

**CSCI 5409 – Advanced Topics in Cloud Computing**

**Term Project**

**Submitted by:**

Anjali Rachel Benjamin

B0094335

**Table of Contents**

[Term Assignment Final Report 3](#_Toc173848591)

[Project Introduction 3](#_Toc173848592)

[Project Goals 3](#_Toc173848593)

[Users and Performance Targets 3](#_Toc173848594)

[Services Selected and Alternatives 3](#_Toc173848595)

[Compute 3](#_Toc173848596)

[Storage 4](#_Toc173848597)

[Network 4](#_Toc173848598)

[General 4](#_Toc173848599)

[Deployment Model 5](#_Toc173848600)

[Delivery Model 5](#_Toc173848601)

[Final Architecture 5](#_Toc173848602)

[Integration of Cloud Mechanisms 5](#_Toc173848603)

[Data Storage 6](#_Toc173848604)

[Programming Languages 6](#_Toc173848605)

[System Deployment 6](#_Toc173848606)

[Architecture Diagram 6](#_Toc173848607)

[Security Considerations 7](#_Toc173848608)

[Data Security 7](#_Toc173848609)

[Security Mechanisms 7](#_Toc173848610)

[Cost Analysis 7](#_Toc173848611)

[Alternative Approaches 7](#_Toc173848612)

[Private Cloud Reproduction 8](#_Toc173848613)

[Monitoring for Cost Management 8](#_Toc173848614)

[Conclusion 8](#_Toc173848615)

# 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: EC2 provides more control over the environment, whereas Elastic Beanstalk abstracts much of the infrastructure management but might be slower for custom configurations.

- 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.

- 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

- Model: Serverless and virtual machines using AWS Lambda and EC2.

- Reason: The serverless model with Lambda allows for scalable and cost-effective function execution, while EC2 provides the flexibility to host the web application with specific configurations.

## Delivery Model

- Model: Continuous Integration/Continuous Deployment (CI/CD) with AWS CloudFormation.

- Reason: CI/CD ensures automated, reliable, and repeatable deployments, reducing manual intervention and the risk of errors.

## 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: Chosen for its scalability and fast performance for storing patient records.

### 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

A diagram of a software company

Description automatically generated with medium confidence

## Security Considerations

### Data Security

- At Rest: DynamoDB encrypts data at rest using AWS KMS.

- In Transit: API Gateway uses HTTPS to encrypt data in transit.

- Access Control: IAM roles and policies are used to restrict access to resources.

### Security Mechanisms

- Encryption: AWS KMS for data encryption.

- IAM: Fine-grained access control using IAM roles and policies.

- VPC: EC2 instances are launched within a VPC for network isolation.

## 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: Estimated $200/month based on storage and read/write operations.

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

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

- SNS: Estimated $20/month for notifications.

## Alternative Approaches

- Alternative Storage: Using AWS S3 instead of DynamoDB could reduce costs but might not provide the same performance.

- Alternative Compute: Using AWS Fargate for containerized applications could offer more control but at a higher cost.

## 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 for Cost Management

Key Cloud Mechanism

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

Future Development

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

- Mechanism: AWS AppSync for GraphQL data synchronization.

- Feature: 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, while the use of CI/CD practices and infrastructure as code (IaC) ensures a robust and maintainable deployment process.