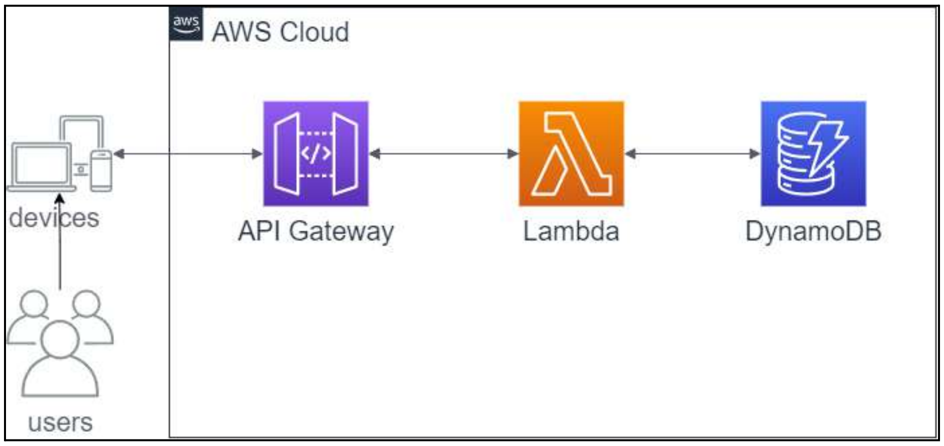
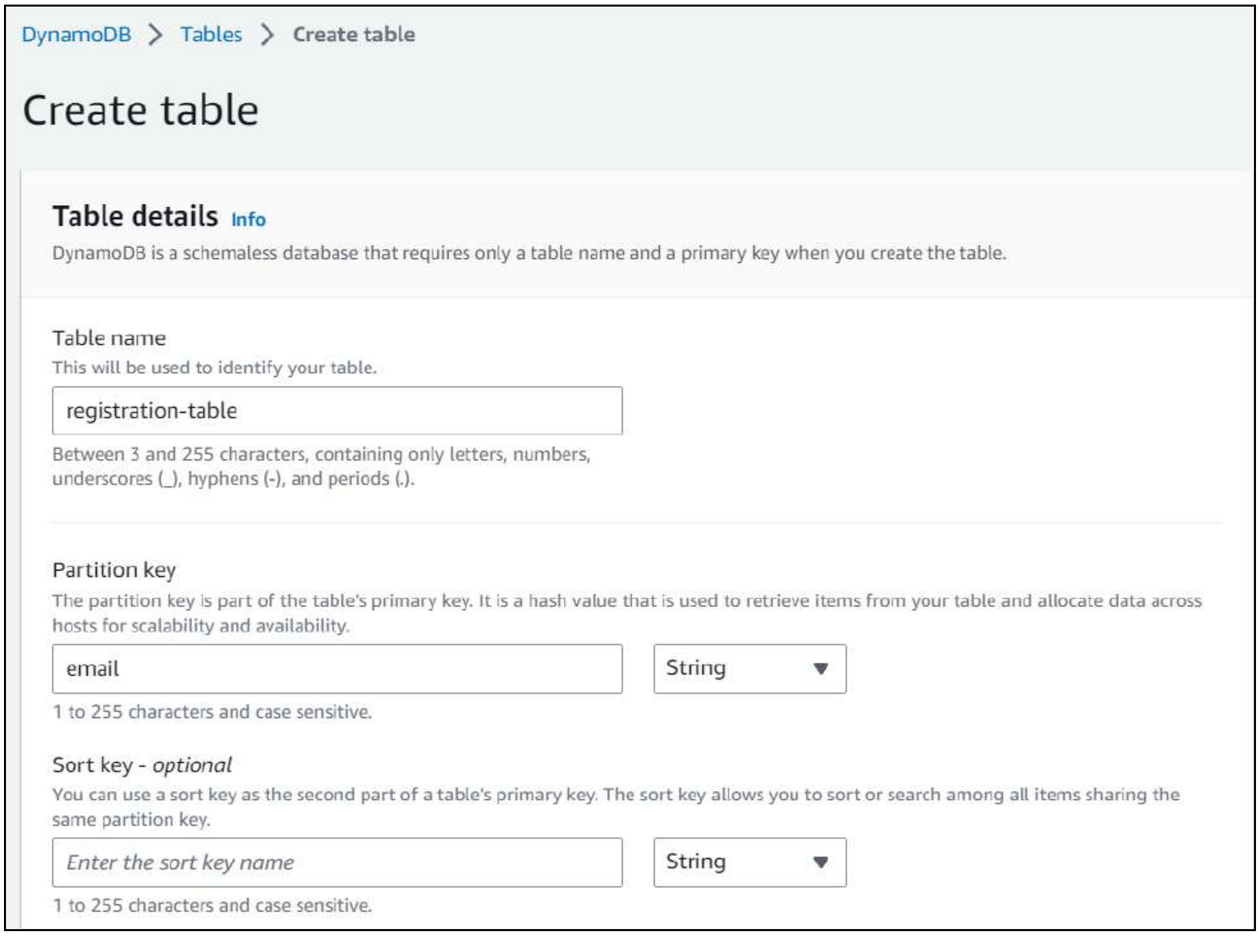
Building Serverless Registration WebApp

Source: https://www.youtube.com/playlist?list=PLjl2dJMjkDjlSARq_6kppW3nvUVIfy0Ut

Architecture:



Create 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.

registration-table

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.

email | String

1 to 255 characters and case sensitive.

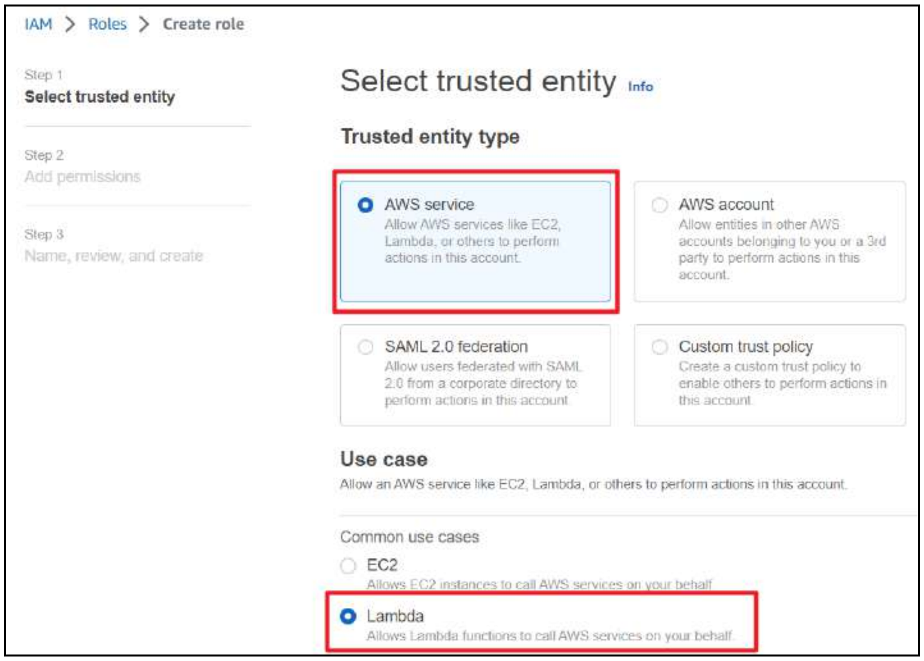
Sort key - *optional*

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.

Create IAM Role for Lambda



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity Info

Trusted entity type

[✓ AWS service]
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

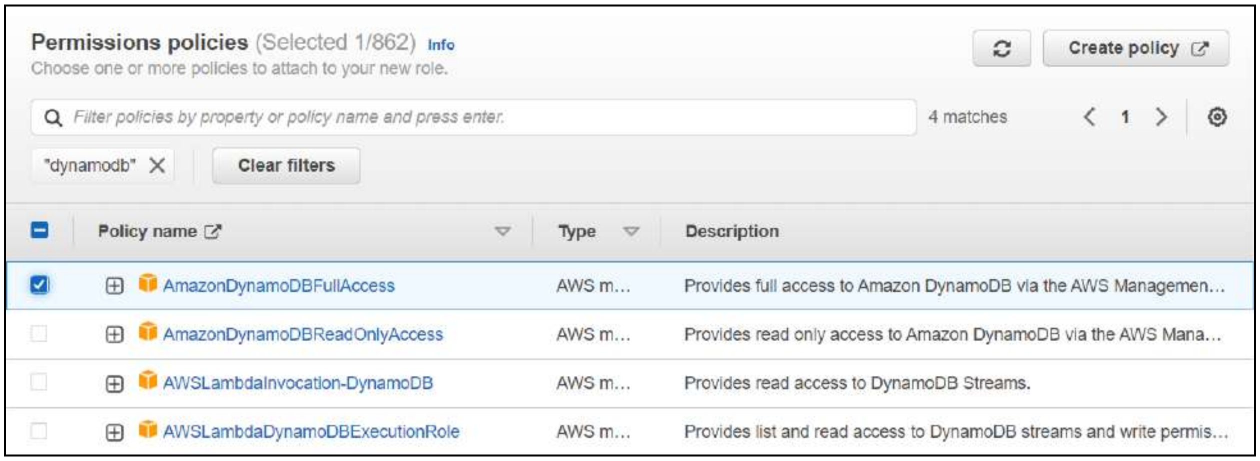
Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

[✓ Lambda]
Allows Lambda functions to call AWS services on your behalf.



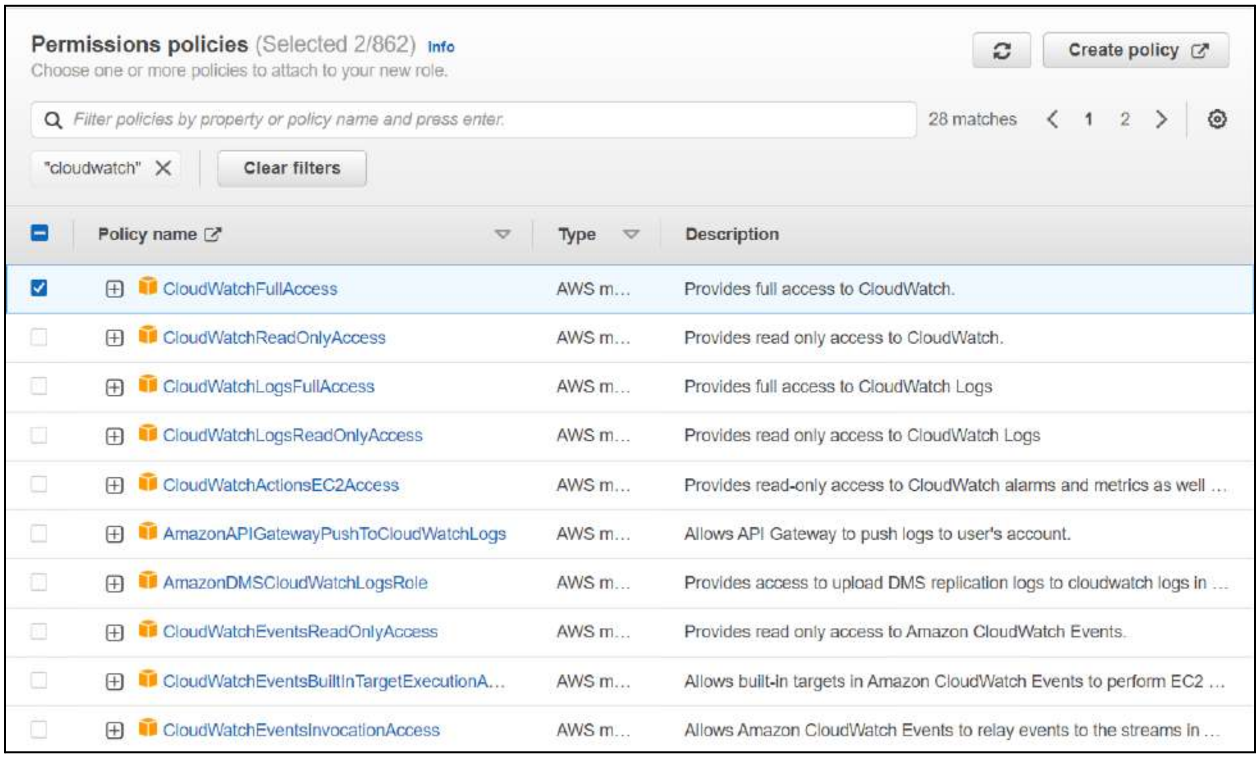
Permissions policies (Selected 1/862) Info

Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter. 4 matches < 1 >

"dynamodb" X Clear filters

	Policy name	Type	Description
✓	AmazonDynamoDBFullAccess	AWS m…	Provides full access to Amazon DynamoDB via the AWS Managemen…
	AmazonDynamoDBReadOnlyAccess	AWS m…	Provides read only access to Amazon DynamoDB via the AWS Mana…
	AWSLambdaInvocation-DynamoDB	AWS m…	Provides read access to DynamoDB Streams.
	AWSLambdaDynamoDBExecutionRole	AWS m…	Provides list and read access to DynamoDB streams and write permis…



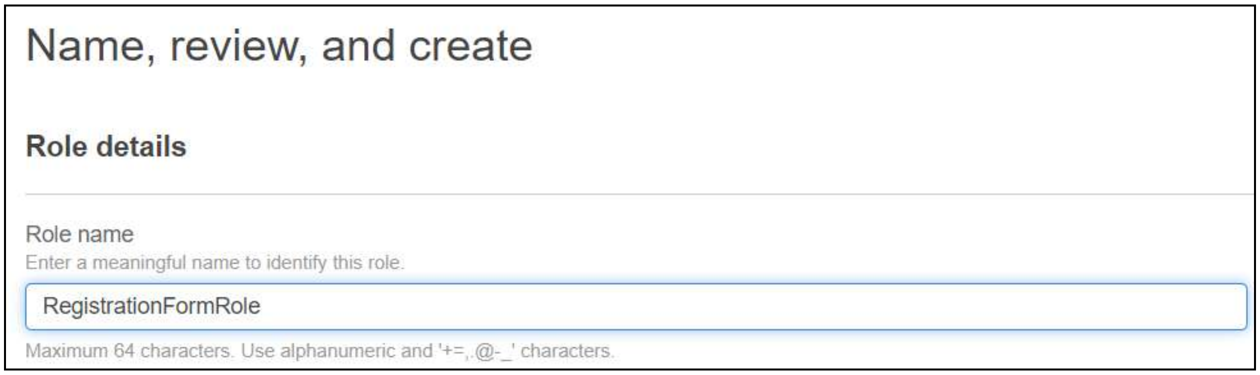
Permissions policies (Selected 2/862) Info

Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter. 28 matches < 1 2 >

"cloudwatch" X Clear filters

	Policy name	Type	Description
✓	CloudWatchFullAccess	AWS m…	Provides full access to CloudWatch.
	CloudWatchReadOnlyAccess	AWS m…	Provides read only access to CloudWatch.
	CloudWatchLogsFullAccess	AWS m…	Provides full access to CloudWatch Logs
	CloudWatchLogsReadOnlyAccess	AWS m…	Provides read only access to CloudWatch Logs
	CloudWatchActionsEC2Access	AWS m…	Provides read-only access to CloudWatch alarms and metrics as well …
	AmazonAPIGatewayPushToCloudWatchLogs	AWS m…	Allows API Gateway to push logs to user's account.
	AmazonDMSCloudWatchLogsRole	AWS m…	Provides access to upload DMS replication logs to cloudwatch logs in …
	CloudWatchEventsReadOnlyAccess	AWS m…	Provides read only access to Amazon CloudWatch Events.
	CloudWatchEventsBuiltInTargetExecutionA…	AWS m…	Allows built-in targets in Amazon CloudWatch Events to perform EC2 …
	CloudWatchEventsInvocationAccess	AWS m…	Allows Amazon CloudWatch Events to relay events to the streams in …


Name, review, and create

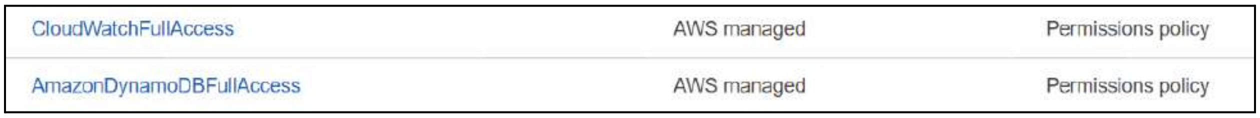
Role details

Role name

Enter a meaningful name to identify this role.

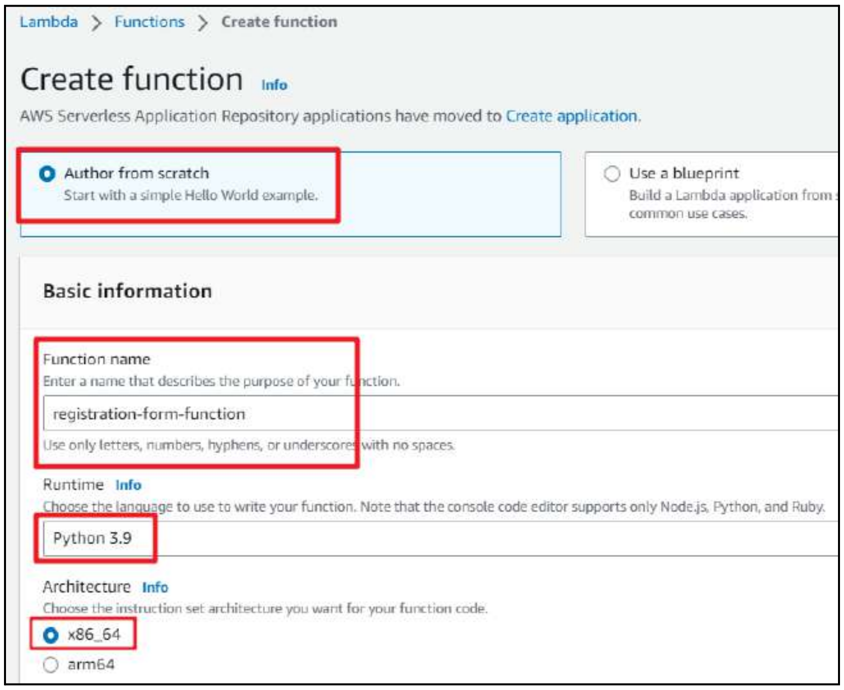
RegistrationFormRole

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.



CloudWatchFullAccess	AWS managed	Permissions policy
AmazonDynamoDBFullAccess	AWS managed	Permissions policy

Create Lambda Function


Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to Create application.

[✓ Author from scratch] Start with a simple Hello World example.

Use a blueprint Build a Lambda application from

common use cases.

Basic information

Function name

Enter a name that describes the purpose of your function.

registration-form-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.

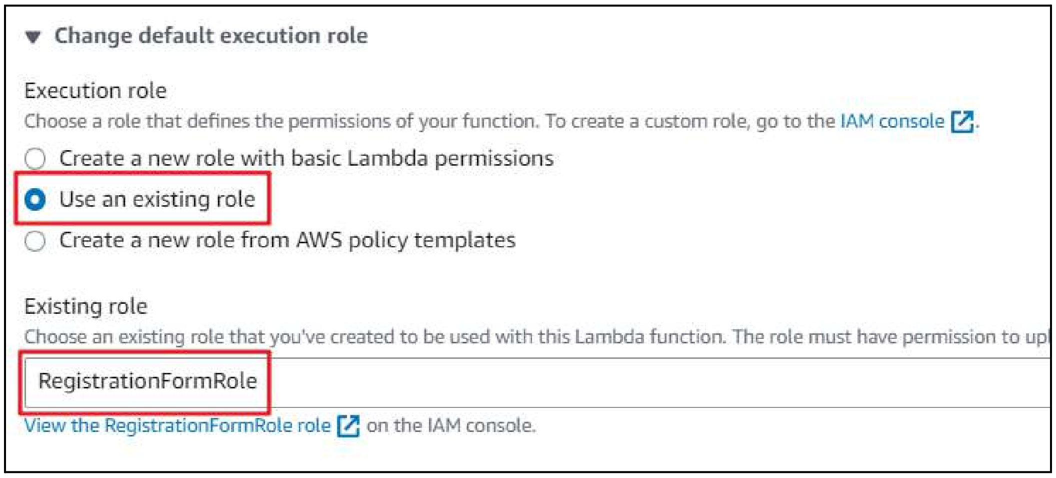
Python 3.9

Architecture Info

Choose the instruction set architecture you want for your function code.

[✓ x86_64]

arm64


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

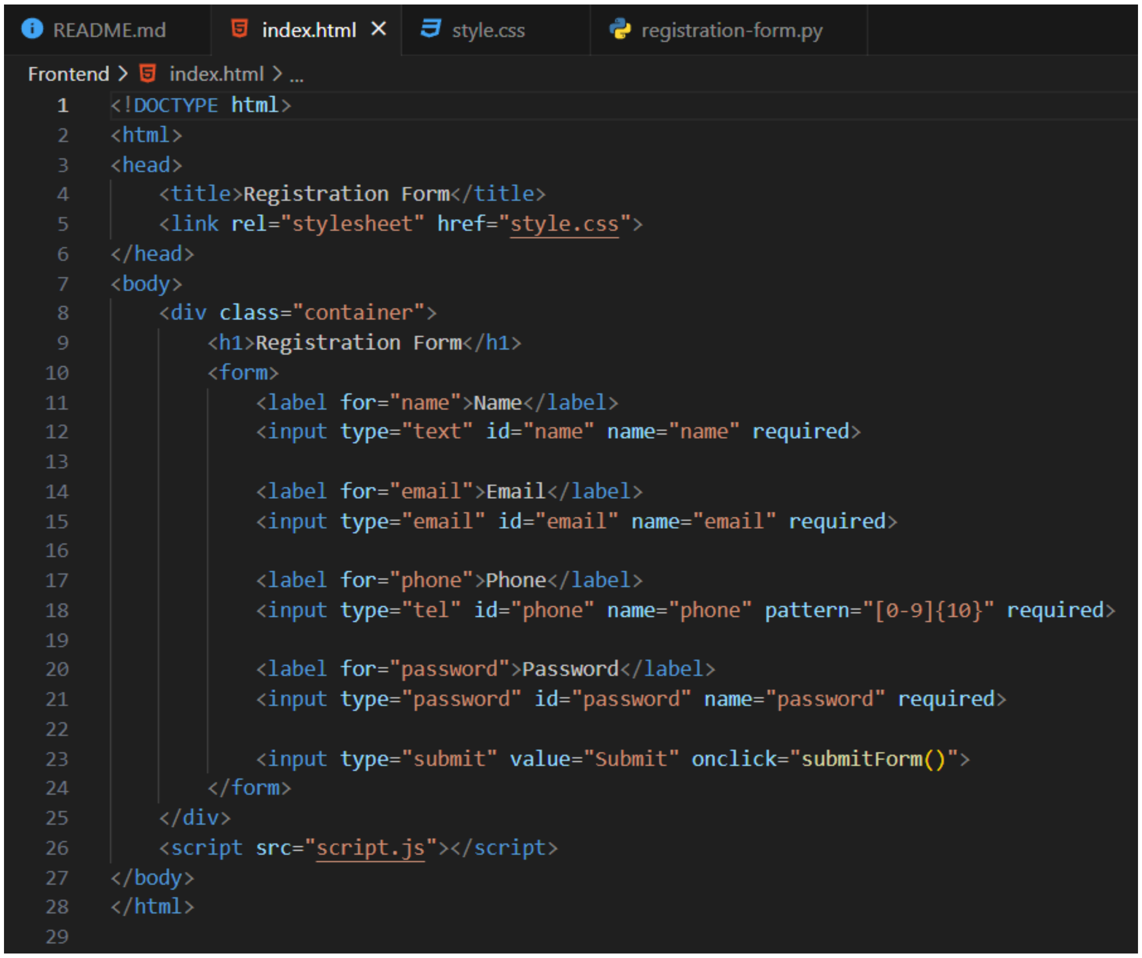
Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to up

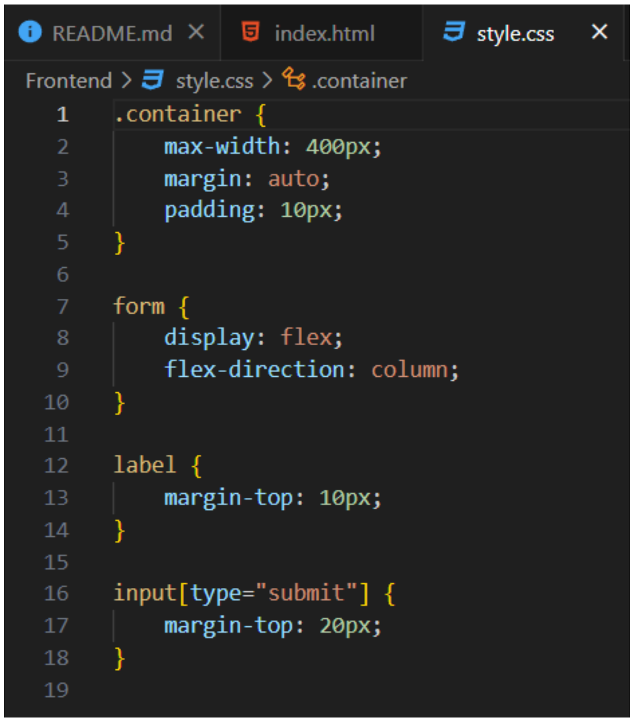
RegistrationFormRole

View the RegistrationFormRole role on the IAM console.

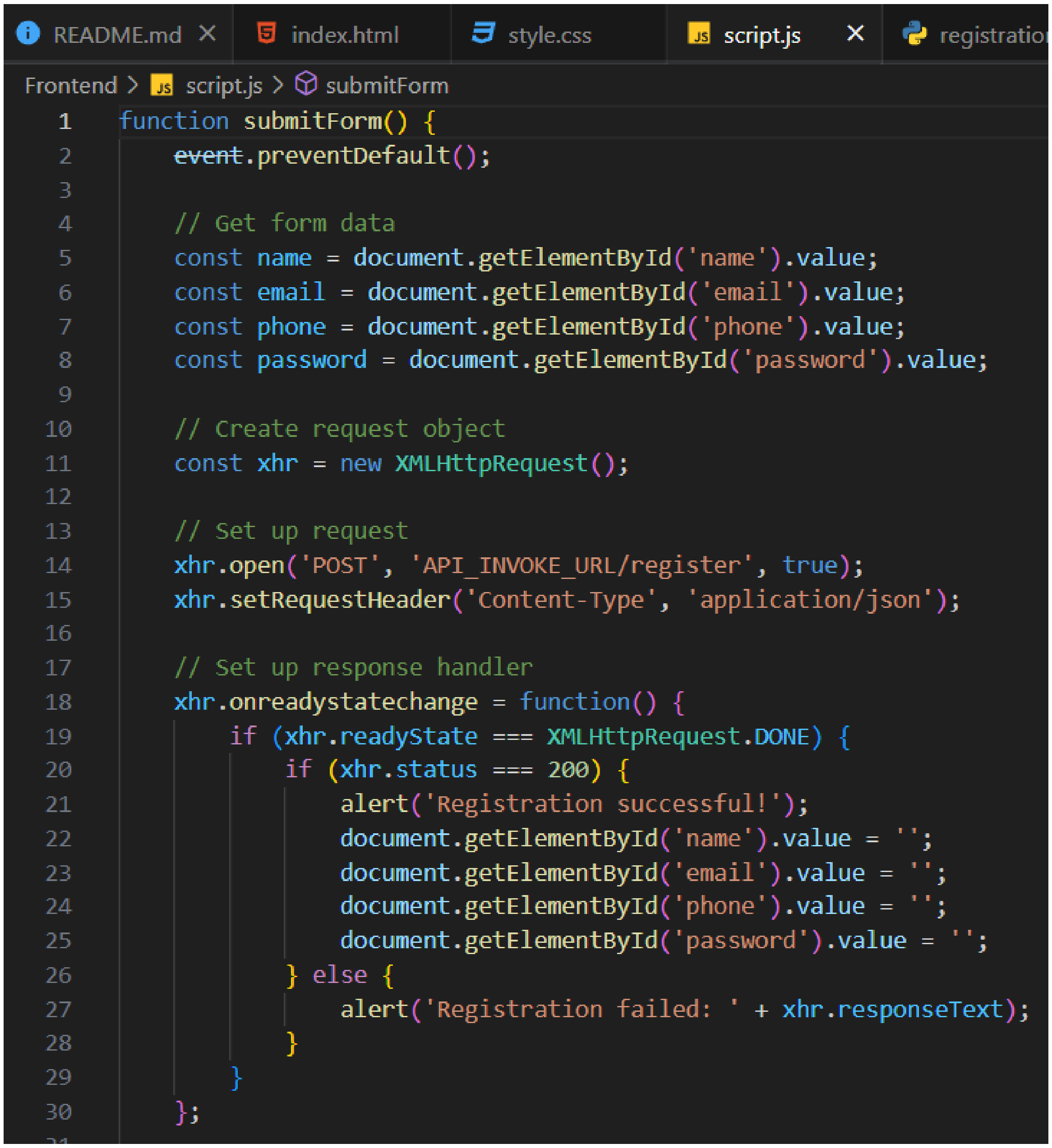
Write Lambda Function



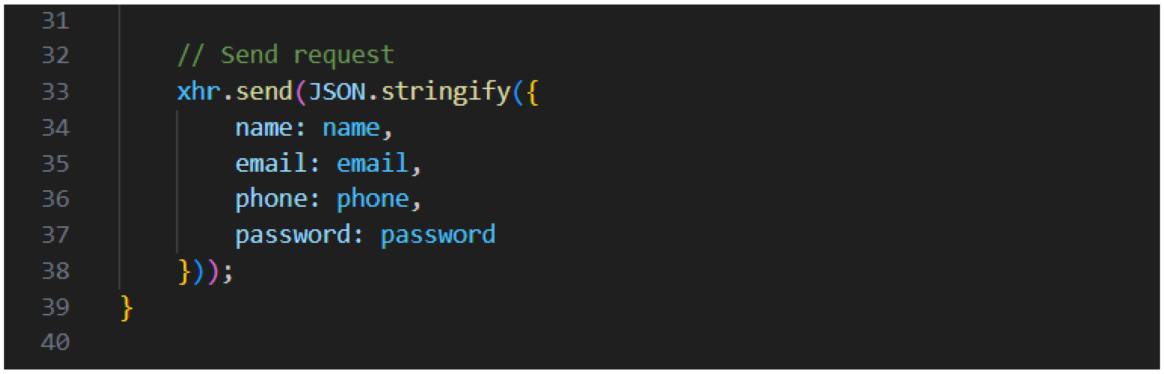
```html
<!DOCTYPE html>
<html>
<head>
    <title>Registration Form</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Registration Form</h1>
        <form>
            <label for="name">Name</label>
            <input type="text" id="name" name="name" required>

            <label for="email">Email</label>
            <input type="email" id="email" name="email" required>

            <label for="phone">Phone</label>
            <input type="tel" id="phone" name="phone" pattern="[0-9]{10}" required>

            <label for="password">Password</label>
            <input type="password" id="password" name="password" required>

            <input type="submit" value="Submit" onclick="submitForm()">
        </form>
    </div>
    <script src="script.js"></script>
</body>
</html>

```



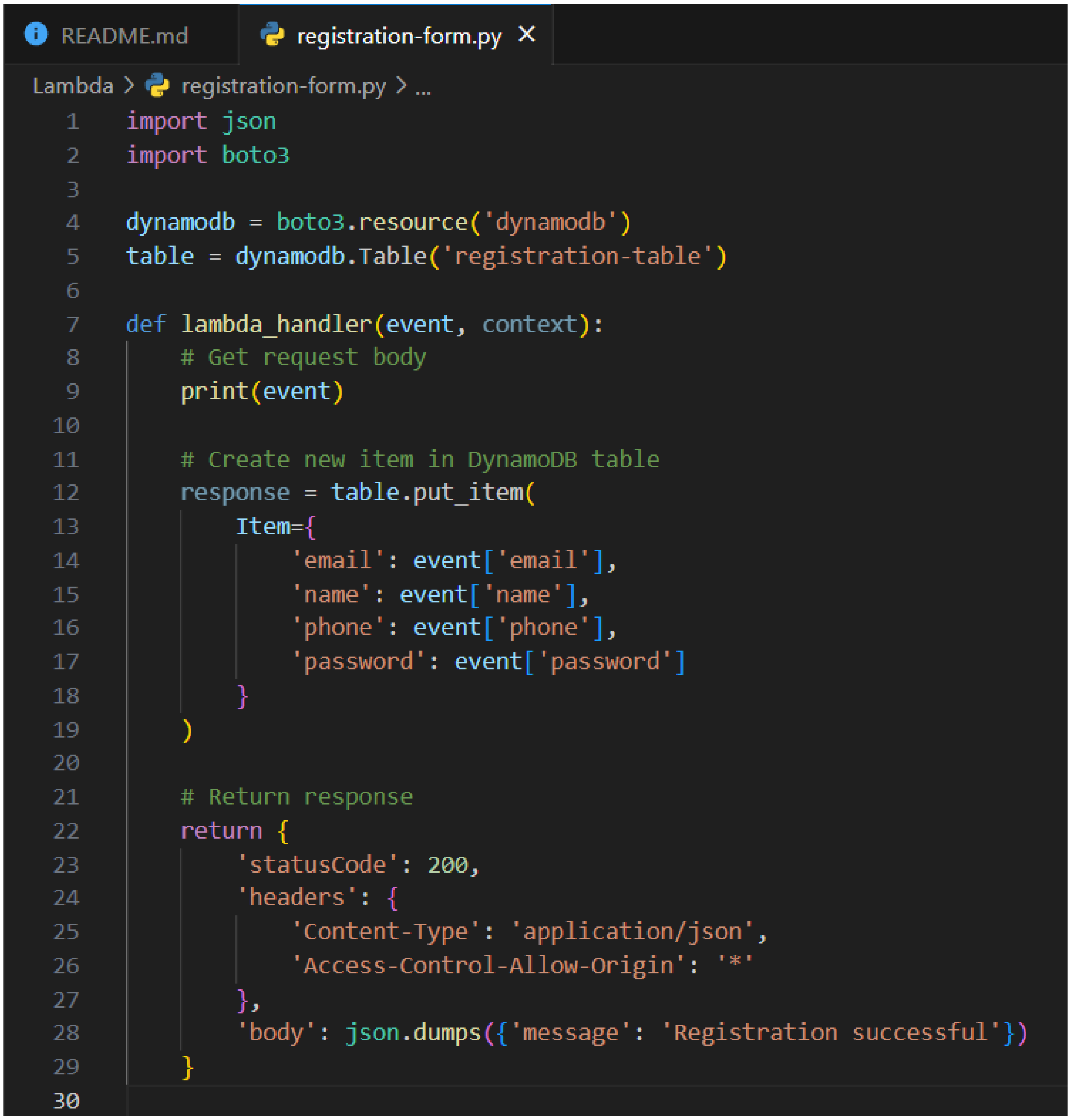
```css
.container {
    max-width: 400px;
    margin: auto;
    padding: 10px;
}

form {
    display: flex;
    flex-direction: column;
}

label {
    margin-top: 10px;
}

input[type="submit"] {
    margin-top: 20px;
}

```



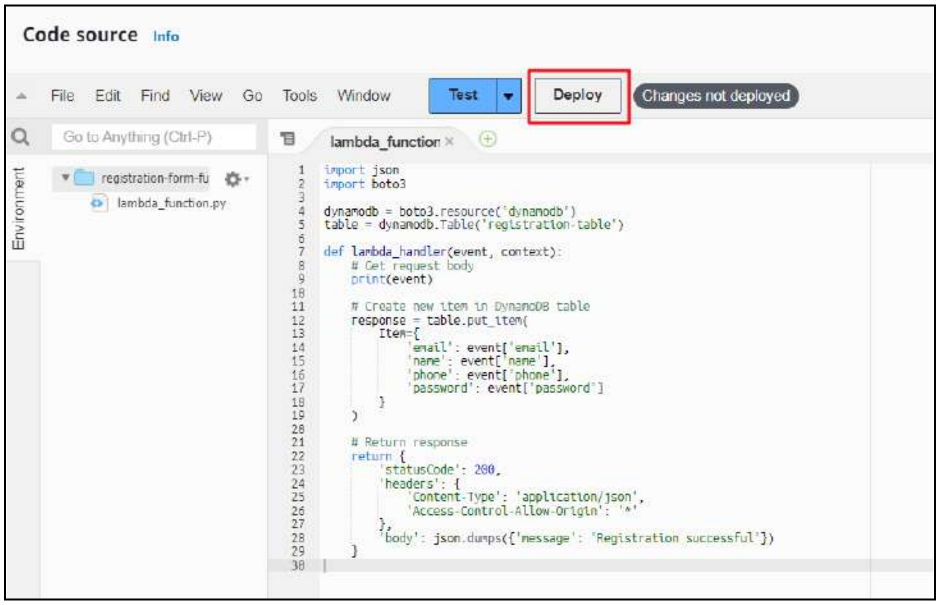
```javascript
function submitForm() {
    event.preventDefault();

    // Get form data
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;
    const phone = document.getElementById('phone').value;
    const password = document.getElementById('password').value;

    // Create request object
    const xhr = new XMLHttpRequest();

    // Set up request
    xhr.open('POST', 'API_INVOKE_URL/register', true);
    xhr.setRequestHeader('Content-Type', 'application/json');

    // Set up response handler
    xhr.onreadystatechange = function() {
        if (xhr.readyState === XMLHttpRequest.DONE) {
            if (xhr.status === 200) {
                alert('Registration successful!');
                document.getElementById('name').value = '';
                document.getElementById('email').value = '';
                document.getElementById('phone').value = '';
                document.getElementById('password').value = '';
            } else {
                alert('Registration failed: ' + xhr.responseText);
            }
        }
    };
```



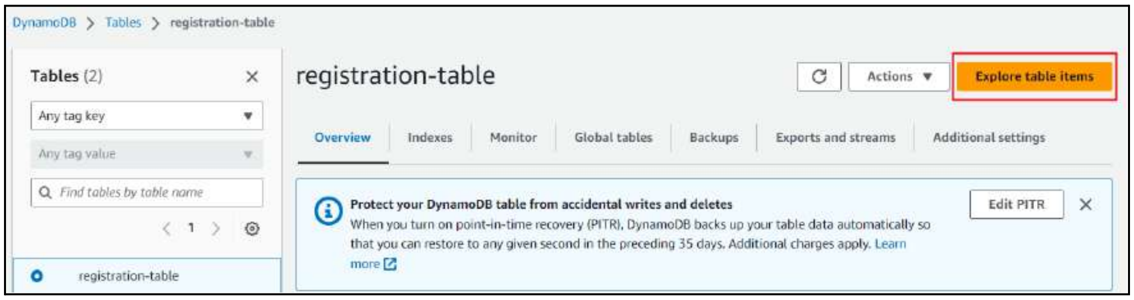
```javascript

    // Send request
    xhr.send(JSON.stringify({
        name: name,
        email: email,
        phone: phone,
        password: password
    }));
}

```



```python
import json
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('registration-table')

def lambda_handler(event, context):
    # Get request body
    print(event)

    # Create new item in DynamoDB table
    response = table.put_item(
        Item={
            'email': event['email'],
            'name': event['name'],
            'phone': event['phone'],
            'password': event['password']
        }
    )

    # Return response
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({'message': 'Registration successful'})
    }

```

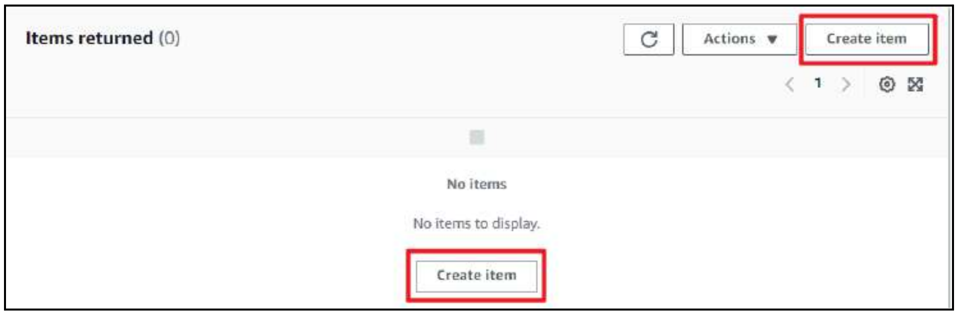


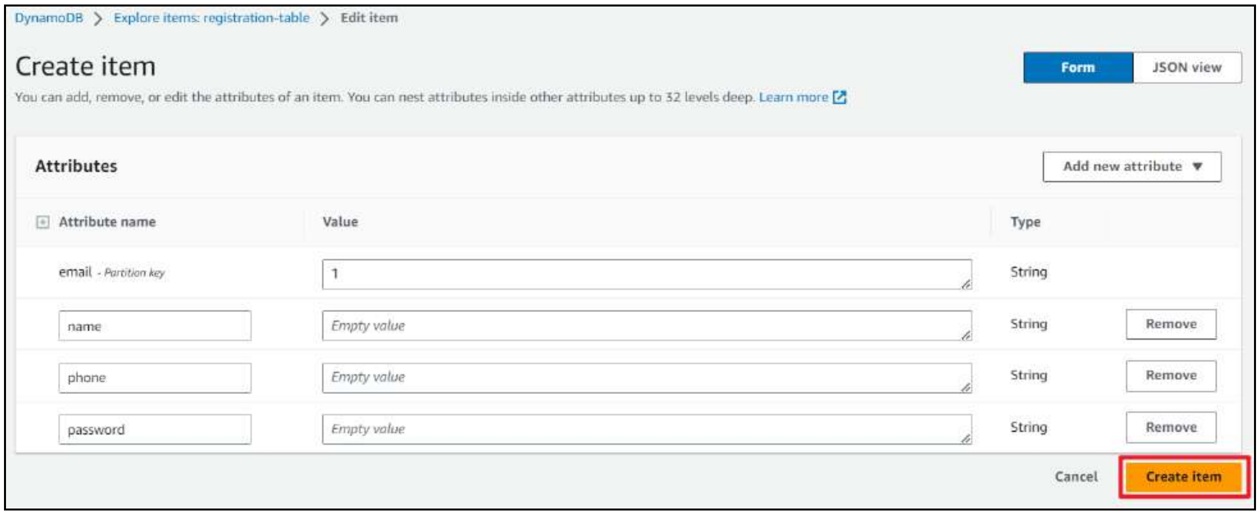
```python
import json
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('registration-table')

def lambda_handler(event, context):
    # Get request body
    print(event)

    # Create new item in DynamoDB table
    response = table.put_item(
        Item={
            'email': event['email'],
            'name': event['name'],
            'phone': event['phone'],
            'password': event['password']
        }
    )

    # Return response
    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({'message': 'Registration successful'})
    }
```



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





DynamoDB > Explore items: registration-table > Edit item

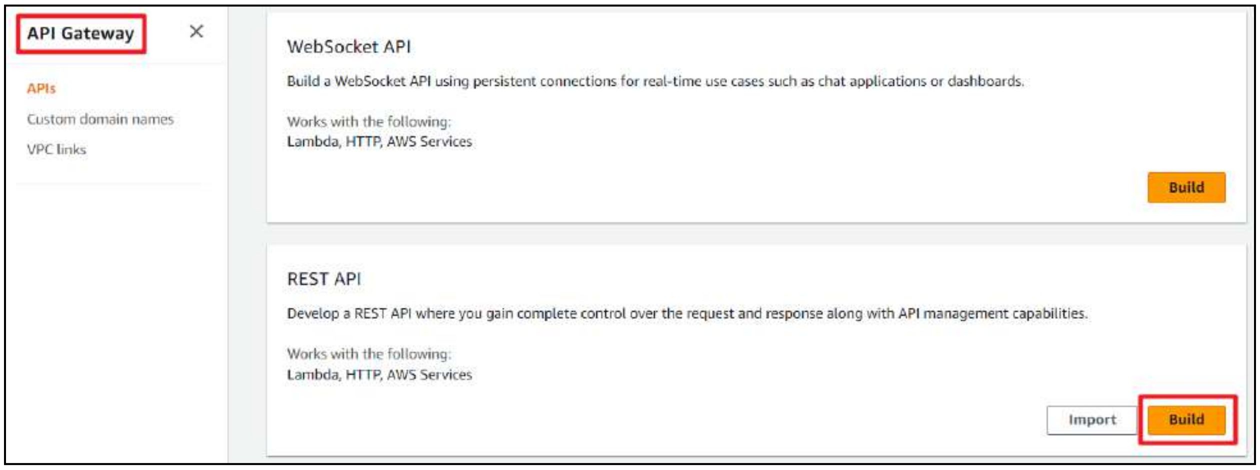
Create item

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

Attributes

Attribute name	Value	Type
email - *Partition key*	1	String
name	*Empty value*	String
phone	*Empty value*	String
password	*Empty value*	String

Create API Gateway



WebSocket API

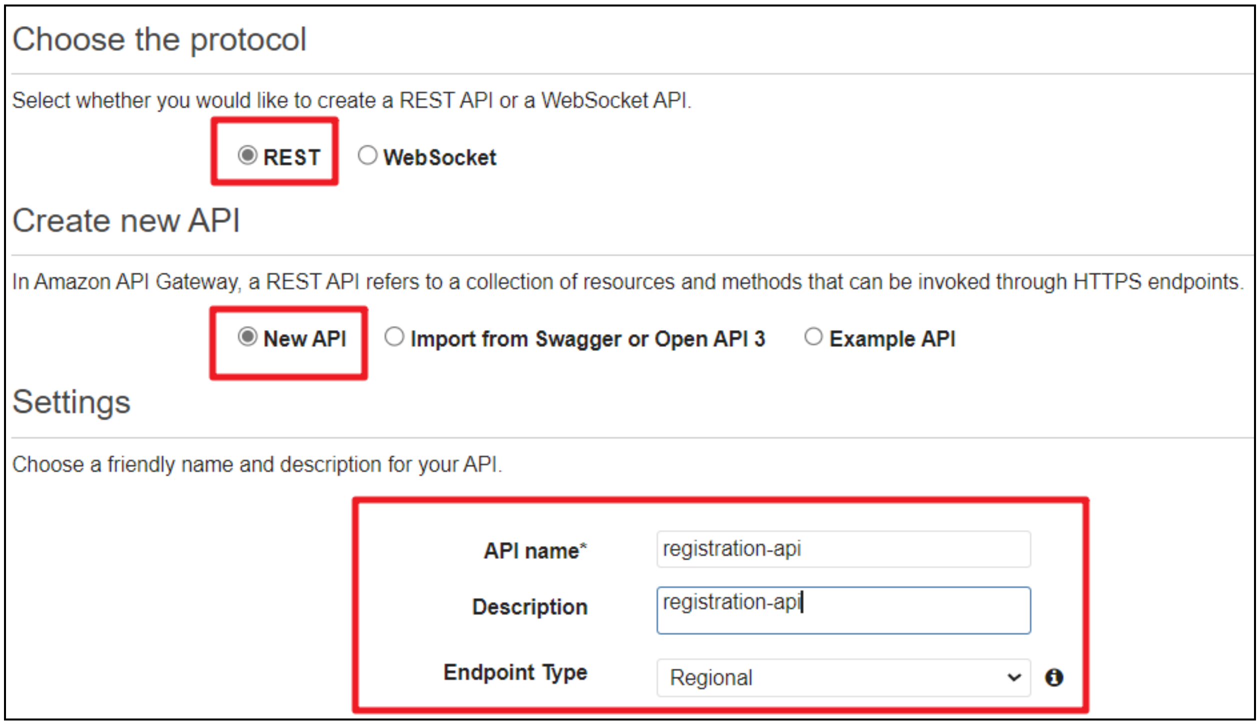
Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services

REST API

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

Works with the following:
Lambda, HTTP, AWS Services



Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

[✓ REST] WebSocket

Create new API

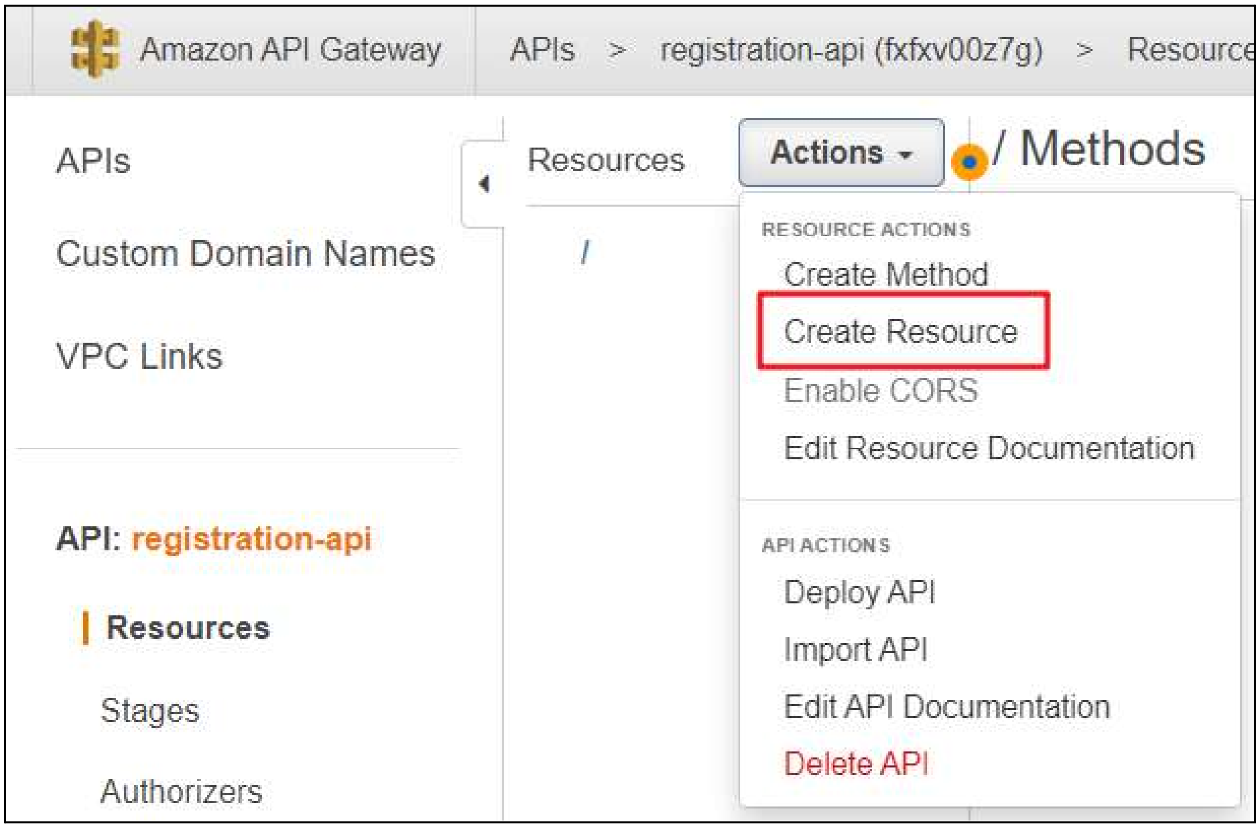
In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

[✓ New API] Import from Swagger or Open API 3 Example API

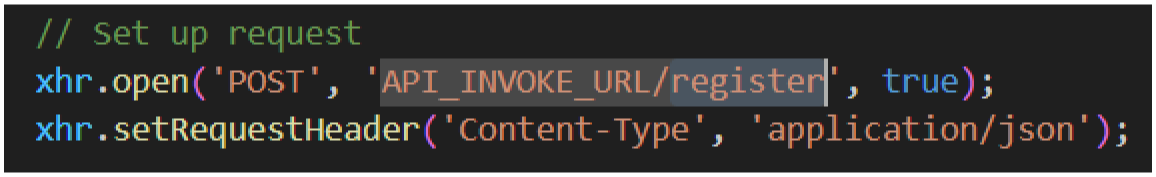
Settings

Choose a friendly name and description for your API.

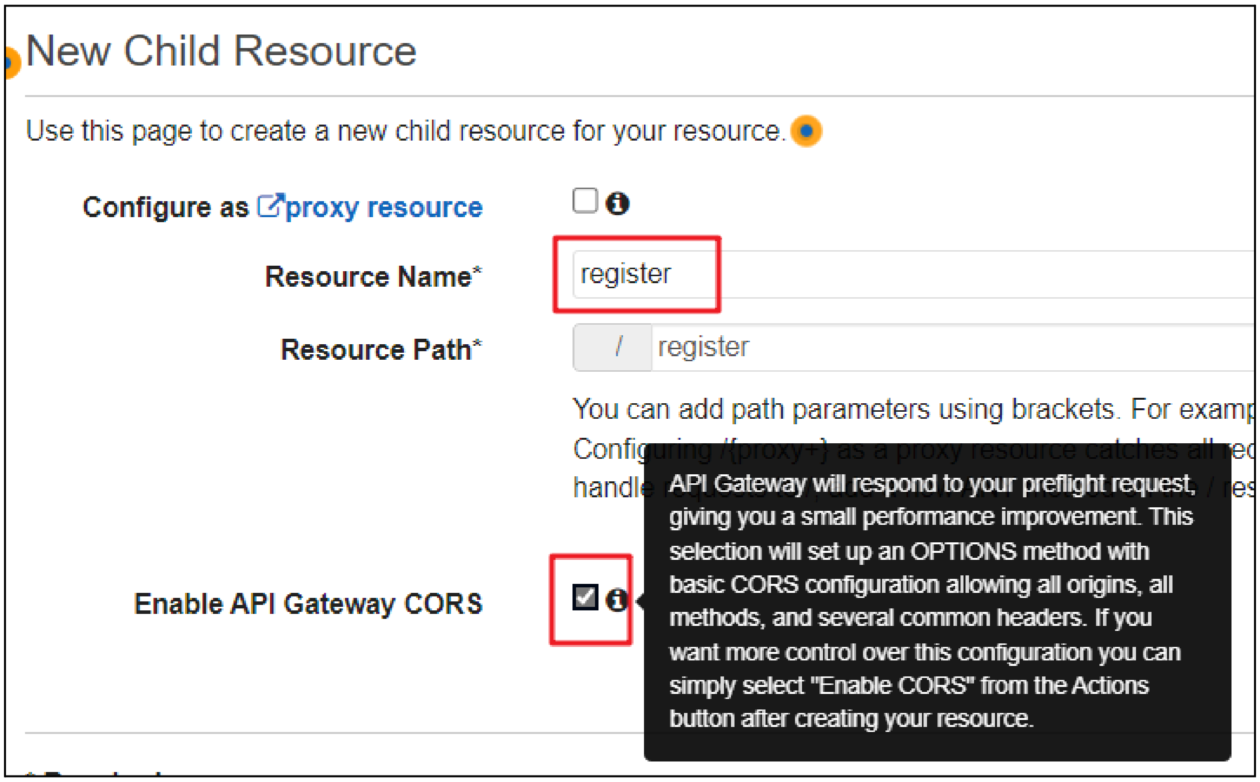
API name*	registration-api
Description	registration-api
Endpoint Type	Regional



Amazon API Gateway APIs > registration-api (fxfxv00z7g) > Resource



```
// Set up request
xhr.open('POST', 'API_INVOKE_URL/register', true);
xhr.setRequestHeader('Content-Type', 'application/json');
```



New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource ☐

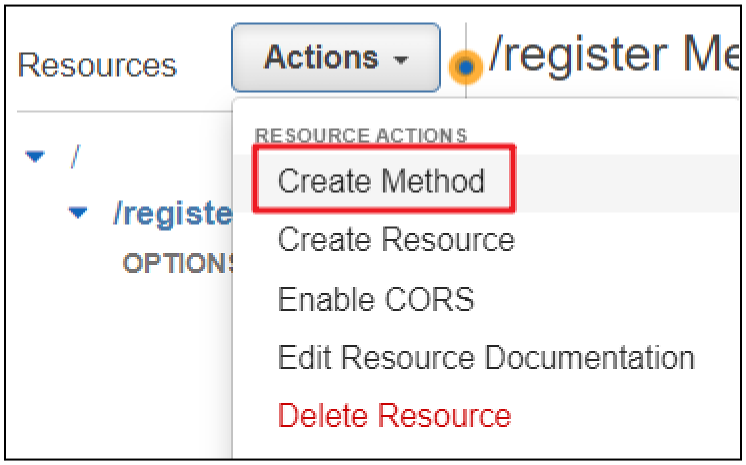
Resource Name* register

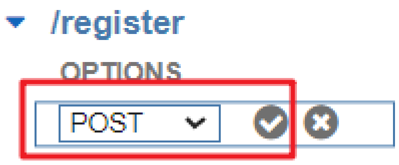
Resource Path* / register

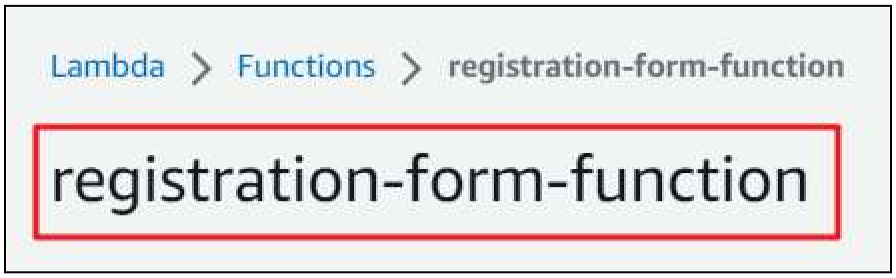
You can add path parameters using brackets. For examp
Config
handle

Enable API Gateway CORS ☑

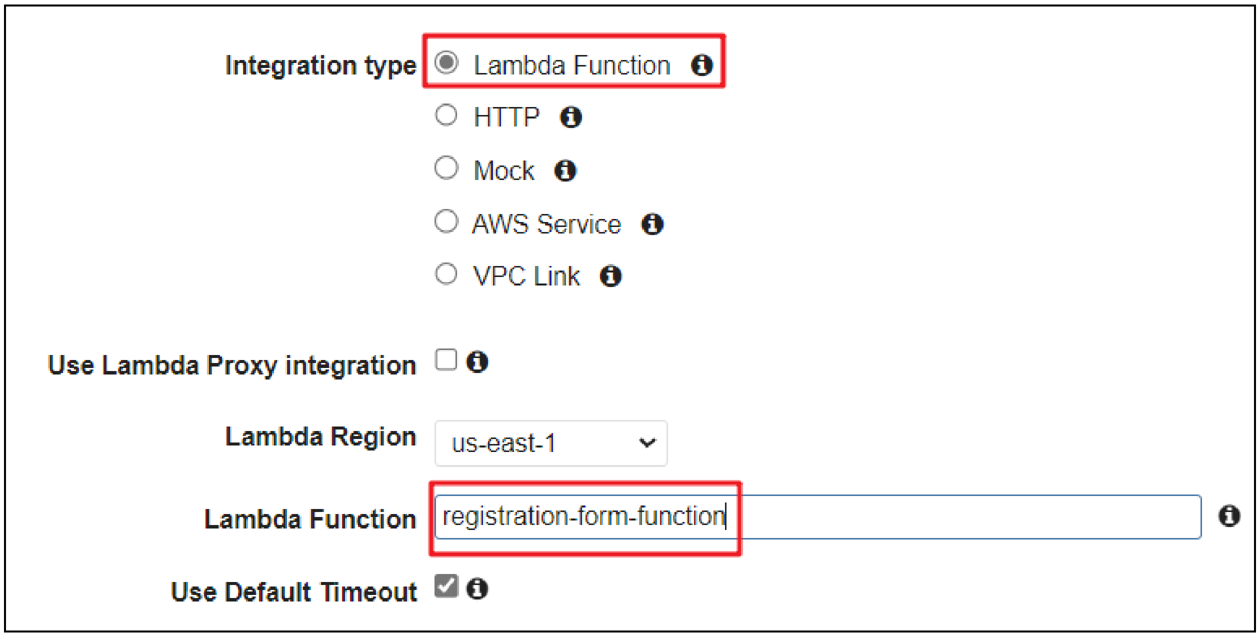
API Gateway will respond to your preflight request, giving you a small performance improvement. This selection will set up an OPTIONS method with basic CORS configuration allowing all origins, all methods, and several common headers. If you want more control over this configuration you can simply select "Enable CORS" from the Actions button after creating your resource.



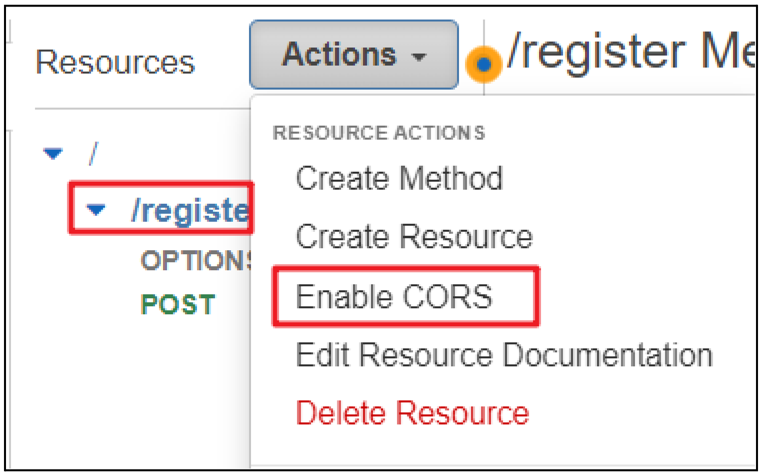


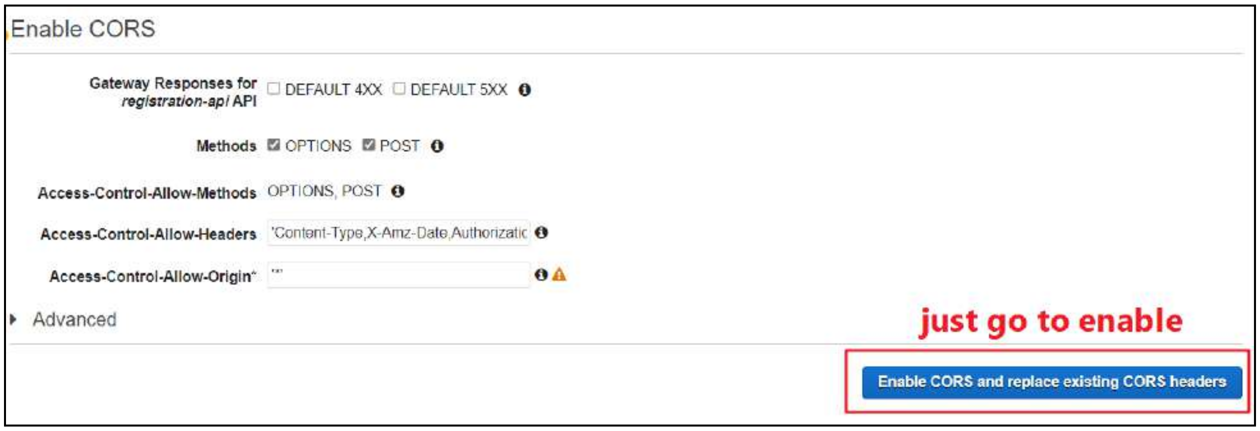


registration-form-function



Integration type	● Lambda Function
	○ HTTP
	○ Mock
	○ AWS Service
	○ VPC Link
Use Lambda Proxy integration	☐
Lambda Region	us-east-1
Lambda Function	registration-form-function
Use Default Timeout	☑



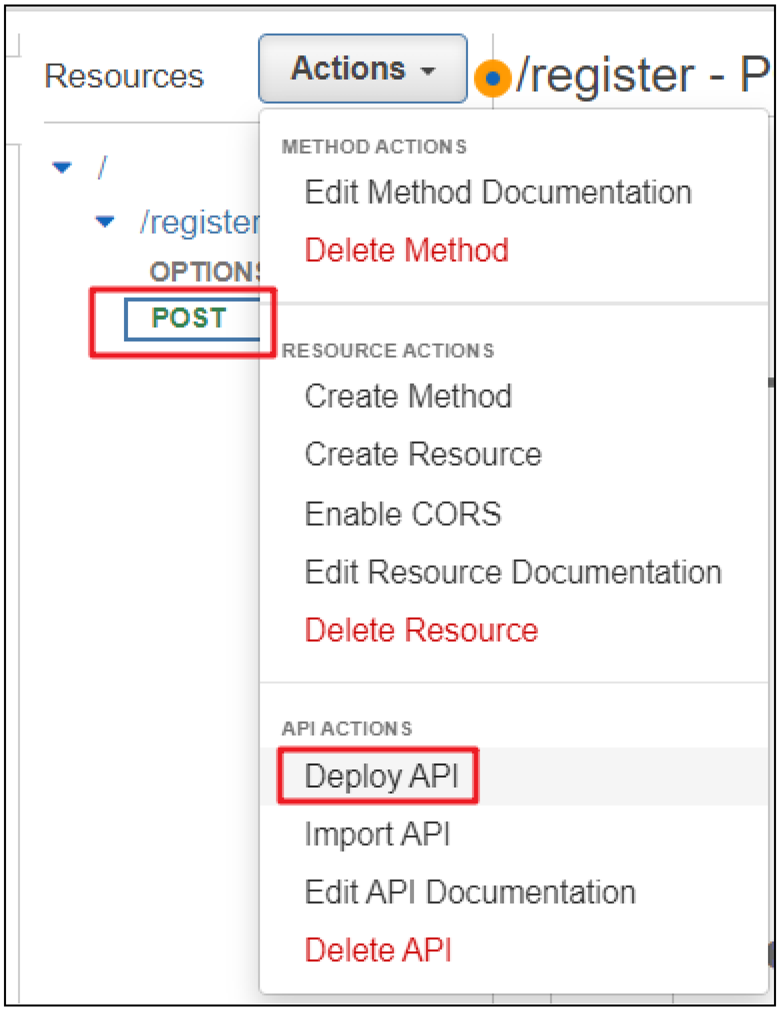


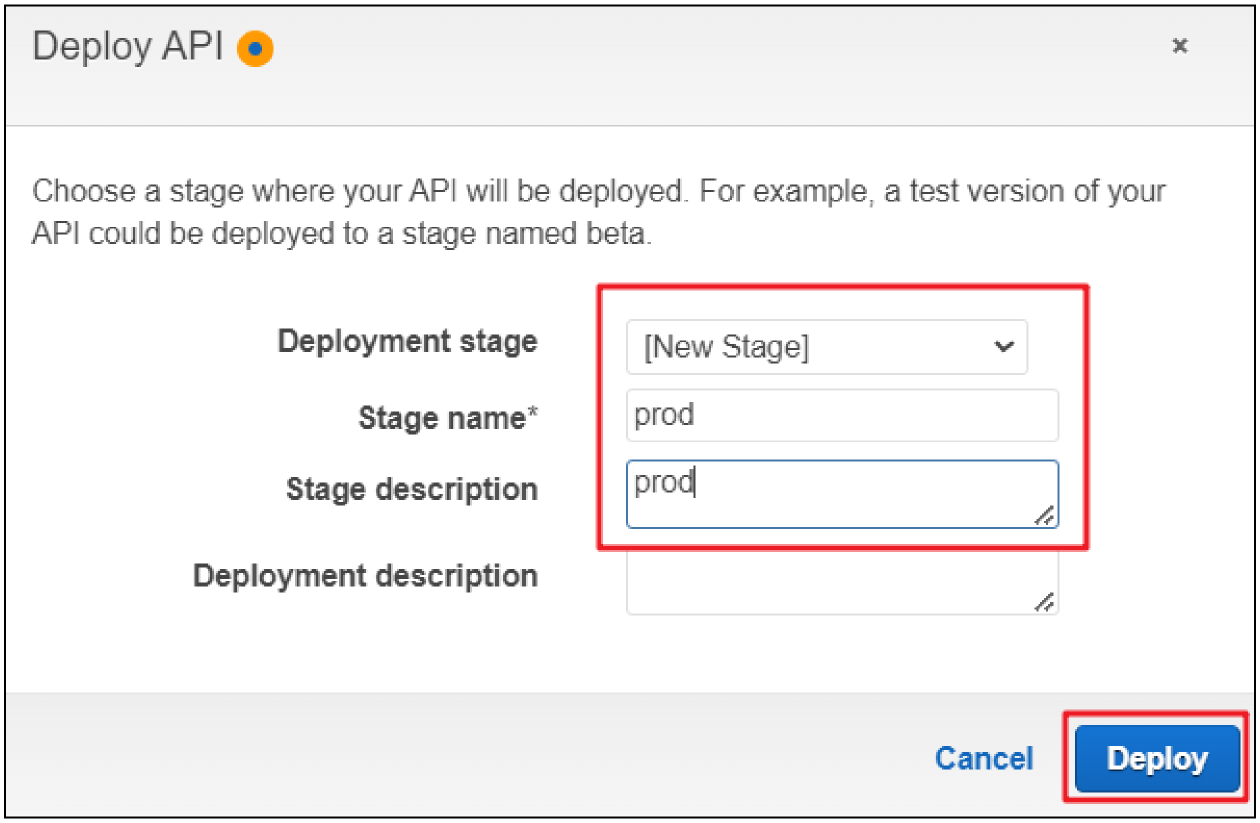
Enable CORS

Gateway Responses for *registration-api* API	☐ DEFAULT 4XX ☐ DEFAULT 5XX
Methods	☑ OPTIONS ☑ POST
Access-Control-Allow-Methods	OPTIONS, POST
Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorizatic
Access-Control-Allow-Origin*	'*'

Advanced

just go to enable


Resources
Actions
/register - P
/
/register
OPTIONS
POST
METHOD ACTIONS
Edit Method Documentation
Delete Method
RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation
Delete Resource
API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API


Deploy API

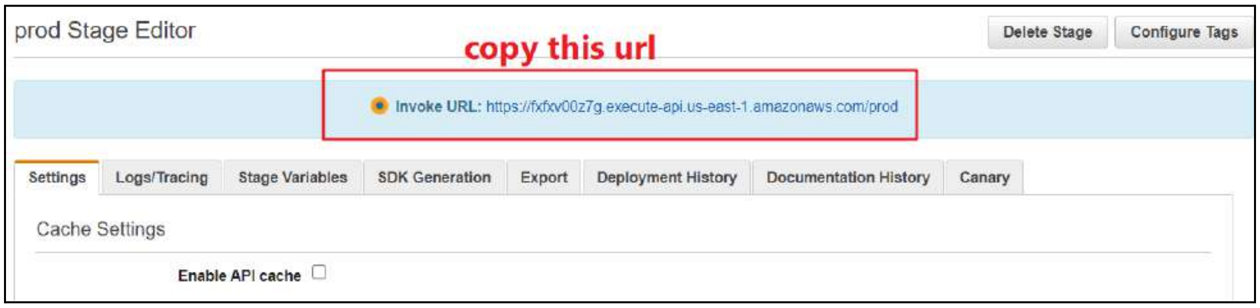
Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage: [New Stage]

Stage name*: prod

Stage description: prod

Deployment description:


prod Stage Editor

copy this url

Invoke URL: https://fxfxv00z7g.execute-api.us-east-1.amazonaws.com/prod

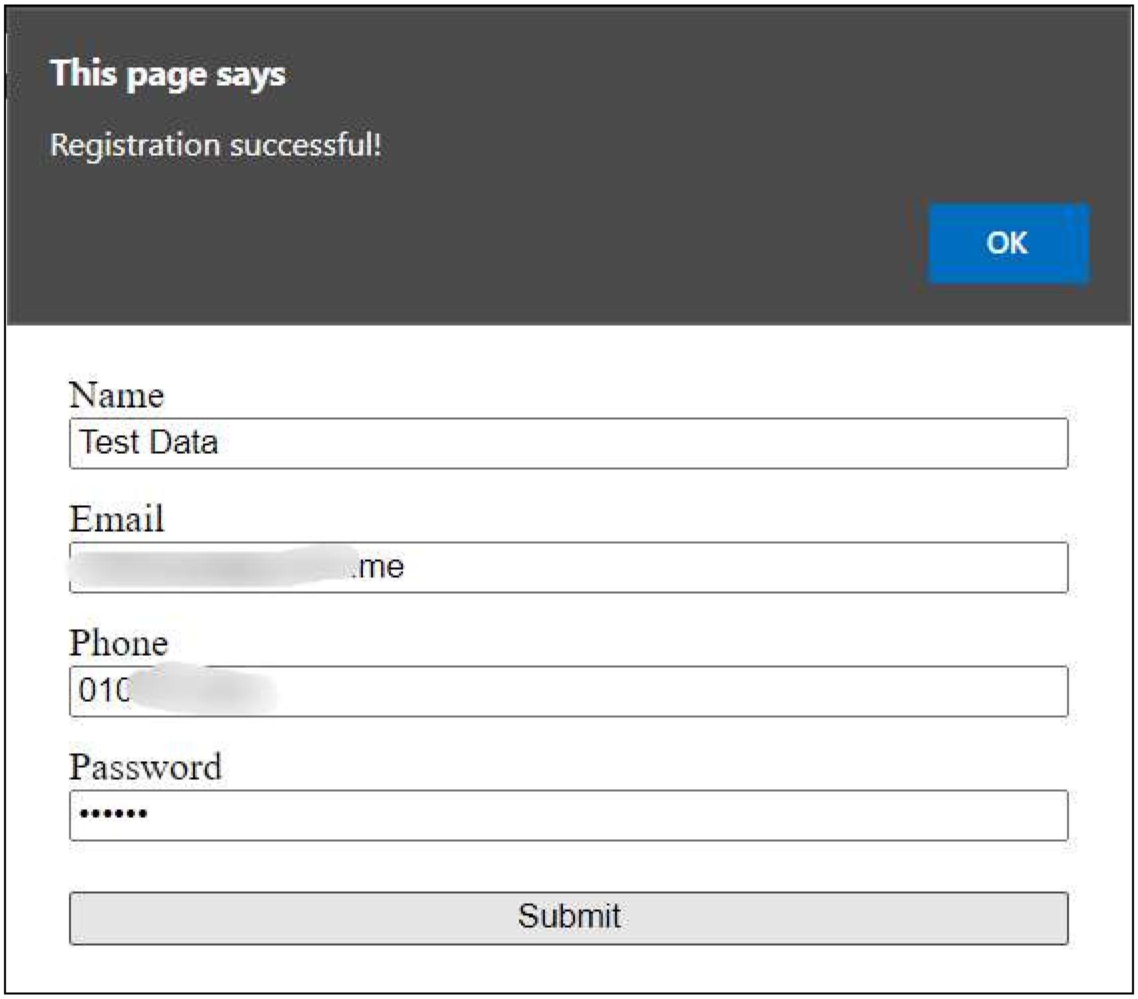
Replace API_INVOKE_URL

```
// Set up request
xhr.open('POST', 'API_INVOKE_URL/register', true);
xhr.setRequestHeader('Content-Type', 'application/json');
```

To the copied URL

```
// Set up request
xhr.open('POST', 'https://fxfxv00z7g.execute-api.us-east-1.amazonaws.com/prod/register', true);
xhr.setRequestHeader('Content-Type', 'application/json');
```

Go to index.html and try to register



This page says

Registration successful!

OK

Name

Test Data

Email

[redacted].me

Phone

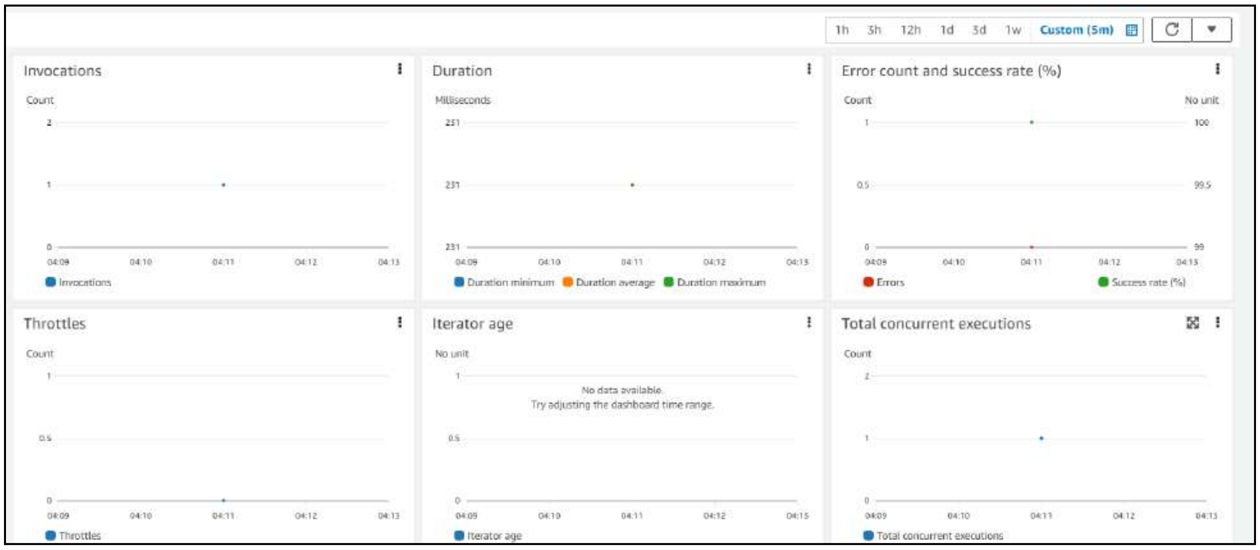
010[redacted]

Password

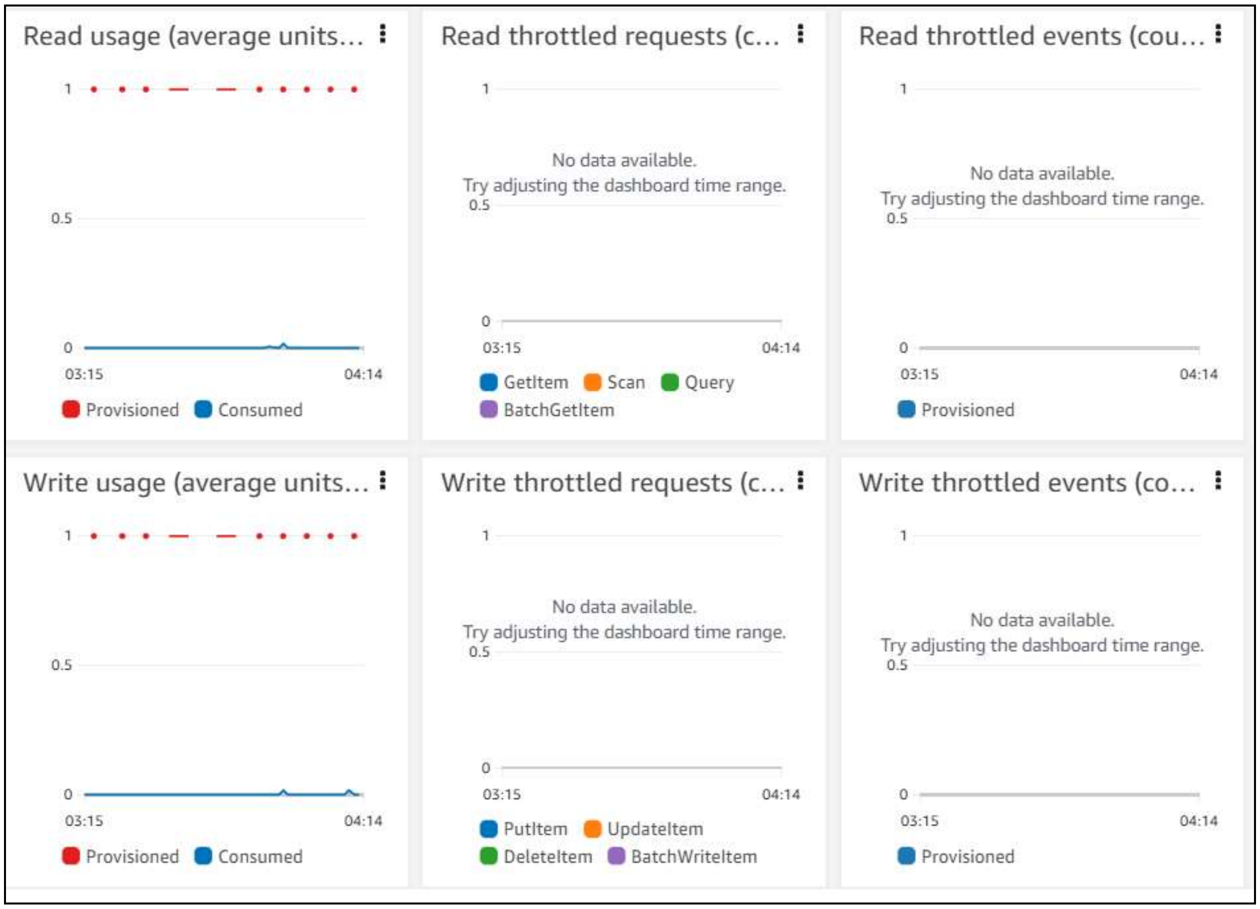
••••••

Submit

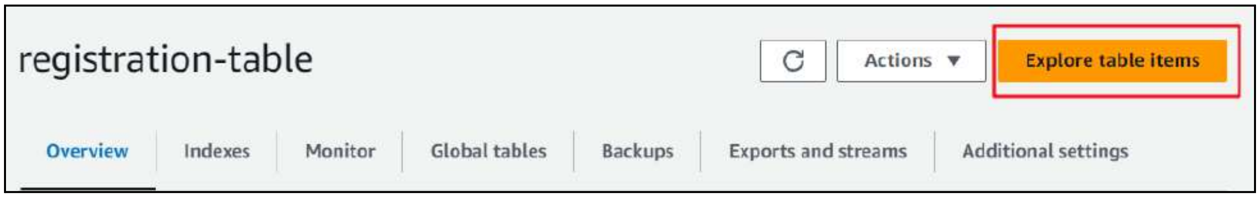
Check Lambda CloudWatch metrics

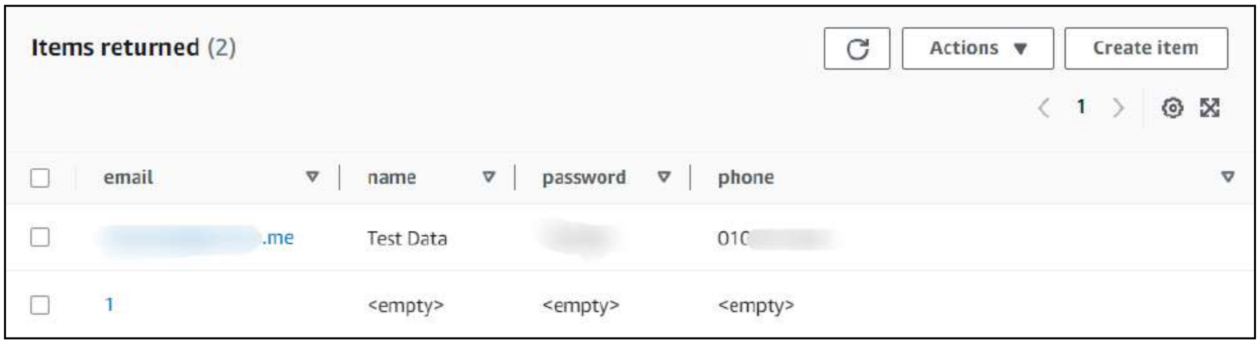


Check DynamoDB capacity metrics



Go to check table items that has registered





Items returned (2)

	email	name	password	phone
☐	[redacted].me	Test Data	[redacted]	010[redacted]
☐	1	<empty>	<empty>	<empty>