: No active alarms	Point-in-time recovery (PITR): Off	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 24, 2024, 23:52:57 (UTC+05:30)	Deletion protection: Off
Integrations: 0			

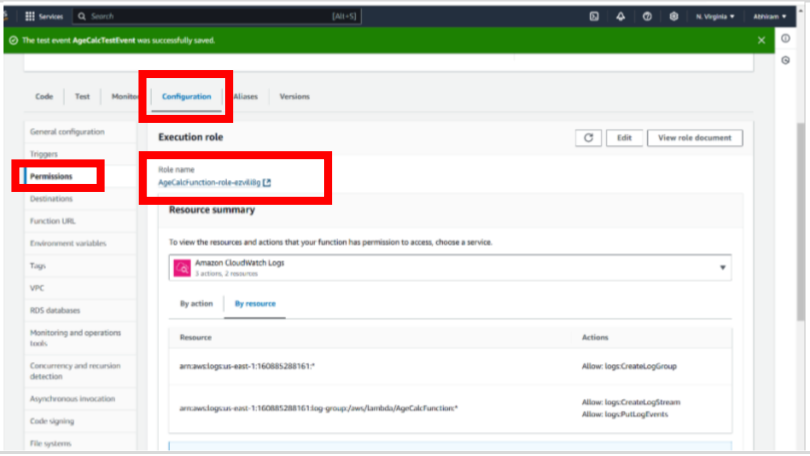
ARN copied
Amazon Resource Name (ARN)
arn:aws:dynamodb:us-east-1:160885288161:table/AgeCalcDatabase

Step 5: Update Lambda Permissions and Code

1. Add DynamoDB Permissions to Lambda:

Go back to the **Lambda** function and navigate to the **Configuration** tab → **Permissions**

Click the **Execution Role** link, which opens in a new tab.



The test event AgeCalcTestEvent was successfully saved.

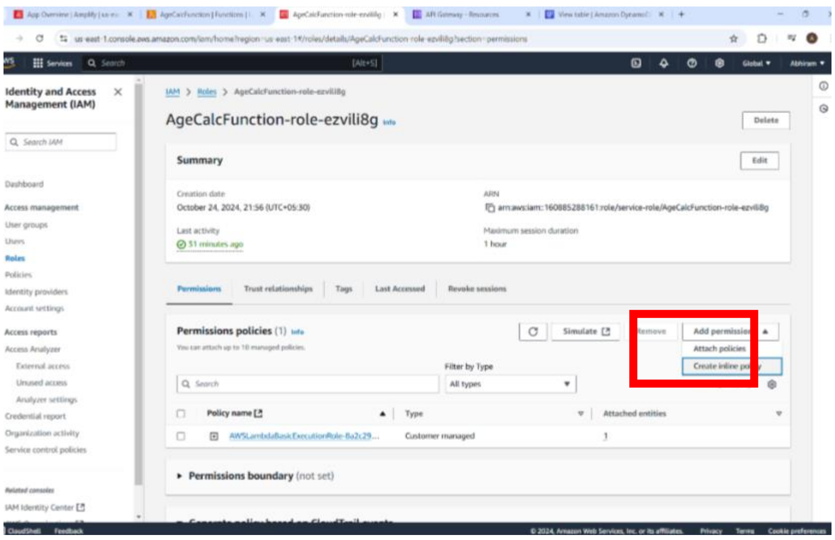
Execution role
Role name
AgeCalcFunction-role-ezvili8g

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:us-east-1:160885288161:*	Allow: logs:CreateLogGroup
arn:aws:logs:us-east-1:160885288161:log-group:/aws/lambda/AgeCalcFunction:*	Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Click ‘**Add Permission**’

Select ‘**create inline policy**’


AgeCalcFunction-role-ezvili8g

Summary

Creation date: October 24, 2024, 21:56 (UTC+05:30)

Last activity: 51 minutes ago

Maximum session duration: 1 hour

Permissions policies (1)

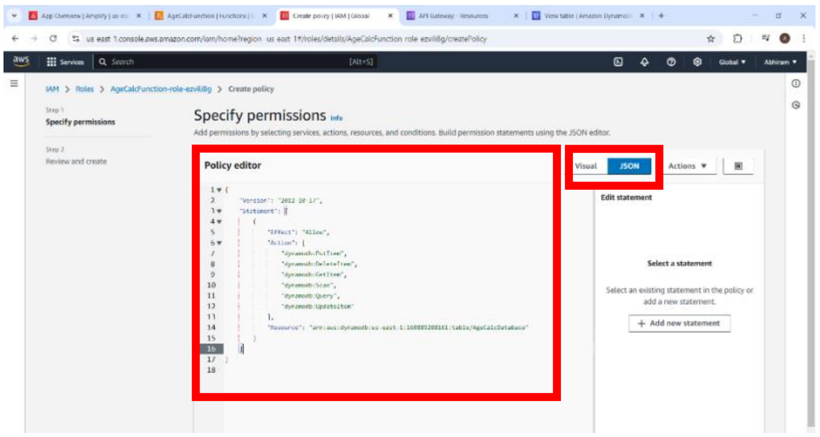
	Policy name	Type	Attached entities
	AWSLambdaBasicExecutionRole-8a2c29...	Customer managed	1

Switch to the **JSON** tab and add a new inline policy by pasting the appropriate policy.

Replace <TABLE_ARN> the ARN that we copied before in the code

Example inline policy code

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "dynamodb:PutItem",
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:Scan",
            "dynamodb:Query",
            "dynamodb:UpdateItem"
         ],
         "Resource": "<TABLE_ARN>"
      }
   ]
}
```


Specify permissions

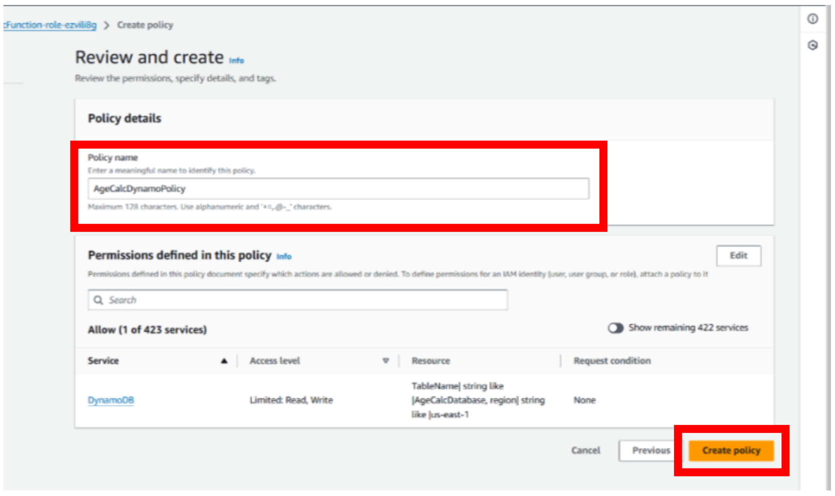
Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

Name the policy.

`eg: AgeCalcDynamicPolicy`

Click **Create Policy.**


Review and create

Review the permissions, specify details, and tags.

Policy details

Policy name

Enter a meaningful name to identify this policy.

AgeCalcDynamoPolicy

Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

Permissions defined in this policy

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

Allow (1 of 423 services)

Service	Access level	Resource	Request condition
DynamoDB	Limited: Read, Write	TableName| string like |AgeCalcDatabase, region| string like |us-east-1	None

2. Update the Lambda Function Code:

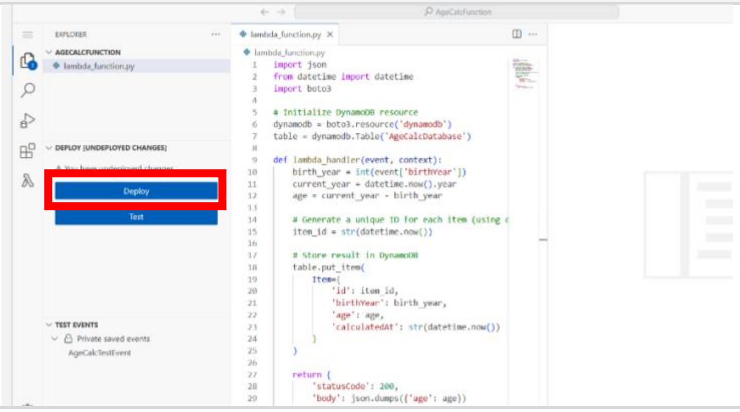
Modify the Lambda function to write data to DynamoDB

Example Lambda code

```
import json
from datetime import datetime
import boto3

# Initialize DynamoDB resource
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('AgeCalcDatabase')

def lambda_handler(event, context):
    # Get the birth year from the event
    birth_year = int(event['birthYear'])

    # Calculate the current year and the age
    current_year = datetime.now().year
    age = current_year - birth_year

    # Generate a unique ID for each item (use a timestamp to avoid duplicates)
    item_id = str(datetime.now().timestamp())

    # Store the result in DynamoDB
    table.put_item(
        Item={
            'ID': item_id,  # Use 'ID' in capital letters to match the partition key in DynamoDB
            'birthYear': birth_year,
            'age': age,
            'calculatedAt': str(datetime.now())  # Store the current timestamp
        }
    )

    # Return the calculated age
    return {
        'statusCode': 200,
        'body': json.dumps({'age': age})
    }
```

Save the changes with **Ctrl + S**

Click **Deploy**



```
import json
from datetime import datetime
import boto3

# Initialize DynamoDB resource
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('AgeCalcDatabase')

def lambda_handler(event, context):
    birth_year = int(event['birthYear'])
    current_year = datetime.now().year
    age = current_year - birth_year

    # Generate a unique ID for each item (using c
    item_id = str(datetime.now())

    # Store result in DynamoDB
    table.put_item(
        Item={
            'id': item_id,
            'birthYear': birth_year,
            'age': age,
            'calculatedAt': str(datetime.now())
        }
    )

    return {
        'statusCode': 200,
        'body': json.dumps({'age': age})
```

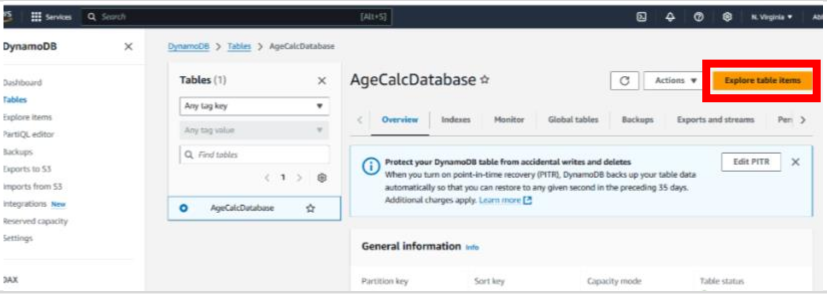
Test the updated function to verify the correct output.

3. Verify Data in DynamoDB:

Go to **DynamoDB** tab

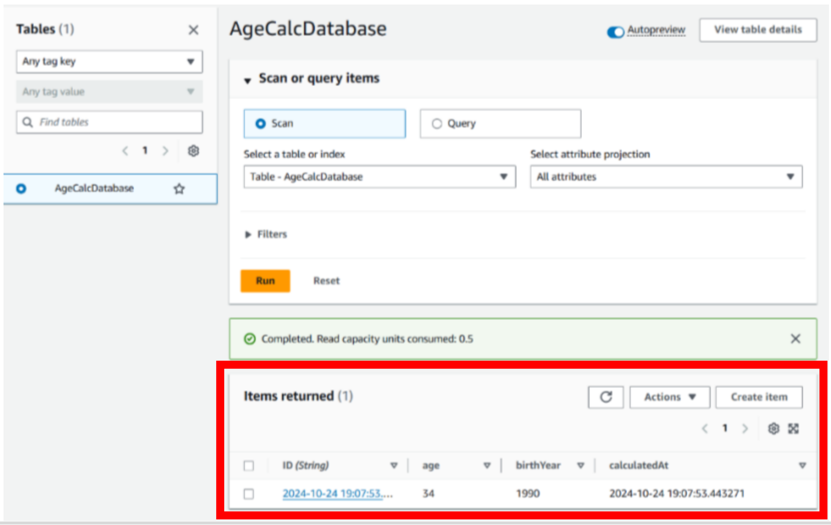
Open the **Database Table**

Click **Explore Table Items**



Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. Learn more

The test **results will be stored** in the table.



Select a table or index: Table - AgeCalcDatabase

Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

ID (String)	age	birthYear	calculatedAt
2024-10-24 19:07:53....	34	1990	2024-10-24 19:07:53.443271

Step 6: Redeploy Updated HTML via Amplify

1. Update the HTML File:

Open the **index.html** file in notepad

Replace **<API_INVOKE_URL>** with the Invoke URL from **API Gateway** (which was copied before)

Example index.html code

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Age Calculator</title>
</head>
<body>
   <h1>Age Calculator</h1>
   <form id="ageForm">
      Enter your birth year: <input type="number" id="birthYear" required><br><br>
      <button type="button" onclick="calculateAge()">Calculate Age</button>
   </form>

   <script>
          function calculateAge() {
         let birthYear = document.getElementById("birthYear").value;

         fetch('<API_INVOKE_URL>', {
                 method: 'POST',
                 body: JSON.stringify({ birthYear: birthYear }),
                 headers: {
                         'Content-Type': 'application/json'
                 }
         })
         .then(response => {
                 // Check if the response is okay and parse JSON
                 if (!response.ok) {
                         throw new Error('Network response was not ok');
                 }
                 return response.json();
         })
         .then(data => {
                 if (data && data.age !== undefined) {
                         alert('Your age is: ' + data.age);
                 } else {
                         alert('Error: Age not calculated');
                 }
         })
         .catch(error => {
                 console.error('Error:', error);
                 alert('Error: Unable to calculate age.');
         });
         }
   </script>

</body>
</html>
```

Save the file and compress this new **index.html** into a **.zip** file

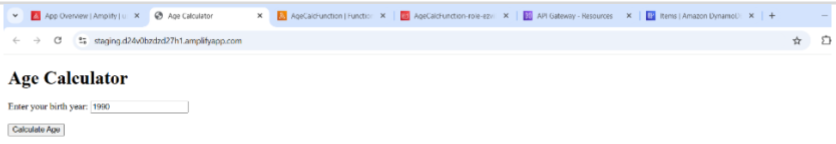
2. Deploy the Updated HTML File:

Go back to the **AWS Amplify** tab and open your app

Click **Deploy Update** and drag the updated .zip file.

Click **Save and Deploy** to publish the changes.

Click the **domain link** to open the website.



Age Calculator

Enter your birth year: 1990