SERVERLESS REGISTRATION FORM

Project Description –

The "Serverless Registration Form using AWS Services" project aims to create a dynamic and scalable registration form application leveraging the power of various AWS services. This application will allow users to register for an event, service, or platform seamlessly, while all backend processing, storage, and handling are handled through serverless architecture.

Key Features:

- User-Friendly Registration Form: Design and develop an intuitive registration form with fields for capturing user information such as name, email, contact details, and additional event-specific inputs.

- Serverless Architecture: Utilize AWS Lambda, Amazon API Gateway, and AWS Step Functions to create a serverless architecture that automatically scales based on demand, ensuring reliable performance without managing underlying infrastructure.

- Data Validation and Storage: Implement client-side and server-side data validation techniques to ensure accurate and consistent information submission. Store user registration data in Amazon DynamoDB for easy retrieval and analysis.

- File Uploads (Optional): If required, enable users to upload files (e.g., profile pictures, supporting documents) as part of their registration. Use Amazon S3 to securely store these files.

- Authentication and Security: Implement user authentication using Amazon Cognito to ensure that only authorized users can access and submit the registration form. Incorporate HTTPS using AWS Certificate Manager to secure data transmission.

- Event-Driven Processing: Utilize AWS Lambda to trigger backend processes like sending confirmation emails, data validation, and notifications in response to user registrations.

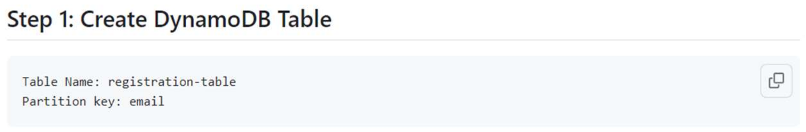
- Serverless Orchestration: Use AWS Step Functions to orchestrate the flow of data and processes, ensuring seamless execution of various components within the application.

- Scalability and Cost Efficiency: Leverage the elasticity of AWS services to automatically scale resources based on user traffic. Since serverless services charge based on actual usage, this architecture helps optimize costs.

Technologies used –

1. **DynamoDB Table Creation:**

    Created a DynamoDB table to store user registration data. Defined the necessary attributes like username, email, and any other relevant fields.

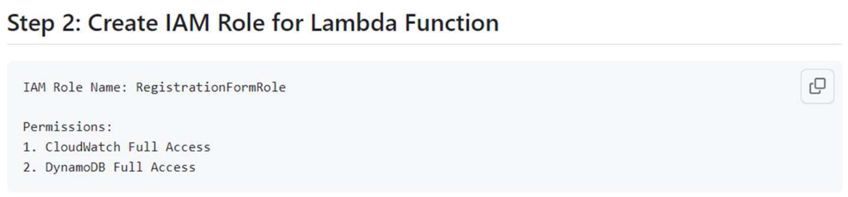


Step 1: Create DynamoDB Table

```
Table Name: registration-table
Partition key: email
```

2. **IAM Role for Lambda Function:**

    Defined an IAM role that grants the Lambda function permissions to access the DynamoDB table. This role should follows the principle of least privilege.



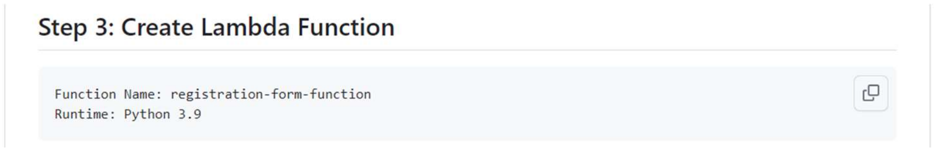
Step 2: Create IAM Role for Lambda Function

```
IAM Role Name: RegistrationFormRole

Permissions:
1. CloudWatch Full Access
2. DynamoDB Full Access
```

3. **Lambda Function Creation:**

    Created a Lambda function that will handle the backend logic of the registration process. This function will validate user input, interact with the DynamoDB table to store registration data, and potentially trigger other actions, such as sending confirmation emails.

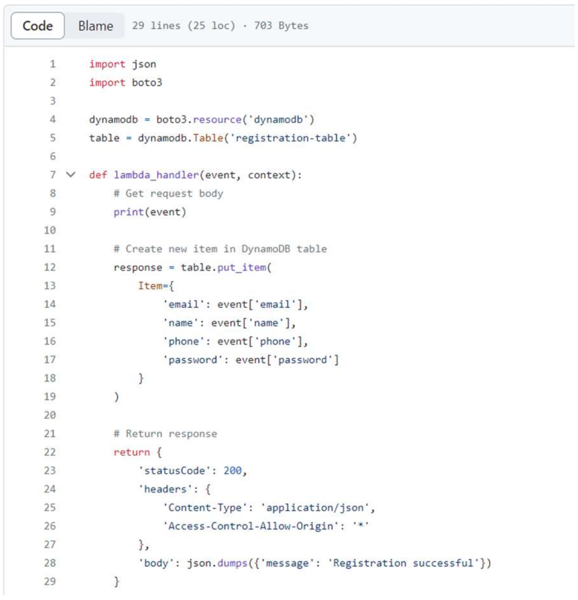


Step 3: Create Lambda Function

```
Function Name: registration-form-function
Runtime: Python 3.9
```

4. **Lambda Function Logic:**

    Written the Lambda function code in a programming language supported by AWS Lambda (e.g., Node.js, Python, Java). Implemented the registration logic, including data validation, database interaction, and any additional functionality required for the project.


Code Blame 29 lines (25 loc) · 703 Bytes

```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'})
    }
```

5. **API Gateway Setup and CORS Configuration:**

Created an API Gateway to expose a RESTful API for the registration form. Enabled CORS (Cross-Origin Resource Sharing) to allow the frontend to interact with the API seamlessly.

6. **Test Application with API Gateway:**

Utilized the API Gateway's test functionality to ensure that the API endpoints are working as expected. Tested the registration form's integration with the Lambda function and DynamoDB table.

7. **Integrate with AWS Cognito:**

Configured AWS Cognito to handle user authentication and authorization. This ensures secure access to the registration form and allows you to manage user identities.

8. **Test the Project:**
Thoroughly test the entire registration process. Verify that user data is correctly stored in the DynamoDB table, and users can register and authenticate using AWS Cognito.

<u>Web Development Tools</u> -

Frontend:

Developed the frontend of the registration form using your preferred web development framework.

<u>Conclusion</u> –

The Serverless Registration Form project harnesses AWS services to craft a dynamic, scalable, and secure registration solution. By eliminating the need for traditional server management, it streamlines application development and ensures seamless scalability to accommodate varying user loads. Leveraging AWS Lambda, API Gateway, DynamoDB, and other components, the project achieves efficient data processing, storage, and user authentication. With automated email notifications, user data validation, and optional file uploads, it caters to diverse registration needs. The project underscores the power of serverless architecture in delivering responsive, cost-effective, and modern applications. Through its successful implementation, I have gained valuable insights into AWS services integration, serverless paradigms, and agile development practices, bolstering my proficiency as a developer in the ever-evolving landscape of cloud technology.

Future Scope –

1. Offer real-time chat support during the registration process to assist users with any queries they might have. This can be implemented using AWS Chatbot integrated with Amazon Connect or WebSocket API for real-time communication.

2. Generate QR codes for registered attendees, which can be scanned at the event for quick check-in. This can be done using AWS Lambda to generate QR codes and Amazon S3 for storage.

3. Allow users to choose their seats or preferred areas during event registration.

4. Add an extra layer of security with multi-factor authentication for user registration.

5. Provide an option to add event details directly to users' calendar apps after registration.Integration with calendar APIs (e.g., Google Calendar API, Microsoft Graph API) via AWS Lambda.