DALHOUSIE UNIVERSITY

CSCI 5409
Advance topics in Cloud Computing

# Term Assignment:
# Project: Volunteer registration for an NGO using Event driven architecture

Ritva Katrodiya
B00930131

Git Lab Repository URL: [Term Assignment · main · Courses / 2023-Summer / csci4145-5409 / rkatrodiya · GitLab (dal.ca)]

## Table of Contents

## Introduction

In this project, I have developed NGO's website using event driven architecture, specifically designed to efficiently handle volunteer registrations and notifications. I have used Amazon Web Services EC2 to host my web app where volunteers can easily register for various events.

When a person registers on the website, AWS Lambda is triggered through API Gateway, ensuring their registration details are securely stored in Amazon web services DynamoDB, the database service. Simultaneously, the NGO owner receives real-time notifications by email using SNS, keeping them promptly informed about the new registrations.

To manage potential spikes in registration traffic, I have used an SQS queue. This queue acts as a buffer, efficiently handling incoming registration messages and ensuring that the system remains reliable and resilient. Additionally, a scheduled Lambda function periodically retrieves and processes messages from the queue, for scheduling lambda function, i have configured event bridge service with lambda function.

The architecture's fault-tolerant and scalable nature, powered by AWS's features, guarantees smooth volunteer management and immediate notifications for the NGO. In nutshell, this project showcases an event-driven approach for NGO websites, utilizing AWS cloud services to streamline registrations and notifications efficiently. The architecture's components, including EC2, Lambda, DynamoDB, SNS, event bridge and SQS, work harmoniously to provide a reliable and scalable system, capable of handling varying levels of traffic with ease and ensuring the NGO stays well-informed about its volunteers and events.

## Services used to implement this project:

1) **Compute:** AWS EC2, AWS lambda
2) **Storage:** AWS DynamoDB
3) **Network:** Amazon API Gateway
4) **General:** AWS SNS, AWS SQS, AWS Event Bridge
5) **Cloud Formation**

### Reason for selecting above listed AWS services:

1) **AWS EC2:** I have selected Amazon web service Elastic cloud compute service to host my web application because EC2 instance provides flexibility to choose the operating system and software. Additionally, it provides security as its security groups and key pair authentication enhance security. Apart from that, EC2 provides a cost-effective and manageable solution for hosting web applications on the cloud.

2) **AWS API Gateway:** I have selected Amazon API Gateway as it will provides Secure and route API requests to lambdas from web application. By API gateway, I can create, maintain, and deploy Plan at nay scale, providing a seamless experience for users to interact with my application. Additionally, API Gateway's seamless integration with other AWS services allows to build serverless applications, taking advantage of the full capabilities of the cloud platform.

3) **AWS SNS:** I have chosen Amazon Simple notification service is designed to send the real time notifications to the number of subscribers using email, text message and other channels. By decoupling the notification logic from primary application, SNS provides seamless integration of new notification types as per needs. With its high availability and durability, SNS guarantees the reliable delivery of notifications, ensuring that messages from applications reach their intended recipients without fail.

4) **AWS SQS:** I have selected Amazon SQS service which offers a fully managed messages queuing service that decouples and scales microservices, distributes systems and serverless applications. it also helps to develop asynchronous message processing, make sure that messages are not lost and handle the number of requests during high traffic period. In addition, SQS provides a flexible and scalable approach to handling message processing, making it suitable for various workloads.

5) **AWS Event Bridge**: I have chosen EventBridge, it is a serverless event bus service that makes it easy to connect different applications using events. It allows to build event-driven architectures and connect various AWS services with third-party applications seamlessly. EventBridge simplifies the management of event routing and processing.so ,in my architecture, it will invoke lambda function after every minute.

6) **AWS DynamoDB:** I have chosen DynamoDB to store all the registration details as DynamoDB is a fully managed NoSQL database service provided by AWS. It offers fast and predictable performance with seamless scalability. Using DynamoDB to store registration details allows to have a scalable and flexible database for application.

7) **AWS lambda function:** I have selected lambda functions to write code as Lambda allows to run code without provisioning or managing servers. It supports event-driven architectures, as lambda can be configured with many services such as SQS, DynamoDB, S3, SNS, etc. Lambda scales automatically and performs efficiently, charging only for the compute time used.

8) **AWS Cloud formation**: I have used AWS CloudFormation  because of its infrastructure-as-code approach, which allowed me to define and manage all the necessary AWS resources such as SQs, SNS topics, lambda functions , DynamoDB table, API gateway , EC2 and  EventBridge in a single template. With cloud formation, it has automated resource provisioning, and helped me to save time and efforts. Additionally, CloudFormation helped me to streamline and simplify my deployment process.

## Architecture of my Project

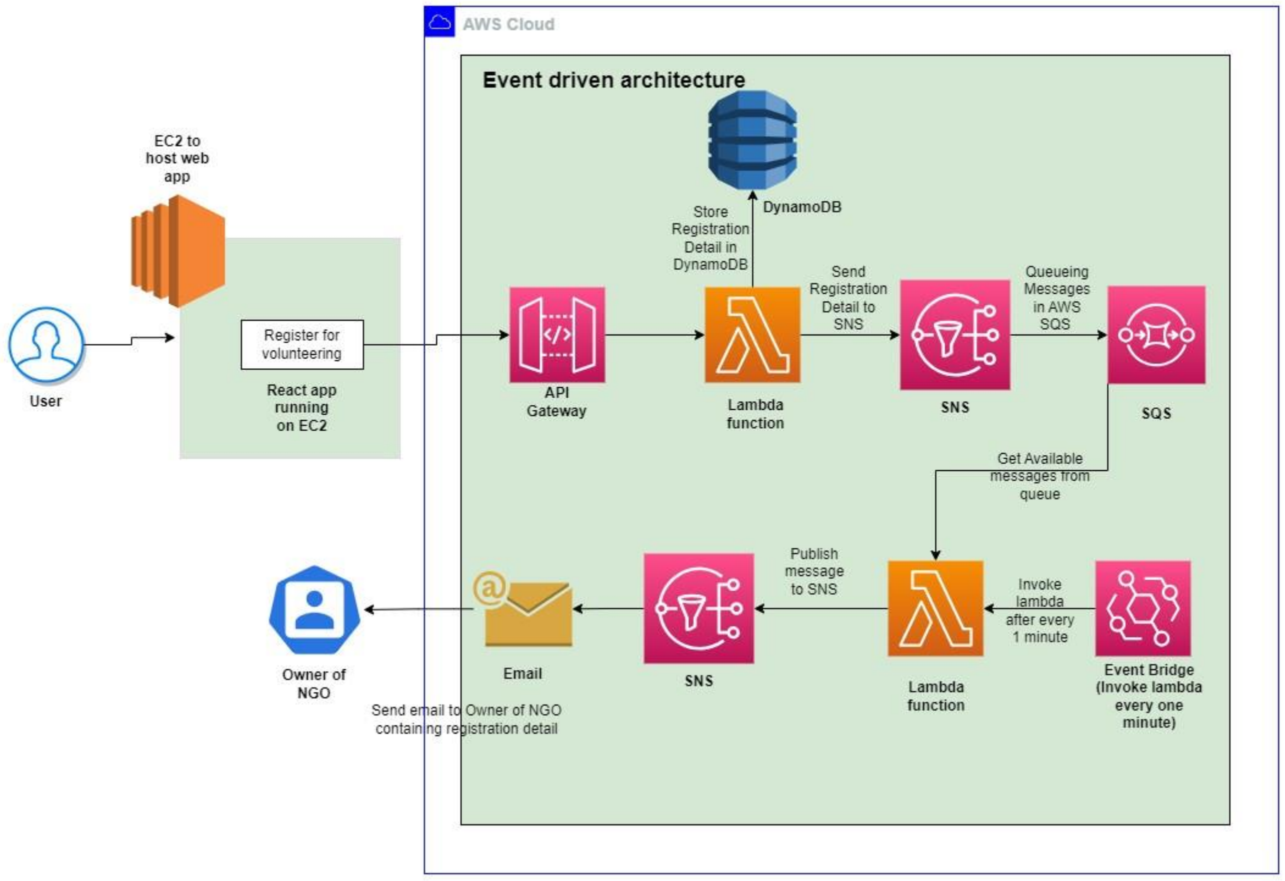
AWS Cloud
Event driven architecture
EC2 to host web app
User
Register for volunteering
React app running on EC2
API Gateway
Store Registration Detail in DynamoDB
DynamoDB
Lambda function
Send Registration Detail to SNS
SNS
Queueing Messages in AWS SQS
SQS
Get Available messages from queue
Invoke lambda after every 1 minute
Event Bridge (Invoke lambda every one minute)
Lambda function
Publish message to SNS
SNS
Email
Owner of NGO
Send email to Owner of NGO containing registration detail

Figure 1 Event Driven Architecture of volunteer management system [9]

1) **User Registration by Web application:** User will register by web app running on EC2 instance. When app get registration data , it will send registration data to lambda fucntion by API Gateway.

2) **Lambda trigger by API Gateway:** When any user registers by web app, app will send the registration details to lambda fucntion API gateway. When lambda gets registration details, lambda will store that details in DynamoDB and send that registration details in SNS topic.

3) **SNS queues registration details in SQS:** When SNS will get registration details from lambda, SNS topic will be queueing registration details in SQS in order to handle potential spikes in traffic and ensure smooth app  performance. This developing allows the app to perform efficiently and provides a buffer for processing message, even during high traffic periods.

4) **Created Event Bridge to invoke lambda:** Event Bridge has been created and configured with lambda fucntion. And schedule expression for minute, so event bridge will invoke lambda fucntion after every one minute.

5) **Invoke second lambda fucntion by Event Bridge:** When second lambda fucntion will be invoked by event bridge, lambda will get available messages from the queue and then publish that message containing registration details in SNS.

6) **SNS sends registration details to owner of an NGO:** When lambda will send registration details to SNS, SNS will send that details via email channel and send it to owner email address. By this, owner can get aware about new volunteer registration and owner will not have to check database every time.

## Deployment Model

For this term assignment, I have chosen Public Cloud deployment model because it is the most suitable deployment model for my serverless architecture.  The Public deployment cloud model offers several advantages such as scalability, cost-effectiveness, and ease of management. In addition, By this deployment model, it is very easy to scale up and down the resources as per demand. And you must pay only for the resources which has been used. This model provides the flexibility and agility needed to quickly adapt to changing requirements and ensure reliable and efficient delivery of your web application for NGO volunteer registration.

## Delivery Model

I have chosen Function as a service model for this term assignment because of its simplicity, scalability, and cost -effectiveness. Using lambda function, I can focus solely on writing the application logic without worrying about server management, allowing faster development. Its event-driven nature ensures automatic processing of volunteer registrations, while seamless integration with other AWS services like DynamoDB, SNS, and SQS enables smooth data flow and reliable notifications to the NGO owner.

## Security Analysis

In this project, security is of utmost importance to protect data at every stage of the system. Data in transit is secured through HTTPS and API Gateway's enforced SSL/TLS encryption. Registration details are stored securely in DynamoDB with encryption at rest, ensuring data remains protected even if physical storage is compromised. Real-time notifications are sent securely via SNS using encrypted communication between the web app and SNS, keeping sensitive information confidential. The system efficiently handles traffic spikes with SQS as a buffer, while IAM roles and permissions ensure proper access control. Infrastructure security is enhanced through VPCs, NACLs, and Security Groups for EC2 instances. Regular monitoring using AWS CloudTrail, AWS Config, and CloudWatch helps detect and respond to potential security incidents promptly. This holistic approach ensures the integrity and privacy of data, providing a robust and secure system for the NGO's website.

## Pricing

**AWS Pricing Overview:**

**AWS lambda:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay-per-request and compute time. | $0 (There is a free tier, but it has limits) | the first 1 million requests per month were free, and after that, it was $0.20 per 1 million requests |

**Amazon API Gateway:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay per API call and data transfer out | $0 | For the REST API: $3.50 per million API calls received, plus data transfer out costs. For the HTTP API (a cheaper option if you only need core features): $1.00 per million API calls received, plus the standard data transfer out costs. |

**AWS SNS:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay per published message and delivered notifications. | $0 (There is a free tier, but it has limits) | $0.50 per 1 million SNS requests |

**AWS SQS:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay per million requests. | $0 (There is a free tier, but it has limits) | $0.40 per 1 million requests after the free tier. |

**AWS DynamoDB:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay for read and write capacity, stored data, and data transfer | $0 (There is a free tier, but it has limits) | Depends on provisioned throughput, storage, and data transfer out. |

**AWS CloudFormation:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| There's no additional charge for AWS CloudFormation. You pay for the AWS resources (e.g., EC2, S3 buckets) created using CloudFormation. | $0 | There's no direct cost for CloudFormation itself. You'd pay for the resources it provisions and manages. |

**AWS Event Bridge:**

| Pricing Model | Upfront Cost | Monthly Cost |
|---|---|---|
| Pay-per-request and compute time. | There is no upfront cost for using AWS EventBridge. | The pricing for AWS EventBridge is based on pay-per-request and compute time. The first 1 million requests per month are free. After that, it costs $0.20 per 1 million requests. |

## Alternative Approach

- **DynamoDB:**
  - Alternatives: Amazon RDS or Amazon Aurora
  - Amazon RDS provides number of relational database options like MySQL and PostgreSQL, while Aurora is a cloud-native, high-performance relational database compatible with MySQL and PostgreSQL. Both alternatives combine the advantages of commercial databases with the cost-effectiveness of open-source solutions, providing flexibility and scalability.

- **Lambda:**
  - Alternatives: AWS Fargate or Amazon EC2.
  - AWS Fargate offers container-based solutions, allowing more control over the environment, while EC2 provides virtual servers for longer runtimes. These alternatives provide greater control but come with increased management overhead.

- **API Gateway:**
  - Alternatives: AWS Elastic Load Balancing (ELB)
  - AWS Elastic Load Balancing (ELB) can be considered as an alternative. Although ELB is primarily a load balancer, it can be configured to route HTTP/HTTPS traffic to various endpoints like EC2 instances, Lambda functions, or containers. While it may lack some API management features, ELB serves as a basic traffic router.

- **SQS:**
  - Alternatives: Amazon Kinesis
  - An alternative of SQS is Amazon Kinesis, which is more geared towards real-time big data processing and analytics. While SQS focuses on decoupling message producers from consumers, Kinesis is suitable for scenarios requiring real-time data analysis and can also function as a message bus.

- **SNS:**
  - Alternatives: Amazon Event Bridge
  - Alternative of SNS is Amazon Event Bridge as Event Bridge is a serverless event bus service that enables easy integration of applications using data from your applications, SaaS applications, and other AWS services. It can replace SNS when you need to filter and transform events before delivering them, providing enhanced event handling capabilities.

## References

[1] [Amazon, “AWS Lambda – Serverless Compute - Amazon Web Services,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 31st July 2023].

[2] AWS, “Amazon Simple Notification Service (SNS) | AWS,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/sns/. [Accessed: 31st July 2023].

[3] AWS, “Amazon Simple Queue Service (SQS) | Message Queuing for Messaging Applications | AWS,” Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/sqs/. [Accessed: 31st July 2023].

[4] “Amazon EventBridge - Amazon Web Services,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/eventbridge/. [Accessed: 31st July 2023].

[5] AWS, “Amazon EC2,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 31st July 2023].

[6] AWS, “Amazon API Gateway,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: 31st July 2023].

[7] AWS, “AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning,” Amazon Web Services, Inc., 2017. [Online]. Available: https://aws.amazon.com/cloudformation/. [Accessed: 31st July 2023].

[8] AWS, “Amazon DynamoDB - Overview,” Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: 31st July 2023].

[9] Draw.io, “Flowchart Maker & Online Diagram Software,” app.diagrams.net, 2023. [Online]. Available: https://app.diagrams.net/. [Accessed: 31st July 2023].