DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

CSCI 5409 – Advanced Topics in Cloud Computing

Term Project

Submitted by:

Anjali Rachel Benjamin

B0094335

Table of Contents

Term Assignment Final Report

Project Introduction

My project is a serverless patient management system for hospitals. This system is designed to streamline the process of managing patient records, including entering patient details, storing them securely, and sending notifications for appointment bookings. The primary users of this system are hospital staff, including doctors, nurses, and administrative personnel, who need to manage patient information efficiently.

Project Goals

The primary objectives of the project are:

- To provide a reliable and efficient way to manage patient records.

- To enable seamless integration with hospital systems for appointment scheduling and notifications.

- To ensure data security and compliance with healthcare regulations.

Users and Performance Targets

The users of this system include:

- Doctors: To access patient records and update treatment plans.
- Nurses: To view patient information and record observations.
- Administrative Staff: To manage patient appointments and contact patients for reminders.

The system should handle a high volume of concurrent users with minimal latency, ensuring fast access to patient records and timely notifications.

Services Selected and Alternatives

Compute

1. AWS EC2: Chosen to host the web application.

- Alternative: AWS Elastic Beanstalk.

- Comparison: Elastic Beanstalk abstracts much of the infrastructure management but might be slower for custom configurations whereas EC2 offers greater flexibility and control over the underlying infrastructure, making it suitable for a wider range of applications and custom configurations.[2]

- Reason for Choice: EC2 allows for customized configurations and better control over the deployment environment, which is crucial for a healthcare application.

2. AWS Lambda: Chosen for serverless functions.

- Alternative: AWS Elastic Beanstalk with Docker.

- Comparison: Lambda offers a serverless environment with automatic scaling and pay-per-use pricing, while Elastic Beanstalk with Docker provides containerized applications but requires more management.

- Reason for Choice: Lambda is cost-effective and simplifies the deployment of individual functions, aligning with the serverless architecture goal.

Storage

- **AWS DynamoDB**: Chosen for NoSQL database storage.

- Alternative: AWS Aurora.

- Comparison: DynamoDB is a NoSQL database offering fast and predictable performance, while Aurora is a managed relational database that can be more expensive. DynamoDB is ideal for applications requiring high-speed transactions and scalability without the need for complex querying capabilities. On the other hand, Aurora is perfect for applications that need relational database features, such as complex queries and transactions, with the added benefit of high performance and scalability. [3]

- Reason for Choice: DynamoDB provides the necessary scalability and performance for managing patient records without the high costs associated with Aurora.

Network

- AWS API Gateway: Chosen for routing API requests to Lambda functions.

- Alternative: AWS Virtual Private Cloud (VPC).

- Comparison: API Gateway simplifies the creation and management of APIs, while VPC provides network isolation but requires more configuration.

- Reason for Choice: API Gateway offers a straightforward way to manage APIs and integrate with other AWS services, making it ideal for a serverless architecture.

General

1. Amazon EventBridge: Chosen for event-driven architecture.

- Alternative: Amazon CloudFront.

  - Comparison: EventBridge is designed for event-driven applications, while CloudFront is a CDN for high-speed content delivery.

- Reason for Choice: EventBridge is suitable for decoupling event sources and handling events in a serverless architecture.

2. AWS SNS: Chosen for sending notifications.

- Alternative: AWS SQS.

  - Comparison: SNS is a pub/sub messaging service for real-time notifications, whereas SQS is a message queue service.

- Reason for Choice: SNS is ideal for sending real-time notifications to patients and staff.

Deployment Model

Approach: We're deploying our application on the public cloud, using a combination of AWS services.

Why: Leveraging the public cloud offers numerous advantages:

Scalability: AWS services like Lambda and EC2 allow us to scale our application based on demand, ensuring performance and cost-efficiency.

Flexibility: EC2 instances provide the flexibility to configure and manage specific application requirements that serverless functions might not address.

Cost-Effectiveness: We only pay for the resources we use, particularly with serverless functions like AWS Lambda.

Reliability: AWS provides high availability and disaster recovery solutions, ensuring our application remains available and resilient.

Delivery Model

Model: SaaS (Software as a Service)

Why: In a SaaS delivery model, we provide our software application over the internet as a service. Users can access the application via a web browser, without the need for installing or maintaining hardware or software on their end. This model offers several benefits:

Ease of Access: Users can access the application from anywhere with an internet connection.

Cost Efficiency: Reduces the need for upfront hardware costs and ongoing maintenance.

Automatic Updates: Ensures users always have access to the latest features and security patches without needing to manually update the software.

Final Architecture

Integration of Cloud Mechanisms

- EC2: Hosts the web application for user interactions.
- Lambda: Executes backend functions for CRUD operations on patient data and sending notifications.
- DynamoDB: Stores patient records securely.
- API Gateway: Routes API requests to Lambda functions.
- EventBridge: Manages event-driven workflows for patient appointments and notifications.
- SNS: Sends notifications to patients and staff.

Data Storage

DynamoDB: Selected for its scalability and rapid performance, making it ideal for managing patient records. Its ability to handle large datasets and provide quick access ensures efficient operation of the application.

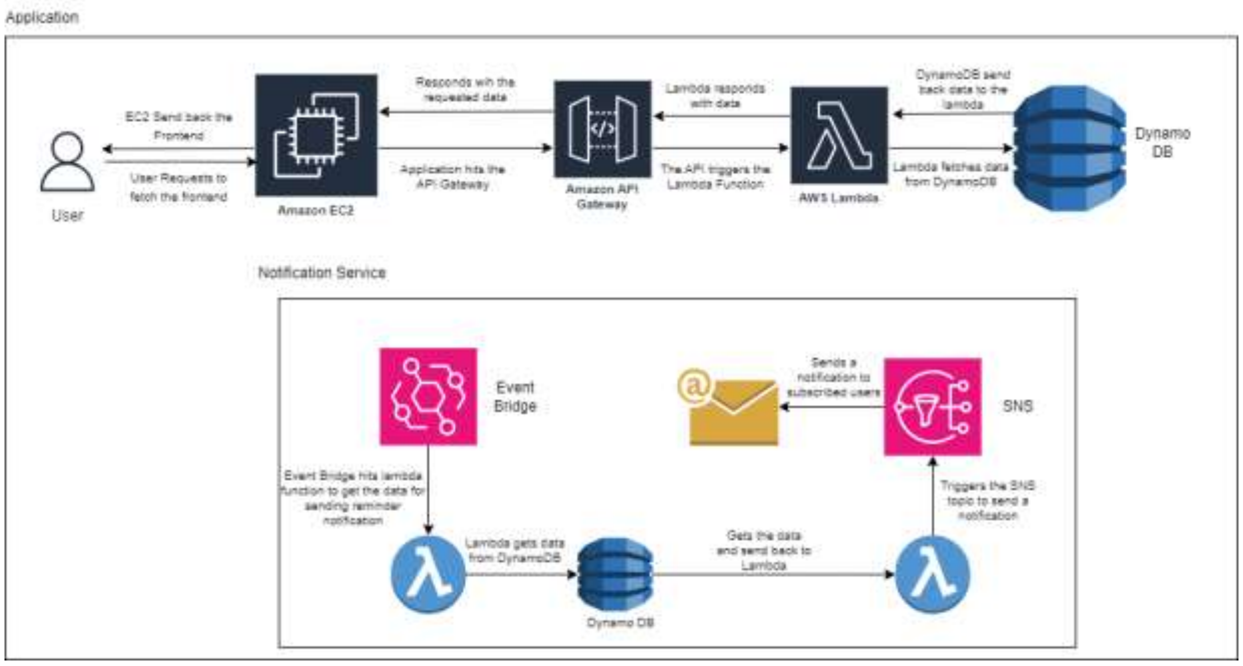
Programming Languages

- **Node.js:** Used for Lambda functions due to its lightweight nature and ease of integration with AWS services.
- **JavaScript/React:** Used for the frontend application for a responsive and dynamic user interface.

System Deployment

- **CloudFormation:** Used to define and provision the entire infrastructure as code, ensuring consistency and repeatability.

Architecture Diagram


Application
User
EC2 Send back the Frontend
User Requests to fetch the frontend
Amazon EC2
Responds wih the requested data
Application hits the API Gateway
Amazon API Gateway
Lambda responds with data
The API triggers the Lambda Function
AWS Lambda
DynamoDB send back data to the lambda
Lambda fetches data from DynamoDB
Dynamo DB
Notification Service
Event Bridge
Event Bridge hits lambda function to get the data for sending reminder notification
Lambda gets data from DynamoDB
Dynamo DB
Gets the data and send back to Lambda
Triggers the SNS topic to send a notification
SNS
Sends a notification to subscribed users

Security Considerations

Data Security

- **At Rest:** If you're using DynamoDB (even though it wasn't explicitly mentioned in the deployment), DynamoDB automatically encrypts data at rest using AWS Key Management Service (KMS).[4]
- **Access Control:** You mentioned using an IAM role provided by your college. This role controls access to your AWS resources.

Security Mechanisms

- **Encryption:** AWS KMS (Key Management Service) is used for data encryption, ensuring that data stored in DynamoDB, S3, or other services is secure.
- **IAM:** The IAM role provided by your college is used to restrict access to AWS resources, ensuring that only authorized users and services can access them.
- **VPC:** If your EC2 instances are launched within a VPC (Virtual Private Cloud), it provides network isolation, adding an extra layer of security.

Cost Analysis

Up-Front Costs

- EC2 Instances: Estimated $500 for initial setup.

- Development Costs: Estimated $2000 for initial development and configuration.

Ongoing Costs

- AWS Lambda: Pay-per-use, estimated $100/month.

- DynamoDB: $0.02 per 100,000 data reading operations

- API Gateway: Estimated $50/month for API requests.

- EventBridge: Estimated $10/month for event handling.

- SNS: Estimated $20/month for notifications.

Private Cloud Reproduction

Estimated Costs and Requirements

- Hardware: Servers for hosting web applications, database servers for patient records.

- Software: Database management systems, application servers, monitoring tools.

- Estimated Costs: Approx. $50,000 for initial hardware and software, plus ongoing maintenance and support costs.

Monitoring Service:

AWS Lambda: Monitoring Lambda usage is crucial as it can scale automatically, potentially leading to higher costs if not managed properly.

Future Development

- Feature 1: Real-time chat between patients and doctors.

- Mechanism: AWS AppSync for GraphQL data synchronization.

- Feature 2: Advanced analytics on patient data.

- Mechanism: AWS Athena for querying data stored in S3.

By integrating these features, the system can enhance patient-doctor interactions and provide deeper insights into patient health data.

Conclusion

This project demonstrates the implementation of a serverless patient management system using AWS services. The choices made in terms of services, deployment model, and security considerations were driven by the need for scalability, reliability, and cost-effectiveness. The system architecture ensures efficient handling of patient records and timely notifications.

References

[1] "draw.io", *draw.io* [Online]. Available: https://www.drawio.com/ . [Accessed: August 4, 2024].

[2] "AWS Elastic Beanstalk vs EC2: A Detailed Comparison", *sitepoint,* [online]. Available: https://www.sitepoint.com/aws-elastic-beanstalk-vs-ec2/ . [Accessed: August 2, 2024].

[3] "DynamoDB vs Amazon Aurora — The Ultimate Comparison", *Dynobase*, [online]. Available: https://dynobase.dev/dynamodb-vs-aurora/ . [Accessed: August 5, 2024].

[4] "DynamoDB encryption at rest", *AWS,* [online], Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html . [Accessed: August 6, 2024].

[5] "What Is AWS? Overview, Services, and Pricing Calculator", *spiceworks*, [online]. Available: https://www.spiceworks.com/tech/cloud/articles/aws-basics/ . [Accessed: August 5, 2024]