## ECE547/CSC547

## Cloud Architecture Fall 2023

*Yannis Viniotis and Ioannis Papapanagiotou*

## Vatsal Palan ([vpalan@ncsu.edu](mailto:vpalan@ncsu.edu) - 200533560)

Ishika Gandhi ([igandhi2@ncsu.edu](mailto:igandhi2@ncsu.edu) - 200534458)

“We, the team members, understand that copying pasting material from any source in our project is an allowed practice; we understand that not properly quoting the source constitutes plagiarism. All team members attest that we have properly quoted the sources in every sentence/paragraph we have copy pasted in our report. We further attest that we did not change words to make copy pasted material appear as our work.”

**1 Introduction**

We have chosen to explore 'MyChart,' Cone Health's patient portal, providing secure online access to personal health records, appointment scheduling, and communication with healthcare providers. It allows patients to access their medical information 24/7, enhancing their engagement in their healthcare journey. The application prioritizes data security, complying with regulations like HIPAA. With an aim to reduce operational costs, it's designed to accommodate variable workloads and optimize profitability through premium service offerings. Overall, MyChart is a comprehensive tool that empowers patients, improves healthcare efficiency, and ensures data privacy.

**1.1 Motivation**

The motivation behind undertaking the MyChart Health Application project stems from a commitment to revolutionize healthcare accessibility and patient engagement. As a cloud architect, the aim is to leverage cloud technologies to create a secure, scalable, and interoperable platform that empowers users to seamlessly manage their health information, facilitates communication with healthcare providers, and enhances overall patient experience. The motivation is grounded in the desire to contribute to the digital transformation of healthcare, making it more patient-centric, efficient, and accessible.

**1.2 Executive summary**

In this project, the MyChart Health Application is conceived and designed as a cloud architect's strategic response to modernizing healthcare interactions. The application seeks to provide a unified platform for patients to access, manage, and communicate their health data securely. As the cloud architect, the focus is on implementing cutting-edge cloud technologies, such as dynamic scaling, disaster recovery best practices, and interoperability standards like HL7 FHIR. This executive summary is crafted for stakeholders, including healthcare providers, patients, and executives, conveying the project's essence of transforming healthcare delivery through innovative cloud architecture, emphasizing security, scalability, and patient engagement.[2][69]

**2 Problem Description**

**2.1 The Problem**

The high-level problem that the MyChart application addresses is the inefficiency and fragmentation in the current healthcare information management system. Patients often struggle to access their medical records and communicate with their healthcare providers in a convenient and timely manner. This lack of accessibility can lead to mismanagement of health, delayed care, and overall dissatisfaction. Healthcare providers, on the other hand, face challenges in coordinating care and accessing critical patient data swiftly, leading to suboptimal healthcare delivery.[2]

**2.2 Business Requirements:**

* BR1 - Optimize operational costs to ensure cost-efficiency[7][10]

* BR2 - Accommodate varying workloads efficiently to maintain consistent performance.[8][9]

* BR3 - Ensure the MyChart application is available 24/7, with minimal downtime.[11][12]

* BR4 - Ensure MyChart application operations are highly secure and compliant with industry standards and regulations, such as HIPAA for healthcare data. [13][55]

* BR5 - Avoid Vendor Lock-In [56]

* BR6 - Maximize Profit

Example: Offer premium features, such as telehealth consultations, for a fee. Collaborate with pharmaceutical companies for sponsored content or clinical trials, generating additional revenue.

* BR7 - Reduce Management Complexity

Example: Use infrastructure as code (IaC) tools like Terraform [57] to automate the provisioning and management of cloud resources, simplifying administrative tasks.

* BR8: Ensure low latency for the MyChart application to provide a responsive user experience. [58][59]

* BR9 - Ensure Data Privacy Compliance [61]

* BR10 - Enhance Interoperability

Example: Implement Health Level 7 (HL7) FHIR standards for data exchange, allowing seamless integration with various healthcare systems and EHRs.

* BR11 - Enable Multi-Platform Access

* BR12 - Establish Disaster Recovery

Example: Set up data replication and backup solutions to geographically diverse data centers. Establish a clear disaster recovery plan with regular testing.

* BR13 - Improve Patient Engagement Features

Example: Offer secure messaging and video consultations with healthcare providers through the MyChart app, making it easier for patients to communicate and receive care.

* BR14 - Monitor and Tune Performance [63]

* BR15 - Incorporate a Feedback Mechanism:

* BR16 - Ensure Tenant Identification

**2.3 Technical Requirements**

* [TR1.1:](#slq4x05oq2nt) Implement automated cost tracking and analysis tools to identify areas of cost optimization, connecting to the cost optimization goal in BR1.[7]

[TR1.2](#slq4x05oq2nt): Create detailed cost reports and dashboards for cost transparency and decision-making, supporting BR1.

* [TR2.1](#bfflq1voz7zn): dynamically scale resources based on workload fluctuations, aligning with BR2.

[TR2.2](#bfflq1voz7zn): Implement automation scripts to manage resource provisioning and deprovisioning in response to varying workloads, addressing the adaptability goal in BR2.

* [TR3.1](#570tjyk2hhj7): Develop and maintain a comprehensive disaster recovery plan that includes backup, data replication, and failover strategies to achieve the 24/7 availability requirement set in BR3.
* [TR4.1](#unjazoi8xyel): Implement monitoring to track the performance, security of MyChart application, ensuring compliance with BR1.

[TR4.2](#unjazoi8xyel): Configure monitoring to promptly detect unauthorized access attempts and alert the appropriate personnel, meeting the security requirement in BR1.

* [TR5.1](#ku07nov3jafm): Develop a vendor-agnostic cloud strategy, focusing on standardization and modular architecture to avoid vendor lock-in and promote flexibility.
* [TR6.1](#bu9f72ncz0p0): Implement a dynamic pricing model that adjusts based on demand, allowing for increased revenue during peak usage periods.

[TR6.2](#bu9f72ncz0p0): Leverage machine learning and AWS data analytics to identify opportunities for upselling or cross-selling premium services.

* [TR7.1](#63xsykq9nhmn): Adopt infrastructure as code (IaC) practices to automate the provisioning and configuration of cloud resources, reducing the complexity of manual management tasks.

[TR7.2](#63xsykq9nhmn): Utilize cloud management platforms and dashboards to centralize resource management and simplify administrative tasks.

* [TR8.1](#fyhpm7qpwdik): cache and distribute content, reducing latency and enhancing user experience.
* [TR9.1](#iemjayf9gm6x): Classify and protect sensitive data, ensuring data privacy compliance.

[TR9.2](#iemjayf9gm6x): Implement continuous monitoring and security controls, maintaining data privacy compliance.

* [TR10.1](#zfgyzcb0qeqx): Utilize API Gateway to create RESTful APIs and facilitate data exchange with external healthcare systems through AWS's serverless architecture.

[TR10.2](#zfgyzcb0qeqx): Implement message queuing and data exchange, enabling real-time interoperability with external systems.

* [TR11.1](#9n9urc4pt9pr): Develop mobile applications to provide a consistent user experience on various platforms, including iOS and Android.
* [TR12.1](#guu2oju64ayl): Set up data replication and backup, ensuring data recovery and application failover capabilities.

[TR12.2](#guu2oju64ayl): Create templates for disaster recovery, enabling the rapid deployment of backup environments during disasters.

* [TR13.1](#71glgfa7xv1s): Integrate secure messaging and video consultations within the MyChart app, enhancing patient engagement and communication.

[TR13.2](#71glgfa7xv1s): Utilize real-time push notifications to keep patients informed and engaged with their healthcare providers.

* [TR14.1](#ubup4cvcwy2x): Implement performance monitoring and optimization, ensuring the continuous tuning of system performance. Total requests per sec, Response time, No of users, No of request failures and availability
* [TR15.1](#hjwj0ruggl2n): Create an AWS Lambda function and Amazon API Gateway to capture and store user feedback in Amazon DynamoDB, allowing for continuous improvement based on user feedback.
* [TR16.1:](#yerwb4ilivy4) Implement a secure and scalable user authentication mechanism to uniquely identify tenants within the myChart application.

[TR16.2:](#yerwb4ilivy4) Integrate a tenant-specific data segmentation strategy to isolate and manage data belonging to different tenants effectively.

**2.4 Tradeoffs**

* **Operational Excellence vs. Cost Optimization:**

Tradeoff: Balancing the efficiency gained through operational excellence (BR1) with the need for cost optimization (BR14, TR1.1).

Justification: Implementing extensive operational processes and procedures may incur additional costs. Striking a balance ensures efficient operations without unnecessary financial burdens.[7][10]

* **Security vs. Performance Efficiency:**

Tradeoff: Achieving stringent security measures (BR4) while optimizing performance efficiency (BR2, TR4.1).

Justification: Introducing high-level security measures, such as encryption and multi-factor authentication, may impact system performance. Finding the right balance ensures a secure yet performant MyChart application.

* **Reliability vs. Operational Excellence:**

Tradeoff: Prioritizing reliability (BR3) versus achieving operational excellence through continuous improvement.

Justification: Strict adherence to reliability might limit the agility for continuous improvements. Balancing these aspects ensures the MyChart application remains reliable while adapting to evolving requirements.

**3 Provider Selection**

Here are three major cloud providers which provides a broad range of services required to fulfill our Technical Requirements:

Amazon Web Services (AWS):[7]

* AWS is known for its extensive service offerings, including AWS Cost Explorer, Amazon Athena, Amazon QuickSight for cost tracking and analysis (TR1).
* It provides container orchestration tools like Amazon Elastic Container Service (ECS) and supports Kubernetes (TR2).
* AWS offers disaster recovery services such as AWS Database Migration Service, AWS Storage Gateway, and CloudFormation templates for backup and failover (TR3, TR12).
* Monitoring tools like Amazon CloudWatch and security tools like Amazon GuardDuty address the performance monitoring and security requirements (TR4, TR14).
* Services like AWS Lambda, Amazon API Gateway, and SQS support the development of RESTful APIs and real-time interoperability (TR10).
* AWS CloudFront is used for content delivery, reducing latency (TR8).
* AWS Data Privacy tools, including Amazon Macie and AWS DataSync, address data privacy compliance (TR9).
* AWS Chime, AWS EventBridge, and Amazon SNS support secure messaging, video consultations, and real-time push notifications (TR13).

Microsoft Azure:

* Azure offers tools for cost tracking and analysis, including Azure Cost Management and Azure Monitor (TR1).
* Azure Kubernetes Service and Azure Container Instances are available for container orchestration (TR2).[12]
* Azure Site Recovery and Azure Backup cater to disaster recovery needs (TR3).
* Azure Monitor and Azure Security Center address performance monitoring and security requirements (TR4, TR14).
* Azure Functions, Azure API Management, and Azure Service Bus support API development and real-time interoperability (TR10).
* Azure Content Delivery Network (CDN) reduces latency for content delivery (TR8).
* Azure Information Protection and Azure Sentinel contribute to data privacy compliance and security monitoring (TR9).
* Microsoft Teams and Azure Notification Hubs can be used for secure messaging and real-time push notifications (TR13).

Google Cloud Platform (GCP):

* GCP provides tools like Google Cloud Billing and BigQuery for cost tracking and analysis (TR1).
* Kubernetes Engine and Google Cloud Functions offer solutions for container orchestration (TR2).
* Google Cloud Disaster Recovery, Google Cloud Storage, and Deployment Manager are available for disaster recovery (TR3, TR12).[11]
* Stackdriver Monitoring and Cloud Security Scanner address performance monitoring and security requirements (TR4, TR14).
* Cloud Endpoints, Cloud Pub/Sub, and Cloud Storage facilitate API development and real-time interoperability (TR10).
* Cloud CDN minimizes latency for content delivery (TR8).
* Data Loss Prevention API and Cloud IAM contribute to data privacy compliance and security (TR9).
* Google Meet and Cloud Pub/Sub support secure messaging and real-time push notifications (TR13). [5][15][50][51]

**3.1 Criteria for choosing a provider** [52]

* Service Offering and Compatibility:

Assess the availability and compatibility of essential services required to meet technical requirements (TRs) outlined in Section 2.3.

* Cost and Pricing Structure:

Evaluate the cost of services and the pricing structure, including any hidden fees. Consider the cost-effectiveness of the selected provider in alignment with the optimization of operational costs (BR1).

* Scalability and Performance:

Examine the provider's scalability features and performance capabilities to ensure efficient accommodation of varying workloads (BR2) and low latency for the MyChart application (BR8).

* Reliability and Availability:

Evaluate the provider's track record for reliability and availability to ensure the 24/7 availability of the MyChart application with minimal downtime (BR3).

* Security and Compliance:

Assess the provider's security measures, compliance certifications, and tools for monitoring and responding to unauthorized access attempts (BR4, BR16).

* Flexibility and Vendor Lock-In Avoidance:

Consider the flexibility of the provider's services and the ability to avoid vendor lock-in, aligning with the goal of avoiding vendor lock-in (BR5).

* Profit Maximization Features:

Evaluate the provider's capabilities to support profit maximization strategies, such as dynamic pricing models and machine learning for identifying revenue opportunities (BR6).

* Management Complexity and Automation:

Assess the provider's tools and services that contribute to reducing management complexity (BR7) and supporting infrastructure as code (IaC) practices (TR7).

* Data Privacy and Compliance Tools:

Evaluate the provider's data privacy tools and capabilities for ensuring compliance with regulations (BR9).

* Interoperability and Integration:

Examine the provider's support for interoperability, including the implementation of standards like HL7 FHIR (BR10).

* Multi-Platform Access:

Assess the provider's tools for developing applications that enable multi-platform access (BR11).

* Disaster Recovery Capabilities:

Evaluate the disaster recovery services and capabilities provided by the cloud provider (BR12).

* Patient Engagement Features:

Assess the provider's tools and services that support the implementation of patient engagement features, such as secure messaging and video consultations (BR13).

* Monitoring and Optimization Tools:

Examine the monitoring and optimization tools provided by the cloud provider to meet the requirements of BR14.

* Feedback Mechanism and User Experience:

Evaluate the tools and services that facilitate the implementation of a feedback mechanism and contribute to a positive user experience (BR15).

* Innovation and Future-Proofing:

Consider the provider's commitment to innovation and its ability to adapt to future technological advancements.

**3.2 Provider Comparison**

| **Criteria** | **AWS** | **Microsoft Azure** | **Google Cloud Platform** |
| --- | --- | --- | --- |
| Service Offering and Compatibility | Comprehensive, extensive service portfolio | Broad range of services, compatible | Varied services, strong integration |
| BR1 - Cost and Pricing Structure | Competitive pricing, transparent structure | Flexible pricing, potential cost savings | Competitive pricing, sustained use discounts |
| BR2 - Scalability and Performance | Highly scalable, strong performance | Scalable, good performance metrics | Strong scalability, high performance |
| BR3 - Reliability and Availability | Proven track record, high availability | Strong reliability, global data centers | Reliable infrastructure, global reach |
| BR4 - Security and Compliance | Robust security measures, compliance tools | Emphasis on security, compliance focus | Security focus, compliance capabilities |
| BR5 - Flexibility and Vendor Lock-In Avoidance | Broad service options, modular architecture | Emphasis on openness, hybrid capabilities | Vendor-agnostic, modular architecture |
| BR6 - Profit Maximization Features | Diverse services for profit maximization | Supports dynamic pricing, analytics | Machine learning for revenue insights |
| BR7 - Management Complexity and Automation | IaC tools (Terraform), automation emphasis | Azure Automation, infrastructure as code | Automation tools, IaC practices |
| BR9 - Data Privacy and Compliance Tools | Robust tools (Amazon Macie, AWS DataSync) | Azure Information Protection, Sentinel | Data privacy tools, compliance focus |
| BR10 - Interoperability and Integration | Broad support, HL7 FHIR standards for data exchange | Integration services, adaptable architecture | Strong interoperability, open standards |
| BR11 - Multi-Platform Access | Tools for mobile app development (AWS Amplify) | Xamarin, App Service for multi-platform | Multi-platform development support |
| BR12 - Disaster Recovery Capabilities | Comprehensive services (AWS Database Migration, Storage Gateway) | Azure Site Recovery, Azure Backup | Disaster recovery services available |
| BR13 - Patient Engagement Features | AWS Chime, EventBridge, SNS for communication | Microsoft Teams, Notification Hubs | Google Meet, Cloud Pub/Sub for engagement |
| BR14 - Monitoring and Optimization Tools | CloudWatch, X-Ray for performance monitoring | Azure Monitor for insights, optimization | Stackdriver, tools for performance tuning |
| BR15 - Feedback Mechanism and User Experience | AWS Lambda, API Gateway for feedback capture | Azure Functions, API Management | Cloud Functions, tools for user feedback |

[5][53][54]

* AWS (Amazon Web Services): AWS is ranked highest due to its extensive service offerings, proven track record, and strong commitment to innovation. It excels in scalability, security, and flexibility, providing a comprehensive solution for the MyChart project. The Cloud provider currently used by Epic for its softwares like myChart is AWS.
* Microsoft Azure: Azure is a close second, offering a broad range of services and a strong focus on security and compliance. It is well-suited for organizations with existing Microsoft technologies and emphasizes hybrid cloud capabilities.
* Google Cloud Platform (GCP): GCP ranks third, providing strong interoperability, an emphasis on open standards, and competitive pricing. It is recognized for its machine learning capabilities and global infrastructure.

**3.3 The final selection**

* **AWS** is the final selection of the cloud provider that we choose

**3.3.1 The list of services offered by the winner**

* The list of services can be found here - [AWS services](https://aws.amazon.com/products/?nc2=h_ql_prod_fs_f&aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=*all&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all) [4]
* Amazon S3 (Simple Storage Service):

Description: Amazon S3 is a scalable object storage service designed to store and retrieve any amount of data from anywhere on the web. It provides industry-leading durability, availability, and performance characteristics.

Use Cases: S3 is commonly used for backup and restore, data archiving, content distribution, and as the storage backend for various AWS services.

* Amazon EC2 (Elastic Compute Cloud):

Description: Amazon EC2 offers virtual computing capacity in the cloud. It allows users to rent virtual servers, known as instances, and run applications on them. EC2 provides flexibility in terms of instance types, operating systems, and security configurations.

Use Cases: EC2 is widely used for web hosting, application development, testing, and running various types of scalable applications.

* Amazon RDS (Relational Database Service):

Description: Amazon RDS is a managed relational database service that makes it easier to set up, operate, and scale a relational database in the cloud. It supports multiple database engines, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

Use Cases: RDS is commonly used for applications that require a relational database, such as e-commerce websites, content management systems, and business applications.

* AWS Lambda:

Description: AWS Lambda is a serverless computing service that runs code without provisioning or managing servers. It automatically scales and manages the computing resources required by the code, allows to focus on writing the code and not managing infrastructure.

Use Cases: Lambda is often used for event-driven computing, real-time file processing, data transformations, and building serverless applications.

**4 The first design draft [0%]**

**4.1 The basic building blocks of the design**

* Cost Optimization (BR1, TR1.1, TR1.2):

AWS Service: AWS Cost Explorer, AWS Budgets

Description: Implement automated cost tracking using AWS Cost Explorer to identify areas of cost optimization. Create detailed cost reports and dashboards for transparency and decision-making using AWS Budgets.

* Workload Efficiency (BR2, TR2.1, TR2.2):

AWS Service: Auto Scaling, AWS Lambda, AWS CloudFormation

Description: Dynamically scale resources based on workload fluctuations using Auto Scaling. Implement automation scripts with AWS Lambda for resource provisioning and deprovisioning. Use AWS CloudFormation for infrastructure as code (IaC).

* 24/7 Availability and Disaster Recovery (BR3, BR12, TR3.1, TR12.1, TR12.2):

AWS Service: AWS RDS (for database replication), AWS Backup, AWS CloudFormation

Description: Develop and maintain a comprehensive disaster recovery plan using AWS Backup. Utilize AWS RDS for database backup, replication, and failover strategies. Implement AWS CloudFormation for disaster recovery template creation.

* Security and Monitoring (BR4, BR9, TR4.1, TR4.2):

AWS Service: AWS CloudWatch, AWS GuardDuty, AWS IAM

Description: Implement monitoring using AWS CloudWatch for tracking performance, security, and availability. Configure AWS GuardDuty to promptly detect unauthorized access attempts. Utilize AWS IAM for robust identity and access management.

* Vendor-Agnostic Cloud Strategy (BR5, TR5.1):

AWS Service: AWS Outposts

Description: Implement a vendor-agnostic cloud strategy using AWS Outposts for on-premises cloud infrastructure, focusing on standardization and modular architecture.

* Dynamic Pricing and Machine Learning (BR6, TR6.1, TR6.2):

AWS Service: AWS Lambda, Amazon Machine Learning

Description: Implement a dynamic pricing model using AWS Lambda for adjusting based on demand. Leverage machine learning and AWS data analytics using Amazon Machine Learning to identify opportunities for upselling or cross-selling premium services.

* Infrastructure as Code (BR7, TR7.1, TR7.2):

AWS Service: AWS CloudFormation, AWS Systems Manager

Description: Adopt infrastructure as code (IaC) practices using AWS CloudFormation for automating the provisioning and configuration of cloud resources. Utilize AWS Systems Manager for centralizing resource management and simplifying administrative tasks.

* Low Latency and Content Delivery (BR8, TR8.1):

AWS Service: Amazon CloudFront

Description: Cache and distribute content using Amazon CloudFront to reduce latency and enhance the user experience for the MyChart application.

* Data Privacy Compliance (BR9, TR9.1, TR9.2):

AWS Service: AWS Key Management Service (KMS), AWS Macie

Description: Classify and protect sensitive data using AWS KMS for encryption. Implement continuous monitoring and security controls for maintaining data privacy compliance with AWS Macie.

* Interoperability and API Gateway (BR10, TR10.1, TR10.2):

AWS Service: Amazon API Gateway, AWS Lambda

Description: Utilize API Gateway to create RESTful APIs and facilitate data exchange with external healthcare systems through AWS's serverless architecture. Implement message queuing and data exchange using AWS Lambda for real-time interoperability with external systems.

* Multi-Platform Access (BR11, TR11.1):

AWS Service: AWS Mobile Hub

Description: Develop mobile applications using AWS Mobile Hub to provide a consistent user experience on various platforms, including iOS and Android.

* Patient Engagement and Feedback Mechanism (BR13, BR15, TR13.1, TR13.2, TR15.1):

AWS Service: Amazon DynamoDB, AWS Lambda, Amazon API Gateway

Description: Integrate secure messaging and video consultations within the MyChart app using Amazon DynamoDB for data storage. Create an AWS Lambda function and Amazon API Gateway to capture and store user feedback, allowing for continuous improvement based on user feedback.

* Performance Monitoring and Optimization (BR14, TR14.1):

AWS Service: AWS CloudWatch, AWS Auto Scaling

Description: Implement performance monitoring and optimization using AWS CloudWatch for continuous tuning of system performance. Utilize AWS Auto Scaling to optimize computing resources.

* Tenant Identification and Data Segmentation (BR16, TR16.1, TR16.2):

AWS Service: Amazon Cognito, AWS Identity and Access Management (IAM)

Description: Implement a secure and scalable user authentication mechanism using Amazon Cognito to uniquely identify tenants within the MyChart application. Integrate a tenant-specific data segmentation strategy using AWS IAM to isolate and manage data belonging to different tenants effectively.

**4.2 Top-level, Informal Validation of the Design:**

* Cost Efficiency (TR1): The design incorporates automated cost tracking, detailed reports, and dashboards to ensure optimal cost utilization.
* Consistent Performance (TR2): Auto Scaling and dynamic resource scaling ensure efficient handling of varying workloads.
* 24/7 Availability (TR3): Disaster recovery strategies and continuous monitoring contribute to achieving 24/7 availability with minimal downtime.
* Security Compliance (TR4, TR9): The design integrates robust security measures, including monitoring, encryption, and access controls, to ensure compliance with industry standards like HIPAA.
* Vendor Agnosticism (TR5): The use of AWS Outposts supports a vendor-agnostic cloud strategy, promoting flexibility.
* Profit Maximization (TR6): Dynamic pricing, machine learning, and upselling strategies contribute to maximizing profits.
* Reduced Management Complexity (TR7): Infrastructure as code practices simplify administrative tasks and reduce management complexity

**4.3 Action items and rough timeline**

Section skipped

**5 The Second Design**

**5.1 Use of the Well-Architected framework** [12]

The Well-Architected Framework is a set of best practices for building secure, high-performing, resilient, and efficient infrastructure. It consists of six pillars: Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimization and Sustainability.Here are the distinct steps suggested by the Well-Architected Framework applied to the MyChart Application:

#### 5.1.1 Operational Excellence

* Best Practice: Implement infrastructure as code (IaC) using tools like Terraform (as mentioned in TR7.1) for automated resource provisioning, management, and versioning.
* Action: Regularly review and update IaC scripts to ensure consistency and eliminate manual errors during resource deployment.

#### 5.1.2 Security

* Best Practice: Leverage AWS Identity and Access Management (IAM) for secure access control and enforce the principle of least privilege (as mentioned in TR4.2).
* Action: Regularly audit and update IAM policies to ensure only necessary permissions are granted, enhancing the overall security posture.

#### 5.1.3 Reliability

* Best Practice: Establish a comprehensive disaster recovery plan (as mentioned in TR12.1) with data replication, backup, and failover strategies.
* Action: Conduct regular disaster recovery drills to validate the effectiveness of the plan and ensure a quick recovery in case of a system failure.

#### 5.1.4 Performance Efficiency

* Best Practice: Utilize content caching and distribution strategies (as mentioned in TR8.1) to reduce latency and enhance the overall user experience.
* Action: Continuously monitor and optimize the performance of the application to ensure responsiveness during varying workloads.

#### 5.1.5 Cost Optimization

* Best Practice: Implement automated cost tracking and analysis tools (as mentioned in TR1.1) to identify areas of cost optimization.
* Action: Regularly analyze cost reports and dashboards, adjusting resource allocations based on usage patterns to optimize operational costs. [7][10]

#### 5.1.6 Sustainability

* Implementation: Adoption of sustainable practices in resource management and data center operations.
* Action: Utilization of energy-efficient services, minimizing the environmental impact of the MyChart Application infrastructure.

[69][70]

**5.2 Discussion of pillars** [5]

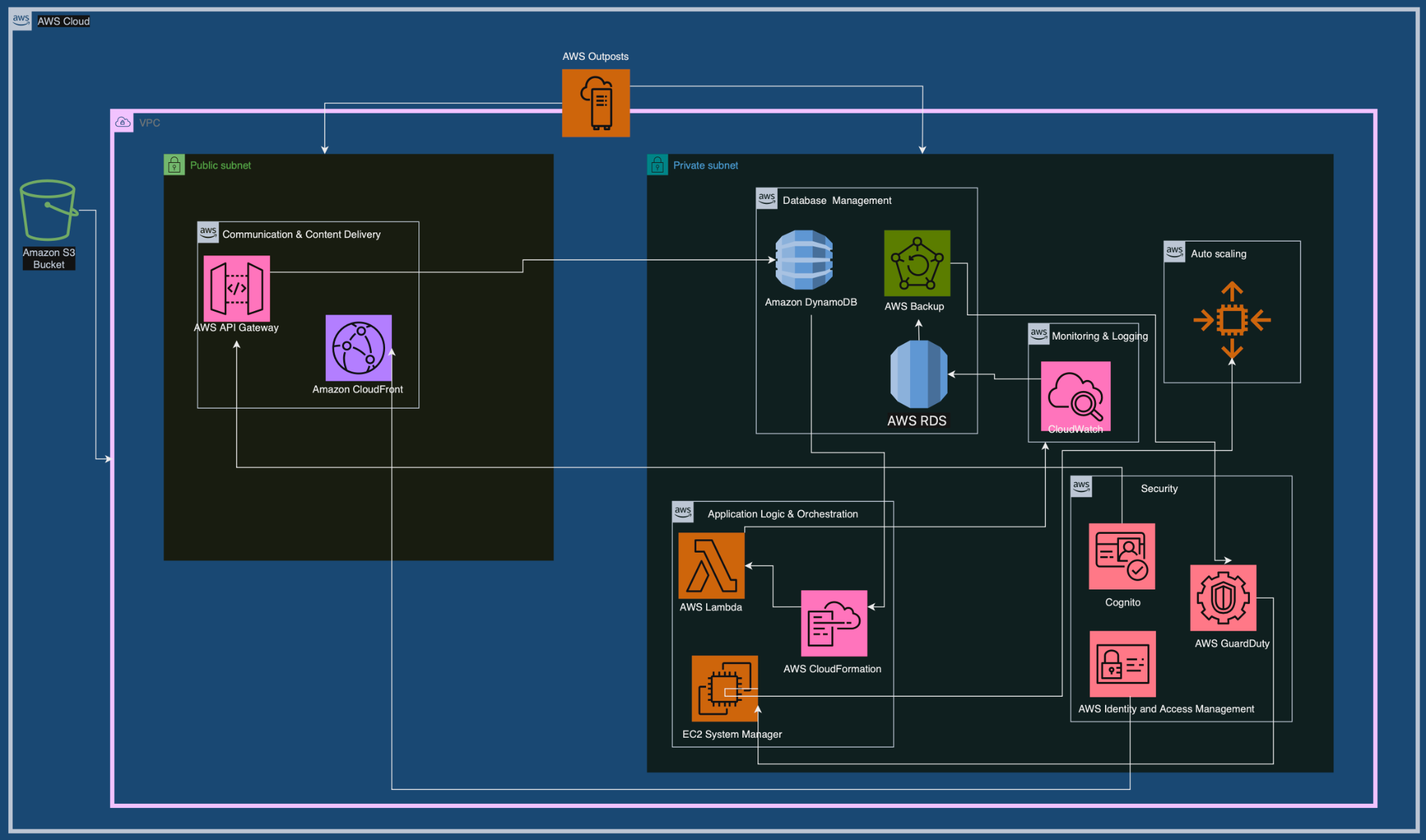
#### 5.2.1 Operational Excellence (Team Member 1 - Vatsal)

* The team member responsible for Operational Excellence focuses on implementing and maintaining infrastructure as code (IaC), ensuring automated resource provisioning and management. This aligns with the Well-Architected Framework's emphasis on operational efficiency and eliminating manual processes.

#### 5.2.2 Security (Team Member 2 - Ishika)

* The team member responsible for Security concentrates on enforcing secure access control through AWS IAM, aligning with the Security pillar of the Well-Architected Framework. Regular audits and updates to IAM policies contribute to maintaining a robust security posture.

**5.3 Use of Cloudformation diagrams** [64][65][69][70]



**5.4 Validation of the design**

* Optimize Operational Costs (BR1):

Validation Argument: The design meticulously incorporates automated cost tracking tools (TR1.1) and dynamic pricing models (TR6.1) that actively utilize metrics of resource utilization. This approach addresses the tradeoff between cost optimization and resource efficiency by ensuring that cost-saving measures do not compromise the application's performance.

* Accommodate Varying Workloads (BR2):

Validation Argument: The chosen dynamic scaling service (TR2.1) uses workload metrics, strategically addressing the tradeoff between responsiveness and cost efficiency. This ensures that resources scale in proportion to demand fluctuations while minimizing unnecessary costs during periods of lower demand.

* Ensure 24/7 Availability (BR3):

Validation Argument: The comprehensive disaster recovery plan (TR3.1) actively considers the tradeoff between availability and cost. By strategically implementing backup, data replication, and failover strategies, the design ensures continuous availability without compromising on operational efficiency or inflating costs unnecessarily.

* Ensure Security and Compliance (BR4):

Validation Argument: IAM for secure access control (TR4.1) aligns with the principle of least privilege, addressing the tradeoff between security and accessibility. Monitoring for security (TR4.2) actively uses relevant metrics, ensuring timely detection of unauthorized access attempts without sacrificing system performance.

* Avoid Vendor Lock-In (BR5):

Validation Argument: The design diligently follows a vendor-agnostic cloud strategy (TR5.1) with standardization and modular architecture, addressing the tradeoff between vendor flexibility and service optimization. This ensures that the chosen services are interchangeable without compromising functionality.

* Maximize Profit (BR6):

Validation Argument: The implementation of dynamic pricing models (TR6.1) and machine learning for upselling (TR6.2) addresses the tradeoff between profitability and user experience. These strategies maximize profit opportunities without compromising the overall user satisfaction or application performance.

* Reduce Management Complexity (BR7):

Validation Argument: Infrastructure as code (IaC) practices using Terraform (TR7.1) address the tradeoff between manual management complexity and automation. By automating resource provisioning and management, the design simplifies administrative tasks without introducing complexity in the IaC scripts.

* Ensure Low Latency (BR8):

Validation Argument: Content caching and distribution (TR8.1) actively use latency metrics to strike a balance between user experience and operational efficiency. By minimizing latency through these measures, the design ensures a responsive user experience without compromising on overall system efficiency.

* Ensure Data Privacy Compliance (BR9):

Validation Argument: The classification and protection of sensitive data (TR9.1) actively address the tradeoff between data privacy and accessibility. Continuous monitoring and security controls (TR9.2) ensure ongoing compliance without hindering data accessibility for authorized users.

* Enhance Interoperability (BR10):

Validation Argument: The implementation of HL7 FHIR standards (TR10.1) and API Gateway for data exchange (TR10.2) addresses the tradeoff between interoperability and system complexity. These standards facilitate seamless integration with external healthcare systems while maintaining a manageable level of complexity.

* Enable Multi-Platform Access (BR11):

Validation Argument: The development of mobile applications for iOS and Android (TR11.1) balances the tradeoff between platform diversity and development effort. This approach ensures a consistent user experience across multiple platforms without exponentially increasing development complexity.

* Establish Disaster Recovery (BR12):

Validation Argument: The disaster recovery plan with data replication and backup solutions (TR12.1, TR12.2) addresses the tradeoff between data security and system recovery time. By regularly testing and refining the plan, the design ensures a swift recovery without sacrificing the integrity of backed-up data.

* Improve Patient Engagement Features (BR13):

Validation Argument: The integration of secure messaging and video consultations (TR13.1) addresses the tradeoff between patient engagement and data security. By implementing secure communication channels, the design enhances patient engagement without compromising the confidentiality of healthcare interactions.

* Monitor and Tune Performance (BR14):

Validation Argument: Performance monitoring and optimization strategies (TR14.1) actively use performance metrics to address the tradeoff between system efficiency and user experience. This ensures continuous tuning of system performance while providing an optimal user experience.

* Incorporate a Feedback Mechanism (BR15):

Validation Argument: The feedback mechanism using AWS Lambda and Amazon API Gateway (TR15.1) addresses the tradeoff between user feedback integration and system overhead. By capturing and storing user feedback efficiently, the design enables continuous improvement without introducing significant operational complexity.

* Ensure Tenant Identification (BR16):

Validation Argument: Secure and scalable tenant identification mechanisms (TR16.1, TR16.2) address the tradeoff between tenant isolation and system efficiency. These mechanisms uniquely identify tenants without introducing unnecessary complexity or compromising system performance.

**5.5 Design principles and best practices used**

* Infrastructure as Code (IaC):

Proposed Implementation: Proposes the use of Terraform (TR7.1) and AWS CloudFormation for automating resource provisioning (BR7). This strategic choice aligns with the Infrastructure as Code principle, aiming to streamline infrastructure management and reduce manual complexity during the implementation phase.

* Security by Design:

Proposed Implementation: The design recommends the use of AWS IAM for secure access (TR4.1) and continuous monitoring (TR4.2). This aligns with the Security by Design principle, ensuring that during the implementation phase, the principle of least privilege is enforced, and unauthorized access attempts are promptly detected.

* Scalability:

Proposed Implementation: Dynamic scaling (TR2.1) to accommodate varying workloads (BR2). This aligns with the Scalability principle, aiming to address the tradeoff between responsiveness and cost efficiency during the implementation phase.

* Interoperability:

Proposed Implementation: Adopting HL7 FHIR standards (TR10.1) and API Gateway for data exchange (TR10.2). This aligns with the Interoperability principle, intending to facilitate seamless integration with external healthcare systems during the implementation phase.

* Continuous Improvement:

Proposed Implementation: Establishment of a feedback mechanism (TR15.1) using AWS Lambda and Amazon API Gateway. This aligns with the Continuous Improvement principle, ensuring that user feedback is captured and stored for continuous improvement during the implementation phase.

* Cost Optimization:

Proposed Implementation: Integrating automated cost tracking (TR1.1) and dynamic pricing models (TR6.1). This aligns with the Cost Optimization principle, aiming to identify areas of optimization and adjust resource allocation based on demand during the implementation phase.

* Vendor-Agnostic Cloud Strategy:

Proposed Implementation: The design recommends following a vendor-agnostic cloud strategy (TR5.1) with standardization. This aligns with the Flexibility principle, intending to mitigate vendor lock-in risks and provide flexibility during the implementation phase.

* Sustainability Measures:

Proposed Implementation: Incorporating sustainability practices in resource management and data center operations. This reflects the Sustainability principle, addressing the environmental impact of the MyChart Application during the implementation phase.

* Data Privacy Compliance:

Proposed Implementation: Classifying and protecting sensitive data (TR9.1) and implementing continuous monitoring (TR9.2). This aligns with the Data Privacy Compliance principle, ensuring adherence to data privacy standards, especially in the healthcare domain, during the implementation phase.

* Mobile Application Development:

Proposed Implementation: Development of mobile applications for iOS and Android (TR11.1). This follows the principle of User Experience Consistency, ensuring a uniform user experience across multiple platforms during the implementation phase.

* Disaster Recovery Best Practices:

Proposed Implementation: Establishing comprehensive disaster recovery plans (TR12.1) and templates for rapid deployment (TR12.2). This aligns with the Disaster Recovery Best Practices principle, ensuring data recovery and application failover capabilities during disasters during the implementation phase.

* Patient Engagement Features:

Proposed Implementation: Integrating secure messaging and video consultations (TR13.1). This addresses the Patient-Centric Design principle, aiming to enhance patient engagement and communication with healthcare providers during the implementation phase.

* Performance Monitoring and Optimization:

Proposed Implementation: Implementing performance monitoring and optimization strategies (TR14.1). This follows the Performance Efficiency principle, ensuring continuous tuning of system performance and optimal responsiveness during the implementation phase.

[66][67][68][69]

**5.6 Tradeoffs revisited** [1][5][Handouts 6.2]

**Operational Excellence vs. Cost Optimization:**

Comparing the enhancement in operational excellence achieved through AWS CloudFormation's infrastructure as code against the cost insights obtained from AWS Cost Explorer. The methodology involves quantifying the time saved in infrastructure management using CloudFormation against the potential cost savings identified through Cost Explorer's detailed reports. The decision point revolves around evaluating whether the time saved in provisioning and managing resources via CloudFormation outweighs the potential cost optimization insights gained from Cost Explorer. The outcome might justify the tradeoff if the time saved significantly enhances operational efficiency or if Cost Explorer's insights significantly impact cost savings, thereby justifying the time investment in manual resource management.

**Security vs. Performance Efficiency:**

Analyzing the impact of robust identity and access management (IAM) on security against the resource usage by CloudWatch for performance monitoring. The methodology includes quantifying the performance impact of IAM policies and role assignments against the resource utilization by CloudWatch monitoring for security checks. The decision point assesses whether the stringent IAM policies impact performance significantly more than the resource utilization by CloudWatch impacts system efficiency. The outcome might support the tradeoff if IAM policies marginally affect performance while providing substantial security or if CloudWatch's monitoring has minimal impact and provides critical performance insights, potentially justifying compromises on IAM stringency.

**Reliability vs. Operational Excellence:**

Evaluating the tradeoff between a comprehensive disaster recovery plan (AWS Backup, AWS RDS) ensuring reliability versus the focus on continuous improvement. The methodology involves quantifying the impact of dedicating resources to disaster recovery planning against the potential gains from agile continuous improvements. The decision point revolves around determining whether the resources dedicated to disaster recovery sufficiently impede continuous improvement efforts. The outcome might justify the tradeoff if the allocation to disaster recovery minimally affects agility, ensuring enhanced reliability, or if the robust disaster recovery plan significantly hampers iterative improvements, warranting a reevaluation of resource allocation.

**Performance Efficiency vs. Cost Optimization:**

Weighing the benefits of reduced latency and enhanced user experience through Amazon CloudFront against the potential cost insights provided by AWS Cost Explorer. The methodology involves quantifying the impact of reduced latency on user engagement against the potential cost savings identified by Cost Explorer. The decision point assesses if the improved user experience through CloudFront significantly justifies the potential cost insights foregone from Cost Explorer. The outcome might justify the tradeoff if improved user engagement via reduced latency is pivotal or if Cost Explorer's insights reveal substantial cost-saving opportunities, potentially reconsidering the emphasis on CloudFront's performance.

**Cost Optimization vs. Sustainability**:

Assessing the impact of leveraging AWS Lambda for cost optimization against investing in sustainable practices. The methodology involves quantifying the cost savings achieved through AWS Lambda's efficient resource utilization against the potential investments required for sustainability initiatives. The decision point evaluates if the immediate cost optimization gained from Lambda outweighs the long-term benefits and costs associated with sustainability efforts. The outcome might support prioritizing immediate cost optimization if the cost savings significantly impact the budget or if sustainability aligns with long-term goals despite initial investments, warranting reconsideration of resource allocation.

**Performance Efficiency vs. Reliability**:

Analyzing the benefits of AWS Auto Scaling for performance efficiency against the resources allocated for enhancing reliability measures. The methodology includes quantifying gains in resource optimization and performance through Auto Scaling against resources dedicated to reliability measures. The decision point assesses if gains in performance efficiency through Auto Scaling justify potentially compromising certain reliability enhancements. The outcome might support the tradeoff if Auto Scaling significantly optimizes resource usage without compromising reliability or if reliability enhancements are critical and cannot be compromised, necessitating reevaluation of the emphasis on performance efficiency.

**Security vs. Infrastructure Management:**

Weighing the benefits of AWS GuardDuty for security against resources allocated for infrastructure management using Systems Manager. The methodology involves quantifying the impact of GuardDuty in detecting security threats against resources required for managing infrastructure using Systems Manager. The decision point assesses if security enhancements from GuardDuty justify potential tradeoffs in resource allocation towards infrastructure management. The outcome might support the tradeoff if GuardDuty significantly strengthens security without overly impacting infrastructure management or if resources allocated to infrastructure management are crucial and cannot be compromised, necessitating reconsideration of the emphasis on GuardDuty.

**Flexibility vs. Standardization**:

Evaluating the flexibility gained through AWS Outposts against the benefits of standardized architectures. The methodology involves quantifying the degree of flexibility and adaptability achieved via Outposts against the efficiency and standardization benefits of a centralized architecture. The decision point assesses if the flexibility of Outposts justifies potential deviations from standardized architectures. The outcome might support the tradeoff if the adaptability provided by Outposts significantly meets specific needs or if adherence to standardized architectures ensures better management and control, warranting reevaluation of the use of Outposts.

**5.7 Discussion of an alternate design**

Skip

**6 Kubernetes experimentation [0%]**

**6.1 Experiment Design**

Here, we are trying to validate the design by choosing TR2 and TR14.

**TR14: Monitor the performance of resources**

Here, the workload performance is monitored in terms of Total requests per sec, Response time, No of users, No of request failures using locust and minikube. The details of monitoring are explained in brief in section 6.2

**TR2 : Scaling of Workloads**

The architecture should elastically add or remove resources to match the current demand at any given time based on the workload that the users generate.

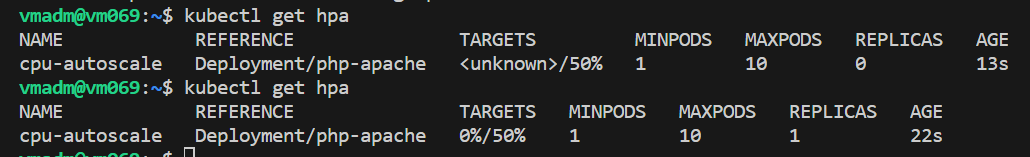
Kubernetes provides a feature called Horizontal Pod Autoscaling (HPA)[62] that allows to automatically adjust the number of running pods in a deployment or replica set based on observed CPU utilization or custom metrics. kubectl is the command-line tool used to interact with and manage resources in a Kubernetes cluster, including autoscaling.

Using kubectl autoscale, we will create the autoscaler.

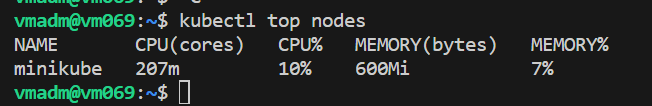
We create a Horizontal Pod Autoscaler (HPA) that maintains between 1 and 10 replicas of the Pods controlled by the php-apache deployment

Via the deployment, HPA will increase and decrease the number of replicas to maintain an average CPU utilization across all Pods. By combining kubectl with Horizontal Pod Autoscaling, it can efficiently manage the resources of applications in a dynamic and automated manner based on demand.

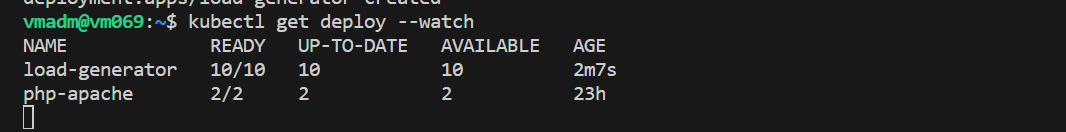
**HPA deployment**



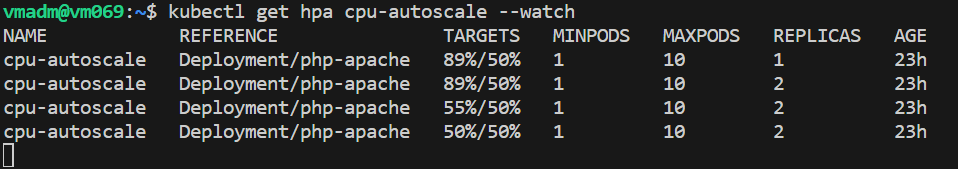
Note that the current CPU consumption is 0% (above screenshot) as there are no clients sending requests to the server and the current replica count is one which was created during the deployment.



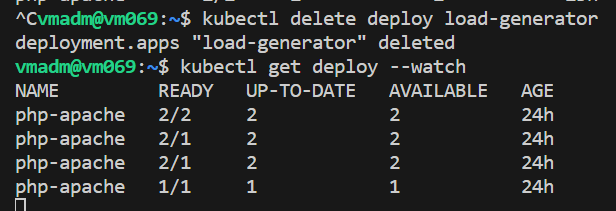
We can see initially the php-apache deployment is currently running on one pod and there is no traffic on it. We deployed a load generator using load.yaml file

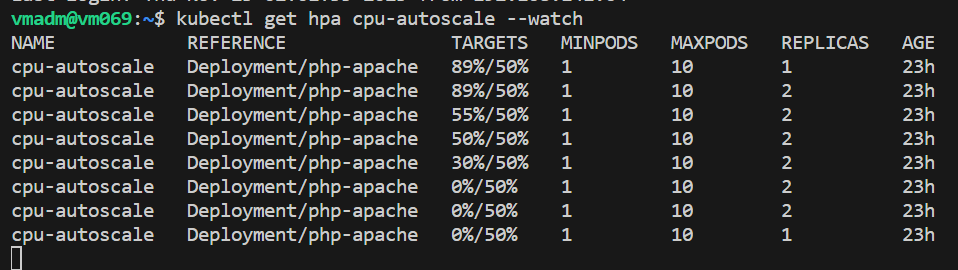


Initially, when there is no load the Kubernetes tries to maintain the number of replica containers as specified in the deployment (1 in this case). Once the load generator deployment is initiated, the load generator creates traffic to the application server running in the pod. We can see that the load gradually increases



We can observe after a while, the load spikes. The autoscaler algorithm kicks in and increases the number of current replicas to 2. Similarly, when the load is removed it automatically removes the running container runtimes.

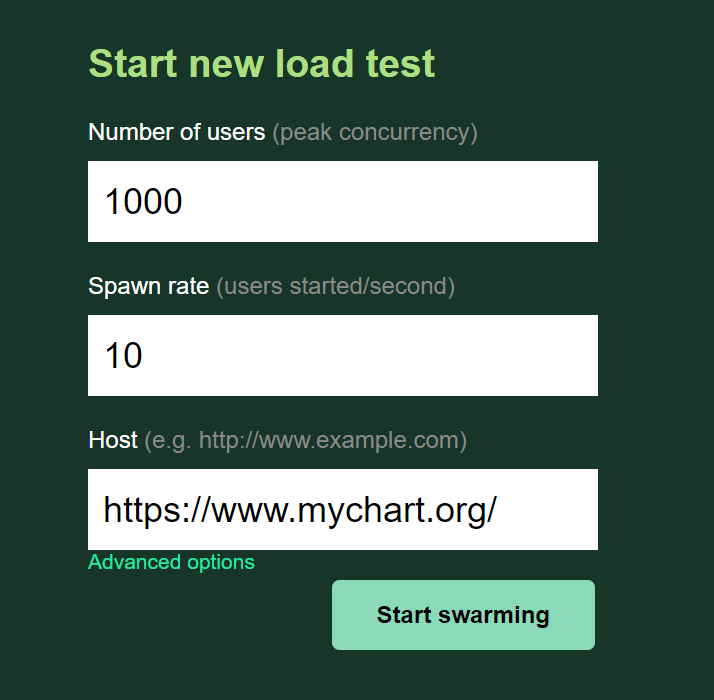




**6.2 Workload generation with Locust**

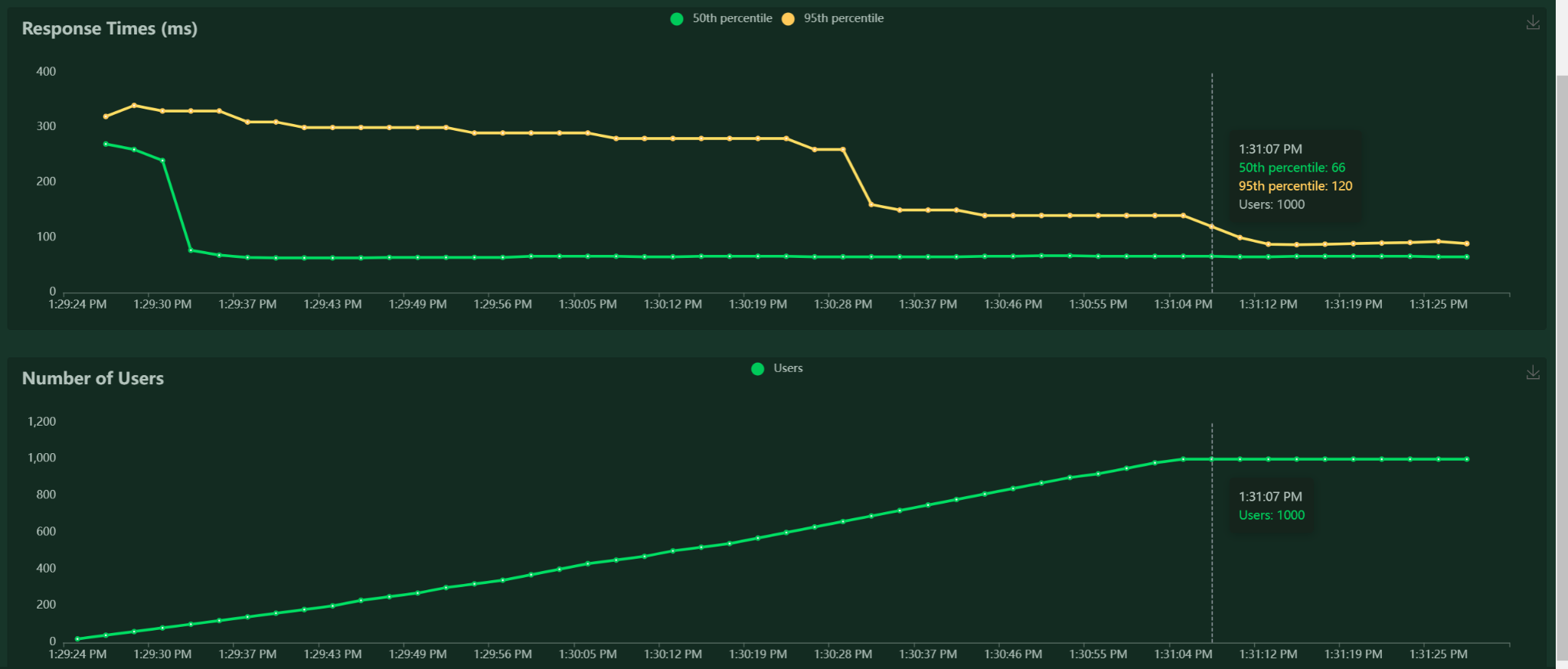
Locust is an open-source Python-based tool for load testing and stress testing applications. It allows to simulate concurrent user activity to evaluate the performance and scalability of the application[63]

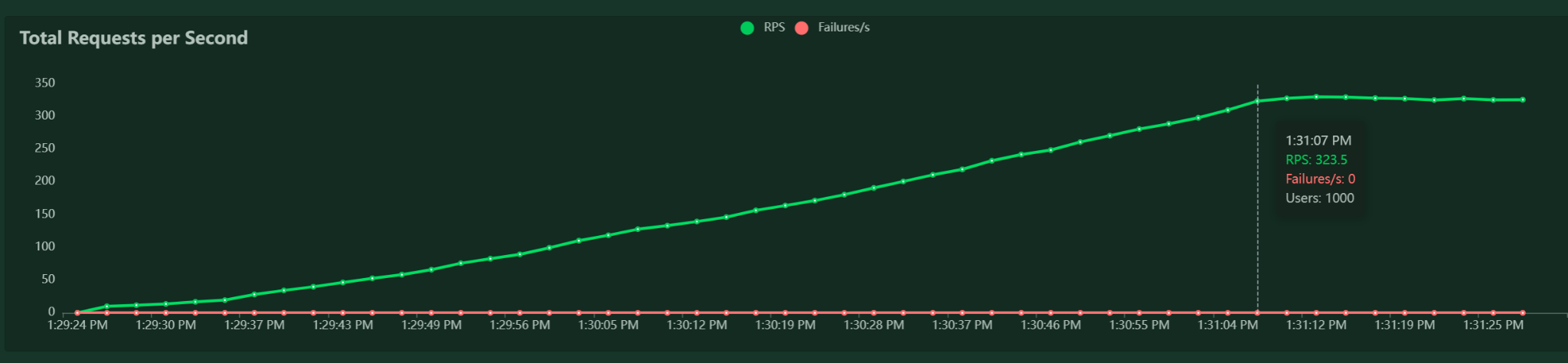
The below are the input parameters for the locust load generation

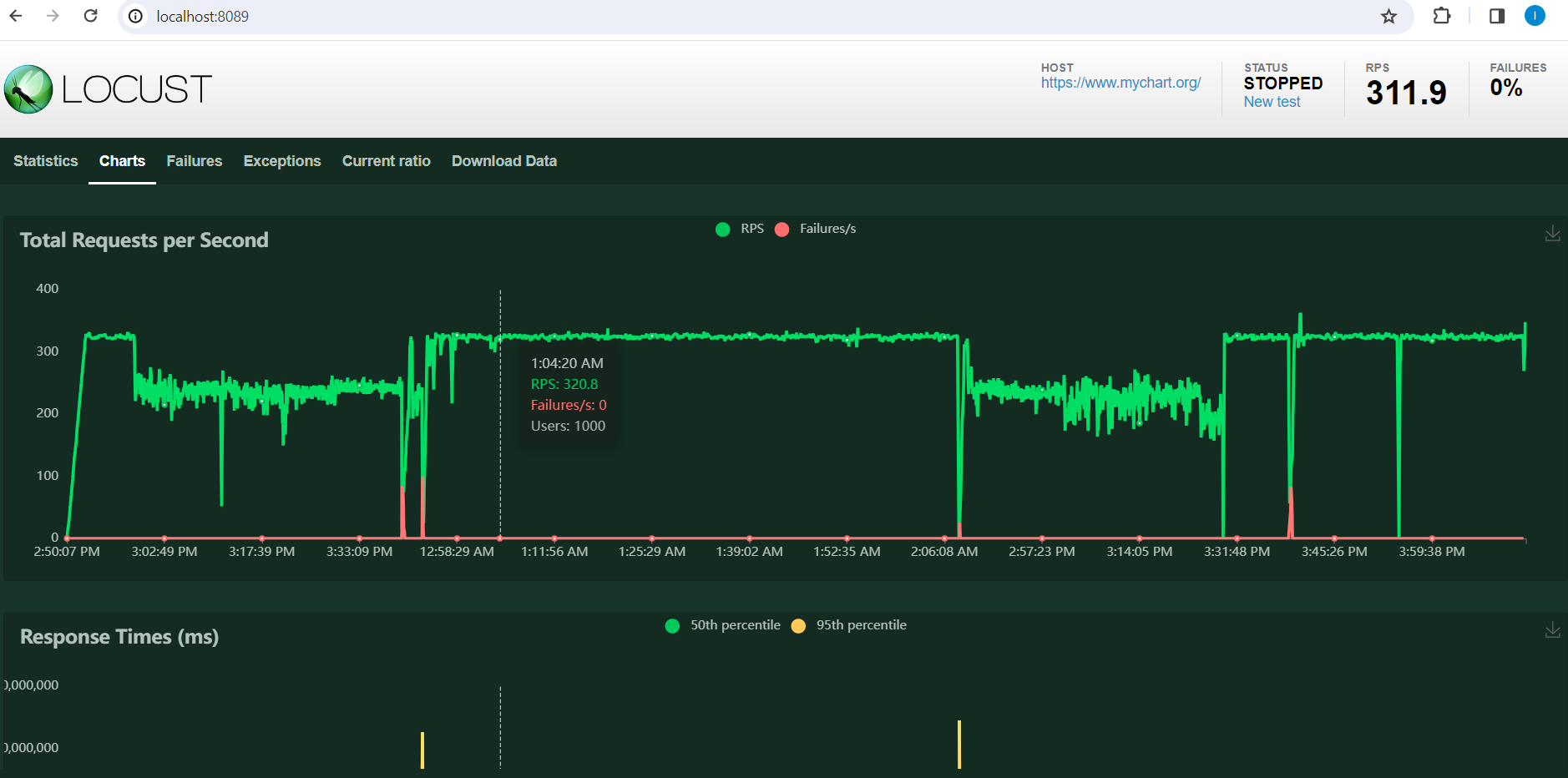


We generate three below graphs as follows:

* Response Time - This graph depicts the response time of the application over time. It shows how quickly the application responds to requests from the simulated users. As the load increases, we observe changes in the response time. We see that as the
* Number of users - With 1000 users and a spawn rate of 10, we saw a gradual increase in the number of users as they are spawned over time. The slope of the graph represents the rate at which users are being added. After 1000 users the graph is a straight line
* Total Requests per Second - This graph represents the total number of requests made to the application per second. As the number of users increases, can see an increase in the total requests per second. It provides a view of the overall load on the application infrastructure.







**6.3 Analysis of the results**

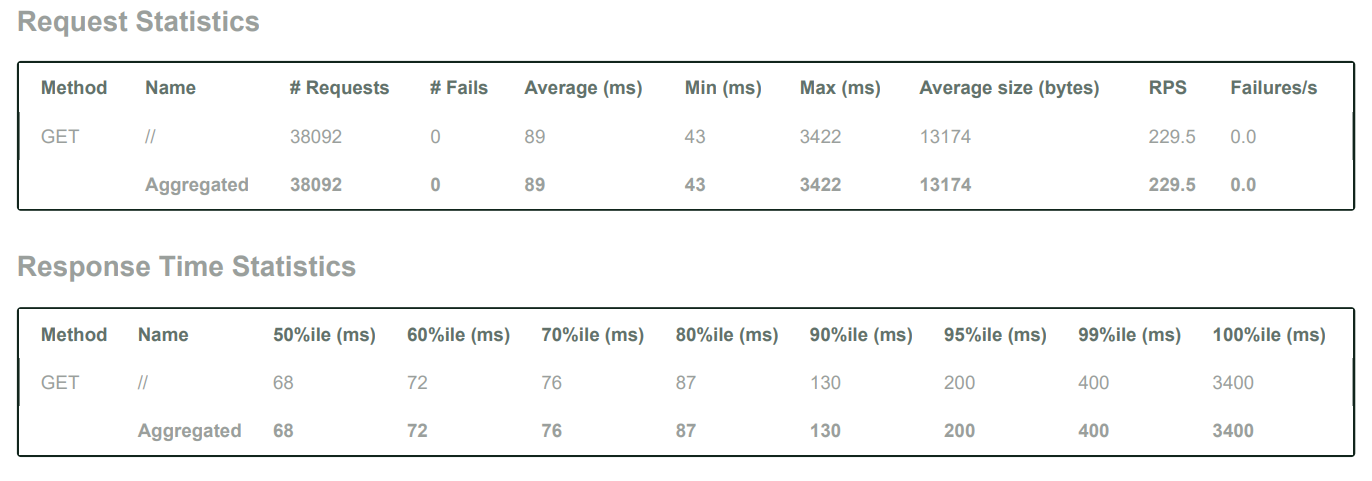
The outputs observed validate the design by showcasing the effective functioning of the Horizontal Pod Autoscaler (HPA) in response to workload changes.

Initially, the php-apache deployment starts with one pod, maintaining minimal CPU utilization due to no incoming traffic. Upon load generation, gradual traffic increase raises CPU utilization. Subsequently, the HPA reacts, dynamically scaling replicas to 2 as CPU usage surpasses the set threshold, exemplifying the system's adaptability. When load diminishes, CPU usage decreases, prompting the HPA to scale down replicas, efficiently optimizing resource allocation in response to reduced demand, solidifying the autoscaler's capability.

These outputs validate the design in several ways:

* Elasticity: The system shows elasticity by seamlessly scaling up and down based on observed workload changes. This flexibility ensures efficient resource utilization.
* Automated Response: The HPA's automatic adjustment of pod replicas showcases the effectiveness of automated resource management. It eliminates the need for manual intervention and ensures that resources match the current demand.
* Optimized Resource Utilization: By adjusting the number of replicas based on CPU utilization, the design optimizes resource allocation. It ensures that there are enough pods to handle the workload while avoiding unnecessary over-provisioning during periods of low demand.

The below are the output statistics for the locust load generation:



* Consistent Performance: The low failure rate, along with consistent response times for most requests, validates the system's ability to handle the imposed load efficiently.
* Response Time Distribution: The majority of requests are handled within an acceptable response time (up to 400 milliseconds), aligning with expectations for good user experience.
* Scalability: The system seems to maintain performance even under high load (as evidenced by the 99th percentile response time), validating its scalability and ability to handle increased traffic.

Overall, The observed outputs validate the effectiveness and efficiency of the design in managing resources dynamically, meeting demands effectively, and optimizing resource usage in a Kubernetes environment through Horizontal Pod Autoscaling. This report also indicates that the system adeptly handles the test's load, upholds a minimal failure rate, and delivers timely responses for most requests. These findings align with the anticipated performance outlined in the system's design, affirming its preparedness to effectively manage analogous real-world workloads.

**7 Ansible playbooks [0%]**

7.1 Description of management tasks

Skip

7.2 Playbook Design

Skip

7.3 Experiment runs

Skip

**8 Demonstration [0%]**

Skip

**9 Comparisons [0%]**

Skip

**10 Conclusion [0%]**

10.1 The lessons learned

Our project involving the MyChart application's documentation was a deep dive into the intricate web of cloud architecture, healthcare technology, and strategic decision-making. Capturing the nuances of a comprehensive patient portal like MyChart was both challenging and immensely intriguing. We delved into multiple facets, from dissecting the motivations behind such a platform to mapping out its technical requirements, provider selection, and the nitty-gritty of architectural design. Each section demanded an understanding not just of technological capabilities but also the balance between user needs, compliance, and operational efficiency.

The core of the challenge was unraveling the complexities inherent in healthcare systems while aligning them with the possibilities offered by cloud technologies. Decisions about tradeoffs between operational excellence and cost optimization, security and performance efficiency, or even reliability and continuous improvement weren't just theoretical. They required a meticulous evaluation of benefits against potential drawbacks and a constant juggle to find the optimal balance. Making these tradeoffs was like navigating through a maze, where each choice had side effects across various aspects of the application, impacting its functionality, security, and user experience.

However, among these intricate decisions, the creation of the CloudFormation diagram was an interesting and fulfilling task. Visualizing the entire architecture in a diagram provided a holistic perspective that consolidated all the components, connections, and interactions within the MyChart application. Additionally, the exhaustive comparison of various cloud providers and their services, although necessary for informed decision-making, might have felt monotonous at times. Analyzing the offerings of AWS, Azure, and Google Cloud Platform to make the final selection based on the application's needs involved a meticulous evaluation of each provider's capabilities. This process, while essential, could have been a bit repetitive, examining similar services across different platforms to identify the best fit for the MyChart application's requirements.

Through this project, we learned the art of balancing intricate technical decisions with user-centric healthcare needs, ensuring a seamless blend of innovation, compliance, and efficiency in the MyChart application. We also learned the implementation of Kubernetes and Locust to gain insights of the technical requirements.

10.2 Possible continuation of the project

Skip

**11 References**

1. YV and YP, “ECE547/CSC547 class notes”.
2. <https://www.mychart.org/>
3. <https://www.mychart.org/Help?topic=faq>
4. <https://docs.aws.amazon.com/>
5. <https://chat.openai.com/>
6. <https://docs.aws.amazon.com/ec2/?icmpid=docs_homepage_featuredsvcs>
7. <https://aws.amazon.com/aws-cost-management/>
8. <https://learn.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling>
9. <https://aws.amazon.com/autoscaling/>
10. <https://www.tierpoint.com/services/cloud-services-solutions/public-cloud/managed-azure/>
11. <https://cloud.google.com/solutions/high-availability>
12. <https://aws.amazon.com/architecture/well-architected/>
13. <https://aws.amazon.com/compliance/shared-responsibility-model/>
14. <https://services.google.com/fh/files/misc/gcp_pci_srm__apr_2019.pdf>
15. <https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility>
16. <https://aws.amazon.com/s3/security/>
17. <https://jayendrapatil.com/google-cloud-storage-security/>
18. [https://learn.microsoft.com/en-us/azure/storage/common/storage-service-encryption?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json](https://learn.microsoft.com/en-us/azure/storage/common/storage-service-encryption?toc=/azure/storage/blobs/toc.json&bc=/azure/storage/blobs/breadcrumb/toc.json)
19. <https://aws.amazon.com/sustainability/resources/?refid=b53e5260-30f6-45cf-9610-30be63969231>
20. <https://cloud.google.com/sustainability>
21. [https://azure.microsoft.com/en-us/explore/global-infrastructure/sustainability/#carbon-benefits](https://azure.microsoft.com/en-us/explore/global-infrastructure/sustainability/%23carbon-benefits)
22. <https://aws.amazon.com/elasticloadbalancing/?nc=sn&loc=1>
23. <https://aws.amazon.com/cloudwatch/>
24. <https://www.simform.com/blog/aws-lambda-vs-ec2/>
25. <https://disaster-recovery.workshop.aws/en/services/databases/dynamodb.html>
26. <https://aws.amazon.com/iam/>
27. <https://aws.amazon.com/cloudwatch/>
28. <https://aws.amazon.com/autoscaling/>
29. <https://aws.amazon.com/elasticloadbalancing/application-load-balancer/>
30. <https://docs.aws.amazon.com/lambda/latest/dg/security-resilience.html>
31. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
32. <https://aws.amazon.com/dynamodb/dax/>
33. <https://aws.amazon.com/sqs/features/>
34. <https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/>
35. <https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/security.html>
36. <https://docs.aws.amazon.com/vpc/latest/userguide/security.html>
37. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html>
38. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucket-encryption.html>
39. <https://www.tinystacks.com/blog-post/battle-of-the-serverless-api-routers-alb-vs-api-gateway-feature-comparison/>
40. <https://docs.aws.amazon.com/wellarchitected/latest/framework/rel-workload-arch.html>
41. <https://docs.aws.amazon.com/wellarchitected/latest/framework/rel-failmgmt.html>
42. <https://docs.aws.amazon.com/wellarchitected/latest/framework/perf-sel.html>
43. <https://docs.aws.amazon.com/wellarchitected/latest/framework/perf-monitoring.html>
44. <https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-aware.html>
45. <https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-cereso.html>
46. <https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-iam.html>
47. <https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-infrastructure.html>
48. <https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-dataprot.html>
49. <https://docs.aws.amazon.com/wellarchitected/latest/framework/sus-user-behavior-patterns.html>
50. <https://cloud.google.com/compute/docs>
51. <https://docs.gcp.databricks.com/index.html>
52. <https://cloudindustryforum.org/8-criteria-to-ensure-you-select-the-right-cloud-service-provider/>
53. <https://www.digitalocean.com/resources/article/comparing-aws-azure-gcp>
54. <https://cloud.google.com/docs/get-started/aws-azure-gcp-service-comparison>
55. <https://www.hhs.gov/hipaa/index.html>
56. <https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/>
57. <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code?in=terraform%2Faws-get-started>
58. <https://www.cloudflare.com/application-services/products/cdn/>
59. <https://aws.amazon.com/cloudfront/>
60. <https://www.ansible.com/overview/how-ansible-works>
61. <https://legal.thomsonreuters.com/en/insights/articles/understanding-data-privacy-and-cloud-computing>
62. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale>/
63. <https://docs.locust.io/en/stable/>
64. <https://app.diagrams.net/>
65. <https://aws.amazon.com/cloudformation/resources/templates/>
66. <https://www.splunk.com/en_us/form/the-state-of-cloud-adoption-digital-transformation.html>
67. <https://www.linkedin.com/pulse/6-cloud-design-principles-successful-environment-sandesh-segu/>
68. <https://docs.aws.amazon.com/wellarchitected/latest/framework/oe-design-principles.html>
69. <https://mychart.hmhn.org/MyChart/quickstart.pdf>
70. <https://mychartonline.umassmemorial.org/mychart/myChart_FAQs.pdf>