CSC 547 Cloud Architecture

Fall 2023

Yannis Viniotis and Loannis Papapanagiotou

Soham Suhas Patil

(200538641)

Vanshika Singh

(200537463)

“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.”

Table of Contents

1.Introduction

1.1. Motivation:

As active users of diverse banking services, we've witnessed the transformation towards a cashless ecosystem, where billions of transactions simplify our lives. This project serves as our inspiration to explore the intricate world of these daily transactions. We're determined to utilize advanced cloud technologies, aiming not only to enhance the banking experience but also to ensure it's secure, convenient, and future-ready.We focus to leverage different cloud technologies to enhance the banking experience.

1.2 Executive summary

This project aims to develop a comprehensive banking application. The primary objectives are to create a secure, user-friendly, and fully functional digital banking platform that adheres to stringent regulatory standards. Key challenges include ensuring data security, optimizing user experience, maintaining compliance, integrating with existing systems, achieving scalability, enhancing performance, and delivering cross-platform compatibility. Efficient customer support and innovative features are integral to maintaining a competitive edge in the dynamic digital banking landscape.

2. Problem Description

2.1 The Problem

Developing a banking application involves addressing critical issues such as security and data protection, user experience, regulatory compliance, integration, scalability, performance, cross-platform compatibility, customer support, and maintaining a competitive edge. Solving these challenges is essential for creating a successful and competitive digital banking solution.

2.2 Business Requirements

- **BR1:** Application should be highly secure.
- **BR2:** The banking application must be available 24/7.
- **BR3:** Optimize costs.
- **BR4:** Accommodate varying traffic over time for bank transactions being processed.
- **BR5:** Ensure the system responds quickly and stays operational. System should be up.
- **BR6:** Develop a robust disaster recovery plan for business continuity.
- **BR7:** Implement robust customer support and communication features.
- **BR8:** Regularly back up data and ensure recoverability.
- **BR9:** Ensure user privacy.
- **BR10:** All operations must follow industry accepted rules and regulations.

2.3 Technical Requirements

Technical requirement 1

- **BR1 :** *Application should be highly secure.*
    - **TR 1.1** Employ robust encryption methods for data at rest and in transit, along with data masking and tokenization techniques to protect sensitive customer and financial information. ENCRYPTION
    - **TR1.2** Implement rigorous access controls and identity and access management (IAM) to ensure that only authorized personnel can access sensitive banking systems and data. ACCESS CONTROL
    - **TR1.3** Choose a scalable relational database management service so as to maintain the data of users in a correct and scalable manner. RDS SELECTION

Technical requirement 2

- **BR2:** *The banking application must be available 24/7.*
    - **TR2.1** Design a robust, high-availability architecture with redundancy and load balancing to minimize single points of failure. DESIGN OF ARCHITECTURE.
    - **TR2.2** Implement auto-scaling to dynamically adjust resources based on traffic, ensuring that the application can handle varying workloads effectively. AUTO SCALING
    - **TR2.3** Implement strong security measures, including firewalls and encryption, to protect against potential threats and vulnerabilities. FIREWALLS

Technical requirement 3

- **BR3:** *Optimize costs.*
    - **TR 3.1** Optimize the resources. Continuously monitor and optimize the use of computing resources, ensuring that you only pay for what you need and efficiently scale resources as demand fluctuates. MONITORING OF RESOURCES.
    - **TR 3.2** We will analyze the usage of resources. Regularly analyze resource usage to identify underutilized or idle resources and take action to reduce or terminate them. RESOURCE USAGE CONTROL.

Technical requirement 4

- **BR4:** *Accommodate varying traffic over time for transactions being processed.*
    - **TR 4.1** Use load balancing to distribute incoming transactions evenly across multiple servers, preventing overload on any single server. LOAD BALANCING

- **TR 4.2** Utilize message queues or transaction processing systems to handle bursts of incoming transactions and manage the order and pace at which they are processed.

Technical requirement 5

- **BR5:** *Ensure the system responds quickly and stays operational. System should be up.*
  - **TR5.1:** Determine the expected scalability of the cloud services to handle traffic spikes. Outline performance requirements during peak loads. SCALABILITY
  - **TR 5.2** Set up real-time monitoring of logs to detect and respond promptly to security incidents or unusual system activities. LOG MONITORING

Technical requirement 6

- **BR6:** *Develop a robust disaster recovery plan for business continuity.*
  - **TR 6.1** Implement automated backup and restore procedures for critical data and systems, including regular testing of data recovery. DATA BACKUPS
  - **TR 6.2** Setup comprehensive performance monitoring and alerting to track the application's health and adjust resources as needed. MONITORING

Technical requirement 7

- **BR7:** *Keep our systems safe.*
  - **TR7.1:** Regularly update and patch all software components, including the operating system, web server, application server, and libraries, to address known security vulnerabilities. MAINTENANCE

Technical requirement 8

- **BR8:** *Regularly back up data and ensure recoverability.*
  - **TR8.1:** Implement a combination of full backups and incremental backups to balance storage and backup time. Full backups capture all data, while incremental backups save changes made since the last backup. DATA RECOVERY

Technical requirement 9

- **BR9:** *Ensure user privacy.*
  - **TR 9.1:** Allow users to easily access, export, and delete their personal data. Implement mechanisms to fulfill user requests for data portability and deletion in compliance with privacy regulations. DATA ACCESS and CONTROL

Technical requirement 10

- **BR10:** *All operations must follow industry accepted rules and regulations.*
    - **TR 10.1** Establish an incident response plan that aligns with regulatory requirements, outlining steps to take in the event of a security incident to comply with breach notification and reporting obligations. INCIDENT RESPONSE

Justifications :

Business Requirements	**Associated TR's**	**Justifications**
BR1	**TR1.1, TR 1.2, TR 10.1**	• **TR1.1:** Implements robust encryption methods, data masking, and tokenization for data security for banking. • **TR1.2:** Enforces rigorous access controls and IAM to restrict authorized personnel access to sensitive banking systems. • **TR10.1:** Establishes an incident response plan aligning with regulations, ensuring compliance with breach notification obligations for enhanced security.
BR2	**TR2.1 , TR2.2, TR7.1**	• **TR2.1**: Designs a resilient architecture with redundancy and load balancing for uninterrupted 24/7 availability, minimizing potential single points of failure. • **TR2.2:** Implements auto-scaling to adapt resources dynamically, ensuring the banking application can efficiently handle fluctuating workloads and maintain continuous availability. • **TR7.1:** Enforces regular software updates and patches across all components, addressing known vulnerabilities to maintain the robustness and security of the banking application's 24/7 availability.
BR3	**TR3.1, TR3.2**	• **TR3.1**: Implements continuous monitoring and

		optimization of computing resources, ensuring cost-effectiveness by dynamically scaling resources based on demand, aligning with the optimization goal. ● **TR3.2:** Regularly analyzes resource usage, identifying and mitigating underutilized or idle resources, thereby controlling costs through efficient resource management in alignment with the optimization objective.
BR4	**TR4.1, TR4.2**	● **TR4.1:** Implements load balancing to evenly distribute incoming transactions across multiple servers, preventing overload on any single server and accommodating varying traffic over time. ● **TR4.2:** Utilizes message queues or transaction processing systems to handle bursts of incoming transactions, managing their order and pace, ensuring efficient processing and meeting the requirement for accommodating varying transaction traffic.
BR5	**TR5.1, TR5.2**	● **TR5.1:** Ensures quick system response and operational stability by determining cloud service scalability for handling traffic spikes and outlining peak load performance requirements. ● **TR5.2:** Establishes real-time log monitoring to promptly detect and respond to security incidents or unusual system activities, contributing to the system's responsiveness and continuous operation.
BR6	**TR6.1, TR6.2, TR1.3**	● **TR6.1:** Establishes a robust disaster recovery plan by implementing automated backup and restore procedures for critical data and systems, ensuring business continuity through regular testing of data recovery. ● **TR6.2:** Sets up comprehensive performance monitoring and alerting to track the application's health, enabling resource adjustments as needed for effective disaster recovery and business continuity. ● **TR1.3:** Selects a scalable relational database management service, ensuring correct and scalable data maintenance for user information as part of the disaster recovery and business continuity strategy.

BR7	**TR7.1**	• **TR7.1:** Ensures system safety by enforcing regular updates and patches across all software components, addressing known security vulnerabilities through consistent maintenance practices.
BR8	**TR8.1, TR1.3**	• **TR8.1:** Enhances data recoverability by implementing a combination of full and incremental backups, balancing storage efficiency and backup time for comprehensive data protection. • **TR1.3:** Selects a scalable relational database management service, ensuring correct and scalable data maintenance for users and contributing to the overall data backup and recovery strategy.
BR9	**TR9.1**	• **TR9.1:** Safeguards user privacy by allowing easy access, export, and deletion of personal data, implementing mechanisms to fulfill user requests for data portability and deletion in compliance with privacy regulations, ensuring robust data control.
BR10	**TR10.1**	• **TR10.1**: Ensures compliance with industry regulations by establishing an incident response plan aligned with regulatory requirements, specifying steps to address security incidents and comply with breach notification and reporting obligations.

2.4 Tradeoffs

1. **Security vs. Accessibility:**

- Trade-off: Cloud banking applications must balance the need for robust security with user accessibility. Strong security measures can sometimes hinder ease of access for customers and employees.

  In this case, we will have the ***TR1.1 and TR1.2 are conflicting*** since TR1.2 which involves access control can be considered a tradeoff since TR 1.1 maintains encryption and security.

2. **Cost vs. Performance:**

- Trade-off: Increasing performance often leads to higher costs. Banking applications need to deliver low-latency responses and high availability, which can be expensive in the cloud.

In this case, we have ***TR3.2 and TR5.1 in conflict***. We will consider cost to be a tradeoff as maintaining low-latency and availability is crucial for usage of banking applications which involves traversal of funds. It is also important for users to have a smooth performance.

3. Compliance vs. Innovation:

- Trade-off: Strict regulatory compliance requirements in the banking industry can slow down innovation and deployment of new features.

Here, we have ***conflict in TR9.1 and TR5.1*** .We know, banking applications work on strict compliance, and hence, we might have innovation as a tradeoff here, as we cannot risk being non-compliant since we are handling other people's funds.

4. Availability vs. Redundancy:

- Trade-off: Ensuring high availability often requires redundancy, which can increase operational complexity and costs.

TR 3.1 and TR4.2 are conflicting here. Availability has been a priority business requirement in our project, and we can trade off redundancy in order to maintain that. Banking applications need to be available 24/7.

5. Access Control vs. User Convenience:

- Trade-off: Stringent access controls enhance security but may hinder user convenience.

A more relaxed approach may improve convenience but compromise security. Hence, ***TR1.2 and TR9.1 have conflict.***

6. Data Security vs. Performance:

- Trade-off: Balancing data security through encryption with the potential performance impact due to encryption/decryption processes.

Conflicts with TR1.1 and 2.1. For this scenario, we will have to trade-off the performance that is impacting data-security, as we have to apply encryption in the banking application for the data in rest as well as data in transaction so as to maintain the security of our application from attacks.

3. Provider Selection

3.1 Criteria for choosing a provider for cloud.

Following are the criterias that are being considered for choosing a cloud service provider:

1. **Security and Compliance**: Ensure the cloud provider adheres to stringent security standards and compliance requirements, especially in the highly regulated banking industry. Look for certifications such as ISO 27001, SOC 2, and compliance with industry-specific regulations like PCI DSS for payment card data.

2. **Scalability and Performance**: Evaluate the scalability and performance capabilities of the cloud infrastructure to accommodate the varying workloads and transaction volumes typical in banking applications. Look for features like auto-scaling, load balancing, and high-performance computing options.

3. **Disaster Recovery and Business Continuity**: Ensure the cloud provider offers robust disaster recovery and business continuity services, including automated backup procedures, geo-redundancy, and data recovery mechanisms. A reliable provider should have a well-documented and tested disaster recovery plan.

4. **Cost**: Assess the overall cost structure, including pricing models, transparency, and potential hidden costs. Opt for a provider with a flexible pricing model that aligns with your usage patterns and growth, allowing you to optimize costs effectively.

5. **Services Offered**: Consider the range of services offered by the cloud provider, such as database management, networking, analytics, and machine learning. Choose a provider that offers a comprehensive suite of services to meet the diverse needs of a banking application, enabling seamless integration and future scalability.

3.2 Provider Comparison

The following table provides a comparison of the “big three” cloud providers and their rankings according to the above criteria:

Criteria	Amazon Web Services (AWS)	Microsoft Azure	Google Cloud Computing
Cost	Offers various pricing models, such as pay-as-you-go and reserved instances. Offers free trials, 12-months free and always free on different services.	Offers flexible pricing options, including pay-as-you-go and reserved instances. Azure Pricing. Offers some services for 12-months free and 55+ services for free always.	Offers only a handful of 20+ products as always free.
Services Offered	Extensive range of over 200 fully-featured services and the largest among all three.	Offers a comprehensive suite of over 200 services and continues to expand but just below AWS.	Features a wide array of over 160 services and continues to grow and the smallest among all three.
Critical acclaim	AWS holds the largest market share among the big three and serves many huge organizations like Formula 1, Airbnb, Netflix, etc.	Microsoft Azure holds the second largest market share and has customers like Intel, Adobe, Verizon, etc.	Has the least market share and has clients like Twitch, LinkedIn, etc.
Security	Complies with various industry standards and regulations, including GDPR, HIPAA, and ISO.	Compliant with a wide range of industry standards and regulations, including GDPR, HIPAA, and ISO.	Complies with industry standards and regulations, including GDPR, HIPAA, and ISO. Offers a variety of compliance certifications.
Locations	With plans to add 15 more Availability Zones and 5 more AWS regions, the AWS Cloud currently spans 102 Availability Zones across 32 geographic regions worldwide.	Azure has over 60 regions across the globe along with more than 100 availability zones.	GCP currently spans across 39 regions and 118 availability zones.

Justification of rankings:

1. **Amazon Web Services:** With over 200 fully-featured services, AWS provides a comprehensive platform for diverse use cases. Its global infrastructure ensures low-latency access and high availability. As a market leader, AWS boasts a large and active community, mature ecosystem, and proven reliability. The scalability, security features, and compliance certifications further contribute to its appeal. AWS's commitment to innovation, continuous improvement, and a track record of supporting customer success stories make it a compelling choice. The wealth of educational resources and documentation facilitates effective utilization of AWS services, aligning with your organization's goals and requirements.

2. **Microsoft Azure:** Ranking Microsoft Azure below AWS is justified based on several factors. While Azure offers a comprehensive suite of services and is a strong player in the cloud market, AWS often takes the lead in terms of service maturity, market share, and global infrastructure. AWS's extensive global presence and longer history contribute to lower-latency access and greater geographical coverage. The perceived performance and reliability of AWS, coupled with a reputation for innovation and early adoption of emerging technologies, make it a preferred choice for organizations with demanding workloads. AWS's larger market share translates into a more mature ecosystem, a robust community, and a broader range of third-party integrations.

3. **Google Cloud Platform (GCP):** Ranking Google Cloud Platform (GCP) below AWS and Microsoft Azure can be justified based on several factors. While GCP provides a diverse range of services and excels in areas like Kubernetes and machine learning, AWS and Azure lead in terms of service maturity, market share, and global infrastructure. AWS and Azure, being more established and widely adopted by enterprises, offer a broader and more mature ecosystem with extensive documentation and a larger community. The global reach of AWS and Azure is often more comprehensive, and their market dominance contributes to a higher level of trust. Additionally, AWS and Azure are recognized for their innovation, early adoption of emerging technologies, and a richer set of security and compliance features.

3.3 The final selection

For our application we have decided to **choose Amazon Web Services (AWS)** as our cloud service provider since AWS offers many services that satisfy our requirements and their attractive pricing models as well as their reputation.

3.3.1 The list of services offered by the winner

A. **Amazon Key Management Service (KMS):**

https://docs.aws.amazon.com/kms/latest/developerguide/overview.html

For encrypting data at rest and in transit in AWS, the primary service we will use is AWS Key Management Service (KMS) for **TR1.1**

AWS Key Management Service (KMS) allows us to create and control the encryption keys used to encrypt your data. It integrates with various AWS services, such as Amazon S3, Amazon EBS, Amazon RDS, and more, enabling you to encrypt data at rest within these services.

B. **AWS Identity and Access Management:**

https://aws.amazon.com/iam/

For implementing rigorous access controls and Identity and Access Management (IAM) in AWS, the primary service you would use is AWS Identity and Access Management (IAM) for **TR1.2**

Amazon Web Services (AWS) offers a web service called AWS Identity and Access Management (IAM) that lets users securely manage who can access AWS resources and services. IAM lowers the risk of unauthorized access and data breaches by assisting organizations in enforcing the concept of least privilege, which guarantees that users and systems have only the rights necessary to carry out their duties.

C. **Amazon Relational Database Service (RDS):**

https://aws.amazon.com/rds

For choosing a scalable relational database management service on AWS, we would use Amazon RDS (Relational Database Service) for **TR1.3, TR6.1, TR8.1.**

A fully managed relational database service provided by Amazon Web Services (AWS) is called Amazon Relational Database Service (RDS). It makes running, scaling, and establishing a relational database on the cloud easier. Several well-known database engines are supported by RDS, including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and Amazon Aurora (an AWS-developed database engine that is compatible with both PostgreSQL and MySQL). Users no longer have to be concerned about maintaining the underlying hardware, operating system, or database software because RDS makes it simple to start, configure, and manage database instances.

By automating repetitive administrative activities like patch management, database setup, and backups, RDS frees users from the burden of managing databases and lets them concentrate on developing applications. Moreover, it offers capabilities like multi-AZ installations for high availability, automatic software patching, and backups. To spread read traffic and boost performance, users can scale their database instances horizontally by adding read replicas or vertically by modifying the compute and memory resources.

D. **Amazon Elastic Load Balancing:**

https://aws.amazon.com/elasticloadbalancing/

For designing a robust, high-availability architecture with redundancy and load balancing to minimize single points of failure on AWS, you would use Amazon Elastic Load Balancing (ELB) for **TR2.1.**

Within one or more availability zones, Amazon Web Services (AWS) offers a fully managed solution called Amazon Elastic Load Balancing (ELB), which automatically divides incoming application traffic among several destinations including IP addresses, containers, and Amazon EC2 instances.

E. **Amazon EC2 Auto Scaling:**

https://aws.amazon.com/ec2/autoscaling/getting-started/

For implementing auto-scaling to dynamically adjust resources based on traffic, the primary AWS service you would use is Amazon EC2 Auto Scaling for **TR2.2, TR3.1**

Using scaling policies that you provide, Amazon EC2 Auto Scaling allows us to automatically add or delete EC2 instances while also assisting you in maintaining application availability..

F. **AWS Web Application Firewall:**

https://aws.amazon.com/waf/features

For implementing strong security measures, including firewalls, on AWS, we would use AWS Web Application Firewall (WAF) for **TR2.3.**

With AWS WAF, you may design rules to filter web traffic according to standards such as custom URIs, IP addresses, and HTTP headers and bodies. This provides you with an extra degree of defense against online threats that aim to take advantage of holes in your own or other people's web applications. Moreover, AWS WAF facilitates the creation of rules that prevent popular online vulnerabilities like SQL injection and cross-site scripting.

G. **AWS Trusted Advisor:**

https://aws.amazon.com/premiumsupport/technology/trusted-advisor/

To regularly analyze and optimize resource usage for bank application on AWS by identifying and addressing underutilized or idle resources, AWS Trusted Advisor is utilized, offering real-time guidance and cost-saving alerts for **TR4.1, TR3.2.**

For regularly analyzing resource usage to identify underutilized or idle resources and taking action to reduce or terminate them on AWS, we would use AWS Trusted Advisor. AWS Trusted Advisor is a service that provides real-time guidance to help you provision your resources following AWS best practices, and it can alert you to potential cost savings by identifying unused or underutilized resources.

H. **Amazon Simple Queue Service:**

https://aws.amazon.com/sqs/features/

To manage surges in incoming transactions and control their processing order and pace, Amazon Simple Queue Service (SQS) would be employed for **TR4.2.**

For handling bursts of incoming transactions and managing the order and pace at which they are processed, you would use Amazon Simple Queue Service (SQS). SQS is a fully managed message queuing service that enables decoupling of the components of a cloud application, allowing them to operate independently and scale separately. It can help you handle bursts of traffic and ensure the orderly processing of messages.

I. **AWS CloudFront:**

https://aws.amazon.com/cloudfront/

To enhance content delivery and minimize latency in a banking application, utilize Amazon CloudFront—a fast, secure CDN service seamlessly integrated with Amazon Web Services for efficient content distribution with low latency and high data transfer speeds for **TR4.3.**

For leveraging CDNs (Content Delivery Networks) to cache and serve content from edge locations, reducing latency and improving response times, you would use Amazon CloudFront. CloudFront is a fast and highly secure CDN service that integrates with other Amazon Web Services products to give developers and businesses an easy way to distribute content to end users with low latency and high data transfer speeds.

J. **AWS CloudWatch:**

https://aws.amazon.com/cloudwatch/

To assess cloud service scalability for handling traffic spikes and establish peak load performance requirements, leverage Amazon CloudWatch for monitoring, observability, and setting alarms in the context of a banking application **for TR5.1, TR6.2.**

For determining the expected scalability of cloud services to handle traffic spikes and outlining performance requirements during peak loads, you would use Amazon CloudWatch. CloudWatch provides monitoring and observability services for AWS resources, enabling you to collect and track metrics, collect and monitor log files, and set alarms.

K. **AWS Systems Manager:**

https://aws.amazon.com/systems-manager/

To address security vulnerabilities and ensure regular updates in a banking application hosted on AWS, AWS Systems Manager is utilized for automated operating system patching and maintenance tasks for **TR7.1**

For regularly updating and patching software components to address known security vulnerabilities in an AWS environment, you would typically use AWS Systems Manager. AWS Systems Manager allows you to automate the patching of operating systems

(including the operating system in EC2 instances), manage patch compliance, and perform maintenance tasks across your AWS resources.

L. **Amazon Simple Storage Service:**

https://aws.amazon.com/s3/

To facilitate data portability in compliance with privacy regulations in a banking application, Amazon S3 is utilized for scalable and configurable user data management for **TR9.1**

For allowing users to easily access, export, and delete their personal data, as well as implementing mechanisms for data portability and deletion in compliance with privacy regulations, you would use Amazon S3 (Simple Storage Service). S3 is a scalable object storage service that can be configured to manage user data, and it provides features and APIs that allow for easy access, export, and deletion of data.

M. **AWS Security Hub:**

https://aws.amazon.com/security-hub/

We will utilize AWS Security Hub for aligning incident response plans with regulatory requirements, ensuring timely and effective response to security incidents in the banking application for **TR10.1.**

AWS Security Hub provides a comprehensive view of your high-priority security alerts and compliance status across your AWS accounts. It can integrate with other AWS services and third-party tools to help you automate response actions in the event of a security incident, ensuring timely and effective incident response.

4. The First Design draft

We have chosen the following technical requirements from the list supplied in section 2.3:

Design steps for Banking Application along with technical requirements:

1. **Define Requirements and Constraints:**
    a. Gather detailed requirements for functionality, security, and compliance (TR1.1 - TR1.3, TR2.1 - TR2.2, TR3.1 - TR3.2, TR4.1 - TR4.2, TR5.1 - TR5.2, TR6.1 - TR6.2, TR7.1 - TR8.1, TR9.1, TR10.1).
    b. Identify regulatory constraints and standards.

2. **System Architecture:**
    a. Design a basic architecture using AWS services (EC2 and RDS for components).
    b. Ensure redundancy by distributing components across multiple Availability Zones (TR2.1).

3. **Security and Compliance:**
    a. Implement basic security measures: encryption using AWS Key Management Service (KMS) (TR1.1), basic access controls with AWS IAM (TR1.2).

4. **Database Management:**
    a. Choose Amazon RDS for a basic, scalable relational database (TR1.3).

5. **High Availability:**
    a. Ensure basic redundancy with multiple Availability Zones (TR2.1).
    b. Implement basic load balancing.

6. **Auto Scaling:**
    a. Implement basic Auto Scaling to adjust resources based on traffic.(TR2.2)

7. **Cost Optimization:**
    a. Perform basic resource monitoring with AWS CloudWatch (TR3.1).
    b. Initiate optimization using AWS Trusted Advisor.

8. **Traffic Management:**
    a. Implement basic load balancing for incoming transactions (TR4.1).
    b. Integrate basic message queues for transaction handling (TR4.2).

9. **System Responsiveness:**
    a. Plan for basic scalability to handle traffic variations (TR5.1).
    b. Implement simple real-time monitoring of logs using Amazon CloudWatch Logs (TR5.2).

10. **Disaster Recovery and Business Continuity:**
    a. Implement basic automated backup and restore procedures using Amazon RDS (TR6.1).
    b. Establish initial performance monitoring and alerting for quick response (TR6.2).

11. **System Security:**
    a. Implement basic regular updates and patching using AWS Systems Manager (TR7.1).

12. **Data Backups and Recovery:**
    a. Implement basic full and incremental backups using Amazon S3 (TR8.1).

13. **User Privacy:**
    a. Implement basic mechanisms for users to access, export, and delete personal data (TR9.1).

14. **Regulatory Compliance:**
    a. Initiate steps towards establishing an incident response plan (TR10.1).

15. **Documentation and Testing:**
    a. Create basic documentation for architecture and configurations.
    b. Conduct initial testing for security, performance, and compliance (TR1.1 - TR10.1).

16. **Deployment:**
    a. Gradually deploy to minimize risks.
    b. Conduct basic monitoring during and after deployment for any issues (TR5.2, TR6.2).

17. **Monitoring and Optimization:**
    a. Implement basic continuous monitoring with AWS CloudWatch.
    b. Begin resource optimization based on changing demands (TR3.1).

This consolidated version provides a high-level overview of the key steps in the AWS design for a basic banking application, addressing technical requirements at each stage.

This is our first design draft for the proposed solution:

4.1 The basic building blocks of the design

The suggested cloud architecture for the banking application includes a number of unique and important components. The following components support the solution's overall functionality, performance, security, and scalability:

1. **Multi-Tier Architecture:**
    - The user interface, application logic (represented by EC2 instances with auto scaling), and data storage (RDS) are clearly segregated in this multi-tiered architecture.

2. **Scalability and Elasticity:**
    - Using AWS EC2 Auto Scaling, the number of instances is automatically changed in response to demand. This guarantees flexibility and scalability, enabling the application to effectively manage changing workloads.
3. **Networking:**
    - By dividing up incoming traffic among several EC2 instances, Amazon Elastic Load Balancer improves fault tolerance and availability. Response times are accelerated and resource utilization is optimized by this load balancing technique
4. **Relational Database Service (RDS):**
    - A managed relational database service with capabilities like automated backups, maintenance, and scaling is provided by using RDS. Data reliability and integrity are thus guaranteed.
5. **Security Layers:**
    - The security levels are enhanced by AWS KMS (Key Management Service) and WAF (Web Application Firewall). WAF safeguards sensitive data from online exploitation and attacks, whereas KMS controls encryption keys.
6. **Queue Service for Decoupling:**
    - The application's components are separated by the use of Amazon SQS (Simple Queue Service), which improves scalability, dependability, and asynchronous processing.
7. **Identity and Access Management (IAM):**
    - IAM is used to control user access to Amazon services, making sure that the right authorization and authentication are in place. This is essential to preserving the banking application's security and privacy.
8. **Monitoring and Logging:**
    - AWS CloudWatch is integrated to track the application's performance, gather log data, and set off alarms depending on pre-established thresholds. This offers perceptions into the functionality and state of the system.
9. **Content Delivery Network (CDN):**
    - As a CDN, AWS CloudFront delivers material from edge locations closer to end users, increasing the performance and latency of the application.
10. **Cost Optimization and Best Practices:**
    - Included is AWS Trusted Advisor, which offers suggestions for cost optimisation, security enhancement, and performance improvement. This guarantees that the solution is economical and adheres to best practises.

4.2 Top-level, informal validation of the design

The banking application's suggested cloud architecture is made to be secure, scalable, efficient, and conformant with industry standards. The following justifies the practicality of this solution:

1. **Scalability and Elasticity:**
    - The programme can dynamically modify its capacity in response to demand by utilizing AWS EC2 Auto Scaling. This guarantees the system's scalability and elasticity, enabling it to effectively manage a range of workloads.
2. **High Availability and Fault Tolerance:**
    - By distributing traffic over several EC2 instances, AWS Elastic Load Balancer improves fault tolerance and availability. Because of this redundancy, the programme is guaranteed to continue functioning even in the event of a server failure.
3. **Managed Database Service (RDS):**
    - With features like patch management, scalability, and automated backups, RDS offers a managed relational database service. This guarantees data integrity, improves reliability, and streamlines database administration.
4. **Security Measures:**
    - Robust security protections are enhanced by AWS KMS and WAF. WAF safeguards against frequently used web attacks, while KMS controls encryption keys to protect confidential information. By complying with industry best practices and legal standards, these actions improve the application's overall security posture.
5. **Decoupling with Queue Service:**
    - Better scalability, resilience, and asynchronous task processing are made possible by the decoupling process using Amazon SQS. This enhances the application's overall responsiveness and efficiency.
6. **Content Delivery Network (CDN):**
    - By serving as a CDN, Amazon CloudFront reduces latency and speeds up content delivery. By delivering material from edge locations nearer to end consumers, this improves user experience.
7. **Identity and Access Management (IAM):**
    - IAM controls user access to AWS services by ensuring appropriate authorization and authentication. This is essential for safeguarding private financial information and making sure that the system is only accessible to those who are authorized.
8. **Monitoring and Logging:**
    - Monitoring and logging features offered by AWS CloudWatch enable proactive problem detection and performance optimisation. This makes it easier to manage and maintain the application effectively.

9. **Cost Optimization and Best Practices:**
    - The advice provided by AWS Trusted Advisor can help with cost optimisation, security enhancement, and performance improvement. Following these best practices guarantees that the solution is affordable and compliant with AWS requirements.

4.3 Action items and rough timeline

1. Design the final architecture of our solution: Oct-29-2023
2. Validate a few aspects of our solution with the help of Kubernetes experimentation: Nov - 20-2023
3. Validate our design after receiving feedback from the instructors: Nov-8-2023

5. The Second Design

5.1 Use of the Well-Architected Framework

The AWS Well-Architected Framework suggests the following steps:

1. Create a list of all business requirements for which an architecture is to be produced.
2. Convert this list into another list of technical requirements.
3. Search for options to address each technical (and hence business) requirement.
4. Evaluate tradeoffs for each option.
5. Select an option for implementation.
6. Document the selection in a design document and/or architectural diagrams.

- We are following the same approach in our design as mentioned in the First Level draft. Our AWS design for the banking application aligns seamlessly with the AWS Well-Architected Framework.
- **We initiated the process by compiling a list of business requirements, meticulously translating them into specific technical requirements. Options from the AWS ecosystem were explored for each technical requirement, considering tradeoffs and evaluating different services.**
- Our selections were made based on their alignment with both business and technical criteria. The entire decision-making process, along with architectural choices, was thoroughly documented in a design document and visualized through architectural diagrams. This approach ensures a systematic and well-documented implementation of the banking application on AWS.

5.2 Discussion of pillars

The Well Architected Framework (WAF) on AWS is a set of guidelines intended to assist users in creating dependable, secure, economical, and long-lasting cloud architecture on the platform. With many years of experience, AWS Solution Architects designed it. To fully utilize a cloud architecture, a business can benefit from best practices, questions to consider, and approaches that the WAF offers. It is supported by six pillars:

1. Operational Excellence
2. Performance Efficiency
3. Cost Optimization
4. Reliability
5. Security
6. Sustainability

We will now discuss two pillars in detail:

1. Operational Excellence

The AWS Well-Architected Framework's Operational Excellence pillar highlights how crucial it is to build and run systems for efficiency, continuous improvement, and the creation of business value. Its guiding ideas include anticipating failure, refining operating procedures, making frequent small reversible modifications, and conducting operations using code (Infrastructure as Code). Preparation through operational mechanism establishment, code operation, and continuing evolution through continuous improvement reviews are the main operations. Defined metrics guarantee alignment with corporate objectives and enable effective reactions to incidents and changes. Monitoring and operations evaluations are essential. In order to achieve operational excellence in cloud environments, this pillar encourages a culture of automation, measurement, and learning. This leads to higher resilience, quicker change adaptation, and continual development that is in line with business objectives.

There are five design principles for operational excellence in the cloud:

1. Perform operations as code
2. Make frequent, small, reversible changes
3. Refine operations procedures frequently
4. Anticipate failure
5. Learn from all operational failures

1. **Perform operations as code:** The same engineering discipline you use for application code can be applied to your entire environment on the cloud. Everything you do (applications, infrastructure, etc.) can be defined as code. You can script operations and automate their execution by triggering events when you treat activities as code. You can reduce human error and promote uniformity in the operations by implementing them as code.
2. **Make frequent, small, reversible changes:** When creating your workloads, make sure that the components may be improved on a regular basis through helpful workload modifications. Small adjustments should be done so that your system can be fully restored in the event of a failure.
3. **Refine operations procedures frequently:** You should strive to get better at using operations procedures as you get better at managing your workload. It should be possible to evaluate and confirm the procedures' efficacy. This should be known to the teams.
4. **Anticipate failure:** It should be possible to locate possible sources of failure and eliminate them in advance of a system breakdown. It's crucial to test failure scenarios and

confirm that you understand their implications. Make sure teams are familiar with the execution of your response protocols for these problems and that they are successful. Evaluate the workload and team dynamics by putting these simulated failure scenarios to the test.

5. **Learn from all operational failures:** Learn from operational events and failures to make improvements to the infrastructure and applications. The whole organization ought to be informed about this.

   According to AWS, achieving operational excellence in the cloud may be accomplished in four ways:
   - Organization
   - Prepare
   - Operate
   - Evolve

2. Reliability

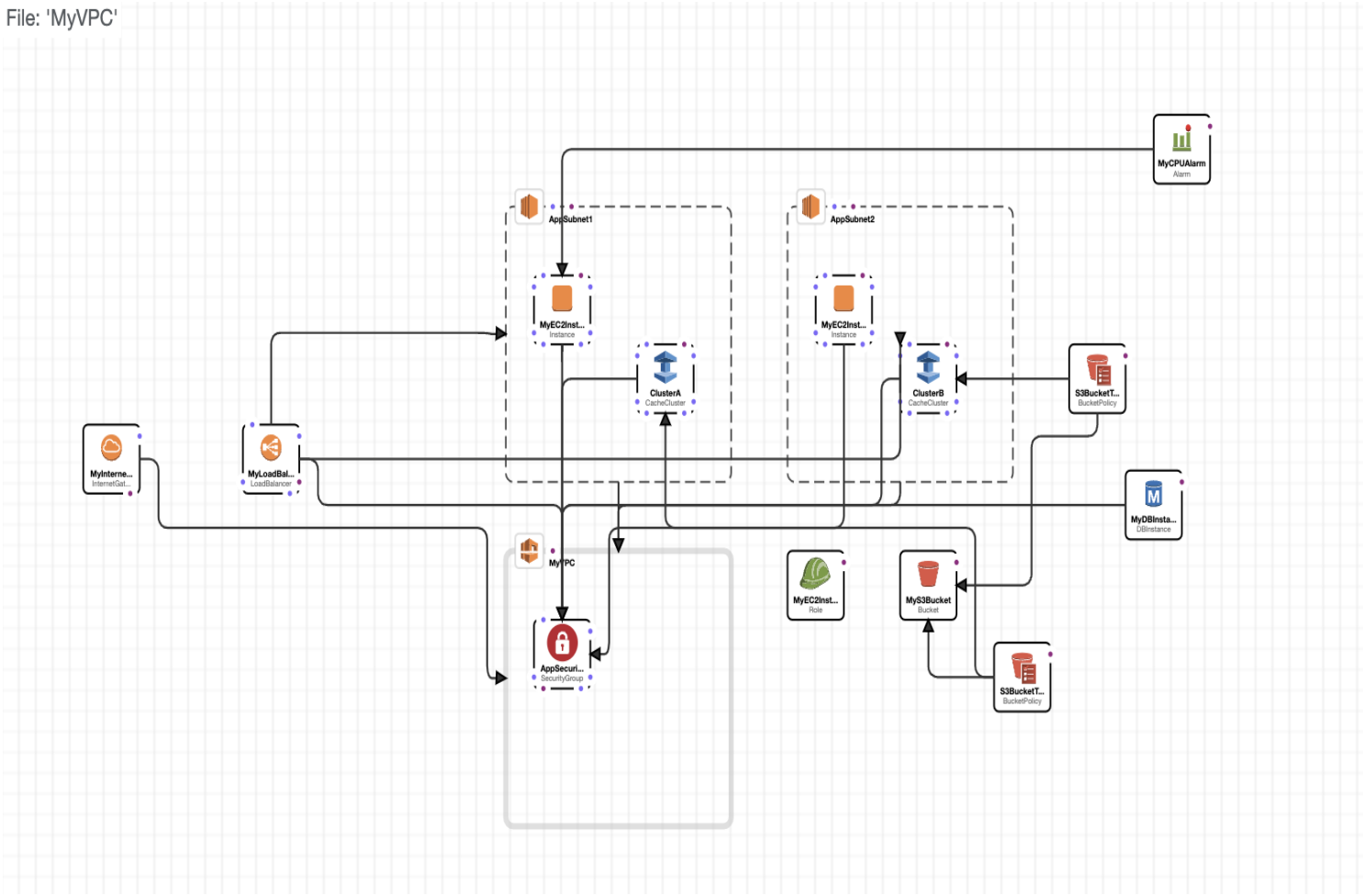
Within the AWS Well-Architected Framework, the Reliability pillar concentrates on operating systems and system design that provide a high degree of reliability. It highlights how crucial it is to put procedures in place for failure detection, prevention, and recovery. Important guidelines incorporate articulating and conveying dependability needs, putting change management procedures for infrastructure and code into practice, setting up thorough monitoring to find problems early, and guaranteeing efficient incident reactions. Using measurements and feedback, the pillar pushes organizations to regularly assess the dependability of their systems. It places special focus on adapting to change, taking failure into account, and putting scalable and effective resource management in place. Organizations may create and manage systems that satisfy user expectations for availability and performance while skillfully handling the effects of changes and failures by following the best practices outlined in the Reliability pillar.

AWS says that these are the best practice guidance for implementing reliable workloads on AWS:

1. Automatically recover from failure
2. Test recovery procedures
3. Scale horizontally to increase aggregate workload availability
4. Stop guessing capacity
5. Manage change in automation

1. **Automatically recover from failure:** Key performance indicators (KPIs) are used to track workloads and can be used to initiate automation in the event that a threshold is crossed. KPIs that gauge commercial value rather than the intricacies of the service's operation must be used. This enables automated failure tracking and alerting, as well as automated recovery procedures that mitigate or fix the failure. With advanced automation, faults can also be predicted and fixed before they happen.
2. **Test recovery procedures:** In an on-premise system, testing is usually done to demonstrate that the workload functions in a certain situation. Recovery strategies are not usually validated by testing. On the other hand, we may verify our recovery protocols and test how our workload fails on the cloud. With automation, we may replicate various failure scenarios and simulate failures that occurred before. By exposing failure paths, this strategy lowers risk by allowing us to test and address them before a genuine failure situation arises.
3. **Scale horizontally to increase aggregate workload availability:** To lessen the effect of a single failure on the total workload, we should replace one large resource with several tiny ones. Requests could avoid sharing a single point of failure if they were split up among several smaller resources.
4. **Stop guessing capacity:** Resource saturation, which occurs when demands on a task surpass its capacity (e.g., DDOS attacks), is a typical reason for on-premises workload failures. We can keep an eye on workload utilization and demand in the cloud. Additionally, we could automate the addition or removal of resources to keep them at the ideal amount to meet demand without going overboard or underprovisioning to the point of redundancy. Although AWS still has certain limitations, some quotas can be managed and others can be controlled.
5. **Manage change in automation:** Automation should be the method used to make changes to our infrastructure. The automation is one of the modifications that needs to be controlled so that it can be monitored and examined.

5.3 Use of CloudFormation diagrams



5.4 Validation of the design

More detailed arguments for why the services we have chosen achieve our TRs?

- **Encryption and Data Protection (TR 1.1)**:
    - **Service Choice:** AWS RDS for MySQL with the specified encryption options using AWS Key Management Service (KMS).
    - **Validation:**
        - **i.** Encryption at Rest: RDS provides transparent encryption at rest using AWS Key Management Service (KMS), ensuring data integrity and confidentiality.
        - **ii.** Example Impact: Without encryption, sensitive financial data stored in the database could be vulnerable to unauthorized access in case of a security breach.

Potential Impact Calculation: -

- Cost of a Data Breach = $150 per compromised record (industry average)
- Number of Records = 10,000
- Potential Impact = $150 x 10,000 = $1,500,000

I. Encryption in Transit: RDS supports SSL/TLS for encrypting data in transit, securing communication between the application and the database.
II. Example Impact: Without encryption in transit, there's a risk of data interception during communication between the application and the database, potentially leading to data leakage.

Potential Impact Calculation:

- Estimated Loss per Incident = $100,000 (industry average)
- Number of Incidents (per year) = 2
- Potential Impact = $100,000 x 2 = $200,000

- **High-Availability Architecture (TR 2.1)**:
    - **Service Choice**: AWS Elastic Load Balancing (ELB) for load balancing.
    - **Validation**:
        - **i.** Redundancy: ELB distributes incoming traffic across multiple EC2 instances, minimizing single points of failure.
        - **ii.** Example Impact: Without a load balancer, a single server failure could lead to service downtime, impacting users' ability to access banking services.

Potential Impact Calculation: - Estimated Loss per Hour of Downtime = $10,000 (industry average) - - Downtime Hours per Incident = 4 - - Potential Impact = $10,000 x 4 = $40,000

        - **iii.** High-Availability: ELB automatically scales its load balancers in response to incoming traffic, enhancing availability.
        - **iv.** Example Impact: Without auto-scaling, the system might struggle to handle sudden traffic spikes, leading to degraded performance and potential service interruptions.

Potential Impact Calculation: - Estimated Loss per Incident = $50,000 (industry average) - Number of Incidents (per year) = 3 - Potential Impact = $50,000 x 3 = $150,000

- **Access Control (TR 1.2):**
    - **Service Choice:** AWS IAM for identity and access management.
    - **Validation:**
        - i. **Rigorous Access Controls:** IAM allows fine-grained access control, enabling precise permission assignment to individuals or systems.
        - ii. **Identity Management:** IAM facilitates centralized control over AWS resources, ensuring only authorized personnel can access sensitive systems.

- **Database Scalability (TR 1.3):**
    - **Service Choice:** Amazon RDS with scalable relational database management.
    - **Validation:**
        i. **Scalability:** RDS allows seamless scaling of database instances to handle increased loads while maintaining performance.
        ii. **Automated Scaling:** Features like Auto Scaling in RDS automatically adjust compute capacity based on demand, optimizing resource usage.

- **Auto-Scaling (TR 2.2):**
    - **Service Choice:** Auto Scaling for dynamic resource adjustment.
    - **Validation:**
        i. **Dynamic Resource Adjustment:** Auto Scaling adjusts the number of EC2 instances based on configured policies, ensuring optimal resource usage.
        ii. **Traffic Handling:** Auto Scaling ensures the application can handle varying workloads effectively by dynamically adjusting resources.

- **Regular Software Updates (TR 7.1):**
    - **Service Choice:** AWS Systems Manager for maintenance.
    - **Validation:**
        i. **Patch Management:** Systems Manager enables automated patching of instances, including the operating system, web server, application server, and libraries, ensuring software components are up-to-date.

- **Data Backup and Recovery (TR 6.1 and TR 8.1):**
    - **Service Choice:** AWS RDS automated backup and Amazon S3 for backup storage.
    - **Validation:**
        i. **Automated Backup:** RDS provides automated daily backups with point-in-time recovery, ensuring critical data is regularly backed up.
        ii. **Incremental Backups:** Using Amazon S3, a combination of full and incremental backups optimizes storage and backup time while providing data recoverability.

- **User Privacy (TR 9.1):**
    - **Service Choice:** AWS IAM and AWS S3 for user data control.
    - **Validation:**
        i. **Data Control:** IAM allows precise control over user access to AWS resources, while S3 provides mechanisms for user data access, export, and

deletion, ensuring compliance with privacy regulations.

- **Compliance with Industry Regulations (TR 10.1):**
  - **Service Choice:** AWS provides a comprehensive set of compliance certifications.
  - **Validation:**
    i. **Incident Response Planning:** AWS services align with regulatory requirements, and AWS provides tools like AWS Security Hub for incident response planning to comply with breach notification and reporting obligations.

- **Monitoring and Log Management (TR 5.2 and TR 6.2):**
  - **Service Choice:** AWS CloudWatch for real-time monitoring and alerting.
  - **Validation:**
    i. **Real-time Monitoring:** CloudWatch enables real-time monitoring of logs, allowing prompt detection and response to security incidents or unusual system activities.
    ii. **Comprehensive Performance Monitoring:** CloudWatch offers comprehensive performance monitoring, including metrics on CPU utilization, ensuring proactive tracking of the application's health.

- **Cost Optimization (TR 3.1 and TR 3.2):**
  - **Service Choice:** AWS Cost Explorer for cost optimization.
  - **Validation:**
    i. **Resource Optimization:** Cost Explorer provides insights into resource usage, allowing continuous monitoring and optimization to ensure efficient scaling and cost control.
    ii. **Usage Analysis:** Regular analysis of resource usage identifies underutilized or idle resources, supporting actions to reduce or terminate them.

- **Scalability of Cloud Services (TR 5.1):**
  - **Service Choice:** AWS services designed for scalability.
  - **Validation:**
    i. **Cloud Service Scalability:** AWS services are designed to be inherently scalable, ensuring the system responds quickly and stays operational during traffic spikes.

These validations highlight how the chosen AWS services align with and fulfill the specified Technical Requirements, offering features, metrics, and guarantees that contribute to the security, availability, and efficiency of the banking application.

5.5 Design principles and best practices used

In our banking application design, we embrace the following design principles and best practices inspired by AWS:

- Think Adaptive and Elastic (Database Scalability):

- We apply the principle of thinking adaptively and elastically by choosing Amazon RDS for database management. This service seamlessly scales database instances, aligning with the adaptive nature required for a banking application that experiences varying workloads.

Treat Servers as Disposable Resources (Auto-Scaling):

- Reflecting the principle of treating servers as disposable resources, our design incorporates Auto Scaling for dynamic resource adjustment. This approach treats EC2 instances as disposable. This not only enhances fault tolerance but also aligns with the principle of automation, contributing to the overall efficiency of the system.

Automate Regular Software Updates:

- We implement the principle of automation by utilizing AWS Systems Manager for regular software updates. This tool automates patch management, treating the process of updating instances, including the operating system and various software components, as an automated and routine task. This practice aligns with the design principle of automation.

Ensure Data Backup and Recovery (Data Backup and Recovery):

- Emphasizing the importance of resilience, our design incorporates AWS RDS automated backup and Amazon S3 for data backup and recovery. This aligns with the principle of removing single points of failure by ensuring that critical data is regularly backed up. The use of both full and incremental backups optimizes the process, aligning with the principle of optimization for cost and efficiency.

Protect User Privacy (User Privacy):

- Reflecting the design principle of implementing loose coupling, AWS IAM and S3 are employed for user data control. IAM allows for precise control over user access, while S3 provides mechanisms for managing user data. This ensures a loosely coupled approach to user data control, aligning with the principle of loose coupling.

Ensure Compliance with Industry Regulations (Compliance with Industry Regulations):

- The design aligns with the principle of focusing on services, not servers, by leveraging AWS's comprehensive set of compliance certifications and tools. This ensures that

compliance is built into the architecture and focuses on overall service-level adherence to industry regulations. Additionally, the use of tools like AWS Security Hub aligns with the principle of security being an integral part of the cloud architecture.

5.6 Tradeoffs revisited

IMPORTANT - arguments for choosing what tradeoffs you made for conflicting TRs - google tradeoffs decision.
Pick one set of conflicting TRs - explain which TR you chose to optimize. Why and how.

- Security vs. Accessibility:

Design Choice: The design prioritizes robust security (TR1.1) over access control (TR1.2) to protect sensitive data. This ensures a secure environment for banking applications.
Trade-off Consideration: While strong encryption might enhance security, they might introduce some level of inconvenience for users. Striking a balance between stringent access controls and user convenience is crucial to maintaining security without hindering accessibility.

- Cost vs. Performance:

Design Choice: The design considers the need for cost optimization (TR3.1 and TR3.2) while maintaining acceptable performance levels. Resource usage is continuously monitored to identify and reduce unnecessary costs, and auto-scaling is employed for efficient resource utilization during varying workloads (TR5.1).
Trade-off Consideration: Balancing low-latency performance with cost-effective resource usage is a trade-off. The challenge is to ensure optimal performance without incurring excessive costs, especially in a resource-intensive environment like a banking application.

- Compliance vs. Innovation:

Design Choice: The design emphasizes strict regulatory compliance (TR9.1) to ensure adherence to industry standards and regulations. This is crucial in the banking sector where compliance is paramount over TR5.1.
Trade-off Consideration: While compliance is prioritized, it may introduce constraints on the speed of innovation. Striking a balance between staying compliant and fostering innovation is essential for the long-term success of the application.

- Availability vs. Redundancy:

Design Choice: The design places a high priority on availability (TR2.1) by incorporating redundancy in the architecture to minimize single points of failure (TR4.2) which is chosen over TR3.1.

Trade-off Consideration: While redundancy enhances availability, it might increase operational complexity and costs. Striking the right balance between availability and redundancy is crucial to ensure uninterrupted service while managing operational overhead.

- Access Control vs. User Convenience:

Design Choice: The design adopts stringent access controls (TR1.2) to enhance security, recognizing the sensitive nature of banking data.
Trade-off Consideration: Stringent access controls might introduce some level of inconvenience for users. The challenge is to implement access controls in a way that maintains a balance between security and user convenience.

- Data Security vs. Performance:

Design Choice: The design prioritizes robust encryption methods (TR1.1) for both data at rest and in transit to ensure data security.
Trade-off Consideration: While encryption enhances data security, it introduces a potential performance impact due to encryption/decryption processes. Balancing data security with performance is crucial, and the design leans towards prioritizing data security, especially in the context of a banking application handling sensitive information.

5.7 Discussion of an alternate design

Skipping

6. Kubernetes experimentation

6.1 Experiment Design

Experiment details

- **Need of Kubernetes:**
    1. Using a single instance is insufficient when our microservice receives a large volume of requests. Our microservice has to run many instances in order to handle all of the requests.
    2. Using numerous instances of it would be a waste of money and resources when the number of requests is low.
    3. The solution automates the need to increase the number of instances of our microservice when there are a lot of requests and lower the number of instances when there are fewer requests in order to overcome this problem.
    4. This task requires a vast amount of manual labor. Therefore, we employ Kubernetes, an automated tool that offers a feature for resource scaling up and down, to address this issue.

- **Use of Kubernetes:**
    1. The ability to autoscale our microservices instances is provided by Kubernetes.
    2. The horizontal pod autoscaler (HPA), which monitors the CPU and MEM utilization by each pod, is employed in this.
    3. The parameters for scaling up and down pods must be defined.
    4. The bare minimum and maximum number of pods that can be used for the service, as well as the first replica of the pods that will be used to launch the service, must all be specified.
    5. The Kubernetes autoscaler will raise the pod replica counts in accordance with the CPU and MEM mean utilization criteria, if the load exceeds them.
    6. HPA will reduce the number of pods if the load drops below a threshold given in the criterion.

- **Handling of load on our application:**
  1 pod is the minimum and 10 pods is the maximum number of pods that we have established.
    1. We have defined the autoscale criterion in our experiment at 50% mean CPU utilization.
    2. To spread the load, the autoscaler creates a new pod in our system if the average CPU utilization is more than 50%.

3. In order to check the load, our autoscaler will reduce the number of pods one by one if the mean CPU utilization is very low. The autoscaler will reduce the number of pods to the bare minimum of replicas indicated in the criterion if the load is so low that it can be handled by a single pod.
4. Workload generation - In order to test a large number of requests to test our Kubernetes autoscaler we made use of the Locust tool.

6.2 Workload generation with Locust

1. **Using Locust tool** - To test an application's load and see whether the system can manage multiple requests at once, use locust.
2. We have used locust to test our application.
3. Total number of users - **50** (Peak concurrency)
4. Total number of requests - **5330**
5. Requests per second - **67**
6. Failure - **1.2** s
7. Since we have a low request failure rate, our autoscaler managed the load and created new pods utilizing the ECR-stored version of our docker image as needed.
8. As anticipated, the autoscaler reduced the replicas to the bare minimum when there was no request; this is demonstrated in the following section.
9. The locust test report is as follows:

Script: locust_load.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/json	5330	136	510	0	2949	20	67.2	1.7
	Aggregated	**5330**	**136**	**510**	**0**	**2949**	**20**	**67.2**	**1.7**

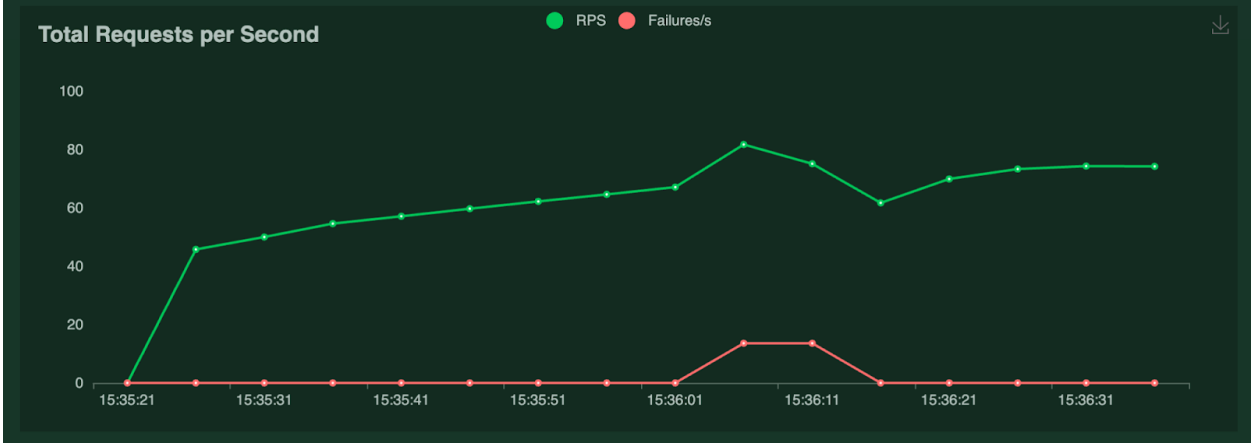
Response Time Statistics

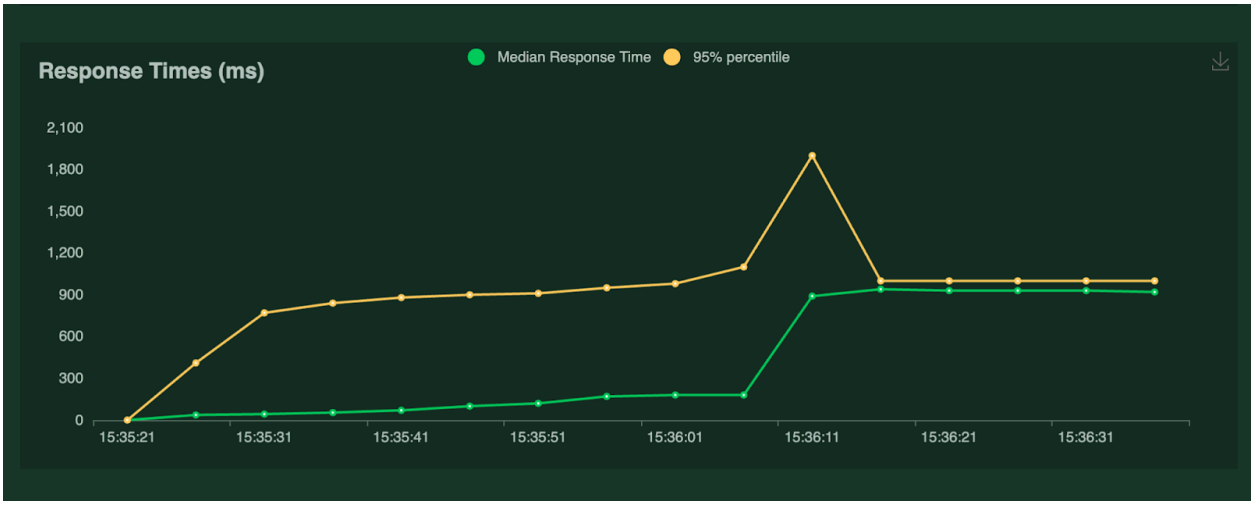
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/json	640	880	920	950	990	1000	1100	2900
	Aggregated	**640**	**880**	**920**	**950**	**990**	**1000**	**1100**	**2900**

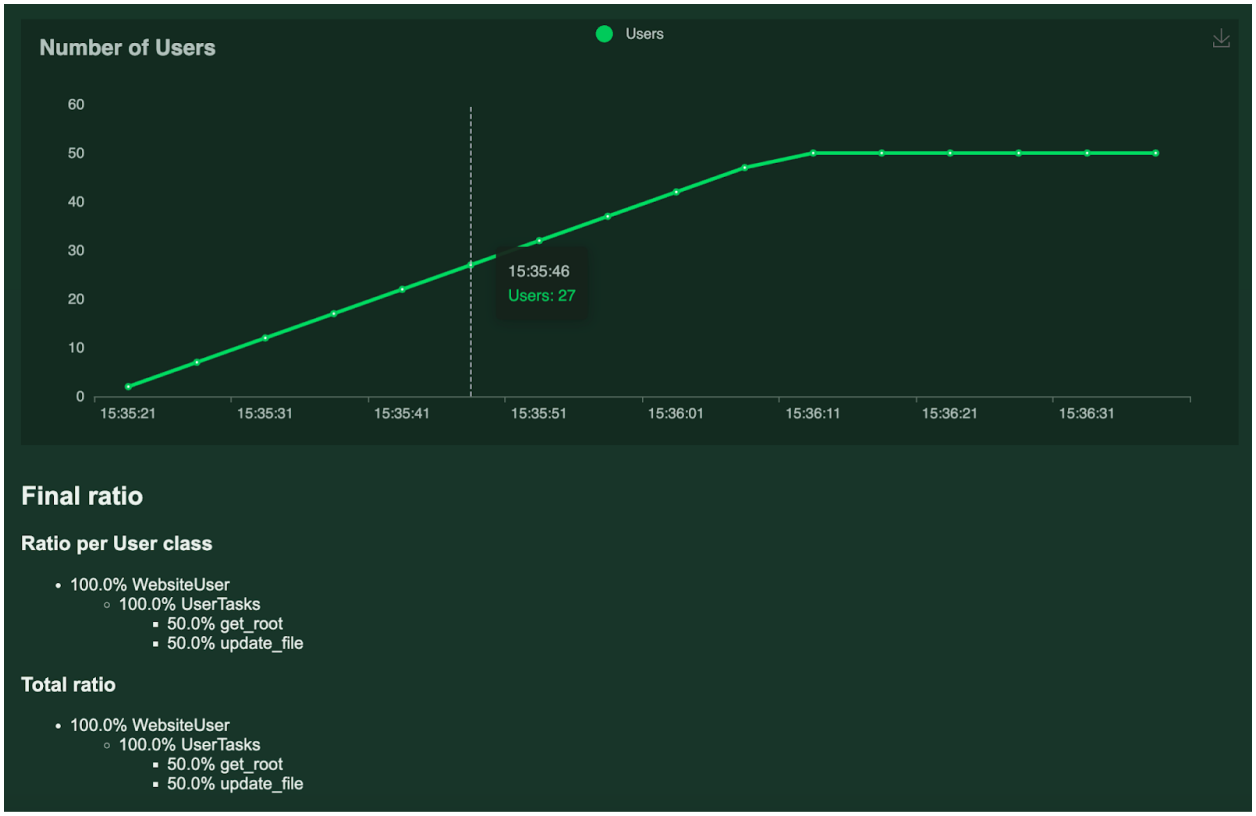
Failures Statistics

Method	Name	Error	Occurrences
POST	/json	[Errno 111] Connection refused	136

Charts


Total Requests per Second
RPS
Failures/s
100
80
60
40
20
0
15:35:21
15:35:31
15:35:41
15:35:51
15:36:01
15:36:11
15:36:21
15:36:31


Response Times (ms)
Median Response Time
95% percentile
2,100
1,800
1,500
1,200
900
600
300
0
15:35:21
15:35:31
15:35:41
15:35:51
15:36:01
15:36:11
15:36:21
15:36:31




```
/LogInjester/locust_csc547$ kubectl get hpa myservice --watch
NAME        REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myservice   Deployment/myservice   1%/50%    1         10        1          23m
myservice   Deployment/myservice   71%/50%   1         10        1          24m
myservice   Deployment/myservice   71%/50%   1         10        2          25m
myservice   Deployment/myservice   19%/50%   1         10        2          25m
myservice   Deployment/myservice   1%/50%    1         10        2          26m
```

6.3 Analysis of the results:

1. Initially we have 1 pod running on our application.
2. **With CPU_percent=60, or mean CPU utilization=50%, minimum pod=1,** or the bare minimum of pods our deployment should have, and maximum pod=**10**, or the maximum number of pods our deployment can have, we add the auto-scalar.
3. Since our minimum replica count was 1, the first replica count in our deployment was also 1.
4. Our pod's CPU utilization jumped to about **71%** when we began the load testing process using the Locust tool. Our load generated **50** peak concurrency and **67** requests/sec.
5. We increased the number of replicas in our deployment from one to two by setting the **CPU utilization in our autoscaler to 50%.**

6. Our application has stabilized because, after spawning an additional copy, there were a total of **2** replicas and a mean CPU utilization of roughly **19%**, which is within our criterion.
7. When we stopped the load testing, the mean CPU utilization went to **1%** so ideally we don't need **2** replicas in our deployment. Hence autoscaler scaled down the number of pods to minimum number of pods i.e. **1**
8. **Result:**

Test Scenario	**Expected Output**	**Actual Output**
When the load is less (mean CPU utilization is < 50%)	Number of pods currently used should be equal to the number of minimum pods required to keep the CPU utilization < 50%	The output was as expected. Initially, when there was no load, the number of pods was 1.
When the load is high (mean CPU utilization is > 50%)	Number of pods currently used should be increase until the mean CPU utilization is < 50%	The output is the same as expected. When the load increased, the number of replicas deployed was equal to 2.

7. Ansible Playbooks

Skipped

8. Demonstration

We chose not to demonstrate.

9. Comparison

Skipped

10. Conclusion

10.1 The Lessons Learned

As we concluded this project, several reflections emerged:

- Identifying conflicting requirements proved to be both the most rewarding and time-consuming aspect of the process.

- Our project journey has been a profound learning experience in architecting a secure, available, and cost-efficient banking application using AWS services. The implementation of robust encryption methods, dynamic resource scaling, and meticulous access controls underscored the project's commitment to safeguarding sensitive financial information. The deployment of high-availability architecture, combined with auto-scaling mechanisms and load balancing, ensured uninterrupted banking services while optimizing resource utilization. Key insights highlight the strategic advantages of utilizing AWS services for creating a resilient, secure, and cost-optimized banking application architecture.

- The continuous monitoring, resource scaling, and strategic service selection not only demonstrated efficient resource utilization but also showcased the project's commitment to cost control. Incorporating disaster recovery plans, data backups, and user privacy controls reflected a commitment to industry standards and regulations, providing a comprehensive guide for creating a banking application that aligns with the highest standards of security, reliability, and efficiency. This project has not only helped understand a robust banking application but also equipped the team with valuable insights into the best practices and considerations for leveraging cloud services effectively.

10.2 Possible continuation of the project

None of the group members plan to take the advanced course in cloud computing in spring 2023; therefore we will not be continuing this project.

11 References

[1] AWS Well Architected Framework

https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc

[2] AWS Elastic Load Balancing

https://aws.amazon.com/elasticloadbalancing/

[3] Amazon Elastic Compute Cloud (EC2)

https://aws.amazon.com/pm/ec2/?gclid=CjwKCAiAsIGrBhAAEiwAEzMlC5mPuyoigo56oqIM_MDvFiijenyBUF9Y8YRobPzOjh88F1zYW7UkXBoCpPoQAvD_BwE&trk=36c6da98-7b20-48fa-8225-4784bced9843&sc_channel=ps&ef_id=CjwKCAiAsIGrBhAAEiwAEzMlC5mPuyoigo56oqIM_MDvFiijenyBUF9Y8YRobPzOjh88F1zYW7UkXBoCpPoQAvD_BwE:G:s&s_kwcid=AL!4422!3!467723097970!e!!g!!aws%20ec2!11198711716!118263955828

[4] AWS Key Management Service

https://aws.amazon.com/kms/

[5] AWS Identity and Access Management

https://aws.amazon.com/iam/

[6] Amazon Simple Storage Service

https://aws.amazon.com/pm/serv-s3/?gclid=CjwKCAiAsIGrBhAAEiwAEzMlC1vTQpUKmq7F4nWEjyFlFre0-mmy_LErG3BgMPOhqVpyyufk4XetrhoCToAQAvD_BwE&trk=fecf68c9-3874-4ae2-a7ed-72b6d19c8034&sc_channel=ps&ef_id=CjwKCAiAsIGrBhAAEiwAEzMlC1vTQpUKmq7F4nWEjyFlFre0-mmy_LErG3BgMPOhqVpyyufk4XetrhoCToAQAvD_BwE:G:s&s_kwcid=AL!4422!3!536452728638!e!!g!!aws%20s3!11204620052!112938567994

[7] AWS CloudFormation

https://aws.amazon.com/cloudformation/

[8] AWS CloudFront

https://aws.amazon.com/cloudfront/

[9] AWS WAF

https://aws.amazon.com/waf/

[10] AWS Systems Manager

https://aws.amazon.com/systems-manager/

[11] AWS CloudWatch

https://aws.amazon.com/cloudwatch/

[12] Amazon Relational Database Service

https://aws.amazon.com/rds/

[13] https://chat.openai.com/

[14] https://www.lucidchart.com/pages/

[15] https://minikube.sigs.k8s.io/docs/drivers/docker/

[16] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

[17] YV and YP, “ECE547/CSC547 class notes”.

[18] https://aws.amazon.com/blogs/architecture/category/aws-cloud-financial-management/

[19] https://awsacademy.instructure.com/

[20]https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer.html

[21] https://us-east-2.console.aws.amazon.com/cloudformation/designer/home?region=us-east-2#

[22] https://www.youtube.com/watch?app=desktop&v=-c-oVK2CJhQ&ab_channel=Cloud4DevOps

[23]https://www.researchgate.net/publication/13119982_Even_Swaps_A_Rational_Method_for_Making_Trade-offs