

LitShelf - An online bookstore

"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.

Authors:

- 1) Sunandini Medisetti (smedise@ncsu.edu)
- 2) Tejaswini Panati (tpanati@ncsu.edu)

Project Mentors:

- 1) Dr. Yannis Viniotis
- 2) Dr. Ioannis Papapanagiotou

1. Introduction

1.1 Motivation

To design the architecture for a cloud based online bookstore application using the cloud architect skills learned in CSC 547.

1.2 Executive Summary

LitShelf introduces a robust, secure, and highly scalable online application that seamlessly integrates a comprehensive bookstore e-library experience. LitShelf is designed to empower users with convenient features, enabling them to browse, purchase books, borrow e-library resources, and access a range of related services by ensuring the utmost security and privacy of customer data and financial transactions, providing a trustworthy and dependable digital space for book enthusiasts. Further the online application also enables a collaborative space for fellow readers to check out each other's interpretations of a story. This will improve customer experience and flexibility thereby maximizing the profits for the company with an online presence.

2. Problem Description

2.1 Problem

LitShelf is introduced to establish a secure and scalable online presence for a combined bookstore and e-library. This involves creating and deploying a cloud-based Infrastructure as a Service (IaaS) application that allows users to browse, purchase books, borrow library resources, and access related services while ensuring data security and privacy for customer data and financial transactions.

We wish to use the services provided by the public cloud providers to deploy the application and make it available to the users.

2.2 Business Requirements

S.NO	Business requirements	Description
BR1	Optimize for high performance	The website or application should respond quickly and seamlessly to user interactions, such as searching for books, viewing book details, and accessing library resources.
BR2	The application data and operations should be highly secure	Require stringent security measures to safeguard user data (user data, including personal information and financial details.) from data breaches.
BR3	Detect unauthorized access	Require robust user management and authentication mechanisms to safeguard user accounts and access control.
BR4	The application should be 24/7 available.	Ensure the LitShelf platform operates continuously and consistently, providing users with uninterrupted access to books, resources, and services. Achieving high availability guarantees that the platform remains accessible, even during unexpected events or system maintenance activities.
BR5	Implement multi-tenancy	Enable separate and secure access for both the bookstore and e-library, with the ability to manage multiple tenants, user accounts, and permissions.
BR6	Retain Data	By retaining data, the organization wishes to maintain historical records of book purchases, library

		borrowings, user reviews, and interactions. This historical data can be valuable for trend analysis, reporting, and understanding user preferences over time and auditing purposes.
BR7	Accommodate Time varying payloads	Users may need to reserve popular books (during book releases), and the availability of these reserved books changes over time. The system must accommodate this by dynamically adjusting reservation queues and notifying users when a reserved book becomes available.
BR8	Implement a reliable architecture	Require the application to be fault tolerant and accessible to users at all times as downtime can result in lost sales. Handle order processing without errors or interruptions, ensuring that orders are correctly placed, payments are processed securely, and users receive their purchases promptly.
BR9	Implement a Cost-Effective Approach	Focus on maximizing efficiency, minimizing unnecessary expenditures, and utilizing budget-friendly technologies and practices.
BR10	Manage billing and subscriptions.	Streamline billing processes and subscription management for users who purchase books, borrow library resources, and access premium services.
BR11	Implement the eco-conscious and sustainable architecture	Focuses on adopting eco-conscious practices,

		minimizing environmental impact, and promoting social responsibility within every aspect of the project's development and deployment.
BR12	The online application should be operated with high excellence	Require to run the workloads effectively and there should continuous support for further development to deliver business value
BR13	Implement monitoring and alerting mechanisms	Track various metrics such as application performance metrics, page load speeds and server resource utilization and trigger when performance thresholds are exceeded. Monitor for any security breaches. Monitor the health and the status of the server resources
BR14	Implement Data Storage Solutions	Store a wide range of data, including book information, user profiles, borrowing records, and reviews, to maintain a comprehensive database of both bookstore and library resources. Need to select appropriate cloud storage solution
BR15	Integrate with external platforms to support multiple payment gateways and for analytics	Integrate various payment methods, including credit/debit cards, digital wallets, and other online payment options, to cater to diverse user preferences and regional payment systems. At the same time, we require a rating system and user review moderation and track past book purchases. Recommend similar books based on previous purchases and

		analytics.
--	--	------------

2.3 Technical Requirements

S.No	TR	Corresponding BR and explanation	Reference
TR1.1	Select and provision high-performance compute capacity	BR1 - Optimize for high performance. Justification: By identifying the compute requirements and evaluating against the options available, you can make your workload more resource efficient therefore increasing performance	Pillar: Performance efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/perf-compute.html
TR1.2	Select a robust database option (relational, key value , graph etc) for data storage.	BR1 - Optimize for high performance. Justification: Choosing the correct DB option helps in indexing, query optimization and so on which will optimize the performance	Pillar: Performance Efficiency
TR1.3	Choose a robust and suitable storage solution	BR1 - Optimize for high performance. Justification: This allows you to determine the most efficient and performant storage technology appropriate for your workload needs.	Pillar: Performance efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/perf-data.html
TR1.4	Configure an optimal networking solution	BR1 - Optimize for high performance.	Pillar Performance efficiency

		Justification: Selecting and configuring appropriate connectivity solutions will increase the reliability of your workload and maximize performance.	
<u>TR1.5</u>	Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization)	BR1 - Optimize for high performance. Justification: Identify the KPIs that indicate whether the workload is performing as intended helps to identify degradation and remediate internal or external factors thereby increasing performance	Pillar: Performance efficiency
<u>TR1.6</u>	Implement load balancing to distribute incoming traffic across multiple servers	BR1 - Optimize for high performance. Justification: This will ensure even workload distribution and prevent overloading of any single server and hence high performance.	Pillar: Performance efficiency
<u>TR1.7</u>	Ensure the application can be easily deployed in multiple regions worldwide	BR1 - Optimize for high performance. Justification: This will enable high performance providing lower latency through global deployment.	Pillar: Performance efficiency

Collective Justification: The above TRs comprehensively address the imperative to optimize for high performance outlined in Business Requirement (BR1). Covering crucial aspects such as compute capacity, robust database options, suitable storage solutions, optimal networking configurations, effective workload monitoring, load balancing, and global deployment, these TRs collectively form a holistic framework that ensures the system is meticulously designed and configured to deliver optimal performance across all dimensions.

<u>TR2.1</u>	Implement a mechanism for secure tenant identification and access	BR2 - The application data and operations should be highly secure. Justification: This will allow only authorized and authenticated users and components to be able to access resources hence making it secure	Pillar: Security https://docs.aws.amazon.com/wellarchitected/latest/framework/security-iam.html
<u>TR2.2</u>	Implement firewalls, intrusion detection and prevention systems, and network segmentation.	BR2 - The application data and operations should be highly secure. Justification: This will safeguard the network infrastructure from threats and unauthorized access	Pillar: Security https://docs.aws.amazon.com/wellarchitected/latest/framework/security-infrastructure.html
<u>TR2.3</u>	Implement real-time monitoring and alerting mechanisms to trace insecure actions and changes within the environment.	BR2 - The application data and operations should be highly secure. Justification: This will track and help us to investigate any insecure actions thereby promoting security.	Pillar: Security https://docs.aws.amazon.com/wellarchitected/latest/framework/security-detection.html
<u>TR2.4</u>	Encrypt data at rest and in transit using industry-standard	BR2 - The application data and operations should be	Pillar: Security

	encryption protocols	highly secure. Justification: This will reduce risk of unauthorized access or mishandling or data being lost thereby helping in securing the appln data.	https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-dataprot.html
TR2.5	Implement sign in mechanisms (Ex:SSO) to allow users to access both the bookstore and lending library services using a single set of credentials.	BR2 - The application data and operations should be highly secure. Justification: This provides secure and convenient access control	Pillar: Security
Collective Justification: The amalgamation of the identified Security Technical Requirements (TRs) forms a robust and comprehensive security framework, aligning with the overarching imperative articulated in Business Requirement (BR2) to establish and uphold a highly secure environment for the application's data and operations. These TRs collectively contribute to a resilient security architecture that systematically addresses access control, network security, real-time monitoring, encryption, and secure authentication mechanisms. This integrated approach ensures that the system is meticulously designed and configured to deliver optimal security across all dimensions, aligning with the overarching security goals outlined in BR2.			
TR3.1	Create a secure user management and authentication system that includes robust mechanisms to protect user accounts and control access.	BR3 - Detect unauthorized access Justification: By implementing a secure user management and authentication system, we establish a foundation for verifying the identity of users. This ensures that only authorized users can access your cloud application.	Pillar: Security
TR3.2	Implement encryption	BR3 - Detect	Pillar: Security

	protocols, multi-factor authentication, and secure password storage practices	unauthorized access Justification: This ensures user data confidentiality and user identity verification.	
<u>TR3.3</u>	Integrate access control measures to restrict unauthorized access and implement audit trails for monitoring user activities.	BR3 - Detect unauthorized access Justification: Access control mechanisms define who can access what resources within your application. This helps restrict access to sensitive data and functionality to only those who have the proper authorization.	Pillar: Security
<p>Collective Justification: Collectively, these Technical Requirements establish a comprehensive security posture for user management and access control, systematically addressing user identity verification, data confidentiality, activity monitoring, and vulnerability mitigation. This integrated approach ensures that the cloud application is meticulously designed and configured to detect and prevent unauthorized access, aligning with the overarching security goals outlined in BR3.</p>			
<u>TR4.1</u>	Utilize redundant servers, and failover mechanisms.	BR4 - The application should be 24/7 available. Justification: This will distribute traffic and maintain seamless operations therefore ensuring high availability.	Pillar: Availability
<u>TR4.2</u>	Implement automated monitoring and alerting systems.	BR4 - The application should be 24/7 available. Justification: This will help in detecting potential issues in real-time and trigger	Pillar: Availability

		rapid response protocols to make the systems more available.	
TR4.3	Deploy geographically distributed backup systems and data replication techniques.	BR4 - The application should be 24/7 available. Justification: This will mitigate the impact of unexpected events or system maintenance activities	Pillar: Availability
Collective Justification: The amalgamation of Technical Requirements (TR4.1, TR4.2, and TR4.3) under the Availability pillar forms a robust and integrated framework, aligning with the imperative outlined in Business Requirement (BR4) to ensure the continuous availability of the application 24/7. These requirements collectively contribute to establishing and maintaining a highly available environment, ensuring uninterrupted operations and a seamless user experienceThis integrated approach ensures that the cloud application is meticulously designed and configured to meet the high availability requirements, providing users with uninterrupted access and a seamless experience as outlined in BR4.			
TR5.1	Isolate tenants and orchestrate their deployments effectively	BR5 - Implement multi-tenancy. Justification: Ensures that tenants' data and resources are kept separate and secure, preventing one tenant from accessing or affecting the data and resources of another thereby allowing multi tenancy	
TR5.2	Create access control policies and separate user roles and permissions for each tenant		
Collective Justification: TR5.1 ensures the effective isolation of tenants and orchestrates their deployments in a multi-tenant environment. This approach guarantees that each tenant's data and resources remain separate and secure, preventing unauthorized access and ensuring the integrity of the multi-tenancy architecture. Additionally, TR5.2 further enhances security by implementing access control policies and distinct user roles and permissions for each tenant, reinforcing the principle of least privilege.			

<u>TR6.1</u>	Establish automated backup and restore procedures to protect against data loss	BR6 - Retain Data Justification: This ensures data protection and continuity and hence data retention	
<u>TR7.1</u>	Implement a system to auto-scale horizontally and vertically.	BR7 - Accommodate Time varying payloads. Justification: This auto scaling ability will dynamically handle varying workload.	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/reli_adapt_to_changes_proactive_adapt_auto.html
<u>TR8.1</u>	Manage your quota increase requests for your workload architecture	BR8 - Implement a reliable architecture Justification: By knowing and managing cloud resource constraints, such as disk or network, which are potentially impactful we make the system more reliable.	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/reli-01.html
<u>TR8.2</u>	Provision redundant connectivity between private networks in the cloud and on-premises environments	BR8 - Implement a reliable architecture Justification: This will ensure resiliency in the failure of the topologies and handle unexpected increase in traffic or use of your services	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/reli_planning_network_topology_ha_conn_private_networks.html
<u>TR8.3</u>	Implement loose coupling using load	BR8 - Implement a reliable architecture	Pillar: Reliability

	balancers	Justification: Enabling load balancers will route traffic to only healthy EC2 instances mitigating failure on any one of them	https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_prevent_interaction_failure_loosely_coupled_system.html
TR8.4	Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand.	BR8 - Implement a reliable architecture Justification: This will help in efficiently managing resource allocation and hence ensuring that the system behaves as expected consistently	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_monitor_aws_resources_monitor_resources.html
TR8.5	Implement a system to auto-scale horizontally and vertically that can dynamically handle varying workload.	BR8 - Implement a reliable architecture Justification: This will scale resources proactively to meet demand and avoid availability impact and hence improving reliability	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_adapt_to_changes_proactive_adapt_auto.html
TR8.6	Establish automated backup and restore procedures to protect against data loss	BR8 - Implement a reliable architecture Justification: Implementing a mechanism to create backups, or being able to reproduce the data from an external source improves the ability to restore and recover data during an outage.	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_backing_up_data_automated_backups_data.html
TR8.7	Implement automated failure recovery mechanisms	BR8 - Implement a reliable architecture Justification: Automated recovery	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_failure_recovery.html

		reduces your recovery time by eliminating the opportunity for manual errors.	d/latest/framework/recovery_auto_recovery.html
TR8.8	Use automation when obtaining or scaling resources	BR8 - Implement a reliable architecture Justification: This reduces errors caused by manual processes and reduces the effort to deploy changes.	Pillar: Reliability https://docs.aws.amazon.com/wellarchitected/latest/framework/rely_adapt_to_changes_autoscale_adapt.html
<p>Collective Justification: The justifications for the listed technical requirements (TRs) predominantly revolve around enhancing the reliability of the architecture. For instance, TR7.1 emphasizes the implementation of auto-scaling mechanisms to dynamically handle varying workloads, contributing to the pillar of reliability. Similarly, TR8.1 encompasses various strategies, such as managing quota increase requests, establishing redundant connectivity, implementing loose coupling with load balancers, monitoring demand, and workload utilization, and automating resource addition or removal to align with demand. These actions collectively aim to proactively adapt to changes, prevent failures, and efficiently manage resources, ultimately reinforcing the reliability of the system. Additionally, TR8.6 and TR8.7 specifically address data protection and recovery, while TR8.8 underscores the use of automation to reduce errors and enhance the reliability of resource obtaining and scaling processes. Each technical requirement aligns with the pillar of reliability and contributes to building a robust and dependable architecture.</p>			
TR9.1	Implement a process to identify and decommission unused resources and automate this process	BR9 - Implement a Cost-Effective Approach Justification: This ensures you shut down or terminates unused resources to reduce waste.	Pillar: Cost efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_decommissioning_resources_implement_process.html
TR9.2	Configure billing and cost management tools	BR9 - Implement a Cost-Effective Approach Justification: Configuration of services, tools, and	Pillar: Cost efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_decommissioning_resources_implement_process.html

		resources to organize and track cost and usage data and further lower cost with resources and pricing optimizations.	d/latest/framework/cost_monitor_usage_configuration_tools.html
TR9.3	Provision resources dynamically	BR9 - Implement a Cost-Effective Approach Justification: This demand based or time based provisioning helps in the least amount of over-provisioning or under-provisioning thereby reducing the cost	Pillar: Cost efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_manage_demand_resources_dynamic.html
TR9.4	Implement dashboards to monitor cost proactively for the workload.	BR9 - Implement a Cost-Effective Approach Justification: This helps us to regularly review the costs with configured tools and monitor/analyze any unnecessary costs	Pillar: Cost efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_proactive_process.html
TR9.5	Use data compression techniques to reduce storage costs and optimize data storage.	BR9 - Implement a Cost-Effective Approach Justification: This optimizes storage costs and efficiency	Pillar: Cost efficiency
<p>Collective Justification: The justifications for TRs for BR9 focus on implementing a cost-effective approach, aligning with the pillar of cost efficiency. TR9.1 underscores the importance of identifying and decommissioning unused resources through automated processes, reducing waste and minimizing unnecessary costs. TR9.2 emphasizes the configuration of billing and cost management tools to organize and track cost and usage data, enabling resource and pricing optimizations for overall cost reduction. TR9.3 advocates for dynamic provisioning of resources based on demand or time, aiming to minimize over-provisioning or under-provisioning and thus, lowering costs. TR9.4 encourages the implementation of dashboards to proactively monitor costs, allowing for regular reviews and</p>			

analysis to identify and rectify unnecessary costs. Lastly, TR9.5 promotes the use of data compression techniques to optimize storage costs and improve efficiency, contributing to the overall cost-effectiveness of the architecture. Each of these strategies collectively reinforces the pillar of cost efficiency within the well-architected framework.

<u>TR10.1</u>	Implement an appropriate usage-based billing or price model	BR10 - Manage billing and subscriptions. Justification: This will help to run systems to deliver business value at the lowest price point with various models like “pay-as-you-go” etc.	Pillar: Cost efficiency https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_select_service_analyze_all.html
<u>TR11.1</u>	Choose regions with less carbon footprint for the workload	BR11 - Implement the eco-conscious and sustainable architecture Justification: Placing a workload close to Amazon renewable energy projects or Regions with low published carbon intensity can help to lower the carbon footprint of a cloud workload thereby improving sustainability	Pillar: Sustainability https://docs.aws.amazon.com/wellarchitected/latest/framework/sus_sus_region_a2.html
<u>TR12.1</u>	Automate build, deployment, and testing of the workload.	BR12 - The online application should be operated with high excellence Justification - Continuous optimization	Pillar: Operational Excellence https://docs.aws.amazon.com/wellarchitected/latest/framework/ops_dev_integ_auto_int

		techniques will streamline the application, enhancing code efficiency, database queries, and network communication. This reduces errors caused by manual processes and reduces the effort to deploy changes.	eg_deploy.html
TR13.1	Monitor and track the workload resource utilization and metrics (ex: application performance metrics, page load speeds, server resource utilization)	BR13 - Implement monitoring and alerting mechanisms. Justification: Identify the KPIs that indicate whether the workload is performing as intended helps to identify degradation and remediate internal or external factors thereby increasing performance	Pillar: Performance efficiency Pillar: Reliability
TR13.2	Implement real-time monitoring and alerting mechanisms to trace insecure actions and changes within the environment.	BR13 - Implement monitoring and alerting mechanisms. Justification: This will track and help us to investigate any insecure actions thereby promoting security.	Pillar: Security
TR14.1	Implement efficient indexing and search capabilities to allow	BR14 - Implement Data storage solutions.	Pillar: Performance efficiency

	users to find books	Justification: It enhances the ability to find books, which is an essential data management requirement.	
TR14.2	Use data compression techniques to reduce storage costs and optimize data storage.	BR14 - Implement Data storage solutions. Justification: This optimizes storage costs and efficiency	Pillar: Cost efficiency
Collective Justification: TR14.1 addresses the imperative to enhance book discovery, crucial for effective data management. This aligns with the performance efficiency pillar, emphasizing improved user experience. On the other hand, TR14.2 responds to the need for cost-effective data storage, aligning with the cost efficiency pillar. By implementing data compression techniques, this initiative aims to optimize storage costs, ensuring an economically viable solution. The combined effect of these efforts is a comprehensive strategy that not only improves performance and user experience but also ensures efficient and cost-effective data storage for the application.			
TR15.1	Implement Seamless Integration with External Platforms	BR15 - Integrate with external platforms to support multiple payment gateways and for analytics Justification: Integrating with external platforms enables diverse payment options, enhancing user convenience and expanding the customer base.	
TR15.2	Establish Secure Data Pipelines for External Platform Integration	BR15 - Integrate with external platforms to support multiple payment	

		gateways and for analytics Justification: Establishing secure data pipelines for external platform integration ensures the safe and compliant exchange of sensitive user information, enhancing payment gateway transactions and analytics processes. This approach guarantees data privacy, regulatory adherence, and user trust.	
Collective Justification: The justification for TR15.1 centers on the enhancement of user experience by seamlessly integrating with external platforms, providing diverse payment options that increase user convenience and expand the customer base. In TR15.2, the emphasis is on establishing secure data pipelines for external platform integration to ensure the safe and compliant exchange of sensitive user information. This not only enhances payment gateway transactions and analytics processes but also guarantees data privacy, regulatory adherence, and fosters user trust by implementing a secure and reliable integration framework.			

2.4 Tradeoffs

S.NO	Conflicting TR's	Discussion	Team member
1)	Performance Vs Security TR- 1.1 Vs TR- 2.4 TR1.1: Select and provision	TR1.1 - High-performance compute resources can provide faster response times, reduced latency, and the ability to process large volumes of data efficiently. TR2.4 - Encrypting data at rest and in transit helps	Tejaswini Panati

	high-performance compute capacity TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols	<p>protect sensitive information from unauthorized access and breaches.</p> <p>Compliance with industry-standard encryption protocols ensures that your application aligns with best practices for security, which is often a requirement for regulatory compliance (e.g., GDPR, HIPAA, etc.).</p> <p>Comment on tradeoff -</p> <ul style="list-style-type: none"> • If you prioritize high-performance compute capacity, you can ensure LitShelf runs efficiently and handles demanding workloads effectively. • It may be more responsive and capable of delivering a better user experience for browsing and lending books. • At the same time, emphasizing data encryption is vital for safeguarding sensitive information, maintaining data integrity, and meeting regulatory requirements. • However, encryption can introduce overhead, potentially impacting the application's performance. Slower data transfer and increased compute (CPU) usage may lead to increased latency or reduced throughput affecting the performance and hence this tradeoff needs to be considered 	
2)	<p>Availability Vs Cost Efficiency</p> <p>TR-4.1 Vs TR-9.5</p> <p>TR4.1: Utilize redundant servers, and failover mechanisms.</p> <p>TR9.5: Use data</p>	<p>TR4.1 - Having redundant servers and failover mechanisms ensures high availability and fault tolerance. In the event of hardware failures or network issues, the workload can seamlessly switch to backup systems, minimizing downtime and ensuring continuous service availability.</p> <p>TR9.5 - Implementing data compression techniques reduces storage costs by optimizing data storage. Compressed data requires less storage space, lowering the overall storage costs. This approach is particularly beneficial when dealing with large volumes of data, saving costs in the long run.</p>	Sunandini Med...

	compression techniques to reduce storage costs and optimize data storage.	Comment on tradeoff - <ul style="list-style-type: none"> ● Redundant servers and failover mechanisms ensure a reliable user experience during peak usage or unexpected failures. ● Seamless switching to backup systems minimizes downtime and ensures continuous service availability. ● Data compression techniques reduce storage costs by optimizing data storage. Compressed data requires less storage space, lowering overall storage expenses. ● Data compression may slightly impact data processing speed due to decompression needs during data access. ● Tradeoff involves a potential minor slowdown in data processing speed for reduced storage costs. ● The choice between high availability and storage optimization should align with project budget constraints and user experience requirements. ● Balancing operational costs with the need for high availability and efficient storage is crucial in decision-making. 	
--	---	---	--

3. Provider Selection

3.1 Criteria for choosing a provider

Based on the literature - [5] and [6], the criteria we plan to use for choosing the provider is as follows :

- Service Offering: We want to ensure that the cloud provider offers the specific services needed to fulfill the TRs listed in Section 2.3. Various such cloud services which we require includes -
 - Compute Service: To host and run the LitShelf application.
 - Storage Service: To store user data and files.
 - Database Service: For data management and retrieval.
 - Network Service: For connectivity and data transfer.
 - Monitoring service: To track and monitor the workload resource utilization and other metrics
 - DNS (Domain Name System): To map domain names to IP addresses.
 - Identity and Access Management services: For tenant identification and access
 - Cost Management and Billing Tools: To monitor and optimize expenses.
 - Security Certificate Provisioning and Management: To ensure secure data transmission.
 - Key Management Service: To safeguard sensitive data.
 - Load balancing service: Services that allow you to scale your infrastructure horizontally and manage traffic distribution effectively.
 - Autoscaling: For automatic resource provisioning based on demand.
 - Backup and disaster recovery service: To safeguard against data loss.
 - Automation and Orchestration Tools: Tools for automating infrastructure provisioning and management
 - Compliance and Governance Services: ensuring regulatory compliance and governance

- Cost/ Pricing Structure: One other criteria is to assess the pricing model and make sure it fits our budget by considering factors like pay-as-you-go, reserved instances, and any potential discounts for long-term commitments.

- SLAs: We also wish to verify if the corresponding service level agreement captures these 3 components -
 - Service level objectives - specific, measurable goals or performance targets that the service provider commits to meeting. These objectives define the expected levels of service quality, such as uptime, response times, availability, or error rates. SLOs are quantifiable and are typically expressed as percentages or time frames
 - Remediation policies and penalties/incentives related to these objectives - outline the steps the service provider will take to address deviations from the defined SLOs. They specify how the provider will respond when service performance falls below the agreed-upon standards. Penalties and incentives are provisions

within SLAs that establish financial consequences for not meeting SLOs or rewards for exceeding them.

- Exclusions and caveats - specify the conditions or circumstances under which the SLA's commitments do not apply or are subject to modification.

Based on the above mentioned criteria , the following section compares the three major cloud service providers - AWS, Azure and GCP and review which of them satisfies the technical requirements to design the cloud infrastructure for LitShelf

3.2 Provider comparison

Considering the service offering to fulfill all the specific TRs in section 2.3 criteria, we compare and rank the major cloud providers - AWS, Azure and GCP below -

Note: We use the corresponding service catalogs for each provider to perform this comparison:

Service:

1. Compute Service:

a. AWS

(<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/compute-services.html>)

- i. Amazon EC2
- ii. AWS Lambda
- iii. AWS ECS and EKS (container orchestration services)

b. Azure (<https://azure.microsoft.com/en-us/products/category/compute>)

- i. Virtual Machines
- ii. Azure App Service
- iii. Azure Functions
- iv. Azure Container Instances
- v. Azure Kubernetes Services

c. GCP (<https://cloud.google.com/products/compute>)

- i. Google Compute Engine
- ii. App Engine
- iii. Google Kubernetes Engine
- iv. Cloud Run

2. Storage Service:

a. AWS

(<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/storage-services.html>)

- i. Amazon Elastic Block Store
 - ii. Amazon Elastic File System
 - iii. Amazon Simple Storage Service
- b. Azure
 - (<https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction#review-options-for-storing-data-in-azure>)
 - i. Azure Files
 - ii. Azure Blobs
 - iii. Azure Elastic SAN
 - iv. Azure Disks
- c. GCP (<https://cloud.google.com/products/#section-24>)
 - i. Cloud Storage
 - ii. Filestore
 - iii. Persistent Disk

3. Database Service:

- a. AWS
 - (<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/database.html>)
 - i. Amazon Relational Database Service (Relational)
 - ii. Amazon Aurora (Relational)
 - iii. Amazon DynamoDB (Key-value)
 - iv. Amazon ElastiCache (In-memory)
- b. Azure (<https://azure.microsoft.com/en-us/products/category/databases>)
 - i. Azure CosmosDB
 - ii. Azure SQL Database
 - iii. Azure Database for PostgreSQL
 - iv. Azure Database for MariaDB
- c. GCP (<https://cloud.google.com/products/#section-8>)
 - i. Cloud BigTable
 - ii. Cloud SQL
 - iii. Firestore
 - iv. Memorystore

4. Networking Service:

- a. AWS
 - (<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/networking-services.html>)
 - i. Amazon API Gateway
 - ii. Amazon VPC
 - iii. Amazon Direct Connect
- b. Azure (<https://azure.microsoft.com/en-us/products/category/networking>)
 - i. Azure Virtual Network

- ii. Azure ExpressRoute
- c. GCP (<https://cloud.google.com/products/#section-19>)
 - i. Cloud CDN
 - ii. VPC
 - iii. Cloud Interconnect
 - iv. Cloud NAT

5. Monitoring Service:

- a. AWS
 - (<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/management-governance.html>)
 - i. Amazon CloudWatch
 - ii. AWS Health Dashboard
 - iii. AWS CloudTrail
 - iv. Amazon GuardDuty
 - v. Amazon Inspector
- b. Azure (<https://azure.microsoft.com/en-us/products/category/management>)
 - i. Azure Monitor
 - ii. Network Watcher
- c. GCP (<https://cloud.google.com/products/#section-20>)
 - i. Cloud Debugger
 - ii. Cloud Monitoring
 - iii. Cloud Logging
 - iv. Cloud Profiler

6. DNS Service:

- a. AWS
 - (<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/networking-services.html#amazon-route-53>)
 - i. Amazon Route 53
- b. Azure (<https://azure.microsoft.com/en-us/products/category/networking>)
 - i. Azure DNS
- c. GCP (<https://cloud.google.com/products/#section-19>)
 - i. Cloud DNS

7. Identity and Access Management Services:

- a. AWS
 - (<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/security-services.html>)
 - i. AWS Identity and Access Management
 - ii. AWS Cognito
- b. Azure (<https://azure.microsoft.com/en-us/products/category/identity>)
 - i. Azure Active Directory (Azure AD)

- c. GCP (<https://cloud.google.com/products/#section-22>)
 - i. Identity and Access Management

8. Cost Management and Billing tools

- a. AWS (<https://docs.aws.amazon.com/account-billing/>)
 - i. AWS Cost Explorer
 - ii. AWS Budgets
 - iii. AWS Billing and Cost Management Dashboard
- b. Azure (<https://learn.microsoft.com/en-us/azure/cost-management-billing/>)
 - i. Azure Cost Management and Billing
 - ii. Azure Budgets
 - iii. Azure Pricing Calculator
- c. GCP (<https://cloud.google.com/billing/docs>)
 - i. Google Cloud Billing Reports
 - ii. Cost Management Tools

9. Key Management Service

- a. AWS (<https://aws.amazon.com/kms/>)
 - i. AWS Key Management Service (KMS)
- b. Azure
(<https://learn.microsoft.com/en-us/azure/security/fundamentals/key-management>)
 - i. Azure Key Vault
- c. GCP (<https://cloud.google.com/security-key-management?hl=en>)
 - i. Google Cloud Key Management Service (KMS)

10. Load balancing service

- a. AWS
(<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/load-balancer-types.html>)
 - i. AWS Elastic Load Balancing (ELB)
 - ii. Application Load Balancer
 - iii. Network Load Balancer
 - iv. Classic Load Balancer
- b. Azure
(<https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/load-balancing-overview>)
 - i. Azure Load Balancer
- c. GCP (<https://cloud.google.com/load-balancing?hl=en>)
 - i. Google Cloud Load Balancing

11. Auto Scaling service

- a. AWS
(<https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html>)

- i. AWS Auto Scaling
 - ii. Target Tracking Scaling
 - iii. Simple Scaling
 - iv. Step Scaling
- b. Azure
(<https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview>)
 - i. Azure Autoscale
- c. GCP (<https://cloud.google.com/compute/docs/autoscaler>)
 - i. Google Cloud Autoscaler

12. Backup and disaster recovery service

- a. AWS
(<https://aws.amazon.com/local/hongkong/solutions/backup-disaster-recovery/>)
 - i. AWS Backup
 - ii. AWS Disaster Recovery
- b. Azure
(<https://learn.microsoft.com/en-us/azure/well-architected/resiliency/backup-and-recovery>)
 - i. Azure Backup
 - ii. Azure Site Recovery
- c. GCP (<https://cloud.google.com/backup-disaster-recovery#>)
 - i. Google Cloud Backup and Restore
 - ii. Google Cloud Disaster Recovery (Cloud Endure Disaster Recovery)

13. Automation and orchestration service

- a. AWS
(<https://docs.aws.amazon.com/whitepapers/latest/next-generation-oss/service-orchestration.html>)
 - i. AWS Systems Manager
 - ii. Automation
 - iii. AWS Step Functions
- b. Azure (<https://learn.microsoft.com/en-us/azure/automation/automation-services>)
 - i. Azure Automation
 - ii. Azure Logic Apps
- c. GCP
(<https://cloud.google.com/blog/topics/developers-practitioners/service-orchestration-google-cloud>)
 - i. Google Cloud Composer
 - ii. Google Cloud Workflows

14. Compliance and Governance service:

- a. AWS
(<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/management-governance.html#aws-config>)
 - i. AWS Config
- b. Azure
(<https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/govern/guides/>)
 - i. Azure Policy (<https://azure.microsoft.com/en-us/services/azure-policy/>)
 - ii. Azure Blueprints (<https://azure.microsoft.com/en-us/services/blueprints/>)
 - iii. Azure Security Center
(<https://azure.microsoft.com/en-us/services/security-center/>)
- c. GCP:
 - i. Google Cloud Security Command Center
(<https://cloud.google.com/security-command-center>)
 - ii. Google Cloud Identity and Access Management (IAM)
(<https://cloud.google.com/iam/>)

Now moving on to the second criteria, cost and pricing structure , we compare AWS, Azure and GCP :

To evaluate the cost and pricing structure of the three major cloud providers – AWS, Azure, and GCP – we employ a systematic approach by utilizing cost analyzer tools and reviewing relevant documentation.

For AWS [8], we utilize the AWS Pricing Calculator (<https://calculator.aws/#/>) to analyze and estimate costs based on our specific technical requirements (TRs). The pricing calculator allows us to input various parameters, such as the type and amount of resources needed, and provides an estimate of the associated costs. AWS follows a pay-as-you-go pricing model, where users are billed for the actual resources consumed. Additionally, AWS offers various pricing plans, such as reserved instances and spot instances, providing flexibility in cost optimization.

Similarly, for Azure, we leverage the Azure Pricing Calculator (<https://azure.microsoft.com/en-us/pricing/calculator/>) to estimate costs tailored to our TRs. Azure adopts a similar pay-as-you-go model and provides various pricing options, including reserved instances and hybrid benefits. The calculator assists in understanding the cost implications of different configurations and services.

In the case of GCP, the GCP Pricing Calculator (<https://cloud.google.com/products/calculator>) is used to project costs based on our TRs. GCP, like its counterparts, follows a pay-as-you-go

pricing model and offers sustained use discounts. The pricing calculator facilitates a detailed breakdown of costs associated with different GCP services.

Upon reviewing the cost analyzer tools and related documentation for all three providers, it is evident that each cloud provider offers transparency in pricing, allowing users to estimate and understand the cost implications of their chosen services. Fair pricing is achieved through the provision of various pricing models, enabling users to optimize costs based on their specific usage patterns and requirements.

In summary, the second criterion is met by all three cloud providers, as they provide accessible tools for cost estimation and adopt fair pricing structures. Users can make informed decisions regarding cost management and optimization based on their unique needs and usage patterns. This ensures that the cost and pricing structures of AWS, Azure, and GCP align with our technical and budgetary considerations.

Finally moving to the third criteria, we study and verify the SLAs in each provider case to ensure they contain the three essential components as mentioned in section 3.1:

For AWS [8] , inspecting the SLA document for one such service - compute service (https://aws.amazon.com/compute/sla/?did=sla_card&trk=sla_card) , we identify that all the three components are specified i.e the SLAs in terms monthly uptime percentage and what will be the corresponding service credit percentage. Further it details on how to claim these incentives i.e SLA credits and finally talks about the AWS Compute SLA exclusion caveats. This is the same with all the above services that satisfy our specific technical requirements.

Similarly for GCP [9], we inspect the SLA document for storage service (<https://cloud.google.com/storage/sla>). The SLOs defined in this case is the uptime percentage as well for each type of storage class in the cloud storage. Further it details the percentage of monthly bill for cloud storage that does not meet SLO that will be credited to customer's future monthly bills as the remediation or incentives related to the objective. Finally it also includes SLA exclusions. It was identified that a similar SLA document was present for the services mentioned above.

For Azure, we inspect the SLA for Azure Expressroute (<https://www.azure.cn/en-us/support/sla/expressroute/>). As the SLO , they guarantee a minimum of 99.95% of dedicated circuit availability. Maximum Available Minutes, Monthly Uptime Percentage are some of the SLOs. Further they indicate the various Service credit claims as the percentage of monthly service fees for the affected service or service resources that is credited to a customer for a validated claim. Finally the SLA exclusions are detailed.

Upon reviewing the SLA documents to assess the fulfillment of the third criterion, it was determined that all service providers had accurately and comprehensively documented the corresponding entities for the various services that support our Technical Requirements (TRs). As a result, it can be confidently stated that this criterion has been successfully met.

Given that the three major cloud providers - AWS, Azure, and GCP - meet the criteria mentioned above, it appears that these providers are equally competitive. Hence we can choose any cloud provider in this scenario and have our use case or technical requirements satisfied. Therefore, in this scenario, we have the flexibility to select any cloud provider, ensuring that our technical requirements are adequately addressed.

3.3 Final selection

For our use case, LitShelf - cloud IaaS application, we choose **AWS** as the cloud service provider.

3.3.1 List of services offered by the winner

The various services offered by AWS for each category mentioned above is available here - <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/amazon-web-services-cloud-platform.html>. Further the services used to satisfy each TR are detailed in section 4 with reasoning.

Elaborating on 4 such services of our choice that will be used in this design is as follows:

Team member : Sunandini Mediseti

- 1) **Amazon VPC:** ([4]) Amazon VPC allows us to create a private, isolated section of the Amazon Web Services (AWS) cloud where you can launch AWS resources in a virtual network that we define. With Amazon VPC, we plan to customize our network configuration, such as selecting our own IP address range, creating subnets, and configuring route tables and network gateways. This enables us to establish a secure and seamless connection between our on-premises data centers and cloud-based resources. Amazon VPC offers features like network access control lists (ACLs) and security groups, allowing us to define rules for inbound and outbound traffic, ensuring a secure environment for our applications. It also supports the creation of Virtual Private Networks (VPNs) and Direct Connect.
- 2) **AWS CloudWatch:** ([5]) Amazon CloudWatch is a monitoring and observability service provided by Amazon Web Services. It allows users to collect and track metrics, collect and monitor log files, and set alarms. With CloudWatch, users can gain valuable insights into their applications, resources, and services running on the AWS platform. It provides

detailed data and analytics, enabling users to monitor performance, troubleshoot issues, and optimize their infrastructure and applications for better efficiency. CloudWatch offers a wide range of monitoring capabilities, including custom metrics, automated actions, and dashboards, allowing users to visualize and understand their system's behavior and performance over time. By utilizing CloudWatch, businesses can ensure the reliable performance of their applications, maintain a seamless user experience, and proactively respond to any operational challenges, thereby enhancing the overall reliability and availability of their AWS-based solutions.

Team member : Tejaswini Panati

- 3) **Amazon Elastic Load Balancing (AWS ELB)** ([16] and [17]): AWS ELB is designed to distribute incoming application traffic across multiple Amazon Elastic Compute Cloud (EC2) instances or other resources within the AWS infrastructure. It ensures that the workload is evenly spread, which enhances the availability and fault tolerance of your applications. offers different types of ELB, including Application Load Balancers (ALB) for HTTP/HTTPS traffic, Network Load Balancers (NLB) for TCP/UDP traffic, and Classic Load Balancers, which are the traditional type for basic load balancing needs.ELB can be used in conjunction with Auto Scaling to create highly scalable and fault-tolerant applications thereby supporting the availability and reliability technical requirements for the application.
- 4) **AWS Identity and Management service (AWS IAM)** ([18] and [19]): AWS Identity and Access Management (IAM) enables secure control over access to AWS resources. IAM allows users to centrally manage permissions for accessing AWS resources, controlling both authentication (signing in) and authorization (permissions) for resource usage. Users can be assigned different permissions for various resources, allowing fine-grained control. IAM allows secure credential management for applications running on Amazon EC2 instances, granting them permissions to access other AWS resources. It helps with tenant identification by allowing you to create and manage distinct identities (users, groups, roles) for each tenant or customer within a multi-tenant environment. Each tenant, whether it's a customer, department, or organization, can have its own set of IAM users, groups, and roles. These identities are unique and separate from one another.

4. The First Design Draft

Before we go ahead with the design, we establish the critical technical requirements that we aim to solve with this design. Following the instructor notes and lectures on 10/30, the following subset has been picked from the list of TRs in section 2.3

Technical Requirement	Category
TR2.1 - Implement a mechanism for tenant identification and access	(Tenant Identification)
TR1.5 - Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization)	Performance efficiency Pillar (Monitoring)
TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers	Performance efficiency Pillar (Elasticity)
TR7.1 - Implement a system to auto-scale horizontally and vertically to accommodate time varying payloads	Reliability Pillar (Elasticity)
TR1.1 - Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols	Performance efficiency Pillar Vs Security Pillar (Conflicting TRs)
TR1.2 - Select a robust database option (relational, key value , graph etc) for data storage.	Performance efficiency Pillar
TR1.3 - Choose a robust and suitable storage solution	Performance efficiency Pillar
TR1.4 - Configure an optimal networking solution	Performance efficiency Pillar
TR4.1: Utilize redundant servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage.	Availability Vs Cost Efficiency Pillar
TR8.8 - Use automation when obtaining or scaling resources	Reliability Pillar (Automation - IaC)
TR10.1 - Implement an appropriate	Cost efficiency pillar

usage-based billing or price model	
------------------------------------	--

4.1 Basic Building Blocks of the design

The basic building blocks of the design includes the various services identified to satisfy the subset of the TRs we aim to base our design on:

S.No	Technical Requirement	Purpose	AWS Service to use (Building Block)
1.	TR2.1 - Implement a mechanism for tenant identification and access	Tenant Identification	AWS Identity and Access Management
2.	TR1.5 - Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization)	Monitoring	AWS CloudWatch and AWS Simple Notification as service
3.	TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers	Load Balancing	AWS Elastic Load Balancer
4.	TR7.1 - Implement a system to auto-scale horizontally and vertically to accommodate time varying payloads	Auto scaling	AWS Auto Scaling and AWS AutoScaling Groups
5.	TR1.1 - Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols	Compute	AWS EC2 Instance
		Encryption at rest	AWS Key Management Service and AWS Certificate Manager
6.	TR1.2 - Select a robust database option (relational, key value , graph etc) for data storage.	Database	AWS RDS
7.	TR1.3 - Choose a robust and suitable storage solution	Storage	AWS S3

8.	TR1.4 - Configure an optimal networking solution	Networking	AWS Virtual Private VPC (Subnets, Internet Gateway, Security Groups, Route Table)
9.	TR4.1: Utilize redundant servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage.	Server Redundancy	AWS Auto Scaling, Multi-AZ Deployments
		Data Storage Optimization	AWS S3 Object Storage with Compression Features
10.	TR8.8 - Use automation when obtaining or scaling resources	Automation (Infrastructure as a code)	AWS CloudFormation template
11.	TR10.1 - Implement an appropriate usage-based billing or price model	Billing / price model	AWS Billing and AWS Cost explorer

4.2 Top-level informal validation of the design

S.No	Technical Requirement	AWS Service	Reason
1.	TR1.1 - Select and provision high-performance compute capacity	AWS EC2 Instance	EC2 provides a wide variety of instance types with different CPU, memory, and storage configurations. This versatility allows you to select an instance type that best matches our specific requirements. EC2 instances offer full control over the virtual machine's configuration, including the choice of operating system, software, and application stack. This level of control is valuable for great customization in case of Litshelf bookstore application. Since we get to manage more aspects of the infrastructure with EC2 instances and hence making it compute intensive, they are preferred in this scenario. [21] [20]

2.	TR1.2 - Select a robust database option (relational, key value , graph etc) for data storage.	AWS RDS (Relational Database)	Litshelf needs a robust database to manage structured data such as book listings, user profiles, transaction records, and user preferences. RDS offers managed database services for various relational database engines, including MySQL, PostgreSQL, and others. It simplifies database management, including backups, patches, and scaling. For applications with structured data such as Litshelf , RDS is a strong choice, providing reliability, security, and scalability. [22] [23]
3.	TR1.3 - Choose a robust and suitable storage solution	AWS S3 (Simple Storage Service)	Litshelf online bookstore needs a robust storage solution for various data types, including book cover images, user-generated content, and binary files. AWS S3 is designed for high availability, durability, and scalability. It offers features like versioning, lifecycle policies, and fine-grained access controls, making it an excellent choice for storing unstructured and structured data. S3 ensures data availability and durability while providing cost-effective storage. [24] [25]
4.	TR1.4 - Configure an optimal networking solution	AWS Virtual Private VPC (Subnets, Internet Gateway, Security Groups, Route Table)	AWS VPC allows the creation of a private and isolated network environment. It provides essential networking components such as subnets, Internet Gateway, Security Groups, and Route Tables, enabling us to configure a secure and scalable network architecture. VPC ensures that your application's network traffic is controlled, isolated, and protected. Configuring a VPC for Litshelf gives us the ability for extreme customization by allowing us to create VPC spanning over various zones for resiliency and in what kind of subnets (private/public), we can place the App instances and DBs. Further it allows us to enable communication between the VPC and the internet via an Internet gateway. [26]
5.	TR1.5 - Monitor and track the workload resource utilization and operational metrics	AWS CloudWatch and AWS SNS	AWS CloudWatch is a comprehensive monitoring and observability service that offers deep insights into the performance and health of your AWS resources and applications. It collects and analyzes metrics, logs, and events from

	(ex: application performance metrics, page load speeds, server resource utilization)		<p>various AWS services, including Amazon EC2 instances, RDS databases, Amazon S3 buckets. This enables comprehensive monitoring across the entire infrastructure.</p> <p>For an IaaS application like LitShelf, the ability to scale resources based on workload and performance metrics is crucial. CloudWatch helps to trigger auto-scaling actions based on defined alarms, ensuring that the application can handle varying traffic loads efficiently. Further it enables us to set up alarms based on thresholds and anomaly detection, allowing you to receive automated notifications via various channels, including email, SMS, and AWS Simple Notification Service (SNS). [27]</p>
6.	TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers	AWS Elastic Load Balancer	<p>ELB distributes incoming traffic across multiple instances or resources in multiple Availability Zones. This enhances the application's availability and fault tolerance. In case one EC2 instance fails, ELB automatically routes traffic to healthy instances, ensuring uninterrupted service.</p> <p>ELB supports auto-scaling, allowing the application to dynamically adjust its capacity based on changing traffic loads. AsLitshelf experiences traffic spikes during special promotions or book releases, ELB ensures that traffic is evenly distributed. [28]</p>
7.	TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols	AWS Key Management Service and AWS Certificate Manager	<p>Litshelf includes sensitive data like user information, payment details, book listings, reviews, content metadata, logs, session tokens, and administrative data stored in a database (Amazon RDS). AWS Key Management Service (KMS) is used to encrypt this data when it's at rest, ensuring protection from unauthorized access and data breaches. KMS plays a crucial role in securing confidential user and payment information, maintaining data integrity, and meeting compliance standards for sensitive data in the cloud.</p> <p>ACM issues and manages SSL/TLS certificates that are used to secure the communication between clients (e.g., web browsers or mobile apps) and the Litshelf application's servers. This</p>

			encryption ensures that data transferred between the client and the server is protected from eavesdropping and interception during transmission. [29]
8.	TR2.1 - Implement a mechanism for secure tenant identification and access	AWS Identity and Access Management (IAM) + Amazon ElastiCache	<p>Litshelf has implemented a robust mechanism for secure tenant identification and access management within its cloud infrastructure. By leveraging the Amazon Web Services (AWS) Identity and Access Management (IAM) service, Litshelf achieves centralized permission management for accessing resources, ensuring a secure authentication and authorization framework. IAM provides fine-grained control, facilitating the creation and management of distinct identities such as users, groups, and roles tailored for each tenant or customer. These identities are meticulously maintained as unique and separate entities, guaranteeing secure, isolated access to specific resources within a multi-tenant environment.</p> <p>In addition to IAM, Litshelf enhances its system by integrating Amazon ElastiCache for an efficient caching mechanism. This integration optimizes performance and ensures that frequently accessed data is readily available, contributing to an improved user experience. The caching mechanism enhances the responsiveness of Litshelf's cloud infrastructure, providing a seamless and swift experience for tenants and customers.</p> <p>This dual-layered approach not only guarantees that tenant-specific data and applications are accessed only by authorized individuals or entities but also contributes to maintaining a high level of security, privacy, and system performance within Litshelf's cloud environment.[31]</p>
9.	TR4.1: Utilize redundant servers, and failover mechanisms.	AWS Elastic Load Balancing (ELB) for distributing	In order to enhance system reliability and minimize downtime, Litshelf implements redundant servers and failover mechanisms within its infrastructure. Leveraging AWS best practices, Litshelf deploys multiple servers

		traffic + Amazon Route 53 can be used for DNS + AWS Multi-AZ Deployment Model	across different Availability Zones, ensuring high availability and fault tolerance. In the event of a server failure, traffic seamlessly transitions to healthy instances, guaranteeing uninterrupted service for users. Additionally, Litshelf utilizes failover mechanisms, automatically redirecting traffic to backup servers or alternate data centers to prevent service disruptions. This proactive approach to redundancy and failover mechanisms ensures continuous operation, mitigates potential risks, and maintains a seamless user experience even in the face of unexpected server failures or other system disruptions.[32]
10.	TR7.1 - Implement a system to auto-scale horizontally and vertically to accommodate time varying payloads	AWS Auto Scaling Group	Litshelf's dynamic online bookstore experiences varying levels of traffic throughout the day, especially during promotions and new book releases. AWS Auto Scaling is the solution of choice for Litshelf to handle these fluctuations effectively. With horizontal scaling, AWS Auto Scaling automatically adds or removes instances based on defined policies, ensuring the platform can seamlessly accommodate increased user demand. Additionally, vertical scaling, achieved through services like Amazon EC2 Auto Scaling groups, enables Litshelf to adjust the capacity of its resources based on specific metrics, ensuring a smooth and responsive user experience during peak traffic hours. [33]
11.	TR8.8 - Use automation when obtaining or scaling resources	AWS CloudFormation	Automating the deployment and management of resources is key to Litshelf's efficient operation. AWS CloudFormation allows Litshelf to automate the provisioning and configuration of AWS infrastructure. By defining templates, Litshelf can create and manage a collection of resources, ensuring consistency and reducing the risk of errors in resource configurations. This automation simplifies the resource scaling process, enabling Litshelf to focus on delivering a seamless user experience while AWS CloudFormation handles the underlying infrastructure management seamlessly. [34]
12.	TR9.5:	Amazon S3,	Efficient data storage and management are vital

	Use data compression techniques to reduce storage costs and optimize data storage.	AWS Glue	for Litshelf. Amazon S3, with its robust features, including versioning, lifecycle policies, and fine-grained access controls, provides Litshelf with a scalable and cost-effective storage solution for various data types, including images and binary files. To further optimize data storage and minimize costs, Litshelf utilizes AWS Glue, which automates the process of compressing data during Extract, Transform, Load (ETL) operations. By leveraging these services, Litshelf ensures data availability, reduces storage expenses, and maintains an optimized and secure data storage environment. [35]
13.	TR10.1 - Implement an appropriate usage-based billing or price model	AWS Cost Explorer and AWS Pricing Calculator	Litshelf prioritizes cost-effectiveness while ensuring a seamless user experience. AWS Cost Explorer offers valuable insights into cost patterns, enabling Litshelf to monitor expenses and optimize resource usage. Additionally, AWS Pricing Calculator assists Litshelf in estimating the cost of its architecture based on usage patterns, services, and geographic regions. By leveraging these tools, Litshelf makes informed decisions about its usage-based billing or pricing model, ensuring efficient cost management and maximizing the value of its AWS resources. [36]

Comment on conflicting TRs (See section 5.6 for detailed analysis):

1) TR1.1 - Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols:

We learnt in section 2.3 that these are conflicting technical requirements for our application. Choosing Amazon EC2 instances for TR1.1 allows LitShelf to prioritize high-performance compute capacity, ensuring efficient and responsive application operation. Simultaneously, using AWS KMS and AWS Certificate Manager for TR2.4 addresses data encryption needs without a significant performance impact. AWS KMS efficiently manages encryption keys and ACM automates the management of SSL/TLS certificates, reducing overhead and allowing LitShelf to maintain data security while still benefiting from high-performance compute resources. This approach strikes a balance between performance and data protection.

2) TR4.1: Utilize redundant servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage :

The demand for redundant servers and failover mechanisms in TR4.1 prompts Litshelf to implement AWS Elastic Load Balancing (ELB) and Amazon Route 53 for DNS management, ensuring high availability and fault tolerance. This redundancy strategy, complemented by failover mechanisms, mitigates risks and maintains a seamless user experience. In a parallel scenario, TR9.5 emphasizes the importance of data compression techniques for efficient storage. Here, Litshelf leverages Amazon S3 and AWS Glue to optimize data storage, reduce costs, and ensure a secure environment. This approach aligns with the imperative of cost-effective data management. This strategic interplay showcases Litshelf's commitment to addressing conflicting technical requirements by striking a balance between system reliability and storage efficiency, ensuring optimal performance and resource utilization.

5. The Second Design

5.1. Use of the Well-Architected Framework

The Well-Architected Framework provides a structured approach to building secure, high-performing, resilient, and efficient infrastructure for applications. In this section a summary of the framework is presented highlighting the key pillars and the best practices which helps in the design process. It consists of six key pillars, and for each pillar, there are distinct steps and best practices to follow:

Operational Excellence:

- Define organizational processes to manage and operate workloads effectively.
- Create a culture that encourages experimentation and learning from failures.
- Develop standard operating procedures and documentation.
- Continuously refine and improve operations to optimize workload performance.

Security:

- Implement identity and access management (IAM) to control access to resources.
- Apply data protection measures, including encryption and key management.
- Use automated security tools for threat detection and remediation.
- Establish incident response and recovery plans for security breaches.

Reliability:

- Design for fault tolerance, ensuring that systems can withstand failures without downtime.
- Automatically scale workloads based on demand.
- Monitor applications and infrastructure to identify and respond to issues proactively.
- Use automation for resource provisioning and configuration management.

Performance Efficiency:

- Use cloud resources efficiently to avoid over-provisioning.
- Optimize workloads for cost and performance by selecting the right instance types, storage, and database solutions.
- Monitor performance and set performance targets.
- Continuously iterate on performance improvements based on data and feedback.

Cost Optimization:

- Monitor and manage costs by setting budgets and utilizing cost management tools.
- Apply resource tagging for cost allocation and accountability.
- Implement cloud cost optimization strategies, such as rightsizing and reserved instances.
- Leverage serverless and managed services to reduce operational overhead.

Further the steps of architecting that is followed in this design document is exactly followed from the instructor notes [2] section 4.4.1 as given here -

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.

5.2 Discussion of Pillars

Pillar: Performance Efficiency

Team Member: Tejaswini Panati

The Performance Efficiency pillar of the AWS Well-Architected Framework [30] focuses on optimizing the use of computing resources to meet system requirements and maintaining that efficiency as demand changes and technologies evolve. This pillar is crucial for designing high-performing, cost-effective, and scalable cloud-based workloads.

Design Principles:

- Democratize Advanced Technologies: Delegate complex tasks to your cloud vendor, allowing your team to consume advanced technologies as services. This frees your team to focus on product development rather than resource provisioning.
- Go Global in Minutes: Deploy workloads in multiple AWS Regions to provide lower latency and a better customer experience at minimal cost.
- Use Serverless Architectures: Eliminate the need to run and maintain physical servers by utilizing serverless computing for tasks such as storage and event hosting. This reduces operational burden and can lower transaction costs.
- Experiment More Often: Leverage virtual and automatable resources to conduct comparative testing using different instances, storage, or configurations.
- Consider Mechanical Sympathy: Align technology approaches with your workload goals, considering factors like data access patterns when selecting database or storage solutions.

Few design principles used in LitShelf cloud application design includes -

- Go global in minutes principle directly aligns with the idea of deploying workloads in multiple AWS Availability Zones / Regions to provide lower latency and a better customer experience globally (TR1.7). Going global in minutes is facilitated by cloud services that allow quick and easy deployment in various regions.
- The principle, Use Serverless Architectures, suggests focusing on high-performance compute capacity (TR1.1), which aligns with the serverless architecture concept of abstracting away physical servers and letting the cloud provider manage the infrastructure. Serverless architectures often involve on-demand, event-triggered compute resources rather than traditional server provisioning.

Best Practices for Performance Efficiency:

- Selection: Optimal solutions may combine multiple approaches and features to improve performance. AWS offers a variety of resource types and configurations to match workload needs. The various resources to be considered for optimal performance is detailed in the section below.
- Review: Regularly evaluate workload components to ensure alignment with evolving AWS services and technologies. Benchmark and load testing can help optimize architecture.

- **Monitoring:** Use monitoring metrics to identify and rectify performance issues before they impact customers. AWS CloudWatch and X-Ray help collect data and insights for operational health.
- **Tradeoffs:** Understand the tradeoffs between consistency, durability, and space versus time and latency. Use tradeoffs to improve performance while meeting workload requirements.

Resource Considerations for Performance Efficiency:

Compute: Select compute resources, such as instances, containers, or functions, that match workload requirements for performance and cost.

Storage: Choose from object, block, or file storage services based on access methods, access patterns, and requirements for scalability and durability.

Database: Select purpose-built database engines like relational, key-value, or in-memory databases to match specific workload needs. Focus on achieving performance while offloading database management tasks.

Network: Consider bandwidth, latency, jitter, and throughput requirements to optimize network performance. Leverage AWS network features and services for efficient data transfer.

Review and Evolution:

Continually review your workload to ensure alignment with the latest AWS technologies and approaches. Keep an eye on AWS releases that could positively impact performance efficiency.

Pillar: Security

Team Member: Sunandini Mediseti

Security is a foundational aspect of any cloud-based workload, ensuring confidentiality, integrity, and availability of resources.

Design Principles:

- **Apply a Strong Identity Foundation:** Establish and enforce a robust identity and access management strategy, ensuring proper authentication and authorization controls across your AWS environment.

- **Enable Traceability:** Implement comprehensive logging and monitoring to enable traceability and auditability of security events, helping to quickly identify and respond to potential security incidents.
- **Automate Security Best Practices:** Leverage automation to enforce security best practices, ensuring consistency and reducing the risk of human error in security configurations.
- **Protect Data in Transit and at Rest:** Implement encryption for data in transit and at rest, safeguarding sensitive information from unauthorized access throughout its lifecycle.
- **Keep People Away from Data:** Minimize human access to data by automating security responses and ensuring that only authorized personnel can access critical systems and information.

Few design principles used in LitShelf cloud application design includes -

- The principle, Apply a Strong Identity Foundation, aligns with LitShelf's approach to secure tenant identification and access management, as implemented using AWS Identity and Access Management (IAM). This ensures proper authentication and authorization controls, upholding a robust identity foundation.
- The design principle, Protect Data in Transit and at Rest, is reflected in LitShelf's strategy of using encryption protocols for secure communication. For instance, SSL/TLS certificates managed by AWS Certificate Manager contribute to securing data in transit, aligning with the principle of protecting data throughout its lifecycle.

Best Practices for Security:

- **Identity and Access Management:** Implement strong identity and access controls, regularly reviewing and refining permissions to align with the principle of applying a strong identity foundation.
- **Detection:** Utilize advanced threat detection tools and regularly conduct security audits to quickly identify and respond to potential security incidents, consistent with the principle of enabling traceability.
- **Automated Security Responses:** Integrate automated security responses to minimize human access to data and enforce security best practices, aligning with the principles of automating security best practices and keeping people away from data.
- **Data Encryption:** Implement encryption for sensitive data in transit and at rest, adhering to the principle of protecting data throughout its lifecycle.

- Incident Response: Establish and regularly test an incident response plan, ensuring quick and effective responses to potential security incidents, consistent with the principles of enabling traceability and protecting data.

Resource Considerations for Security:

Network Security: Leverage AWS network features and services to optimize network security, considering factors like bandwidth, latency, and throughput, aligned with the security best practices for network optimization.

Encryption: Choose appropriate encryption mechanisms for different data types, ensuring that data in transit and at rest is securely protected, as detailed in the principles of protecting data in transit and at rest.

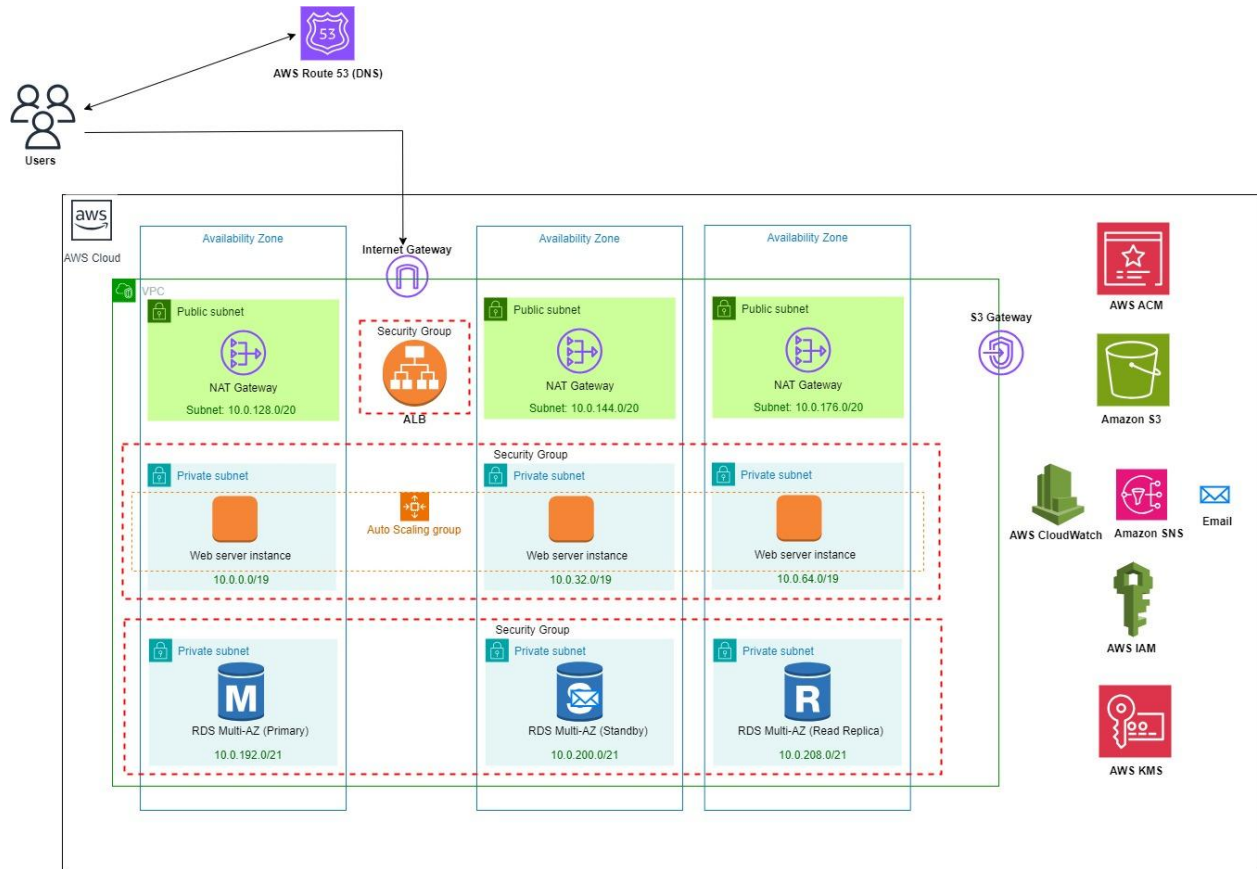
Access Control: Implement access controls at multiple layers, from identity management to resource-level permissions, in accordance with the principles of applying a strong identity foundation.

Review and Evolution:

Continually review security measures to ensure alignment with evolving AWS services and emerging threats. Regularly update security configurations and practices to address any potential vulnerabilities. Keep abreast of AWS releases and security updates to enhance the security posture of the workload.

5.3 Use of Cloud Formation Diagrams

The following architecture diagram represents the design of the Litshelf cloud application -



Design tool used - draw.io as suggested by AWS Architectural Icons [37]; source file - [AWS Architecture diagram](#)

From the above diagram,

Users generate the traffic destined to the web server that enters the VPC via the Internet Gateway whose address is obtained from the Route 53 DNS service. The internet gateway forwards the traffic to the public facing elastic load balancer (Application load balancer). The load balancer runs the algorithm to distribute the traffic to the three web sever EC2 instances hosted in 3 different availability zones in the same region. The VPC is associated with a route table (main route table) which contains the default local route for traffic routing/ communication within the VPC. Further the public subnet is associated with a custom route table which has both the default local route for inter-VPC communication and all other IPv4 subnet traffic from the public subnet to the internet over the internet gateway. In the case of the private subnets, they are associated with a route table with an entry for default local route for inter-VPC communication and the second entry that routes all other IPv4 subnet traffic from the private subnet to your network/ internet over the virtual NAT gateway. EC2 webserver and RDS instances are stored in these subnets and do not interact with the internet directly. This enhances security by isolating them from direct public access. Security Groups are attached to the load balancer and both EC2 and RDS instances to control inbound and outbound traffic. This helps enforce security policies and

restrict access based on rules (mainly allow HTTP, HTTPS and SSH traffic to the web server and additionally SQL traffic to RDS). Autoscaling is configured and implemented on the EC2 web server instances present in the private subnets. This ensures that the number of instances adjusts automatically based on demand. AWS CloudWatch communicates with the Load Balancer for health information. This setup facilitates high-performance monitoring and logging of the application. Further alarms are configured accordingly for auto scaling and alerting via AWS simple notification as a service to your email. IAM roles and policies are configured separately for different subnet groups in the Amazon VPC and further for the tenants to facilitate identification. AWS Certificate Manager (ACM) is used for managing SSL/TLS certificates. In a scenario where user traffic is received from the internet, the Load Balancer likely terminates SSL/TLS connections. ACM allows you to provision, manage, and deploy SSL/TLS certificates for securing communication between users and the load balancer. AWS Key Management service (KMS) is used for key management, and it can be involved in encrypting sensitive data. Further AWS Simple Storage Service (AWS S3) is used for storing book cover images or ebooks or any user uploads and is accessed from the VPC via the S3 Gateway Endpoint.

5.4 Validation of Design

A) TR1.1 - Select and provision high-performance compute capacity

Design Choice: AWS EC2 instance.

We deploy the online bookstore web application as a monolithic application running on a t2 micro instance in the private subnet in each of the three availability zones. It hosts various functionalities - Browse for books, view all book listings, View each book's details, Read Book reviews, Purchase books (payment processing) and user registration/authentication.

Validation of design choice: EC2 instances offer the necessary flexibility (general purpose to compute-optimized) to deploy the web server application because as an architect we have the entire choice of CPU, memory and storage that can be chosen for the instance housing the application. The bookstore application requirement moderate computational capabilities, choosing t2 micro instances suffices and satisfies the above requirement for our use case. Further the application is stored as an AMI (Machine Image) and the instances are deployed via an Autoscaling group with a launch template referencing the corresponding AMI in three different AZ's private subnet (for security). An Elastic load balancer (Application load balancer) distributes the received HTTP/HTTPS traffic to the corresponding instances. Amazon EC2 employs live migration for instances when there is a need to relocate them from one server to another.

This relocation occurs for purposes such as hardware maintenance without disrupting the service , hence making it an ideal choice in this use case.

B) TR1.2 - Select a robust database option (relational, key value , graph etc) for data storage.

Design Choice: AWS RDS (Relational Database)

The Database servers are deployed in the private subnets across the three AZs and isolated from the external world for increased security. They are connected to the EC2 instance which are deployed in the private subnets too. The Amazon RDS Database service is used to launch the database in Multi-AZ mode to bring High Availability and use Amazon RDS Read-Replicas feature to separate the writes and read requests. Primarily the database will house different DB tables - user data (store user profile (username, passwords (hashed), email addresses and user preferences), Book data (store book info including titles, authors, description, ISBN No and cover image references), Book reviews (user book reviews along with user ratings and comments or user generated content), Payment transactions table (when users purchase books) and use the MySQL DB engine.

Validation of design choice: Amazon RDS supports the relational database model, which is well-suited for structured data and relationships between different data entities. This is particularly relevant for online bookstore application that require organized and related data as described above. Deploying RDS in Multi-AZ mode enhances reliability by automatically replicating the database to a standby instance in a different Availability Zone. This ensures that in the event of a hardware failure or maintenance, there is minimal disruption, contributing to high availability. Leveraging the Read-Replicas feature allows the separation of read and write requests. This improves scalability by distributing read traffic across multiple database instances, reducing the load on the primary database. It's beneficial for our application with a high read-to-write ratio. Amazon RDS provides automated backups and point-in-time recovery, ensuring data durability. This feature allows you to restore your database to any point in time within the retention period, offering protection against accidental data loss or corruption. RDS is a managed service, handling routine database management tasks such as patching, backups, and hardware provisioning. This reduces the operational overhead. RDS provides options to vertically scale (scale up) by increasing the compute and memory resources of the instance, or horizontally scale (scale out) by adding Read-Replicas. This flexibility accommodates growing workloads therefore making it an ideal choice for our use case.

C) TR1.3 - Choose a robust and suitable storage solution.

Design Choice: AWS S3 (Simple Storage Service)

AWS S3 is utilized to store the user generated content like book reviews or images and book cover images for the users to view the cover images while they browse. The DB in the VPC's private subnet connects to the S3 via the S3 gateway endpoint and use the references/ links specified within the DB to identify the location of the S3 content. This allows applications to retrieve and display the relevant content stored in S3.

Validation of the design choice: AWS S3 is designed to provide scalable object storage. It can handle a virtually unlimited amount of data, making it suitable for storing large amounts of user-generated content, including book reviews and images. S3 automatically replicates data across multiple devices and facilities within a region, providing high durability. This ensures that user-generated content is protected against data loss, hardware failures, or other issues. By using an S3 gateway endpoint, the database in the VPC's private subnet can securely connect to S3 without the need for internet access. This ensures that retrieval of content stored in S3 is efficient and controlled within the VPC environment. Therefore its scalability, durability (99.9999999% as per the notes in topic 06g), cost-effectiveness, security features, and seamless integration with other AWS services make it an optimal choice for this application as it requires efficient and reliable storage of various types of data, such as book reviews, images, and cover images

D) TR1.4 - Configure an optimal networking solution.**Design Choice: AWS Virtual Private Cloud - VPC (Subnets, Internet Gateway, Security Groups, Route Table)**

The whole infrastructure for this online book application is deployed in the AWS Cloud VPC that spans across three availability zones. Each availability zone has one public subnet and two private subnets. The VPC is connected to the internet via an internet gateway. The routing inter-VPC and with the outside internet is possible via the route tables associated with the corresponding subnets. The instances, the databases and the ALB have their own security groups defined which act at an Instance level firewall based on the defined inbound and outbound rules.

Validation of the design:

VPC: VPC allows us to logically isolate our resources, creating a virtual network. This isolation provides a secure and controlled environment for our applications and data. It is entirely managed by AWS therefore relieving us from the management and administration overhead. They are highly scalable and secure. AWS provides a default VPC but for our use case, we create a custom VPC with a CIDR block - 10.0.0.0/16.

IP Address Range Definition: Further we have the flexibility to define our own IP address range for the VPC and the corresponding subnets, allowing us to design a network that aligns with the corresponding needs. Therefore we assigned the public subnet in AZ1 with 10.0.128.0/20 , public subnet in AZ2 with 10.0.144.0/20, and public subnet in AZ3 with 10.0.176.0/20 in AZ3 such that each subnet has 4096 IP addresses available. These subnets host the NAT devices which enable the internet connectivity to the instances in the private subnet. There are 3 NAT gateway devices in different AZs to avoid single point of failure.

Further the private subnet hosting the web server EC2 instance is assigned 10.0.0.0/19 in AZ1 , 10.0.32.0/19 in AZ2 and 10.0.64.0/19 in AZ3. This provides each of the subnets with 8192 available IP addresses. Similarly for the private subnet hosting DB servers , the following subnets are assigned - 10.0.192.0/21 in AZ1, 10.0.200.0/21 in AZ2 and 10.0.208.0/21 in AZ3 which provides each subnet 2048 IP addresses. This allocation was used in the design to take into account the fact that we have a large number of web server instances that are part of the auto scaling group which can scale out and in and hence needs more number of available IP addresses as compared to the DB subnet group. Further to accommodate the growing environment we have spare subnet capacity in each AZ.

Subnets: We have flexibility to create public and private subnets within the VPC. Public subnets can be used for resources that need direct internet access, while private subnets can host backend databases or application servers that should not be directly accessible from the internet.

In our design we have 3 subnets - 1 public subnet and 2 private subnets.

Hence as you can see, the public subnet houses the NAT device in the architecture and the web server instances and database servers are placed in the private subnet enhancing security. The outbound access for the web servers is provided via the NAT gateway in the public subnet which performs the private to public address translation and sends the traffic via the internet gateway. Further the inbound traffic to the web servers hits the public facing Application load balancer which runs an algorithm to distribute the traffic across the web servers.

Route Table: Route table is responsible for routing traffic within the subnet or to the internet based on the routes configured. Each subnet is associated with a single route table while each route table can be associated with multiple subnets. The VPC is attached with a route table called the main route table which has a default local route: Destination: 10.0.0.0/16 and target: local. This enables intra-VPC communication. Further the public subnets are associated with a custom route table with the first entry as the default local route and a second entry routes all other IPv4 subnet traffic from the public subnet to the internet over the internet gateway (destination:0.0.0.0/0 and target:igw-id). The private subnets with the web server instance have a similar first default local route entry while the second entry routes all other IPv4 subnet traffic from the private subnet to your

network over the virtual Nat gateway (destination:0.0.0.0/0 and target:nat-gateway-id). The private subnets with the DB servers on the other hand are just associated with the main route table as it doesn't need to send traffic to the internet and participates only in the communication with the VPC therefore providing more isolation.

Security Groups: VPC enables the use of security groups and network ACLs to control inbound and outbound traffic at the instance and subnet levels. This granular control enhances security by specifying who can access resources. In the above design the security group is configured with mainly three inbound allow rules for IPv4 traffic - SSH traffic at port 22, HTTP traffic at port 80 and HTTPS traffic at port 443.

Hence this flexibility of defining our virtual network as per our needs and yet having AWS manage the VPC providing us a secure environment to host the infrastructure makes it an ideal choice to satisfy the TR1.5 design requirement.

E) TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers.

Design Choice: AWS Elastic Load Balancer (AWS Application Load balancer)

A public-facing load balancer (Application level load balancer) is configured in the VPC to distribute incoming traffic to the EC2 web instances in the private subnet.

Validation of the design choice:

The application load balancers listeners are configured with protocol as HTTP (HTTPS) and port as 80 (443). Every inbound request passes through ALB and it checks for connection requests using the port and protocol we configure. The rules that we define for a listener determine how the load balancer routes requests to its registered targets defined by the target group that groups the corresponding instances participating in the load balancing. AWS Application Load Balancer operates at Layer 7 of the OSI model, enabling intelligent routing decisions based on content. This is well-suited for our applications with different web services, allowing for more granular control over traffic. ALB performs health checks on backend instances to ensure they are operational. Unhealthy instances are automatically taken out of service, contributing to improved reliability. ALB integrates seamlessly with Auto Scaling groups, allowing for automatic scaling of EC2 instances based on demand. This ensures that the application can handle varying levels of traffic efficiently. ALB automatically distributes traffic evenly across multiple availability zones as in the architecture, enhancing availability and fault tolerance.

As discussed in lecture 11/13, here's a detailed validation of the choice of the load balancer: In our design, the load is defined as the server CPU time, specifically the EC2

instance's CPU utilization time. The metric chosen for load balancing is the number of requests per workload. AWS ALB is well-suited for this scenario. After the load balancer receives a request, it evaluates the listener rules in priority order to determine which rule to apply, and then selects a target from the target group for the rule action. Therefore we can configure listener rules to route requests to different target groups based on the content of the application traffic. By the definition of load balancing, choosing ALB will ensure that for any given time period the difference between the loads served by any two EC2 instances will not be greater than 1 (€) . ALB can perform load balancing in a way that helps optimize the usage of server CPU time, aligning with our defined load.

Further ALB uses a round robin algorithm by default to route the traffic accordingly to the corresponding servers [38]. Round Robin is a simple and effective method for distributing traffic when all targets are expected to handle a similar amount of load. Since our design uses only targets which are of the same type - EC2 t2 micro instances, Round robin is indeed an effective algorithm satisfying the TR1.6. It also supports the "Least Outstanding Requests" routing algorithm as an alternative to Round Robin which is useful when targets have different capacities or when you want to direct traffic to the target that can respond most quickly. It helps in optimizing resource utilization and reducing response times.. Further in case of ALB, the feedback action is processed and enforced locally to make the corresponding routing decisions.

ALB provides Layer-7 aware routing which is more suitable for routing based on the content such as HTTP/HTTPS headers than the network level load balancer (NLB) which routes based on the Layer 4 data (used for applications with TCP/UDP traffic). We chose ALB for our use case with a prime motive of routing based on the HTTP requests received by the load balancer to the corresponding web server instances. Further ALB supports SSL/TLS termination, enabling offloading of SSL/TLS decryption at the load balancer, reducing the load on backend instances.

Hence the choice of AWS ALB is well-justified satisfying the TR1.6

F) TR7.1 - Implement a system to auto-scale horizontally and vertically to accommodate time varying payloads

Design Choice: AWS Auto Scaling Group, AWS Auto Scaling Policy and Launch Template

An auto scaling group is configured to maintain a specified number (3) of EC2 instances across multiple Availability Zones, automatically adjusting the number of instances to meet changing application demand or workloads. The EC2 instances are deployed as a result of this configuration as the desired capacity is set to 3.

A launch template is used to define consistent configurations for EC2 instances. This ensures that newly launched instances have the same specifications, including the AMI,

instance type, key-pair, security groups, and other settings. This is referenced when creating the Auto scaling group. Further we can attach it to the existing application load balancer.

The group size is defined with a desired capacity of 2 while the minimum capacity is 2 and maximum capacity is 5. A target tracking auto scaling policy is defined where if the CPU utilization is beyond 50% , an alarm is triggered and a new instance is spun up in one of the chosen AZ's. Similarly if the utilization is below 35% for 15 minutes, then the alarm is triggered to remove the added instances and set to the default capacity. We employ horizontal auto scaling algorithm where the replicas are scaled according to this - $\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} * (\text{currentMetricValue} / \text{desiredMetricValue})]$

Validation of design choice: AWS Auto Scaling Group allows for horizontal scaling by automatically adjusting the number of EC2 instances in response to changes in demand. This is crucial for accommodating time-varying workloads. As demand increases, new instances are added, and as demand decreases, excess instances are terminated. : Launch Templates within Auto Scaling Group provide a way to define consistent configurations for EC2 instances. They also support versioning, allowing you to manage and roll back to previous configurations if needed. This ensures that instances launched during scaling events have the desired specifications. Auto Scaling Policies define rules for adjusting the capacity of the Auto Scaling Group based on various metrics, such as CPU utilization, network traffic, or custom metrics. In our design we use the CPU utilization as the metric based on which auto scaling occurs. This facilitates dynamic adjustments in response to changing workloads. Auto Scaling Groups can be configured to distribute instances across multiple Availability Zones (AZs). This enhances availability and fault tolerance by ensuring that instances are spread across different physical locations. ASGs can be associated with Elastic Load Balancers (ELB), allowing instances to be automatically registered and deregistered as they are launched or terminated. This ensures that traffic is evenly distributed among healthy instances.

Further AWS Auto Scaling integrates seamlessly with Amazon CloudWatch, allowing us to monitor various metrics. CloudWatch alarms can trigger scaling actions based on thresholds we define, ensuring that the Auto Scaling Group responds to changes in workload characteristics. Finally AWS Auto Scaling helps optimize costs by automatically adjusting the number of instances based on demand. This prevents overprovisioning during low-demand periods and ensures that the infrastructure scales up when needed.

Therefore AWS Auto Scaling Group, along with associated features like Launch Templates and Auto Scaling Policies, is a robust choice for achieving both horizontal and vertical scaling to accommodate time-varying payloads satisfying TR7.1.

G) TR1.5 - Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization)

Design Choice: AWS CloudWatch and AWS Simple Notification as Service

Cloudwatch Alarms are configured to send SNS notifications to the corresponding topic in any of the following events - launch, terminate, failed to launch, failed to terminate. These SNS notifications are received via e-mail. Further Clouwatch Dashboard for the auto scaling group to track the CPU Utilization (%) and the NetworkPacketsIn (Bytes) Metrics primarily.

Validation of the design choice:

AWS CloudWatch is a robust monitoring service that provides a comprehensive set of tools for collecting, analyzing, and visualizing various operational metrics and logs from AWS resources. It allows us to monitor resource utilization metrics such as CPU utilization, memory usage, disk I/O, and network performance. This provides insights into the health and performance of the EC2 instances running the workload. We primarily monitor the CPU Utilization and Network Packets In (Bytes) in our use case. CloudWatch supports custom metrics. This flexibility allows you to define and monitor application-specific metrics relevant to your workload, such as application performance metrics and page load speeds. CloudWatch enables the creation of alarms based on metric thresholds. When a metric breaches a predefined threshold, CloudWatch can trigger actions such as sending notifications or automatically scaling resources. This is essential for proactive monitoring and alerting. In the above design when the target tracking policy is enabled for the auto scaling group it automatically creates two Cloudwatch Alarms - TargetTracking- AlarmLow and TargetTracking-AlarmHigh for CPU utilization < 35 % and CPU utilization > 50% respectively.

AWS Simple Notification Service (SNS) on the other hand is a fully managed notification service that enables the sending of real-time notifications to a variety of endpoints, including email, SMS, and application endpoints. SNS seamlessly integrates with CloudWatch Alarms. When a CloudWatch Alarm is triggered due to a metric breach, it can send notifications through SNS. This ensures that relevant stakeholders are promptly notified of any operational issues. SNS supports multiple protocols and endpoints for notifications, allowing us to reach different teams or individuals through their preferred channels (e.g., email, SMS, HTTP/HTTPS, SQS, Lambda). SNS facilitates a fanout architecture, allowing you to broadcast notifications to multiple subscribers. This is beneficial for scenarios where notifications need to be sent to different teams or individuals based on the nature of the alert. For this design, the SNS topic is created and

added as a notification for the auto scaling thereby sending automatic notifications via email when the alarm created by the auto scaling policy is triggered.

The combination of CloudWatch and SNS creates an autonomous monitoring system. CloudWatch continuously monitors the defined metrics, and when an issue is detected, it triggers SNS notifications, ensuring that relevant parties are informed without manual intervention. Therefore this makes it an ideal choice to satisfy the TR1.5 design requirement.

H) TR2.1 - Implement a mechanism for secure tenant identification and access

Design Choice: AWS Identity and Access Management and Amazon ElastiCache

The chosen design for secure tenant identification and access, particularly leveraging Amazon ElastiCache with the Redis engine, aligns seamlessly with LitShelf's requirements. LitShelf, an application with diverse user interactions and dynamic content, benefits from the high-performance caching capabilities of ElastiCache. The Redis engine ensures swift retrieval of session data, authentication tokens, and other relevant information critical for user authentication and session management in a multi-tenant environment. In tandem with ElastiCache, AWS Identity and Access Management (IAM) integrates seamlessly into LitShelf's architecture to fortify security and access control. IAM roles and policies are meticulously defined to grant precise access to ElastiCache resources based on roles and responsibilities. Upholding the principle of least privilege, each IAM user or role possesses only the essential permissions required for their specific tasks. The potential enforcement of Multi-Factor Authentication (MFA) further enhances the overall security posture. Within LitShelf's web application backend logic, IAM roles come into play for user authentication, ensuring that only authorized individuals access tenant-specific data stored in ElastiCache. The identification of tenants through unique identifiers in request headers aligns with LitShelf's need for a robust and scalable solution. Additionally, AWS CloudTrail and CloudWatch are employed to monitor IAM-related events in real-time, providing LitShelf with a comprehensive view of access to ElastiCache and other AWS resources.

Validation of the design choice:

Amazon ElastiCache provides seamless scalability by dynamically adjusting the number of nodes based on demand, ensuring optimal system performance even under varying loads from different tenants. The low-latency access to data offered by Redis, as an in-memory data store, is particularly crucial for swift and efficient tenant identification

and access validation, significantly enhancing the overall responsiveness of the application. ElastiCache reinforces security through both at-rest and in-transit encryption, safeguarding tenant data during transmission and while stored in the cache. This security measure aligns perfectly with the imperative of secure tenant identification.

Incorporating Amazon ElastiCache in a Multi-Availability Zone (Multi-AZ) deployment ensures high availability and fault tolerance. In the event of a failure in one Availability Zone, the system seamlessly transitions to another, minimizing downtime and guaranteeing continuous tenant access. ElastiCache's status as a fully managed service further streamlines operations by handling routine tasks like patching, backups, and hardware provisioning. This operational efficiency allows the concentration on developing and maintaining secure and efficient tenant identification and access mechanisms.

The integration of Amazon ElastiCache with AWS Identity and Access Management (IAM) adds an extra layer of security. IAM facilitates fine-grained access control, enhancing the overall security posture of the multi-tenant application. This collaboration ensures that access to ElastiCache resources is meticulously governed, aligning with LitShelf's commitment to robust security practices. In summary, the combination of Amazon ElastiCache with the Redis engine, bolstered by IAM, perfectly aligns with the design requirements for secure tenant identification and access. Its scalability, performance, security features, and seamless integration with AWS services collectively make it a robust choice for efficiently managing tenant data in a dynamic multi-tenant environment.

D) TR4.1: Utilize redundant servers, and failover mechanisms.

Design Choice: AWS Elastic Load Balancing (ELB) and Amazon Route 53

Litshelf's design choice encompasses the strategic utilization of AWS Elastic Load Balancing (ELB) and Amazon Route 53 to establish a resilient infrastructure for redundant servers and failover mechanisms. ELB dynamically distributes incoming traffic across multiple Amazon EC2 instances, preventing a single point of failure and optimizing overall system performance. Complemented by Auto Scaling, ELB ensures the system seamlessly scales based on demand. Amazon Route 53, as the chosen DNS management solution, efficiently routes user requests to ELB endpoints and employs intelligent failover mechanisms for continuous service availability. The Multi-AZ deployment model further enhances high availability by replicating resources across multiple Availability Zones, establishing a robust foundation for disaster recovery. This

strategic combination ensures Litshelf's web portal maintains optimal performance, scalability, and fault tolerance, providing users with a seamless and reliable experience.

Validation of the Design Choice:

Elastic Load Balancing (ELB): ELB automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, in multiple Availability Zones (AZs). This ensures even load distribution and prevents any single point of failure. ELB seamlessly integrates with AWS Auto Scaling, dynamically adjusting the number of instances in response to changes in demand. This ensures that the application can handle varying levels of traffic, enhancing scalability and fault tolerance.

Amazon Route 53: DNS Management: Route 53, as Litshelf's chosen DNS service, plays a crucial role in directing user requests to the appropriate ELB endpoint. It provides domain registration and routing capabilities, allowing for the effective management of traffic and failover.

Health Checks and Failover: Route 53 conducts health checks on the registered resources and automatically routes traffic away from unhealthy instances to healthy ones. This proactive health monitoring enhances system reliability and minimizes downtime.

Multi-AZ Deployment Model: The Multi-AZ deployment model involves replicating resources in multiple Availability Zones. In the event of a failure in one AZ, traffic is automatically redirected to healthy instances in another, ensuring high availability and minimizing service disruptions.

Multi-AZ deployment serves as a disaster recovery strategy, providing redundancy and resilience against unforeseen events. This aligns with the goal of maintaining a reliable and available system.

Overall Redundancy and Failover: The combination of ELB, Route 53, and Multi-AZ deployment forms a robust strategy for redundancy and failover. It mitigates risks associated with server failures, network issues, or other unforeseen events that could impact system availability.

Seamless User Experience: The failover mechanisms provided by ELB and Route 53 contribute to a seamless user experience by automatically directing traffic away from compromised or underperforming instances.

In conclusion, the design choice of AWS Elastic Load Balancing, Amazon Route 53, and Multi-AZ deployment aligns with TR4.1's requirement for redundant servers and failover mechanisms. This strategic approach reflects Litshelf's commitment to ensuring high availability, fault tolerance, and an optimal user experience for its web portal.

J) TR8.8 - Use automation when obtaining or scaling resources

Design Choice: AWS CloudFormation

Litshelf has chosen to leverage AWS CloudFormation as the primary tool for automation when obtaining or scaling resources. AWS CloudFormation enables the automation of the deployment and management of AWS infrastructure through the use of templates. CloudFormation facilitates the seamless management of updates and changes to the infrastructure. Whether modifying configurations, adding new resources, or scaling existing ones, CloudFormation streamlines the process. This reduces the effort required to deploy changes and ensures that updates are applied consistently. CloudFormation integrates seamlessly with various AWS services, allowing Litshelf to automate the provisioning of a wide range of resources, including EC2 instances, databases, networking components, and more. This comprehensive integration ensures that Litshelf can efficiently manage its entire AWS infrastructure using a standardized and automated approach.

Validation of the design choice:

CloudFormation allows Litshelf to define and provision infrastructure as code. Templates written in JSON or YAML provide a declarative way to specify the resources needed, their configurations, and the relationships between them. This approach aligns with best practices for Infrastructure as Code, facilitating version control, collaboration, and repeatability. Templates in CloudFormation ensure consistency in the provisioning of resources. Whether deploying a single resource or an entire stack, CloudFormation ensures that the infrastructure is reproducible across different environments, reducing the risk of configuration errors and promoting reliability. Automation is crucial for scaling resources efficiently. As Litshelf experiences changes in demand or growth, CloudFormation allows for the automated scaling of infrastructure. This is particularly valuable for adapting to varying workloads, ensuring optimal resource utilization, and enhancing the overall scalability of the application. Manual processes in resource provisioning and scaling can introduce errors. By utilizing CloudFormation, Litshelf minimizes the reliance on manual intervention, reducing the likelihood of configuration mistakes. This aligns with the goal of implementing a reliable architecture by mitigating human errors in resource management.

K) TR9.5: Use data compression techniques to reduce storage costs and optimize data storage.**Design Choice: Amazon S3 and AWS Glue**

Amazon S3 provides Litshelf with a scalable and durable object storage solution. With features like versioning, lifecycle policies, and fine-grained access controls, S3 enables Litshelf to efficiently store and manage a variety of data types, including images and

binary files. This choice aligns with the need for a scalable and cost-effective storage solution. S3's pricing model is designed to be cost-effective, with tiered storage classes that allow Litshelf to choose the most suitable storage class based on the access patterns of their data. By optimizing the storage class for each data type, Litshelf can reduce storage costs without compromising data availability. AWS Glue automates the Extract, Transform, Load (ETL) process, and in the context of TR9.5, it plays a crucial role in optimizing data storage. During ETL operations, AWS Glue can apply compression techniques to data, reducing its size and optimizing storage efficiency. This automation ensures that data is compressed consistently, promoting efficient storage practices.

Validation of the design Choice:

Utilizing AWS Glue for data compression during ETL operations contributes to cost efficiency. Compressed data occupies less storage space, resulting in reduced storage costs. This aligns with the overarching goal of TR9.5, which is to use data compression techniques to optimize storage costs effectively. Both Amazon S3 and AWS Glue provide Litshelf with a secure and managed environment for data storage and processing. S3's access controls and encryption options ensure data security, while AWS Glue's managed ETL service reduces the operational overhead, allowing Litshelf to focus on optimizing data storage without compromising on security. The scalability of Amazon S3 and the automation capabilities of AWS Glue together ensure that Litshelf can efficiently handle growing volumes of data while maintaining data availability. This design choice supports scalability requirements while keeping storage costs in check. In conclusion, the design choice of using Amazon S3 and AWS Glue for data compression aligns with Litshelf's commitment to implementing a cost-effective approach (BR9) and optimizing storage costs while ensuring data availability and security. This strategic combination of services supports Litshelf in maintaining an efficient, scalable, and secure data storage environment.

L) TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols

Design Choice: AWS Key Management System and AWS Certificate manager

AWS KMS is used to manage cryptographic keys that are essential for encrypting user sensitive data at rest that is stored in the AWS RDS instance. ACM is used for obtaining and managing SSL/TLS certificates therefore handling secure HTTPS connections by encrypting the data in transit.

Validation of the design choice: When data is stored in Amazon RDS, AWS KMS helps in encrypting this data. KMS allows the creation and management of Customer Master Keys (CMKs), which are used to encrypt and decrypt data. ACM simplifies the process

of obtaining, renewing, and managing SSL/TLS certificates required for secure communication. Certificates issued by ACM can be easily integrated with AWS services and resources. AWS KMS and ACM adhere to industry-standard encryption protocols, ensuring strong cryptographic mechanisms for securing data both at rest and in transit. They are user-friendly, providing straightforward interfaces for managing encryption keys and SSL/TLS certificates. This simplicity contributes to the ease of implementation, which is important for maintaining operational efficiency. Both AWS KMS and ACM are managed services, reducing the operational overhead associated with key management and certificate deployment. AWS handles tasks such as key rotation, certificate renewal, and other maintenance activities. KMS provides efficient key management capabilities, including the ability to create, rotate, and disable encryption keys. The management of cryptographic keys is crucial for maintaining the security of encrypted data at rest. ACM handles the issuance, renewal, and deployment of SSL/TLS certificates, ensuring secure communication between clients and the online bookstore application.

M) TR10.1 - Implement an appropriate usage-based billing or price model

Design Choice: AWS Cost Explorer and AWS Pricing Calculator

Litshelf's implementation of AWS Cost Explorer and Pricing Calculator enhances cost visibility, enabling proactive monitoring and optimization. These tools empower informed decision-making by estimating costs and maximizing the value of AWS resources. Aligned with the Cost Efficiency pillar, Litshelf adapts billing models, ensuring business value at the lowest cost and fulfilling TR10.1's goal of a usage-based model for optimal cost management.

Validation of the design Choice: AWS Cost Explorer for Cost Visibility

AWS Cost Explorer provides Litshelf with a comprehensive view of its AWS costs and usage patterns. This tool allows Litshelf to analyze historical data, identify cost trends, and gain insights into resource utilization. This visibility is crucial for making informed decisions about resource optimization and cost-effectiveness. With AWS Cost Explorer, Litshelf can monitor costs in near-real-time and set up custom reports to track spending across different dimensions, such as services, accounts, or tags. AWS Pricing Calculator empowers Litshelf to estimate the cost of its architecture based on usage patterns, services, and geographic regions. This tool allows Litshelf to explore different scenarios, understand cost implications, and make decisions that align with its budget and cost-efficiency objectives. By utilizing AWS Cost Explorer and Pricing Calculator, Litshelf is equipped to make informed decisions about its usage-based billing or pricing model. The tools provide the necessary data and insights to align the billing model with business requirements, ensuring that the chosen model is both cost-effective and

conducive to delivering business value. The combination of AWS Cost Explorer and Pricing Calculator supports Litshelf in maximizing the value of its AWS resources. By understanding the cost implications of various configurations and usage patterns, Litshelf can optimize resource utilization, allocate budgets effectively, and achieve the lowest price point for running its systems. This design choice aligns with the Cost Efficiency pillar by enabling Litshelf to manage billing and subscriptions efficiently. The tools contribute to the proactive management of costs, allowing Litshelf to adapt its usage patterns and resource allocation to deliver business value at the lowest possible cost. In conclusion, the design choice of implementing AWS Cost Explorer and AWS Pricing Calculator demonstrates Litshelf's commitment to efficient cost management and aligns with TR10.1.

5.5 Design principles and best practices used (Using [1] as reference)

Pillar: Operational excellence

Design Principles:

- **Perform operations as code:**
 - This principle emphasizes treating infrastructure, configurations, and operational procedures as code, enabling automation, repeatability, and consistency.
 - **How was it used in the design?** Utilizing Infrastructure as Code tool such as AWS CloudFormation, the entire AWS infrastructure for the Litshelf (VPC, subnets, EC2 instances, databases, etc.) is defined in code. Changes to the infrastructure are made by updating the code. This allows for version control, review processes, and automated deployments. It aligns with the idea of treating infrastructure changes as code changes.
 - **Why did this design principle turn out to be beneficial for our use case/design?** Treating operations as code reduces the chances of manual errors in configuration and deployment processes, enhancing the overall reliability of the system. With infrastructure and configurations defined in code, there is consistency across different environments, ensuring that the development, testing, and production environments are aligned. Automation allows for quicker responses to changes or events, enabling a more agile development and operations lifecycle.
 - This principle was used in formulating the following TRs: TR 12.1: Automate build, deployment, and testing of the workload. ; TR8.8: Use automation when obtaining or scaling resources

- **Make frequent, small, reversible changes:**
 - This principle emphasizes making incremental updates to the workload in a scalable and loosely coupled manner, allowing for faster, safer, and reversible changes.
 - **How was it used in the design?** AWS CloudFormation scripts are used to define infrastructure as code. The modular nature of these scripts allows for making small, reversible changes to specific components without affecting the entire infrastructure. Infrastructure changes are version-controlled, enabling the ability to roll back to previous configurations in case of issues. This aligns with the principle of reversible changes. CI/CD pipelines are implemented using services like AWS CodePipeline or Jenkins, facilitating automated and frequent deployment of small changes.
 - **Why did this design principle turn out to be beneficial for our use case/design?** Making small, reversible changes reduces the risk associated with each deployment. If an issue arises, it is easier to identify and roll back specific changes without affecting the entire application. Scalability and loose coupling allow for efficient resource utilization, ensuring that only the necessary components are scaled based on demand.
 - This principle was used in formulating the following TRs: TR 12.1: Automate build, deployment, and testing of the workload. And TR8.8: Use automation when obtaining or scaling resources

Best Practices:

- **OPS01-BP06 Evaluate tradeoffs:**
 - It emphasizes the need to assess the impact of tradeoffs between competing interests or alternative approaches when deciding where to focus efforts or choosing a course of action. This principle helps in prioritizing options based on the specific goals and requirements of the workload.
 - **How was it used in the design?** While formulating various technical requirements satisfying the BRs for our design, there were certain requirements which were conflicting such that purely focussing on one impacts the other. These are briefly described in section 2.3 (Performance efficiency Vs Security; Availability Vs Cost efficiency) and are detailed in the following section.
 - **Why did this design principle turn out to be beneficial for our use case/design?** By evaluating tradeoffs, the design ensures that decisions are made based on a comprehensive understanding of the implications and consequences. This led to more informed and strategic choice thereby contributing to resource utilization.

- This best practice was used in formulating the conflicting TRs: TR1.1: Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols; TR4.1: Utilize redundant servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage
- **OPS05-BP01 Use version control**
 - It advocates the adoption of version control systems to track changes and releases, ensuring collaboration on code, consistent merges, and easy reversion of errors through correct versioning.
 - **How was it used in the design?** Version-controlled AWS CloudFormation templates are used to define and manage the bookstore's infrastructure. Changes to infrastructure are tracked, enabling the ability to deploy new versions, detect modifications, and revert to prior versions if needed.
 - **Why did this design principle turn out to be beneficial for our use case/design?** Version control ensures that code consistency is maintained, and changes are applied uniformly across the development environment. In case of errors or unintended changes, the design can easily revert to known good states or previous versions, reducing the risk of data or code loss.
 - This best practice was used in formulating the following TRs: TR 12.1: Automate build, deployment, and testing of the workload. And TR8.8: Use automation when obtaining or scaling resources
- **OPS05-BP09 Make frequent, small, reversible changes:** Explained in the design principles section
- **OPS05-BP10 Fully automate integration and deployment**
 - **How was it used in the design?** Infrastructure changes are managed as code using AWS CloudFormation templates. These templates are version-controlled, and the deployment of infrastructure updates is automated through the CI/CD pipeline.
 - **Why did this design principle turn out to be beneficial for our use case/design?** Automation reduces the risk of errors caused by manual processes during build, testing, and deployment.
 - This best practice was used in formulating the following TRs: TR 12.1: Automate build, deployment, and testing of the workload. And TR8.8: Use automation when obtaining or scaling resources
- **OPS08-BP01 Analyze workload metrics and OPS08-BP05 Create dashboards**

- **How was it used in the design?** The design utilizes Amazon CloudWatch dashboards to create a centralized view of system metrics, performance, and business KPIs. It helps to monitor the workload periodically based on various metrics like CPU utilization, NetworkPacketsIn, NetworkPacketsOut etc defined for the corresponding Auto scaling group or other health metrics generated by the load balancer.
 - **Why did this design principle turn out to be beneficial for our use case/design?** This provides immediate visibility into critical system metrics and rapid insight into the impact of operational issues, enabling quick responses and problem resolution.
 - This best practice was used in formulating the following TRs: TR1.5: Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization); TR2.3 Implement real-time monitoring and alerting mechanisms to trace insecure actions and changes within the environment.; TR4.2: Implement automated monitoring and alerting systems.; TR8.4: Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand; TR9.4: Implement dashboards to monitor cost proactively for the workload.
- **OPS08-BP04 Create actionable alerts**
 - **How was it used in the design?** Cloudwatch Alarms are configured based on the workload metrics. In this design, one such alarm is configured for the CPU utilization such that if the utilization is greater than 50% for few minutes the Alarm is triggered and a notification is sent via email using the Amazon SNS. Further the necessary action of auto scaling takes place to mitigate the issue.
 - **Why did this design principle turn out to be beneficial for our use case/design?** Improved system uptime and reliability are achieved through proactive detection and mitigation of potential issues.
 - This best practice was used in formulating the following TRs: TR1.5: Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization); TR2.3 Implement real-time monitoring and alerting mechanisms to trace insecure actions and changes within the environment.; TR4.2: Implement automated monitoring and alerting systems.; TR8.4: Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand

Pillar: Performance Efficiency

Design Principles:

- **Democratize advanced technologies:**
 - **How was it used in the design?** Instead of setting up and managing a complex database system, we use managed database services like Amazon RDS Multi-AZ (MySQL) for storing and retrieving data. This simplifies database management tasks and allows the team to focus on application development.
 - **Why did this design principle turn out to be beneficial for our use case?** By consuming advanced technologies as services, the team can concentrate on developing features and improving the bookstore application rather than spending time on the complexities of technology implementation.
 - This satisfies the technical requirement - TR1.2: Select a robust database option (relational, key value , graph etc) for data storage., for efficient and scalable data storage without the need for in-depth database administration skills.
- **Go global in minutes:**
 - **How was it used in the design?** The cloud-based bookstore application is deployed in multiple AWS Availability Zones globally. This means that there are redundant instances of the application running in different geographic locations. The database used by the bookstore is configured for multi-region replication. This ensures that data is replicated across multiple Regions, providing data redundancy and the ability to failover in case of a regional outage.
 - **Why did this design principle turn out to be beneficial for our use case?** Users from different parts of the world experience lower latency and faster response times, leading to an improved overall experience. The redundant deployments and multi-region architecture enhance the availability of the application
 - This satisfies the technical requirement for providing lower latency and improved performance to users accessing the bookstore from various parts of the world, TR1.7 - Ensure the application can be easily deployed in multiple regions worldwide and TR4.3 - Deploy geographically distributed backup systems and data replication techniques.. It also addresses the need for high availability by having redundant deployments.This satisfies the technical requirement for ensuring data availability and resilience in the face of potential regional failures.

Best practices:

- **PERF01-BP02 Use guidance from your cloud provider or an appropriate partner to learn about architecture patterns and best practices:**
 - **How was it used in the design?** We reviewed AWS documentation, whitepapers, and best practices related to AWS services used in the architecture to come up with the technical requirements as well as our design solution. This includes referring to AWS Well Architected Framework, user guides, architectural best practices, and case studies.
 - **Why did this best practice turn out to be beneficial for our use case?** By following AWS guidance and best practices, it helped us make informed decisions that align with industry standards and AWS-recommended approaches.
 - This helped in formulating most of the Technical requirements described in section 2.2.
- **PERF01-BP03 Factor cost into architectural decisions**
 - **How was it used in the design?** We used Well-Architected cost optimization best practices to right-size workload components, enable elasticity, and achieve optimization. Continual monitoring and analysis of cost data are performed to identify opportunities for further cost optimization. AWS tools such as AWS Cost Explorer are utilized.
 - **Why did this best practice turn out to be beneficial for our use case?** By factoring cost into architectural decisions, we optimize resource utilization, avoiding unnecessary expenses and promoting cost efficiency.
 - This helped in formulating most of the Technical requirements described in section 2.2, specifically, TR9.3: Provision resources dynamically.
- **PERF01-BP04 Evaluate how trade-offs impact customers and architecture efficiency**
 - **How was it used in the design?** While formulating various technical requirements satisfying the BRs for our design, there were certain requirements which were conflicting such that purely focussing on one impacts the other. These are briefly described in section 2.3 (Performance efficiency Vs Security; Availability Vs Cost efficiency) and are detailed in the following section.
 - **Why did this best practice turn out to be beneficial for our use case/design?** By evaluating tradeoffs, the design ensures that decisions are made based on a comprehensive understanding of the implications and consequences. This led to more informed and strategic choice thereby contributing to resource utilization.
 - This best practice was used in formulating the conflicting TRs: TR1.1: Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols; TR4.1: Utilize redundant

servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage

- **PERF02-BP01 Select the best compute options for your workload and PERF02-BP02 Understand the available compute configuration and features**
 - **How was it used in the design?** We thoroughly reviewed compute requirements of the online bookstore workload. This includes processing needs, traffic patterns, data access patterns, scaling requirements, and latency considerations. Further we explored different compute options available on AWS, such as Amazon EC2, Amazon ECS, Amazon EKS, AWS Lambda, AWS Batch, and Amazon Lightsail for its characteristics and common use cases. And finally chose Amazon EC2 instance to be the most relevant for our requirement.
 - **Why did this best practice turn out to be beneficial for our use case/design?** By selecting the best compute options, the design team ensures that the bookstore workload is more resource-efficient, avoiding over-provisioning and minimizing unnecessary costs.
 - This best practice was used in formulating the following TR: TR1.1 - Select and provision high-performance compute capacity.
- **PERF02-BP04 Configure and right-size compute resources and PERF02-BP05 Scale your compute resources dynamically**
 - **How was it used in the design?** To avoid the common anti-patterns of over-provisioning or under-provisioning compute resources, an auto scaling group was used. This ensures that the resources allocated are in line with the actual needs of the workload, preventing unnecessary costs or performance issues. AWS monitoring tools like Amazon CloudWatch are employed to monitor resource usage continually. This involves analyzing metrics like CPUUtilization for ephemeral workloads and regularly checking for optimization opportunities in stable workloads.
 - **Why did this best practice turn out to be beneficial for our use case?** Configuring and right-sizing compute resources ensures that the cloud infrastructure operates efficiently and cost-effectively by accommodating time varying payload, meeting the business needs.
 - This best practice was used in formulating the following TRs: TR1.1 - Select and provision high-performance compute capacity, TR1.5 - Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization) , TR7.1: Implement a system to auto-scale horizontally and vertically.
- **PERF02-BP03 Collect compute-related metrics**

- **How was it used in the design?** AWS monitoring tools like Amazon CloudWatch are employed to monitor resource usage continually. We capture various compute related metrics like CPU utilization, NetworkPacketsIn, NetworkPacketsOut and so on.
- **Why did this best practice turn out to be beneficial for our use case?** These metrics are a part of a data-driven approach to actively tune and optimize our workload's resources.
- This best practice was used in formulating the following TRs: TR1.5 - Monitor and track the workload resource utilization and operational metrics (ex: application performance metrics, page load speeds, server resource utilization)
- **PERF03-BP01 Use a purpose-built data store that best supports your data access and storage requirements and PERF03-BP02 Evaluate available configuration options for data store**
 - **How was it used in the design?** Based on the documented data characteristics, the team selects purpose-built data stores from the available AWS options like Amazon S3 for object storage and Amazon RDS for relational databases. Evaluating various data stores aligns with the technical requirement to choose the most suitable technology for different data types and access patterns.
 - **Why did this best practice turn out to be beneficial for our use case?** By selecting purpose-built data stores, the workload can efficiently manage data based on its characteristics, leading to improved performance.
 - This best practice was used in formulating the following TRs: TR1.2 - Select a robust database option (relational, key value, graph etc) for data storage., TR1.3 - Choose a robust and suitable storage solution
- **PERF04-BP01 Understand how networking impacts performance**
 - **How was it used in the design?** The design involves various key AWS networking services, such as Virtual Private Cloud (VPCs), Elastic Load Balancing (ELB), and Amazon Route 53. The infrastructure deployed in VPC is configured with the internet gateway, NAT devices, subnets and route tables to enable seamless communication and functioning. Further we monitor the throughput to check how it impacts network performance.
 - **Why did this best practice turn out to be beneficial for our use case?** Understanding networking's impact helps identify potential bottlenecks and areas for improvement in the overall system. A well-optimized network contributes to improved user experience by reducing latency and ensuring efficient data transfer.
 - This best practice was used in formulating the following TR: TR1.4 - Configure an optimal networking solution

- **PERF04-BP04 Use load balancing to distribute traffic across multiple resources**
 - **How was it used in the design?** An appropriate load balancer type based on workload characteristics is selected. The choice of the load balancer is Application Load Balancer (ALB) as the application uses HTTP/HTTPS workloads. Further the round robin algorithm routing algorithm is used to distribute workload to the target variations.
 - **Why did this best practice turn out to be beneficial for our use case?** Configuring and right-sizing compute resources ensures that the cloud infrastructure operates efficiently and cost-effectively by accommodating time varying payload, meeting the business needs.
 - This best practice was used in formulating the following TR: TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers

Pillar: Reliability

Design Principles:

- **Automatically recover from failure:**
 - **How was it used in the design?** Upon detecting a breach in a critical KPI like CPU utilization, an automated notification system sends alerts allowing us to investigate the issue properly. These alerts are based on the cloudwatch alarms and are published as notifications by the AWS SNS service to a specific topic we define. Hence in this design, if the CPU utilization is above the defined threshold then the cloudwatch alarm for target tracking goes off sending notification to the SNS topic. Further the recovery actions are also automated which may include restarting failed components, rerouting traffic, or scaling up resources to handle increased demand as part of the auto scaling group.
 - **Why did this best practice turn out to be beneficial for our use case?** Automated recovery ensures that the application remains available even in the event of failures, minimizing downtime and maintaining user satisfaction. Further it eliminates the need for manual intervention in case of failures, reducing the workload on operations personnel and allowing them to focus on more strategic tasks.
 - This principle was used in formulating the TR: TR7.1 - Implement a system to auto-scale horizontally and vertically and TR8.4 - Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand and TR8.8 - Use automation when obtaining or scaling resources
- **Scale horizontally to increase aggregate workload availability:**

- **How was it used in the design?** Considering that this principle drives distribution of requests across multiple, smaller resources to verify that they don't share a common point of failure, this design uses an application level load balancer which distributes the received HTTP/HTTPS traffic to different targets or multiple EC2 instances in a round robin fashion rather than overloading a specific EC2 instance.
 - **Why did this best practice turn out to be beneficial for our use case?** By distributing the workload across multiple instances, the application is less susceptible to the impact of a single failure, improving its overall resilience to disruptions.
 - This principle was used in formulating the TR: TR7.1 - Implement a system to auto-scale horizontally and vertically and TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers and TR8.8 - Use automation when obtaining or scaling resources
- **Manage change in automation:**
 - **How was it used in the design?** The entire application infrastructure, including servers, databases, network configurations, and other resources, are defined and managed through IaC tool like AWS Cloudformation. Infrastructure changes are version-controlled, enabling the ability to roll back to previous configurations in case of issues. CI/CD pipelines are implemented using services like AWS CodePipeline or Jenkins, facilitating automated and frequent deployment of small changes.
 - **Why did this best practice turn out to be beneficial for our use case?** Managing change in automation ensures that infrastructure changes are performed consistently and reliably, reducing the risk of human error and configuration drift.
 - This principle was used in formulating the TR: TR8.8 - TR8.8 - Use automation when obtaining or scaling resources

Best practices:

- **REL01-BP01 Aware of service quotas and constraints**
 - **How was it used in the design?** AWS Cloudwatch and AWS Auto scaling group enable monitoring key metrics like CPU utilization, infrastructure reviews, and automation remediation steps to verify that services quotas and constraints are not reached that could cause service degradation or disruption.
 - **Why did this best practice turn out to be beneficial for our use case?** Fluctuations in customer traffic can lead to service disruptions or performance issues if appropriate measures are not in place. By continuously monitoring and

managing traffic patterns across all regions and accounts, applications can be made more resilient to unexpected or adverse events.

- This best practice was used in formulating the TR: TR8.1 - Manage your quota increase requests for your workload architecture and TR7.1 - Implement a system to auto-scale horizontally and vertically
- **REL02-BP01 Use highly available network connectivity for your workload public endpoints**
 - **How was it used in the design?** Using highly available network connectivity to the public endpoints is enabled by using an elastic load balancer - AWS ALB which is highly available and distributes the received requests only to the healthy nodes / targets/ instances. Further the highly available Route 53 DNS enables users in the internet to connect to the ELB via the internet gateway attached to the VPC.
 - **Why did this best practice turn out to be beneficial for our use case?** Implementing a network architecture that prioritizes high availability and resilience guarantees that your users can consistently access and utilize your workload
 - This best practice was used in formulating the TR: TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers, TR1.4 - Configure an optimal networking solution, TR1.7 - Ensure the application can be easily deployed in multiple regions worldwide.
- **REL04-BP02 Implement loosely coupled dependencies**
 - **How was it used in the design?** Tight coupling enforces a failure of the EC2 instance to affect the application directly. On the other hand, loose coupling is implemented in the design by using multiple EC2 instances along with a load balancer - AWS ALB such that the received traffic is distributed evenly across only healthy instances ensuring high availability of the application.
 - **Why did this best practice turn out to be beneficial for our use case?** Loose coupling isolates the impact of failures, ensuring that a failure in one component does not cascade to others.
 - This best practice was used in formulating the TR: TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers, TR8.3 - Implement loose coupling using load balancers
- **REL06-BP01 Monitor all components for the workload (Generation) and REL06-BP02 Define and calculate metrics (Aggregation)**
 - **How was it used in the design?** Logs and metrics for AWS services such as Amazon ECS, Amazon RDS, Amazon S3 used in the design is monitored and

analyzed by AWS Cloudwatch used in the design. Further thresholds for the extracted metrics like CPU utilization is set to invoke alarms.

- **Why did this best practice turn out to be beneficial for our use case?** Comprehensive monitoring allows for the early detection of issues at various tiers of the workload, enabling proactive measures before they impact the user experience.
- This best practice was used in formulating the TR: TR8.4 - Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand.
- **REL06-BP03 Send notifications (Real-time processing and alarming)**
 - **How was it used in the design?** AWS Cloudwatch alarms are configured for monitoring certain metrics. When the alarm threshold is breached, Amazon SNS enables real time notifications and alerts published to a specific topic.
 - **Why did this best practice turn out to be beneficial for our use case?** These notifications allow for the early detection of issues, enabling proactive measures before they impact the user experience.
 - This best practice was used in formulating the TR: TR8.4 - Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand.
- **REL06-BP04 Automate responses (Real-time processing and alarming)**
 - **How was it used in the design?** When the Cloudwatch alarm breaches and the alert is invoked, then the Auto Scaling is invoked thereby scaling out or in the necessary resources like EC2 instances to manage the demand. This is an automated process without any human intervention
 - **Why did this best practice turn out to be beneficial for our use case?** The automated response mechanism ensures that the workload can dynamically adapt to changing demand. During peak periods, additional resources are automatically provisioned, optimizing performance and user experience. Conversely, during periods of low demand, resources are automatically scaled in, reducing costs and increasing reliability
 - This best practice was used in formulating the TRs: TR8.4 - Monitor demand and workload utilization, and automate the addition or removal of resources to align with demand and TR8.8 - Use automation when obtaining or scaling resources
- **REL07-BP01 Use automation when obtaining or scaling resources**
 - **How was it used in the design?** Auto Scaling is configured for EC2 instances hosting the bookstore's applications. This enables automated scaling based on predefined auto scaling policy (target tracking), ensuring the right number of

instances to handle varying workloads. It helps maintain steady performance and reduces costs during low-demand periods. Elastic Load Balancer (Application Load Balancer) is employed to distribute incoming traffic among multiple EC2 instances. ELB ensures load balancing across Availability Zones, enhancing fault tolerance. It integrates seamlessly with Auto Scaling to manage demand effectively. Amazon Route 53 is configured to manage DNS for the Load balancers. It allows users to access the bookstore's workloads using user-friendly domain names, distributing this information globally.

- **Why did this best practice turn out to be beneficial for our use case?** The implementation ensures that resources scale automatically based on demand, optimizing performance and costs without manual intervention. The use of load balancing, and DNS management contributes to enhanced availability and fault tolerance.
 - This best practice was used in formulating the TRs: TR8.5 - Implement a system to auto-scale horizontally and vertically that can dynamically handle varying workload and TR8.8 - Use automation when obtaining or scaling resources
- **REL07-BP02 Obtain resources upon detection of impairment to a workload and REL07-BP03 Obtain resources upon detection that more resources are needed for a workload**
 - **How was it used in the design?** Health checks are configured for critical components of the workload, such as EC2 instances and DynamoDB tables to detect when availability is impacted by resource shortages by the load balancer. Upon detection, automation is initiated to scale resources reactively via auto scaling group, restoring workload availability.
Amazon EC2 instances are configured for automatic scaling based on usage metrics corresponding to the demand for the workload. Metrics such as average CPU utilization and load balancer request count are utilized to dynamically scale resources to meet demand.
 - **Why did this best practice turn out to be beneficial for our use case?** Reactive scaling in response to workload impairment enhances the overall availability of the bookstore's services. This ensures that any resource shortages are promptly addressed to minimize downtime or service disruptions. Proactive scaling based on demand metrics ensures that the workload can efficiently handle varying levels of traffic. This enhances the scalability of the system and responsiveness to changes in demand.
 - This best practice was used in formulating the TRs: TR8.5 - Implement a system to auto-scale horizontally and vertically that can dynamically handle varying workload and TR8.8 - Use automation when obtaining or scaling resources

- **REL09-BP03 Perform data backup automatically**
 - **How was it used in the design?** The database and storage services used in the design - Amazon RDS instances can be automatically backed up almost continuously every five minutes and Amazon S3 objects can be backed up almost continuously every fifteen minutes.
 - **Why did this best practice turn out to be beneficial for our use case?** This provides extreme fault tolerance and prevents data being lost
 - This best practice was used in formulating the TR: TR8.7 - Implement automated failure recovery mechanisms

- **REL10-BP01 Deploy the workload to multiple locations**
 - **How was it used in the design?** The architecture of the cloud-based bookstore spans multiple Availability Zones within a single AWS Region. This ensures that the workload is distributed across physically separated and independent zones, reducing the risk of correlated failures and enhancing fault tolerance. Amazon EC2 instances and Amazon RDS databases configured to operate in a Multi-AZ setup.
 - **Why did this best practice turn out to be beneficial for our use case?** Leveraging multiple Availability Zones enhances system availability and fault tolerance. Further this reduces the latency as well. Distributing workload data and resources across multiple locations mitigates the risk of single points of failure.
 - This best practice was used in formulating the TRs: TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers, TR1.4 - Configure an optimal networking solution, TR1.7 - Ensure the application can be easily deployed in multiple regions worldwide.

Pillar: Security

Design Principles:

Implement Security as Code:

This principle underscores the importance of integrating security measures into both development and operational processes by representing security policies and configurations as code.

How it was used in the design:

In the LitShelf cloud architecture, security is implemented as code through the use of AWS Identity and Access Management (IAM) policies, AWS Key Management Service (KMS)

configurations, and other relevant security settings. Security policies are defined and managed alongside the application code, ensuring a unified and consistent approach to security.

Benefit for the use case:

LitShelf's adoption of security as code provides a systematic and automated way to manage security configurations. This approach minimizes the risk of human error, ensures consistency across environments, and facilitates efficient version control for security policies. As a result, LitShelf can maintain a secure and resilient system, aligning with its commitment to safeguarding sensitive data and operations.

AWS Technologies Leveraged:

AWS IAM: Utilized to define and enforce access policies, ensuring that only authorized entities have the necessary permissions.

AWS KMS: Employed for secure key management, including encryption and decryption of sensitive data.

AWS CloudFormation: Used to codify infrastructure and security configurations, enabling version control and consistent deployment.

Resource Considerations:

Implementing security as code involves leveraging version control systems for security configurations, integrating security checks into the continuous integration/continuous deployment (CI/CD) pipeline, and ensuring that security policies are codified for automation.

Alignment with Pillar:

LitShelf's adoption of security as code aligns seamlessly with the Security pillar. By integrating security practices directly into the codebase and leveraging AWS technologies such as IAM and KMS, LitShelf ensures a robust security posture, contributing to the overall operational excellence of its cloud environment. LitShelf's strategic use of AWS technologies for security as code exemplifies its dedication to maintaining a secure, automated, and well-managed cloud architecture. The alignment with the AWS Security pillar reinforces LitShelf's commitment to providing a secure and reliable platform for its users.

Best Practices: Security - Amazon Web Services (AWS)

SEC01-BP01 - Aware of service quotas and constraints

How it was used in the design:

AWS CloudWatch and AWS Auto Scaling group are integral components of LitShelf's security strategy. These tools actively monitor key metrics such as CPU utilization, allowing LitShelf to stay within service quotas and constraints. Automation, including infrastructure reviews and

remediation steps, verifies that these limits are not exceeded, preventing service degradation or disruptions.

Benefit for the use case:

By continuously monitoring and managing traffic patterns, LitShelf ensures that potential fluctuations in customer traffic do not lead to service disruptions. This proactive approach aligns with the best practice of being aware of service quotas and constraints, contributing to the overall security posture of the application.

AWS Technologies Leveraged:

AWS CloudWatch: Monitors key metrics for infrastructure and application components.

AWS Auto Scaling: Automates the adjustment of capacity to maintain service levels within defined constraints.

Resource Considerations:

Managing service quotas is crucial for preventing service degradation due to unexpected traffic variations. LitShelf's approach aligns with this best practice to ensure a secure and reliable workload.

SEC02-BP01 - Use highly available network connectivity for your workload public endpoints

How it was used in the design:

LitShelf ensures highly available network connectivity to its public endpoints by leveraging AWS services. The use of an elastic load balancer (AWS ALB) provides high availability and distributes requests only to healthy nodes or instances. Additionally, Route 53 DNS, configured for high availability, facilitates user connections to the ELB through the internet gateway attached to the Virtual Private Cloud (VPC).

Benefit for the use case:

Implementing a network architecture prioritizing high availability guarantees consistent access and utilization of LitShelf's workload. The redundant and highly available configuration aligns with the best practice, contributing to the overall security and reliability of the application.

AWS Technologies Leveraged:

AWS ALB (Elastic Load Balancer): Ensures high availability and even distribution of incoming requests.

Amazon Route 53: Facilitates highly available DNS resolution for user connections.

Resource Considerations:

Configuring optimal networking solutions, including highly available public endpoints, is fundamental for maintaining a secure and resilient architecture. LitShelf's design aligns with this best practice for enhanced security.

SEC04-BP02 - Implement loosely coupled dependencies

How it was used in the design:

LitShelf prioritizes a loosely coupled architecture to mitigate the impact of potential failures. The design utilizes multiple EC2 instances and an AWS ALB (Application Load Balancer) to distribute incoming traffic evenly across healthy instances. This ensures high availability of the application even in the face of individual component failures.

Benefit for the use case:

Loose coupling isolates the impact of failures, preventing a single component failure from cascading to others. In LitShelf's design, the use of an ALB and multiple EC2 instances enables the application to maintain availability despite individual component failures.

AWS Technologies Leveraged:

AWS ALB (Application Load Balancer): Distributes incoming traffic across multiple EC2 instances.

Amazon EC2: Multiple instances working together to create a redundant and resilient architecture.

Resource Considerations:

Implementing loose coupling through load balancing ensures that the failure of one instance does not compromise the overall availability of the application. LitShelf's design aligns with this best practice for enhanced security.

SEC06-BP01 - Monitor all components for the workload (Generation) and SEC06-BP02 - Define and calculate metrics (Aggregation)

How it was used in the design:

LitShelf employs AWS CloudWatch to monitor logs and metrics for various AWS services such as Amazon ECS, Amazon RDS, and Amazon S3. By continually analyzing these metrics, LitShelf gains comprehensive insights into the performance and health of its workload components.

Benefit for the use case:

Comprehensive monitoring enables the early detection of issues at different tiers of the workload, allowing LitShelf to take proactive measures before any problems impact the user.

experience. Setting thresholds for metrics like CPU utilization ensures that alarms are triggered when deviations occur.

AWS Technologies Leveraged:

AWS CloudWatch: Monitors logs and metrics for AWS services, allowing real-time analysis.

Amazon ECS, Amazon RDS, Amazon S3: Monitored services for comprehensive insights.

Resource Considerations:

Continuous monitoring and analysis of metrics contribute to early issue detection, fostering a proactive approach to workload management. LitShelf's use of AWS CloudWatch aligns with these best practices for enhanced security.

SEC06-BP03 - Send notifications (Real-time processing and alarming)

How it was used in the design:

AWS CloudWatch alarms are configured in LitShelf's design to monitor specific metrics. When thresholds are breached, Amazon SNS (Simple Notification Service) enables real-time notifications and alerts, published to a specific topic. This ensures that relevant stakeholders are promptly informed of critical events.

Benefit for the use case:

Real-time notifications allow for the early detection of issues, enabling LitShelf to take proactive measures before they impact the user experience. By promptly alerting stakeholders, the organization can respond swiftly to potential security threats or performance issues.

AWS Technologies Leveraged:

AWS CloudWatch Alarms: Configured to monitor metrics and trigger alerts.

Amazon SNS (Simple Notification Service): Facilitates real-time notifications and alerts.

Resource Considerations:

Swift notification mechanisms contribute to minimizing the time between issue detection and response. LitShelf's implementation aligns with the best practice of sending real-time notifications for enhanced security.

SEC06-BP04 - Automate responses (Real-time processing and alarming)

How it was used in the design:

In LitShelf's design, when a CloudWatch alarm is triggered and an alert is invoked, the Auto Scaling feature is invoked. This automated response mechanism dynamically scales resources,

such as EC2 instances, to manage demand. This process occurs without manual intervention, ensuring a rapid and automated response to workload changes.

Benefit for the use case:

Automated responses ensure that LitShelf's workload can adapt dynamically to changing demand. During peak periods, additional resources are automatically provisioned, optimizing performance and user experience. Conversely, during periods of low demand, resources are automatically scaled in, reducing costs and increasing reliability.

AWS Technologies Leveraged:

AWS CloudWatch Alarms: Triggered when predefined thresholds are breached.

Auto Scaling: Automatically adjusts the number of EC2 instances based on demand.

Resource Considerations:

Automated responses contribute to dynamic workload management, optimizing resource usage based on demand. LitShelf's use of automated scaling aligns with the best practice of automating responses for enhanced security.

Pillar: Cost Optimization

Design Principles:

Implement Cost Optimization as Code:

This principle emphasizes embedding cost optimization practices into the development and operational processes by representing cost management policies and configurations as code.

How it was used in the design:

In LitShelf's cloud architecture, cost optimization is implemented as code through the use of AWS Cost Explorer, AWS Pricing Calculator, and other relevant cost management tools. Cost management policies are defined and managed alongside the application code, ensuring a unified and consistent approach to cost efficiency.

Benefit for the use case:

LitShelf's adoption of cost optimization as code provides a systematic and automated way to manage cost configurations. This approach minimizes the risk of overspending, ensures consistency across environments, and facilitates efficient budget allocation for cost-effective resource utilization. As a result, LitShelf can achieve the lowest price point for running its systems while delivering business value.

AWS Technologies Leveraged:

AWS Cost Explorer: Utilized for comprehensive visibility into AWS costs and usage patterns.

AWS Pricing Calculator: Empowered LitShelf to estimate the cost of its architecture based on usage patterns and services.

AWS Budgets: Employed for setting custom cost and usage budgets, providing proactive cost management.

Resource Considerations:

Implementing cost optimization as code involves leveraging cost management tools, regularly reviewing and refining budgets, and integrating cost checks into the CI/CD pipeline to ensure cost efficiency throughout the development lifecycle.

Alignment with Pillar:

LitShelf's adoption of cost optimization as code aligns seamlessly with the Cost Optimization pillar. By integrating cost management practices directly into the codebase and leveraging AWS tools such as Cost Explorer and Pricing Calculator, LitShelf ensures efficient cost management. This approach contributes to the overall operational excellence of its cloud environment, reinforcing LitShelf's commitment to delivering business value at the lowest possible cost. The alignment with the AWS Cost Optimization pillar demonstrates LitShelf's dedication to achieving optimal resource utilization and cost-effectiveness in its cloud architecture.

COP01-BP01 Leverage AWS Pricing Models:

How it was used in the design:

LitShelf leverages AWS Pricing Models effectively by utilizing the AWS Pricing Calculator. This tool enables LitShelf to estimate costs based on various scenarios, helping make informed decisions aligned with budget and cost-efficiency objectives.

Why this best practice is beneficial:

By understanding and leveraging AWS Pricing Models, LitShelf can optimize resource utilization and make cost-effective decisions. The use of the AWS Pricing Calculator allows for exploring different scenarios, ensuring alignment with the budget and overall cost efficiency.

COP02-BP02 Monitor, Analyze, and Attribute Expenditure:

How it was used in the design:

AWS Cost Explorer is employed by LitShelf to monitor costs in near-real-time, providing insights into historical data and usage patterns. This tool enables LitShelf to analyze and attribute expenditures, contributing to informed decision-making.

Why this best practice is beneficial:

Monitoring and analyzing expenditures using AWS Cost Explorer empower LitShelf to identify trends, track spending across different dimensions, and gain insights into resource utilization. This proactive approach to cost monitoring ensures efficient cost management.

COP03-BP03 Right Size Your Workloads:

How it was used in the design:

LitShelf incorporates the best practice of right-sizing workloads by using AWS tools like AWS Cost Explorer. Continuous monitoring and analysis of cost data allow LitShelf to identify opportunities for right-sizing and optimizing resource utilization dynamically.

Why this best practice is beneficial:

Right-sizing workloads ensure that LitShelf optimally allocates resources, avoiding unnecessary expenses. This contributes to cost efficiency by dynamically adjusting resources based on actual workload needs.

COP04-BP04 Use Managed Services to Reduce Cost of Ownership:

How it was used in the design:

LitShelf employs managed services like Amazon RDS Multi-AZ for database management, aligning with the best practice of using managed services to reduce the cost of ownership.

Why this best practice is beneficial:

By leveraging managed services, LitShelf minimizes the operational overhead and complexity associated with managing certain infrastructure components. This results in lower costs and allows the team to focus more on application development.

COP05-BP05 Analyze and Attribute Costs:

How it was used in the design:

LitShelf uses AWS tools, including Cost Explorer and Pricing Calculator, to analyze and attribute costs to different dimensions. This aligns with the best practice of having a comprehensive understanding of costs and optimizing resource allocation.

Why this best practice is beneficial:

Analyzing and attributing costs enable LitShelf to make informed decisions about resource allocation and identify areas for optimization. It promotes transparency and accountability in cost management.

COP06-BP06 Optimize Over Time:

How it was used in the design:

LitShelf follows the best practice of optimizing over time by continually monitoring and analyzing cost data. This involves regularly reviewing configurations, making adjustments, and optimizing resource utilization based on changing requirements.

Why this best practice is beneficial:

Continuous optimization ensures that LitShelf's cloud environment remains cost-effective over time. Regular reviews and adjustments help adapt to evolving business needs and technological changes.

COP07-BP07 Use Tagging to Allocate Costs:

How it was used in the design:

LitShelf utilizes tagging to allocate costs and track spending across different dimensions. This practice provides granular visibility into resource usage and cost allocation.

Why this best practice is beneficial:

Tagging allows LitShelf to allocate costs accurately, facilitating a more detailed understanding of resource usage. It supports efficient budgeting and helps in identifying areas for cost savings.

COP08-BP08 Leverage Savings Plans and Reserved Instances:

How it was used in the design:

LitShelf leverages AWS Savings Plans and Reserved Instances to optimize costs by committing to consistent, predictable usage. This aligns with the best practice of using savings plans and reservations for cost savings.

Why this best practice is beneficial:

By committing to reserved capacity, LitShelf can benefit from significant cost savings compared to on-demand pricing. Savings Plans provide flexibility and help optimize costs for variable workloads.

COP09-BP09 Monitor AWS Trusted Advisor Recommendations:

How it was used in the design:

LitShelf monitors AWS Trusted Advisor recommendations regularly to identify opportunities for optimization and adherence to best practices. This aligns with the best practice of leveraging Trusted Advisor for cost optimization.

Why this best practice is beneficial:

AWS Trusted Advisor provides personalized recommendations for cost optimization, security, and performance improvements. Monitoring these recommendations ensures LitShelf stays aligned with best practices and maximizes cost efficiency.

COP10-BP10 Utilize AWS Free Tier:

How it was used in the design:

LitShelf takes advantage of the AWS Free Tier for certain services, allowing for experimentation and initial development without incurring additional costs.

Why this best practice is beneficial:

Utilizing the AWS Free Tier enables LitShelf to explore and test AWS services at no cost, supporting cost-conscious development practices and encouraging experimentation without financial commitments.

COP11-BP11 Consider Data Transfer Costs:

How it was used in the design:

LitShelf considers data transfer costs by optimizing the architecture to minimize unnecessary data transfer across regions and services.

Why this best practice is beneficial:

Considering data transfer costs helps LitShelf avoid unnecessary expenses associated with data movement between different AWS regions and services. Optimizing data transfer contributes to overall cost efficiency.

COP12-BP12 Align Resources with Business Value:

How it was used in the design:

LitShelf aligns resources with business value by regularly assessing the relevance and impact of each resource in terms of delivering business objectives.

Why this best practice is beneficial:

Aligning resources with business value ensures that LitShelf invests in resources that contribute directly to business goals. This practice supports efficient resource allocation and cost optimization.

Pillar: Sustainability**Design Principles:****Implement Sustainability as Code:**

How it was used in the design:

In LitShelf's cloud architecture, sustainability practices are implemented as code through the use of AWS tools and services dedicated to environmentally conscious resource management. AWS Compute Optimizer, AWS Power Scheduler, and other relevant tools are integrated into the

development and operational processes. Sustainability policies and configurations are defined and managed alongside the application code, ensuring a unified and consistent approach to environmental responsibility.

Benefit for the use case:

LitShelf's adoption of sustainability as code provides an automated and systematic way to manage environmental impact. By embedding sustainability practices into the codebase, LitShelf minimizes its carbon footprint, ensures eco-friendly resource allocation, and supports a greener cloud environment. This approach aligns with LitShelf's commitment to environmental responsibility and contributes to the broader goal of sustainable cloud computing.

AWS Technologies Leveraged:

AWS Compute Optimizer:

Utilized for analyzing the utilization of AWS resources and providing recommendations to improve efficiency.

AWS Power Scheduler:

Empowered LitShelf to schedule the operational hours of instances, allowing for energy-efficient resource usage.

AWS Tools for Sustainable Practices:

Other AWS tools dedicated to sustainability practices are integrated into the architecture for comprehensive environmental impact management.

Resource Considerations:

Implementing sustainability as code involves integrating AWS sustainability tools, continuously monitoring resource usage for environmental impact, and aligning practices with broader sustainability goals.

Alignment with Pillar:

LitShelf's adoption of sustainability as code aligns seamlessly with the Sustainability pillar. By integrating sustainability practices directly into the codebase and leveraging AWS tools dedicated to eco-friendly resource management, LitShelf ensures a greener cloud environment. This approach contributes to operational excellence with a focus on minimizing environmental impact. The alignment with the AWS Sustainability pillar demonstrates LitShelf's dedication to sustainable cloud computing and responsible resource usage.

Best Practices:

SUS01-BP01 Implement Environmentally Responsible Practices:

How it aligns with the design:

LitShelf adopts AWS environmentally responsible practices by utilizing AWS Compute Optimizer and AWS Power Scheduler. These tools assist in optimizing resource usage, reducing carbon footprint, and ensuring energy-efficient operations.

SUS02-BP01 Leverage Sustainable Data Centers and Infrastructure:

How it aligns with the design:

LitShelf prioritizes the use of AWS data centers and infrastructure designed with sustainability in mind. AWS Regions and Availability Zones are strategically chosen to align with AWS's commitment to sustainability, contributing to overall eco-friendly cloud operations.

SUS03-BP01 Utilize Renewable Energy Sources:

How it aligns with the design:

LitShelf aligns with AWS's commitment to using renewable energy sources by leveraging regions and facilities powered by renewable energy. This practice ensures that the cloud architecture has a reduced impact on the environment and supports the use of clean energy.

SUS04-BP01 Optimize Resource Utilization for Sustainability:

How it aligns with the design:

LitShelf employs AWS Compute Optimizer and other optimization tools to continuously monitor and optimize resource utilization. This practice ensures that resources are used efficiently, contributing to reduced energy consumption and supporting sustainable cloud computing.

SUS05-BP01 Continuously Monitor and Improve Sustainability Metrics:

How it aligns with the design:

LitShelf incorporates sustainability metrics into its monitoring practices. By continuously monitoring and analyzing the environmental impact of cloud operations, LitShelf can make informed decisions to further improve its sustainability practices over time.

SUS06-BP01 Participate in AWS Sustainability Programs:

How it aligns with the design:

LitShelf actively engages in AWS sustainability programs and initiatives. By participating in AWS's commitment to sustainability, LitShelf contributes to broader industry efforts to reduce the environmental impact of cloud computing.

SUS07-BP01 Educate Teams on Sustainable Practices:

How it aligns with the design:

LitShelf ensures that its teams are educated on sustainable practices and the importance of environmentally responsible cloud computing. This commitment to education aligns with AWS best practices for fostering a culture of sustainability across organizations.

Resource Considerations:

Incorporating AWS sustainability best practices involves leveraging tools for optimization, selecting regions powered by renewable energy, monitoring sustainability metrics, and actively participating in AWS sustainability programs.

Alignment with Pillar:

LitShelf's adoption of AWS sustainability best practices aligns with the Sustainability pillar, reinforcing its commitment to environmentally responsible cloud operations. By following these practices, LitShelf contributes to a sustainable and eco-friendly cloud computing environment. The alignment with AWS Sustainability best practices demonstrates LitShelf's dedication to minimizing its environmental impact and promoting sustainable cloud practices.

5.6 Tradeoffs visited

A) TR1.1 - Select and provision high-performance compute capacity Vs TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols:

Tradeoff:

- High-performance compute capacity ensures that the application runs effectively providing quick loading times, smooth navigation, and swift responses to user interactions, contributing to overall user satisfaction and hence handling the workloads effectively without latency.
- Data encryption ensures that sensitive information, such as customer details and transaction data, remains confidential and protected from unauthorized access or potential breaches whether at rest or at in transit. Emphasizing data encryption is vital for safeguarding sensitive information, maintaining data integrity, and meeting regulatory requirements.
- Opting for high-performance compute capacity prioritizes computational efficiency, vital for resource-intensive tasks and real-time processing. However, the introduction of encryption, a critical security measure for safeguarding data, adds computational overhead and potential latency thereby impacting its performance.

Analysis:

Performance Efficiency (TR1.1 - Select and provision high-performance compute capacity):

- Pros: Faster application response, scalability, adaptability to varying workloads.
- Cons: Potential resource costs, especially if scaled continuously.

Security - Data Encryption (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols):

- Pros: Data protection, regulatory compliance.
- Cons: Computational overhead, potential performance impact.

Using reference [39] - Even swap approach, let's see how the tradeoff can be approached or balanced.

Consequences Table:

Step1: List your objectives down the left side of a page and your alternatives along the top. This will give you an empty matrix. In each box of the matrix, write a concise description of the consequence that the given alternative (indicated by the column) will have for the given objective (indicated by the row).

The three alternatives for the given objectives (TR1.1, TR2.4):

- A) High performance compute without data encryption enabled
- B) Data encryption enabled but with a compute having compromised capacity
- C) High performance compute with encryption enabled using AWS KMS and ACM

Further to study the consequences for each objective, Performance and Security are split into potential parameters for evaluation.

Objectives		Alternatives		
		High performance compute without data encryption enabled	Data encryption enabled but with a compute having compromised capacity	High performance compute with encryption enabled using AWS KMS and ACM
Performance (TR1.1 - Select and provision high-performance)	Application response time	low	high	Moderate
	Scalability	high	low	Moderate

compute capacity)	Resource cost	low	Moderate	high
Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols)	Efficient key management and certificate handling	low	Moderate	high
	Overhead	low	high	Moderate
	Performance impact	low	high	Moderate

Objectives

Performance (TR1.1 - Select and provision high-performance compute capacity):

1) High-performance compute without data encryption enabled:

- Application response time: Low, as there is no encryption overhead, resulting in faster response times.
- Scalability: High impact, as the absence of encryption simplifies scalability and resource allocation.
- Potential resource costs: Low, as there are no additional costs associated with internal encryption.

2) Data encryption enabled but with a compute having compromised capacity:

- Application response time: High impact, as compromised compute capacity may introduce significant computational overhead, affecting response times.
- Scalability: Low impact, as compromised compute capacity may limit scalability.
- Potential resource costs: Moderate, as the compromised compute capacity may require additional resources to maintain encryption, impacting overall costs.

3) High-performance compute with encryption enabled using AWS KMS and ACM:

- Application response time: Moderate impact, as the encryption process handled by AWS KMS and ACM may introduce some computational overhead.
- Scalability: Moderate impact, as the encryption handled by external services could influence scalability, depending on the efficiency of AWS KMS and ACM.

- Potential resource costs: High, as the use of AWS KMS and ACM may incur additional costs for the encryption services

Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols):

1) High-performance compute without data encryption enabled:

- Efficient key management and certificate Handling: Low, as there is no encryption, and key management is unnecessary.
- Overhead: Low, as there is no encryption overhead, resulting in minimal computational load.
- Potential performance impact: Low, as there is no encryption to impact performance.

2) Data encryption enabled but with a compute having compromised capacity:

- Efficient key management and certificate handling: Moderate, as compromised compute capacity may hinder efficient key management.
- Overhead: High, as compromised compute capacity may increase computational overhead associated with encryption.
- Potential performance impact: High, as compromised compute capacity may introduce significant performance impact, compromising the overall security of data encryption.

3) High-performance compute with encryption enabled using AWS KMS and ACM:

- Efficient key management and certificate handling: High, as AWS KMS provides robust and efficient key management, enhancing overall data security.
- Overhead: Moderate, as the use of AWS KMS and ACM may reduce the computational overhead associated with encryption compared to internal encryption capabilities.
- Potential performance impact: Moderate, as the external encryption services may introduce some performance impact but are generally efficient when managed by AWS KMS and ACM.

Eliminating Dominated Alternatives:

Step 2: To identify alternatives that can be eliminated, follow this simple rule: if alternative A is better than alternative B on some objectives and no worse than B on all other objectives, B can be eliminated from consideration.

In order to do so, we first rank the alternatives with 1 being the best and 3 being the worst -

Objectives		Alternatives		
		High performance compute without data encryption enabled	Data encryption enabled but with a compute having compromised capacity	High performance compute with encryption enabled using AWS KMS and ACM
Performance (TR1.1 - Select and provision high-performance compute capacity)	Application response time	1	3	2
	Scalability	1	3	2
	Resource cost	1	2	3
Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols)	Efficient key management and certificate handling	3	2	1
	Overhead	1	3	2
	Performance impact	1	3	2

Now we color format it with Blue being the most coolest color and red being the most warm color -

Objectives		Alternatives		
		High performance	Data encryption enabled but with	High performance compute with

		compute without data encryption enabled	a compute having compromised capacity	encryption enabled using AWS KMS and ACM
Performance (TR1.1 - Select and provision high-performance compute capacity)	Application response time	1	3	2
	Scalability	1	3	2
	Resource cost	1	2	3
Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols)	Efficient key management and certificate handling	3	2	1
	Overhead	1	3	2
	Performance impact	1	3	2

Therefore by the principle of dominance, “**High performance compute with encryption enabled using AWS KMS and ACM**” is more cool than “**Data encryption enabled but with a compute having compromised capacity**” i.e “**Data encryption enabled but with a compute having compromised capacity**” is dominated by “**High performance compute with encryption enabled using AWS KMS and ACM**” and can be eliminated -

Objectives		Alternatives		
		High performance compute without data encryption enabled	Data encryption enabled but with a compute having compromised capacity	High performance compute with encryption enabled using AWS KMS and ACM
Performance (TR1.1 - Select and provision	Application response time	1	3	2

high-performance compute capacity)	Scalability	1	3	2
	Resource cost	1	2	3
Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols)	Efficient key management and certificate handling	3	2	1
	Overhead	1	3	2
	Performance impact	1	3	2

There is no scope for applying practical dominance principle here anymore, so with the leftover two alternatives, we perform even swaps.

Even Swaps

Objectives		Alternatives	
		High performance compute without data encryption enabled	High performance compute with encryption enabled using AWS KMS and ACM
Performance (TR1.1 - Select and provision high-performance compute capacity)	Application response time	low	Moderate
	Scalability	high	Moderate
	Resource cost	low	high
Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption)	Efficient key management and certificate handling	low	high
	Overhead	low	Moderate
	Performance	low	Moderate

protocols)	impact		
------------	--------	--	--

In our use case, let's delve into the narrative of our decision-making process through a series of swaps while considering qualitative metrics. In focusing on Scalability, it becomes apparent that in order to attain high scalability with the **"High performance compute with encryption enabled using AWS KMS and ACM"** option, there is a corresponding need to augment resource costs. This observation underscores the critical trade-off between scalability and resource expenditures.

Highlighting the qualitative metric of security, the impact on effective key management and certificate handling is marked as high for the **"High performance compute with encryption enabled using AWS KMS and ACM"** option. In light of the overarching importance of securing sensitive user data, which involves critical elements such as payment transactions and personnel information, a strategic decision is made to tolerate a moderate application response time and a moderate level of overhead. This compromise is deemed acceptable, given the paramount significance of safeguarding sensitive information.

Within this context, the adoption of "High performance compute with encryption enabled using AWS KMS and ACM" emerges as the optimal choice. Further researching on AWS KMS and ACM, the reason of choice is simplified more:

Performance (TR1.1 - Select and provision high-performance compute capacity):

- Faster application response: The use of AWS KMS and ACM suggests a more specialized and optimized approach to encryption, potentially minimizing the impact on application response times.
- Scalability: AWS KMS and ACM are designed to efficiently handle encryption tasks, making them well-suited for scalable architectures. This could result in better scalability compared to internal encryption capabilities.
- Potential resource costs: Although using AWS KMS and ACM may incur additional costs, the optimized services they provide could lead to more efficient resource utilization, potentially balancing or even reducing overall costs compared to internal encryption.

Security (TR2.4: Encrypt data at rest and in transit using industry-standard encryption protocols):

- Efficient key management: AWS KMS offers robust key management, ensuring secure key storage and distribution. This enhances the overall security posture compared to internal key management.

- Reduced overhead: While any encryption introduces some level of computational overhead, AWS KMS and ACM are specialized services designed to minimize this impact, ensuring efficient and secure data encryption.
- Potential performance impact: The performance impact of using AWS KMS and ACM is likely to be moderate, considering their optimization for cloud-based encryption tasks. This impact is generally outweighed by the enhanced security and key management they provide.

B) TR4.1: Utilize redundant servers, and failover mechanisms. Vs TR9.5: Use data compression techniques to reduce storage costs and optimize data storage :

Comment on tradeoff -

Redundant servers and failover mechanisms ensure a reliable user experience during peak usage or unexpected failures. Seamless switching to backup systems minimizes downtime and ensures continuous service availability. Data compression techniques reduce storage costs by optimizing data storage. Compressed data requires less storage space, lowering overall storage expenses. Data compression may slightly impact data processing speed due to decompression needs during data access. Tradeoff involves a potential minor slowdown in data processing speed for reduced storage costs. The choice between high availability and storage optimization should align with project budget constraints and user experience requirements. Balancing operational costs with the need for high availability and efficient storage is crucial in decision-making.

Analysis:

Availability (TR4.1 - Utilize redundant servers, and failover mechanisms) vs. Cost Efficiency (TR9.5: Use data compression techniques to reduce storage costs and optimize data storage):

TR4.1 - Utilize redundant servers, and failover mechanisms:

Pros: Ensures high availability and fault tolerance.

Seamless switch to backup systems minimizes downtime.

Continuous service availability during hardware failures or network issues.

Cons: Requires additional infrastructure investment.

Increased operational and maintenance costs for redundant systems.

TR9.5 - Use data compression techniques to reduce storage costs and optimize data storage:

Pros: Reduces storage costs significantly.

Optimizes data storage, particularly beneficial for large volumes of data.

Long-term cost savings.

Cons: Potential minor impact on data processing speed due to decompression needs.

Balancing storage optimization with processing speed is crucial.

Using reference [39] - Even swap approach, let's see how the tradeoff can be approached or balanced:

Considerations:

Project Budget Constraints: If the budget allows for additional infrastructure costs, prioritizing high availability through redundant servers (TR4.1) might be preferable.

Data Storage Needs: If the workload deals with extensive data, prioritizing storage optimization through data compression (TR9.5) can lead to significant cost savings.

User Experience Requirements: If the project demands seamless user experience and continuous service availability, prioritizing TR4.1 might be necessary.

Balancing the Tradeoff:

Implementing a mix: Employ redundant servers for critical components that demand high availability while utilizing data compression for less critical or archival data.

Periodic Evaluation: Regularly assess the changing needs of the project and adjust the balance between high availability and cost efficiency accordingly.

Conclusion:

Balancing the tradeoff between TR4.1 and TR9.5 involves strategic decision-making based on project-specific requirements, budget considerations, and the criticality of high availability. Regular evaluations and a flexible approach to the implementation of these technical requirements can ensure optimal resource utilization aligned with the project's objectives.

Consequences Table:

Step1: List your objectives down the left side of a page and your alternatives along the top. This will give you an empty matrix. In each box of the matrix, write a concise description of the consequence that the given alternative (indicated by the column) will have for the given objective (indicated by the row).

Alternatives:

Utilize redundant servers and failover mechanisms (TR4.1).
 Use data compression techniques to reduce storage costs (TR9.5).

Objectives \ Alternatives		Utilize Redundant Servers	Use Data Compression
Availability	Utilize Redundant Servers	High availability and fault tolerance: Incorporating redundant servers ensures continuous service availability, fault tolerance, and the ability to seamlessly switch to backup systems in the event of hardware failures or network issues. This mitigates downtime risks and enhances overall system reliability.	Long-term cost savings: While redundant servers enhance reliability, they may require additional upfront infrastructure investment. However, in the long term, the benefits of increased availability and minimized downtime-related costs contribute to significant cost savings.
Availability	Use Data Compression	Requires additional infrastructure investment: Utilizing redundant servers involves additional hardware and infrastructure investment to build redundancy. While this enhances reliability, it may lead to increased operational and maintenance costs in the short term.	Potential minor impact on data processing speed: The use of data compression may introduce a slight impact on data processing speed due to decompression needs. However, this tradeoff is crucial for balancing storage optimization with processing speed. The efficient utilization of resources contributes to overall cost efficiency
Cost Efficiency	Utilize Redundant Servers	Increased operational and maintenance costs: While redundant servers improve availability, they may result in increased operational and maintenance costs due to the management of additional hardware. However, the efficient utilization of resources, including storage, contributes to overall cost efficiency.	Efficient utilization of resources: Utilizing redundant servers allows for efficient resource utilization, ensuring that hardware resources are optimally utilized. This leads to cost efficiency, particularly in terms of resource utilization during peak usage or unexpected failures.
Cost Efficiency	Use Data	Enhanced reliability during	Efficient storage usage,

	Compression	hardware failures or network issues: Data compression enhances storage efficiency, reducing overall storage expenses. This is crucial for achieving cost efficiency while maintaining reliable storage systems	reducing overall storage expenses: Implementing data compression techniques optimizes data storage, resulting in efficient storage usage and reduced overall storage expenses. This contributes to long-term cost savings and efficient resource management.
--	-------------	--	--

Objectives

Availability (TR4.1 - Utilize redundant servers and failover mechanisms):

Utilize Redundant Servers:

- High availability and fault tolerance: Ensures continuous service availability during failures and mitigates downtime risks.
- Seamless switch to backup systems: Enhances reliability during hardware failures or network issues.
- Continuous service availability during failures: Minimizes downtime-related costs.
- Mitigation of downtime risks: Redundancy ensures reliability during peak usage or unexpected failures.
- Effective scalability: Low to Moderate, as redundant servers contribute more to availability than scalability.
- Resource allocation: Simplified, ensuring efficient use of resources but not specifically enhancing scalability.
- Potential impact on scalability: Moderate, as the redundancy measures may introduce complexity that affects scalability.

Cost Efficiency (TR9.5 - Use data compression techniques to reduce storage costs and optimize data storage):

Use Data Compression:

- Requires additional infrastructure investment: Increased operational and maintenance costs, but enhanced reliability during failures.
- Potential minor impact on data processing speed due to decompression needs: Balancing storage optimization with processing speed is crucial.

- Efficient storage usage, reducing overall storage expenses: Efficiently balances storage optimization with processing speed.
- Tradeoff involves a potential minor slowdown in data processing speed for reduced storage costs: Balances operational costs with the need for high availability and efficient storage.
- Storage optimization for scalability: High, as data compression reduces storage costs and supports efficient scaling of data.
- Processing speed and scalability: Moderate, as data compression may introduce minor processing overhead but generally supports scalability.
- Resource efficiency for scalable workloads: High, as optimized storage allows for effective resource utilization and scalability.

Eliminating Dominated Alternatives:

Step 2: To identify alternatives that can be eliminated, follow this simple rule: if alternative A is better than alternative B on some objectives and no worse than B on all other objectives, B can be eliminated from consideration.

Objectives		Utilize Redundant Servers	Use Data Compression
Availability	Utilize Redundant Servers	1	2
Availability	Use Data Compression	2	1
Cost Efficiency	Utilize Redundant Servers	2	1
Cost Efficiency	Use Data Compression	1	2

Now we color format it with Blue being the most coolest color and red being the most warm color -

Objectives		Utilize Redundant Servers	Use Data Compression
Availability	Utilize Redundant Servers	1	2
Availability	Use Data Compression	2	1
Cost Efficiency	Utilize Redundant Servers	2	1
Cost Efficiency	Use Data Compression	1	2

The table reflects a balanced tradeoff between "Utilize Redundant Servers" (TR4.1) and "Use Data Compression" (TR9.5) across the objectives of Availability and Cost Efficiency.

For Availability:

Utilize Redundant Servers receives a score of 1, indicating a favorable outcome for enhancing availability.

Use Data Compression receives a score of 2, suggesting a slightly less favorable outcome regarding availability.

For Cost Efficiency:

Utilize Redundant Servers receives a score of 2, indicating a tradeoff with slightly higher costs but improved availability.

Use Data Compression receives a score of 1, suggesting a more cost-efficient solution but with a potential impact on availability.

Overall, there is no clear dominance of one alternative over the other. The decision may depend on specific project requirements, budget considerations, and the desired balance between availability and cost efficiency. It's a matter of weighing the tradeoffs based on the priorities and constraints of the project.

Even Swaps:

In the decision-making process for LitShelf, let's consider the trade-offs between Utilize Redundant Servers (TR4.1) and Use Data Compression (TR9.5) in the context of key objectives such as Availability and Cost Efficiency.

Analyzing Availability, LitShelf places a high premium on ensuring continuous service availability and fault tolerance. The "Utilize Redundant Servers" option aligns well with these objectives, offering a high level of availability and fault tolerance. This is crucial for a platform like LitShelf, where uninterrupted service is paramount, especially during peak usage or unexpected failures.

Considering Cost Efficiency, LitShelf aims for efficient resource utilization and long-term cost savings. The "Use Data Compression" option stands out in terms of cost efficiency, as it enhances storage efficiency and contributes to reduced overall expenses. However, this comes with a trade-off of moderate availability, which might be acceptable depending on the specific context and priorities.

Now, let's delve into the specific context for LitShelf:

Availability:

High availability and fault tolerance: Utilizing redundant servers ensures that LitShelf maintains continuous service availability, fault tolerance, and seamless switches to backup systems. This is particularly valuable for LitShelf's commitment to providing a reliable platform for users.

Cost Efficiency:

Moderate operational and maintenance costs: While redundant servers enhance availability, there might be some increase in operational and maintenance costs due to managing additional hardware. However, the efficiency gained in resource utilization contributes to overall cost efficiency.

Efficient utilization of resources: Utilizing redundant servers allows LitShelf to efficiently use resources, ensuring optimal utilization of hardware resources. This aligns with the goal of achieving cost efficiency, especially during peak usage or unexpected failures.

In the even swap analysis, the decision to Utilize Redundant Servers is favored, given LitShelf's emphasis on high availability and the strategic balance between availability and cost efficiency. Here is the summary:

Even Swaps:

In the decision-making process for LitShelf, let's consider the trade-offs between Utilize Redundant Servers (TR4.1) and Use Data Compression (TR9.5) in the context of key objectives such as Availability and Cost Efficiency.

Analyzing Availability, LitShelf places a high premium on ensuring continuous service availability and fault tolerance. The "Utilize Redundant Servers" option aligns well with these objectives, offering a high level of availability and fault tolerance. This is crucial for a platform like LitShelf, where uninterrupted service is paramount, especially during peak usage or unexpected failures.

Considering Cost Efficiency, LitShelf aims for efficient resource utilization and long-term cost savings. The "Use Data Compression" option stands out in terms of cost efficiency, as it enhances storage efficiency and contributes to reduced overall expenses. However, this comes with a trade-off of moderate availability, which might be acceptable depending on the specific context and priorities.

Availability:

High availability and fault tolerance: Utilizing redundant servers ensures that LitShelf maintains continuous service availability, fault tolerance, and seamless switches to backup systems. This is particularly valuable for LitShelf's commitment to providing a reliable platform for users.

Cost Efficiency:

Moderate operational and maintenance costs: While redundant servers enhance availability, there might be some increase in operational and maintenance costs due to managing additional hardware. However, the efficiency gained in resource utilization contributes to overall cost efficiency.

Availability is a critical factor for LitShelf, favoring the "Utilize Redundant Servers" option, which provides high availability and fault tolerance.

While "Use Data Compression" offers high cost efficiency, there is a trade-off with moderate availability, which may not align with LitShelf's priority for continuous service availability.

In the context of LitShelf, the decision to Utilize Redundant Servers is well-founded, emphasizing the platform's commitment to providing a reliable and continuously available service to its users.

6. Kubernetes Experimentation

In this section we validate some aspects of our design by performing few experiments using AWS.

6.1 Experiment Design

In this phase, we have selected specific Technical Requirements (TRs) to experimentally validate aspects of our design. We wish to validate if the following technical requirements are achieved -

- TR1.6 - Implement load balancing to distribute incoming traffic across multiple servers
- TR7.1 - Implement a system to auto-scale horizontally and vertically to accommodate time varying payloads
- TR1.5 - Monitor and track the workload resource utilization and operational metrics

Along with TR1.1 - Select and provision high-performance compute capacity and TR1.4 - Configure an optimal networking solution.

EXPERIMENT 1

The first set of experiments will be conducted locally using a Killercoda/Minikube environment, focusing on validating TR7.1 (Implement auto-scaling horizontally and vertically). For this experiment, we will configure the Killercoda environment to simulate a web-server setup with auto-scaling mechanisms in place.

Objective:

The objective is to monitor resource utilization and operational metrics to validate TR1.5 and if the pods scale accordingly. To observe if the HPA controller will increase and decrease the number of replicas to maintain an average CPU utilization across all Pods of 50%.

Configuration of the Environment:

Kubernetes Cluster:

- A single node Kubernetes cluster is set up in Killercoda and kubectl command-line tool is configured to communicate with the cluster.

Metrics Server:

- Unlike Minikube Killercoda doesn't come with a metrics server configured. So we deploy and configure the metric server in the cluster.
- The Metrics Server collects resource metrics from the kubelets in the cluster.

PHP-Apache Deployment:

- A sample web application (Apache httpd with PHP) as defined in [40] is deployed using a Kubernetes Deployment.
- The Deployment is defined in the php-apache.yaml manifest [40].
- It specifies resource limits for CPU (500millicores) and requests (200millicores).

HorizontalPodAutoscaler (HPA):

- A HorizontalPodAutoscaler is created for the PHP-Apache Deployment.
- The HPA is configured to maintain between 1 and 10 replicas of the Pods.
- The autoscaling is based on average CPU utilization across all Pods, targeting 50%.

Inputs:

Load Generation:

- The load generator is started by deploying a separate Pod that continuously sends queries to the PHP-Apache service.
- This is done using the load-generator Pod with an infinite loop of HTTP requests to the php-apache service.

Experiment Execution Steps:

A) Deploy the metric server

- Run `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml` to deploy the metrics server in the Kubernetes cluster.

B) Deploy PHP-Apache:

- Run `kubectl apply -f https://k8s.io/examples/application/php-apache.yaml` to deploy the PHP-Apache Deployment and Service.
- The php-apache.yaml defines the resource limits for the CPU and requests as shown below [40] -

Unset

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
```

```

spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
        - name: php-apache
          image: registry.k8s.io/hpa-example
          ports:
            - containerPort: 80
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
    - port: 80
  selector:
    run: php-apache

```

- The requests field specifies the amount of resources that Kubernetes will guarantee for a container. It is the amount of CPU or memory that a container gets initially when it starts running. The container is guaranteed to get at least 200 milliCPU when it is running. If there is not enough CPU available, the container might not be scheduled.

- The limits field specifies the maximum amount of resources that a container can use. If a container tries to use more than the specified limit, it might be throttled or terminated, depending on the resource type. The container can use up to 500 milliCPU, but not more.

C) Create HPA:

- Run ``kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10`` to create the HorizontalPodAutoscaler.
- The HPA controller will increase and decrease the number of replicas to maintain an average CPU utilization across all Pods of 50%.
- We set the minimum and maximum replicas to be between 1 and 10 and CPU Utilization of the php-apache pod to be 50%.

D) Check HPA Status:

- The HPA's status is monitored by running `kubectl get hpa`.
- Initially, the CPU consumption is 0% as there are no requests.

E) Increase Load:

- The load generator is started in a separate terminal to simulate increased traffic to the PHP-Apache service.

F) Monitor/Watch HPA Reaction:

- Watch the HPA's reaction to the increased load using ``kubectl get hpa php-apache --watch``.
- The HPA adjusts the replica count based on the configured CPU utilization target (50%).

G) Check Deployment Replica Count:

- Verify that the PHP-Apache Deployment scales up by checking ``kubectl get deployment php-apache.``

H) Stop Load Generation:

- Terminate the load generation by stopping the load-generator Pod.

I) Check HPA and Deployment Status:

- Monitor the HPA and Deployment to observe the scaling down of replicas after the load decreases

J) Further this experiment is tried by changing the CPU percent (to 70% and 20%) and observing how the scaling up and down happens.

Expected Outputs:

HPA Status:

- Initially, the HPA shows 0% CPU consumption as there are no requests.
- During increased load, the HPA adjusts the replica count to maintain the target CPU utilization (50%).

Deployment Scaling:

- During increased load, the Deployment scales up, and you should see an increased number of replicas.

Autoscaling Down:

- After load generation stops, the HPA detects decreased CPU utilization and scales down the replicas.

Stabilized State:

- After a few minutes, the system stabilizes, and the HPA maintains the desired state based on the configured metrics.

EXPERIMENT 2

In the second set of experiments, AWS will be utilized to deploy a basic infrastructure and a web application which spans across multiple Availability Zones, creating a near-real-world setting. The focus here is to validate TR1.6 (Implement load balancing), TR7.1 (Implement auto-scaling horizontally and vertically), and TR1.5 (Monitor and track resource utilization and operational metrics) along with TR1.1 (Select and provision high-performance compute capacity) and TR1.4 (Configure an optimal networking solution).

Configuration of the Environment:

Objective:

The objective is to assess how effectively incoming traffic is distributed across multiple servers, ensuring load balancing is functioning as intended. Additionally, we will implement auto-scaling

measures to accommodate varying workloads, and monitor resource utilization and operational metrics to validate TR1.5.

Prerequisites:

A) Load Balancer and Target Group:

- A target group is created by specifying the protocol to be HTTP and port as 80 for the IPv4 targets namely the web server instances. Further the health checks are enabled such that the associated load balancer periodically sends requests to the registered targets to test their status.
- An internet-facing application level load balancer is created to route incoming HTTP and HTTPS traffic (metric being no of requests) across multiple targets - Amazon EC2 instances based on request attributes. When the load balancer receives a connection request, it evaluates the listener rules in priority order to determine which rule to apply, and if applicable, it selects a target from the target group for the rule action.
- Availability zone - us-east-1a and us-east-1d is chosen. The load balancer routes traffic to targets in these Availability Zones only.
- Now the LB is ready to be tied to an Auto scaling group.

B) Security Group:

A security group is created for our launch template or load balancer.. The security group allow access from the load balancer on both the listener port (port 80 for HTTP traffic, port 443 for HTTPS traffic and port 22 for SSH traffic) and the port used for health checks.

C) Amazon Machine Image (AMI):

We create an Amazon Machine Image defined as the source template for our Amazon EC2 instances. This includes custom configurations, software installations, etc. The web application used for our experiments is a fibonacci application discussed in the ECE 547 Lab lecture. [41]

D) Virtual Private Cloud (VPC):

We used the default VPC for our experimentation

Environment:

Step 1: Set up a Launch Template:

- We specified various details like the above created AMI, instance type as t2.micro EC2 instance, network settings, security group as created above.
- We ensured the security group allows traffic from the load balancer.

Step 2: Create an Auto Scaling Group:

- On the Auto Scaling group tab in EC2 dashboard , we chose create an Auto scaling group. We provided a name for your Auto Scaling group and chose the launch template created above.
- The default VPC was chosen and the corresponding AZs were configured similar to the load balancer - us-east-1a and us-east-1d.
- The Auto Scaling group to the existing load balancer.
- We configured health checks using Elastic Load Balancing health checks.
- In group size section we choose the desired capacity to be 2 with an intention of deploying 1 instance each in each Availability zone everytime. The minimum capacity and maximum capacity provided is 2 and 5 i.e in case of scaling out based on a metric, the maximum instances that can be deployed are 5 and when scaling in the minimum should 2.
- An auto scaling policy is configured - in this experiment we used Target Tracking policy where the metric - CPU utilization is monitored every 5 minutes and if the CPU utilization is greater than 50% then the auto scaling group invokes creation of a new instance to handle the load and when it is less than 35% then the instances are scaled in accordingly to prevent under utilization. An initial warmup interval of 300 seconds is defined
- Then we add a notification by creating a SNS topic and providing the email address. (The subscription was confirmed)
- On creation of the Target Tracking policy two new cloudwatch Alarms are automatically created - TargetTrackingAlarmHigh and TargetTrackingAlarmLow which is breached when the CPU utilization verifies and sends a notification and invokes auto scaling.

Step 3: We further created a Cloudwatch dashboard with different widgets to monitor CPU utilization and NetworkPackets (In) and NetworkPackets (Out) and so on.

Step 4: Checks after the Auto scaling group is launched:

- From the Auto Scaling groups page, verify that the load balancer is attached to the Auto Scaling group.
- Check the details tab to see attached load balancer target groups or Classic Load Balancers.

- Check the activity tab to verify that instances have launched successfully.
- On the Instance management tab, verify that instances are in-service.

Inputs:

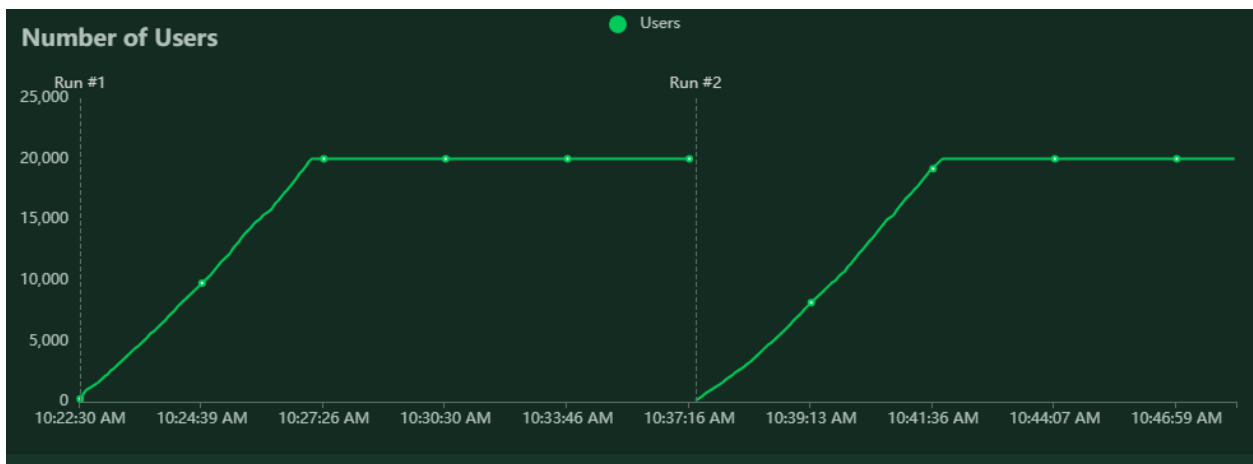
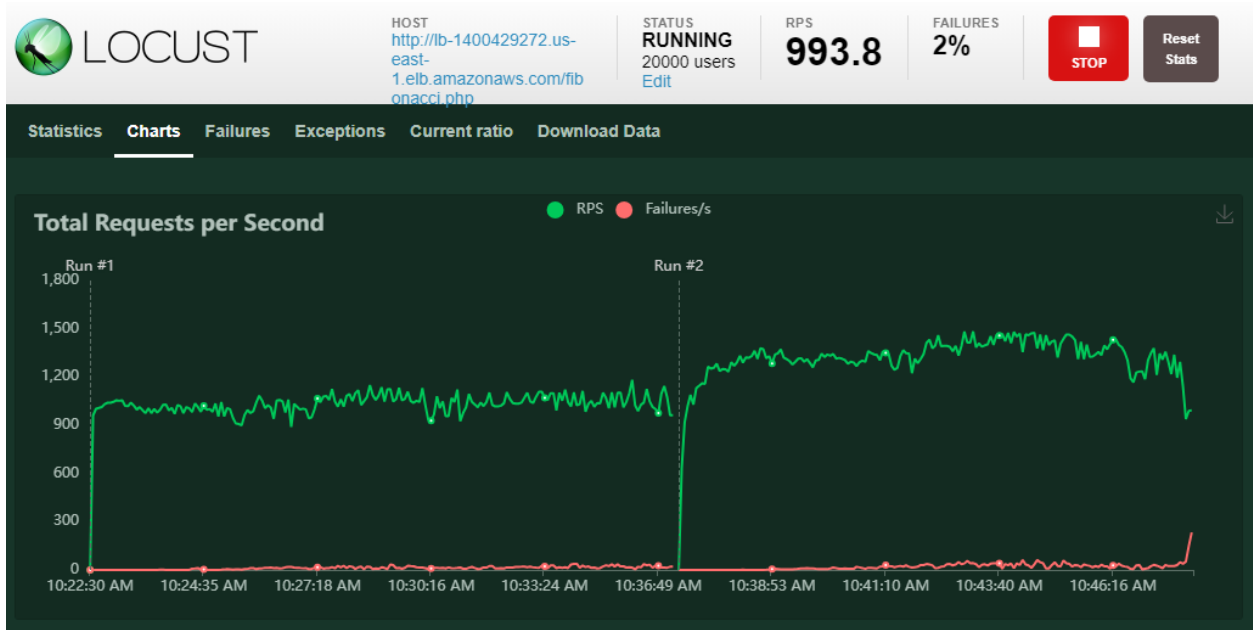
Locust tool is used for load generation and hitting the load balancer with the corresponding load to validate the behavior of the load balancer

Expected Outputs:

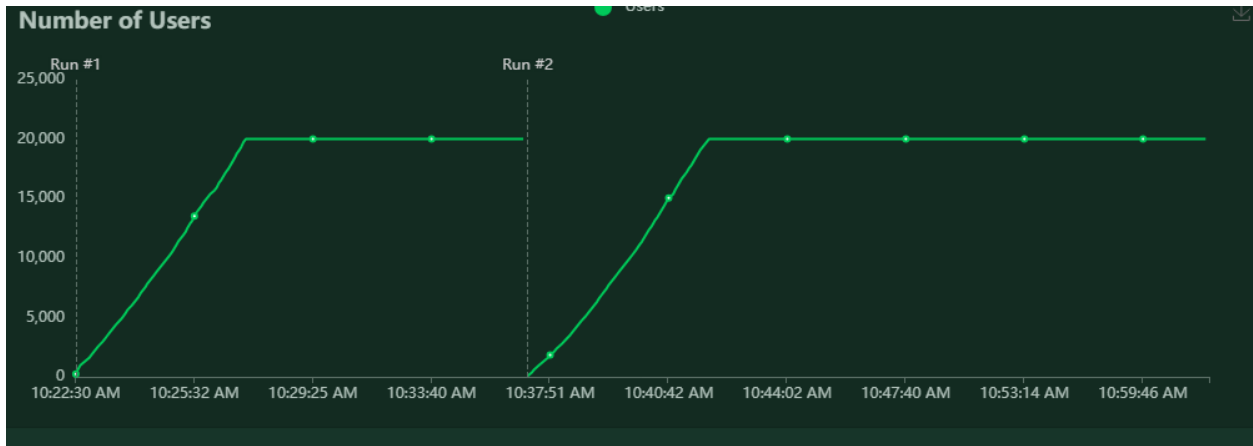
- Successful creation of the Auto Scaling group.
- Instances launched and registered with the load balancer.
- Instances passing health checks.
- Auto Scaling group configured to handle traffic through the load balancer.
- The instances are automatically scaled up and down when the CPU utilization varies. A SNS notification is sent to our email in that case.

6.2 Workload generation with Locust

In this experiment, Locust, a load testing tool, has been employed to simulate a controlled workload consisting of HTTP requests. From the Locust graph, the configuration includes a maximum of 20,000 users being spawned into the system at a rate of 80 users per second over a duration of 3600 seconds. The average number of requests per second is observed to be around 993.8, and instances of failure are documented, dependent on the health of the EC2 instances. Every run is specifically analyzed to understand the system's behavior under these predefined conditions. By utilizing Locust, the experiment aims to mimic real-world scenarios, providing insights into the responsiveness of the system i.e whether the instances are scaled out or in and if the load is balanced across all instances. The gathered metrics serve as valuable data for assessing and improving the overall performance of the system under varying workloads. The following figures show the total number of users and the total requests per second for different runs but for the same load profile described above. As shown the load profile spawns 20000 users initially and then remains constant till the duration of the test.



Further in the graph below, there was a brief scenario where an instance was unhealthy so briefly all the requests to the load balancer failed till the auto scaling group spawned another healthy instance after detecting the failure as part of the health check.



6.3 Analysis of the result

Experiment 1:

In Tab-1 of the terminal,

The metrics server is deployed in the kubernetes cluster. It collects resource utilization data from various sources in the cluster. It then exposes this data through the Kubernetes Metrics API, allowing other components and tools to access and use it. Once the metrics server pod is in the ready state, we deploy the php-apache web app using a deployment manifest.

Further the horizontal pod scaler is deployed by specifying the target CPU utilization as 50% with min pods as 1 and max pods that can be spawned as 10.

In Tab 2,

We deploy the load generator pod by using the busybox:1.28 image, `/bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"` is the command that will be run inside the pod. It starts an infinite loop that uses wget to make HTTP requests to the "http://php-apache" endpoint every 0.01 seconds. This effectively simulates a continuous load on the specified endpoint.

[illegible]

Simultaneously in Tab 3, we observe how the pods are getting auto scaled as the load increases and hence CPU utilization increases as shown in Tab 1 (HPA).

Editor	Tab 1	Tab 2	Tab 3	+
NAME			READY	STATUS RESTARTS AGE
load-generator			1/1	Running 0 3m18s
php-apache-598b474864-67n8j			1/1	Running 0 2m54s
php-apache-598b474864-7b4r4			1/1	Running 0 8m11s
php-apache-598b474864-9j6sn			1/1	Running 0 2m54s
php-apache-598b474864-9j2qf			1/1	Running 0 2m24s
php-apache-598b474864-1wk6h			1/1	Running 0 2m24s
php-apache-598b474864-rf5qm			1/1	Running 0 2m54s
php-apache-598b474864-x82zx			1/1	Running 0 2m24s
controlplane \$ kubectl get pods				
NAME			READY	STATUS RESTARTS AGE
php-apache-598b474864-67n8j			1/1	Running 0 4m54s
php-apache-598b474864-7b4r4			1/1	Running 0 10m
php-apache-598b474864-9j6sn			1/1	Running 0 4m54s
php-apache-598b474864-9j2qf			1/1	Running 0 4m24s
php-apache-598b474864-1wk6h			1/1	Running 0 4m24s
php-apache-598b474864-rf5qm			1/1	Running 0 4m54s
php-apache-598b474864-x82zx			1/1	Running 0 4m24s
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	7/7	7	7	10m
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	7/7	7	7	10m
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	7/7	7	7	11m
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	7/7	7	7	12m
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	1/1	1	1	13m
controlplane \$ kubectl get deployment php-apache				
NAME			READY UP-TO-DATE AVAILABLE	AGE
php-apache	1/1	1	1	13m
controlplane \$				

When the CPU utilization crosses 50%, the HPA scales up the pods to manage the load as observed in the Tab1 and 3. The autoscaling uses this algorithm -

$$\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} * (\text{currentMetricValue} / \text{desiredMetricValue})]$$

For example, desired replicas = $\text{ceil}[1 * (183/50)] = 4$ as observed in Tab1. Finally once the pods are auto scaled to 7 replicas the CPU utilization is stabilized within the target percentage. When the load generated pod is terminated, the replicas are again scaled down to min replicas - 1.

Now the next test is conducted by changing the CPU target percentage to 70% and we validate the horizontal auto scaling ability.

```
controlplane $ kubectl delete hpa php-apache
horizontalpodautoscaler.autoscaling "php-apache" deleted
controlplane $ kubectl autoscale deployment php-apache --cpu-percent=70 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
controlplane $ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/70%	1	10	1	8s

```
controlplane $ kubectl get hpa php-apache --watch
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/70%	1	10	1	71s
php-apache	Deployment/php-apache	167%/70%	1	10	1	106s
php-apache	Deployment/php-apache	250%/70%	1	10	3	2m1s
php-apache	Deployment/php-apache	121%/70%	1	10	4	2m16s
php-apache	Deployment/php-apache	73%/70%	1	10	4	2m31s
php-apache	Deployment/php-apache	73%/70%	1	10	4	2m46s
php-apache	Deployment/php-apache	71%/70%	1	10	4	3m1s
php-apache	Deployment/php-apache	66%/70%	1	10	4	3m16s
php-apache	Deployment/php-apache	76%/70%	1	10	4	3m31s
php-apache	Deployment/php-apache	69%/70%	1	10	4	3m46s
php-apache	Deployment/php-apache	73%/70%	1	10	4	4m1s
php-apache	Deployment/php-apache	74%/70%	1	10	4	4m16s
php-apache	Deployment/php-apache	38%/70%	1	10	4	4m31s
php-apache	Deployment/php-apache	0%/70%	1	10	4	4m46s
php-apache	Deployment/php-apache	0%/70%	1	10	4	9m16s

```
php-apache Deployment/php-apache 0%/70% 1 10 3 9m31s
^Ccontrolplane $
```

```

controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    4/4     4             4           21m
controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    4/4     4             4           21m
controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    4/4     4             4           23m
controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    4/4     4             4           24m
controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    4/4     4             4           25m
controlplane $ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    1/1     1             1           28m
controlplane $ 

```

As observed, when the load is getting generated, the CPU utilization crosses 70% which invokes the HPA and the replicas are spun up -

$$\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} * (\text{currentMetricValue} / \text{desiredMetricValue})]$$

$$= \text{ceil} [1 * (250/70)] = 4$$

Similarly when the load is terminated, the replicas scale down as visible.

Finally we test with the target CPU utilization as 20% -

```

controlplane $ kubectl delete hpa php-apache
horizontalpodautoscaler.autoscaling "php-apache" deleted
controlplane $ kubectl autoscale deployment php-apache --cpu-percent=20 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
controlplane $ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     <unknown>/20%   1         10        0           8s
controlplane $ kubectl get hpa
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     0%/20%          1         10        1           82s
controlplane $ kubectl get hpa php-apache --watch
NAME                REFERENCE                TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache          Deployment/php-apache     0%/20%          1         10        1           88s
php-apache          Deployment/php-apache     7%/20%          1         10        1           2m
php-apache          Deployment/php-apache     248%/20%        1         10        1           2m15s
php-apache          Deployment/php-apache     248%/20%        1         10        4           2m30s
php-apache          Deployment/php-apache     130%/20%        1         10        8           2m45s
php-apache          Deployment/php-apache     46%/20%         1         10        10          3m
php-apache          Deployment/php-apache     40%/20%         1         10        10          3m15s
php-apache          Deployment/php-apache     38%/20%         1         10        10          3m30s
php-apache          Deployment/php-apache     36%/20%         1         10        10          3m45s
php-apache          Deployment/php-apache     39%/20%         1         10        10          4m
php-apache          Deployment/php-apache     37%/20%         1         10        10          4m15s
php-apache          Deployment/php-apache     39%/20%         1         10        10          4m30s
php-apache          Deployment/php-apache     38%/20%         1         10        10          5m
php-apache          Deployment/php-apache     39%/20%         1         10        10          5m15s
php-apache          Deployment/php-apache     40%/20%         1         10        10          5m30s
php-apache          Deployment/php-apache     38%/20%         1         10        10          5m45s
php-apache          Deployment/php-apache     35%/20%         1         10        10          6m
php-apache          Deployment/php-apache     42%/20%         1         10        10          6m15s
php-apache          Deployment/php-apache     35%/20%         1         10        10          6m31s
php-apache          Deployment/php-apache     38%/20%         1         10        10          6m46s
php-apache          Deployment/php-apache     35%/20%         1         10        10          7m1s
php-apache          Deployment/php-apache     44%/20%         1         10        10          7m16s
php-apache          Deployment/php-apache     11%/20%         1         10        10          7m31s
php-apache          Deployment/php-apache     2%/20%          1         10        10          7m46s
php-apache          Deployment/php-apache     0%/20%          1         10        10          8m1s

```

In this case the $\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} * (\text{currentMetricValue} / \text{desiredMetricValue})] = \text{ceil}[1 * (248/20)] = \text{ceil}[12.4] = 13$

Since the max number of replicas can be 10 which limits auto scaling to 12, we can see that all the 10 pods are spun up to handle the load being generated.

In summary, the autoscaler is effectively responding to changes in the target CPU utilization, scaling up when needed and scaling down when the load decreases, with appropriate constraints set by the minimum and maximum replicas configuration.

Experiment 2:


As described in section 6.1, the configuration was performed and an auto scaling group was created. As seen in the activity section below, this lead to two instances being deployed in the 2 AZs provided.

Activity notifications (1)				
<input type="text" value="Filter notifications"/> < 1 >				
<input type="checkbox"/>	Send to	▼	On instance action	▼
<input type="checkbox"/>	TOPIC-3 (tpanati@ncsu.edu)		Launch, Terminate, Fail to launch, Fail to terminate	

Activity history (2)				
<input type="text" value="Filter activity history"/> < 1 >				
Status	Description	Cause	Start time	End time
Successful	Launching a new EC2 instance: i-063f3c72e1159cea4	At 2023-11-23T02:05:44Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2023-11-23T02:05:46Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2.	2023 November 22, 09:05:48 PM -05:00	2023 November 22, 09:06:20 PM -05:00
Successful	Launching a new EC2 instance: i-0a2373dedee35c2f7	At 2023-11-23T02:05:44Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2023-11-23T02:05:46Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2.	2023 November 22, 09:05:48 PM -05:00	2023 November 22, 09:06:21 PM -05:00

On creation of the pods, the Auto scaling group sends a notification to the provided email address using AWS SNS indicating a launch. During the creation of the auto scaling group, we configured that the AWS SNS notification should be sent on launch, terminate, fail to launch and fail to terminate actions.

Auto Scaling: launch for group "ASG-T3" External Inbox x



AWS Notifications

<no-reply@sns.amazonaws.com>

to me

9:06 PM (5 minutes ago) ☆ ↶ ⋮

Service: AWS Auto Scaling

Time: 2023-11-23T02:06:20.420Z

RequestId: 92a62fe7-b47b-3ad2-0110-9bac77a46734

Event: autoscaling:EC2_INSTANCE_LAUNCH

AccountId: 353246777341

AutoScalingGroupName: ASG-T3

AutoScalingGroupARN: arn:aws:autoscaling:us-east-1:353246777341:autoScalingGroup:b455fe41-aa26-47f9-a7ec-3b56ed6f9919:autoScalingGroupName/ASG-T3

ActivityId: 92a62fe7-b47b-3ad2-0110-9bac77a46734

Description: Launching a new EC2 instance: i-063f3c72e1159cea4

Cause: At 2023-11-23T02:05:44Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 2. At 2023-11-23T02:05:46Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 2.

StartTime: 2023-11-23T02:05:48.622Z

EndTime: 2023-11-23T02:06:20.420Z

Status: InProgress

StatusMessage:

Progress: 50

EC2Instanceid: i-063f3c72e1159cea4

Details: {"Subnet ID": "subnet-049b180e289dc109b","Availability Zone": "us-east-1a"}

Origin: EC2

Destination: AutoScalingGroup

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:

<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:353246777341:TOPIC-3:a80a6f0c-c1ae-4f41-ae53-bba6aa2ff552&Endpoint=tpanati@ncsu.edu>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

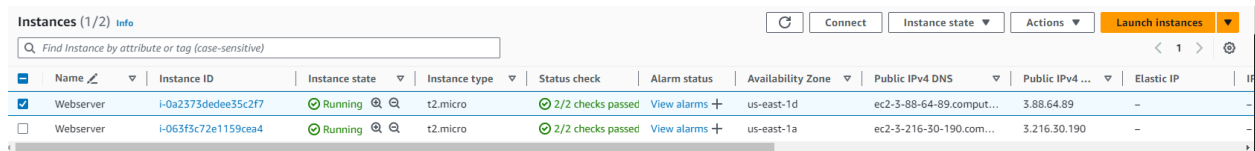
Now we checked if the application is reachable by entering the public facing load balancer’s DNS name in the URL and was able to see the application running successfully as shown below

← → ↻ ⚠ Not secure lb-588504112.us-east-1.elb.amazonaws.com/fibonacci.php/

Fib1Number of terms in the sequence: 10

Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

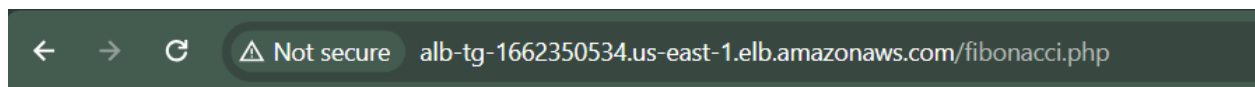
Further the health checks have passed and the web server instances are continuously serving the load (requests) generated by locust.



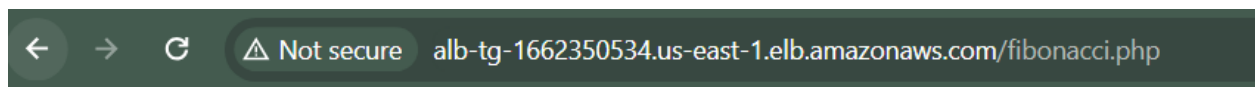
The screenshot shows the AWS Management Console 'Instances' page. It displays two EC2 instances, both in a 'Running' state. The first instance is in the 'us-east-1d' availability zone, and the second is in the 'us-east-1a' availability zone. Both instances have passed their health checks (2/2 checks passed).

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input checked="" type="checkbox"/>	Webserver	i-0a2373dedee35c2f7	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1d	ec2-3-88-64-89.comput...	3.88.64.89	-
<input type="checkbox"/>	Webserver	i-063f3c72e1159cea4	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	ec2-3-216-30-190.com...	3.216.30.190	-

It can be seen that the load balancer is balancing the corresponding requests received to the web server instances in a round robin fashion as you can see here -



Fib2 Number of terms in the sequence: 10
Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,



Fib1 Number of terms in the sequence: 10
Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

Further the various metrics that are monitored and are included in the AWS Cloudwatch dashboard -

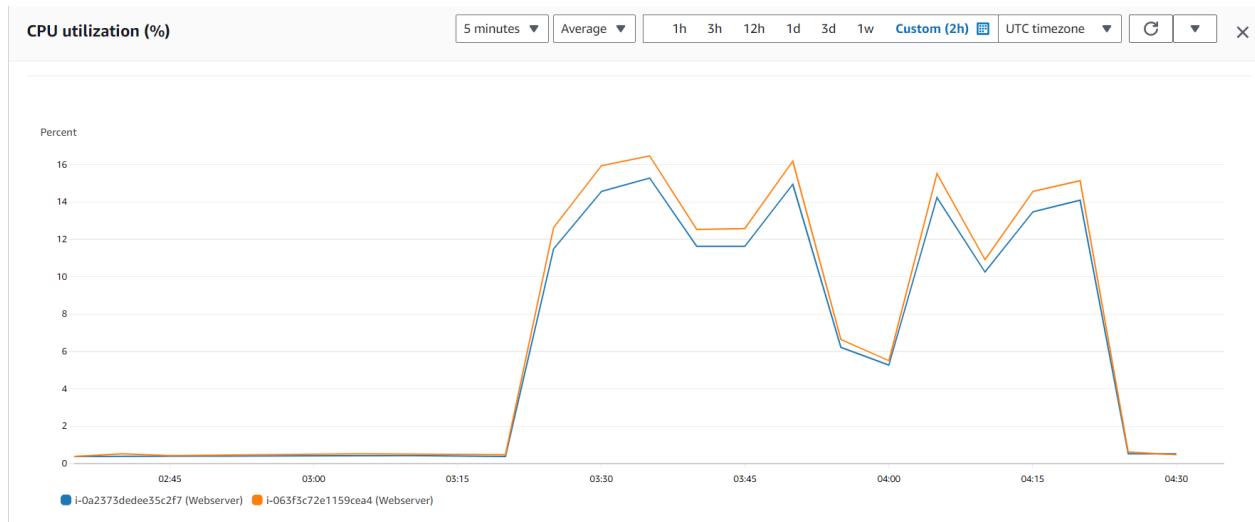
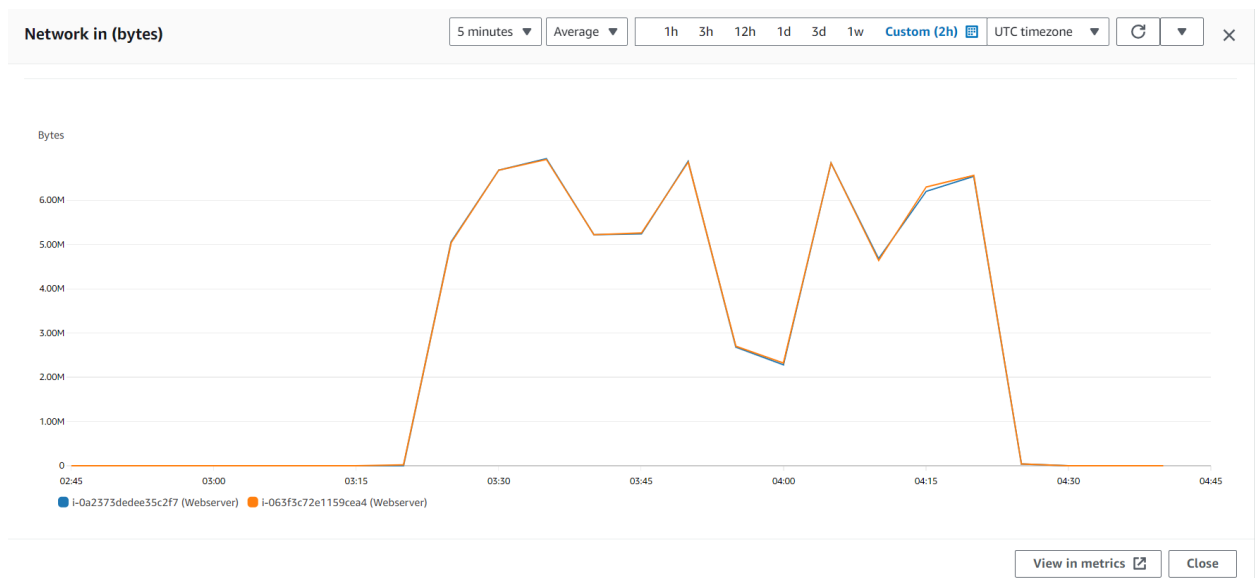
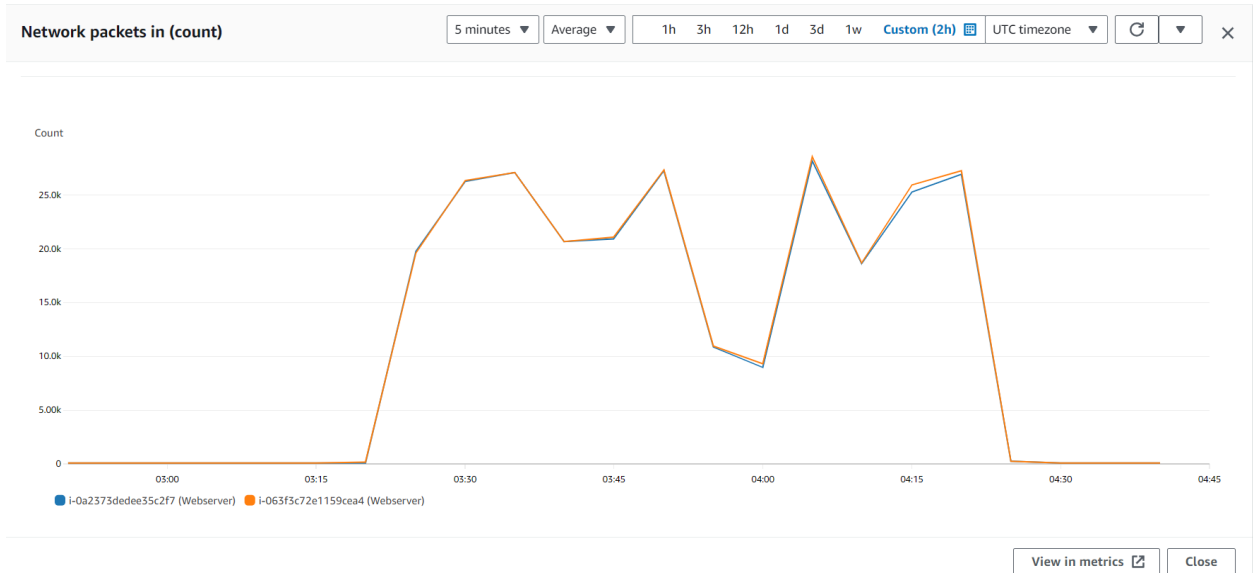


Fig: CPU Utilization for both the instances

The following graph shows the no of the packets (bytes and count) received by the web server instances. This further proves that the load is balanced between both the instances as each of them receive almost same no of requests for the given time interval.

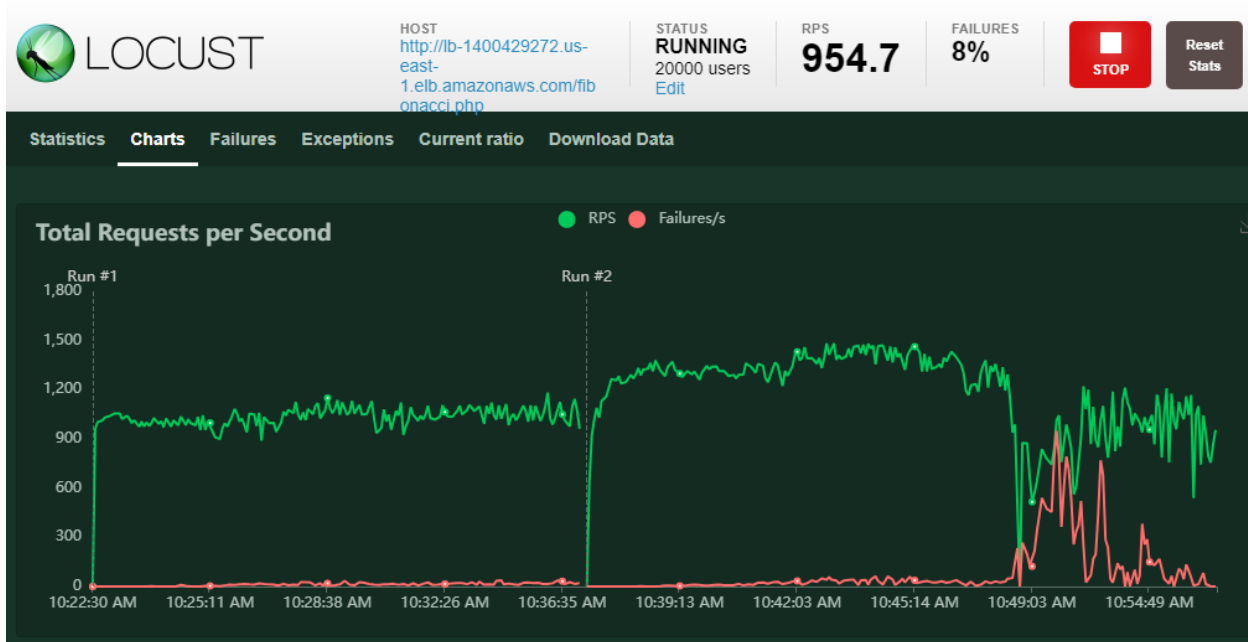




When an instance gets unhealthy the auto scaling group takes care of terminating the instance and spinning up a new one. The corresponding health status of the instance is tracked periodically by the load balancer which drives this decision. This also invokes a cloudwatch alarm and hence an SNS notification.

The load generated by locust as per the same profile mentioned in section 6.2 hits the load balancer. As shown here there is a brief period of time when the total requests per second becomes 0 and failures are observed during which it was identified that the instance became unhealthy. Immediately a new instance is spun up.





Here is the AWS SNS Notification sent to the email address notifying the same -

Auto Scaling: termination for group "ASG-T2" External

AWS Notifications <no-reply@sns.amazonaws.com> Tue, Nov 21, 10:56 AM (1 day ago)
to me ▾

Service: AWS Auto Scaling
Time: 2023-11-21T15:56:06.765Z
RequestId: d0b62fca-4d4b-376e-8845-70e602ac44a0
Event: autoscaling:EC2_INSTANCE_TERMINATE
AccountId: 353246777341
AutoScalingGroupName: ASG-T2
AutoScalingGroupARN: arn:aws:autoscaling:us-east-1:353246777341:autoScalingGroup:5b8732f5-b158-485b-82b0-c576d60a4b67:autoScalingGroupName/ASG-T2
ActivityId: d0b62fca-4d4b-376e-8845-70e602ac44a0
Description: Terminating EC2 instance: i-0aa9414220e59b76b
Cause: At 2023-11-21T15:50:23Z, an instance was taken out of service in response to an ELB system health check failure.
StartTime: 2023-11-21T15:50:23.181Z
EndTime: 2023-11-21T15:56:06.765Z
StatusCode:InProgress
StatusMessage:
Progress: 50
EC2InstanceId: i-0aa9414220e59b76b
Details: ("Subnet ID": "subnet-00745d294a410e1bb", "Availability Zone": "us-east-1d")
Origin: AutoScalingGroup
Destination: EC2

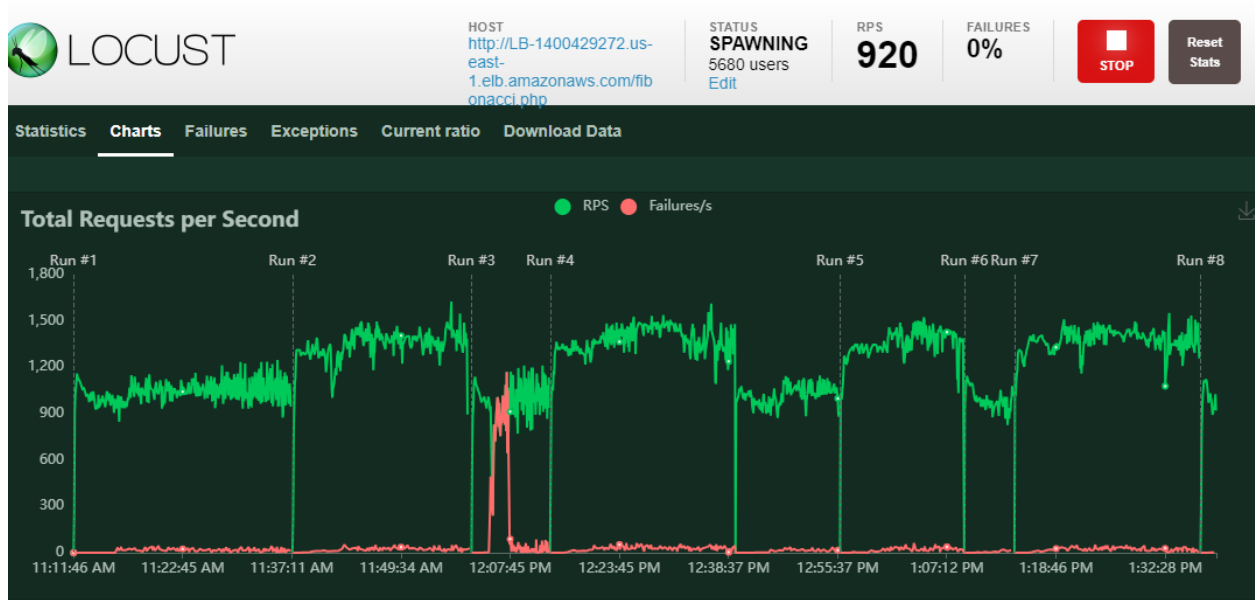
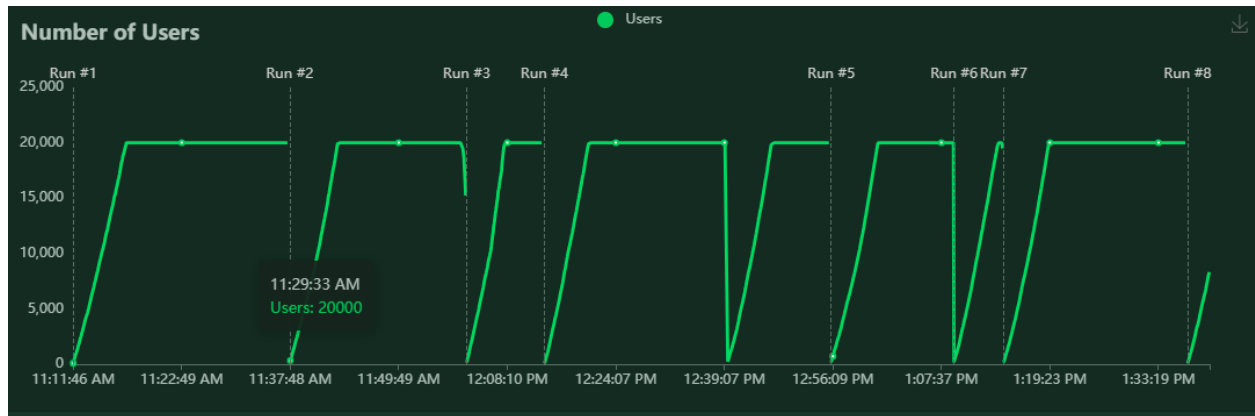
--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:353246777341:TOPIC:3e80a6f0-c1ae-4f41-ae53-bba6aa2ff552&Endpoint=tpanati@ncsu.edu>

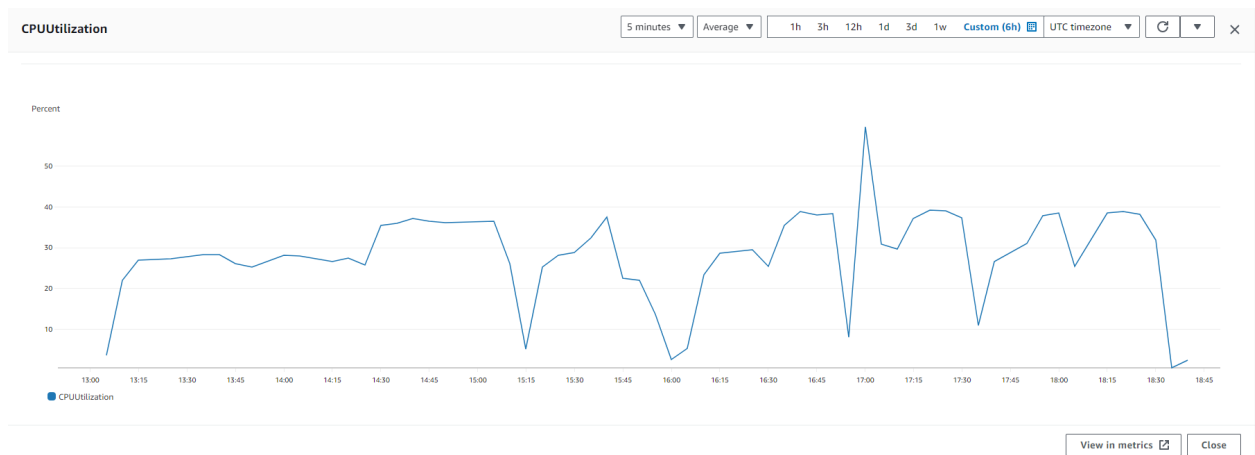
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Now to test the auto scaling functionality, we perform the same load testing (almost 5 different runs) so that the CPU utilization peaks 40% as per the Target tracking policy.

Here are the locust results -



It can be observed from the CloudWatch widget that the CPU utilization went above 40% for a brief period of time -



And immediately, the auto scaling group was invoked due to the Cloudwatch Alarm breach to spin up another instance to manage the load and prevent overutilization



Successful	Launching a new EC2 instance: i-02f24e83fb969394e	At 2023-11-21T15:56:25Z a monitor alarm TargetTracking-ASG-T2-AlarmHigh-b4d0654d-ea8b-4f04-9027-c35f857fdd09 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 3. At 2023-11-21T15:56:34Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2023 November 21, 10:56:36 AM -05:00	2023 November 21, 11:02:08 AM -05:00
------------	---	--	--------------------------------------	--------------------------------------

The above picture indicates the TargetTrackingAlarmHigh which has been breached in the process and the Auto scaling group activity notification. Hence this also invokes a SNS notification to the corresponding TOPIC and an email is received -

```
Service: AWS Auto Scaling
Time: 2023-11-21T15:57:08.206Z
Requestid: 75a62fca-6414-42b0-d73c-a0ef5d607a58
Event: autoscaling EC2_INSTANCE_LAUNCH
Accountid: 353246777341
AutoScalingGroupName: ASG-T2
AutoScalingGroupARN: arn:aws:autoscaling:us-east-1:353246777341:autoScalingGroup:5b8732f5-b158-485b-82b0-c576d60a4b67:autoScalingGroupName/ASG-T2
ActivityId: 75a62fca-6414-42b0-d73c-a0ef5d607a58
Description: Launching a new EC2 Instance: i-02f24e83fb969394e
Cause: At 2023-11-21T15:56:25Z a monitor alarm TargetTracking-ASG-T2-AlarmHigh-b4d0654d-ea8b-4f04-9027-c35f857fdd09 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 3. At 2023-11-21T15:56:34Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.
StartTime: 2023-11-21T15:56:36.540Z
EndTime: 2023-11-21T15:57:08.206Z
StatusCode: InProgress
StatusMessage:
Progress: 50
EC2Instanceid: i-02f24e83fb969394e
Details: [{"SubnetID":"subnet-00745d294a410e1bb","AvailabilityZone":"us-east-1d","InvokingAlarms":[{"AlarmArn":"arn:aws:cloudwatch:us-east-1:353246777341:alarm:TargetTracking-ASG-T2-AlarmHigh-b4d0654d-ea8b-4f04-9027-c35f857fdd09","Trigger":{"MetricName":"CPUUtilization","EvaluateLowSampleCountPercentile":"","ComparisonOperator":"GreaterThanThreshold","TreatMissingData":"","Statistic":"AVERAGE","StatisticType":"Statistic","Period":60,"EvaluationPeriods":3,"Unit":null,"Namespace":"AWS/EC2","Threshold":40},"AlarmName":"TargetTracking-ASG-T2-AlarmHigh-b4d0654d-ea8b-4f04-9027-c35f857fdd09","AlarmDescription":"DO NOT EDIT OR DELETE. For TargetTrackingScaling policy arn:aws:autoscaling:us-east-1:353246777341:scalingPolicy:8350558b-0d77-4501-965c-8bf645bd654d:autoScalingGroupName/ASG-T2:policyName/TargetTrackingPolicy","AWSSAccountid":"353246777341","OldStateValue":"OK","Region":"US East (N. Virginia)","NewStateValue":"ALARM","AlarmConfigurationUpdatedTimestamp":1700572195386,"StateChangeTime":1700582185117}]}]
```

And a new instance is created -

Instances (3) Info										
Find Instance by attribute or tag (case-sensitive)										
<div> <div>Instance state running X</div> <div>Clear filters</div> </div>										
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs
webserver	i-0aa9414228e59b76b	Running	t2.micro	2/2 checks passed	No alarms	us-east-1d	ec2-18-206-222-170.co...	18.206.222.170	-	-
webserver	i-0380cd8c30b83fc57	Running	t2.micro	Initializing	No alarms	us-east-1d	ec2-50-19-169-107.co...	50.19.169.107	-	-
webserver	i-0cad55648bb6e6a6e	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	ec2-3-239-52-32.comp...	3.239.52.32	-	-

As soon as the utilization comes below 25%, the Auto scaling group is invoked again and one instance is terminated to prevent underutilization and maintain desired capacity of the instances



Auto Scaling group: ASG-T2				
Successful	Terminating EC2 instance: i-0380cd8c30b83fc57	At 2023-11-21T16:14:50Z a monitor alarm TargetTracking-ASG-T2-AlarmLow-35c2963e-fb1e-4b95-aec7-61f650735759 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 2. At 2023-11-21T16:14:57Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2023-11-21T16:14:57Z instance i-0380cd8c30b83fc57 was selected for termination.	2023 November 21, 11:14:57 AM -05:00	2023 November 21, 11:20:42 AM -05:00

An AWS SNS Notification is sent in this case as well -



The implemented AWS infrastructure, including Auto Scaling Groups, Elastic Load Balancer, CloudWatch, and SNS, demonstrates a robust and responsive architecture. The auto-scaling policies efficiently handle fluctuations in demand, ensuring optimal resource utilization and maintaining a balance between performance and cost.

The use of CloudWatch alarms to trigger auto-scaling actions based on CPU utilization is a key aspect of achieving scalability. The notifications sent through AWS SNS provide administrators with timely alerts and insights into the system's behavior.

The successful handling of load testing scenarios, including instances becoming unhealthy and breaching CPU utilization thresholds, demonstrates the effectiveness of the auto-scaling configuration in adapting to changing conditions.

In summary, the implemented AWS infrastructure exhibits scalability, resilience, and effective load balancing, meeting the demands of varying workloads while providing administrators with the necessary visibility and control over the system.

10. Conclusion

10.1 The lessons learned

The lessons learned during this project were multifaceted. One of the challenges encountered was the necessity to balance between the tradeoffs like performance objectives with security requirements. Deciding on the appropriate trade-offs between the conflicting TRs posed a significant challenge. The inherent conflict between these requirements demanded careful consideration and research to ensure an optimal solution.

Navigating through the various alternatives and their consequences was an intellectually stimulating aspect of the project. Especially coming up with the business requirements and the corresponding technical requirements required a significant amount of research of the AWS Well-Architected Framework. It also required a deep dive into the intricacies of cloud architecture of various cloud providers like AWS, GCP and Azure, considering factors such as application response time, scalability, resource costs, and security concerns. The process of evaluating different providers, comparing their offerings, and making a final selection demanded thorough research and analysis.

Additionally, dealing with the tradeoffs inherent in cloud architecture design was both interesting and challenging. Balancing the need for a high-performance compute infrastructure with the imperative of robust data encryption added layers of complexity to the decision-making process. Each alternative presented unique advantages and drawbacks, requiring careful consideration of the project's specific requirements and objectives.

On the flip side, certain aspects of the project were unexpectedly engaging. Experimenting with Kubernetes, exploring Auto scaling aspects on AWS, and delving into the intricacies of cloud provider services added a practical dimension to the theoretical concepts learned in class. These hands-on experiences provided valuable insights into the real-world application of cloud architecture principles.

In summary, this project exposed us to the responsibilities of a cloud architect for a consumer and provided a deep understanding of various concepts revolving around cloud architecture and automation.

10.2 Possible continuation of the Project

As for the possible continuation of the project, an avenue for further exploration could involve extending the scope to cover additional aspects of the online bookstore application.

This involves extending the architecture to support deployment across multiple cloud providers (AWS, Azure, GCP) and exploring how our design adapts to the differences in services and infrastructure between providers. Further we can refine our architecture to adopt a microservices approach. Break down the application into smaller, independently deployable services. And we can conduct extensive experimentation using tools like Kubernetes for managing microservices.

Further incorporating serverless computing into our architecture would be an interesting feature by exploring the use of AWS Lambda, Azure Functions, or Google Cloud Functions for different functionalities within our application.

Implementing a hybrid cloud approach, integrating on-premises infrastructure with cloud resources would be exciting. This involves exploring solutions like AWS Outposts, Azure Arc, or Google Anthos for hybrid deployments.

Alternatively, delving deeper into the comparison between different cloud solutions, such as AWS, Azure, IBM, and Google, could provide valuable insights into the evolving landscape of cloud technologies.

Additionally, considering the rapid advancements in cloud technologies, we can work on further experimentation aspects using Kubernetes which could have potential avenues for publishing research findings or contributing to the broader discourse on cloud architecture.

11. References

[1] AWS Well Architected Framework -

<https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>

[2] Prof Ioannis Viniotis and Prof Ioannis Papapanagiotou notes and lectures -

<https://moodle-courses2324.wolfware.ncsu.edu/course/view.php?id=312#section-13>

<https://moodle-courses2324.wolfware.ncsu.edu/course/view.php?id=312#section-15>

[3] ASA Restaurant application - ECE 547 Project by - Sandeep Grande, Anirudh Vijay, Arjun

[4] CHATGPT - <https://chat.openai.com/>

[5] 7 factors that help to choose correct cloud service provider -

<https://www.binfire.com/blog/7-factors-help-choose-right-cloud-service-provider/>

[6] 8 criteria to ensure you select the right cloud service provider -

<https://cloudindustryforum.org/8-criteria-to-ensure-you-select-the-right-cloud-service-provider/>

[7] <https://allcode.com/top-aws-services/>

[8] AWS SLAs -

https://aws.amazon.com/legal/service-level-agreements/?aws-sla-cards.sort-by=item.additionalFields.serviceNameLower&aws-sla-cards.sort-order=asc&awsf.tech-category-filter=*all

- [9] GCP SLAs - <https://cloud.google.com/terms/sla>
- [10] Various services provided by AWS - <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/amazon-web-services-cloud-platform.html>
- [11] Various services provided by AWS - <https://www.antvaset.com/c/21gjd17gz4>
- [12] Various services provided by GCP - <https://cloud.google.com/products/>
- [13] Various services provided by GCP - <https://cloud.google.com/terms/services>
- [14] Various services provided by Azure - <https://azure.microsoft.com/en-us/products/>
- [15] Various services provided by Azure - <https://www.antvaset.com/c/21h7dv3g91>
- [16] AWS ELB - <https://aws.amazon.com/elasticloadbalancing/?nc=sn&loc=1>
- [17] AWS ELB Features - <https://aws.amazon.com/elasticloadbalancing/features/?nc=sn&loc=2&dn=1>
- [18] AWS IAM - <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
- [19] AWS IAM - <https://aws.amazon.com/iam/>
- [20] AWS EC2 Instance Types - <https://aws.amazon.com/ec2/instance-types/>
- [21] AWS EC2 - <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
- [22] AWS RDS - <https://aws.amazon.com/rds/>
- [23] AWS RDS Guide - <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>
- [24] AWS S3 Guide - <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html> \
- [25] AWS S3 - <https://aws.amazon.com/s3/>
- [26] AWS VPC - <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>

- [27] AWS Cloudwatch - https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_architecture.html
- [28] AWS ELB Guide - <https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html>
- [29] AWS KMS Guide - <https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html>
- [30] Performance efficiency pillar - <https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/welcome.html>
- [31] AWS Identity and Access Management (IAM):
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>
AWS ElastiCache:
<https://aws.amazon.com/pm/elasticache/>
- [32] AWS Elastic Load Balancing (ELB) and Amazon Route 53:
<https://aws.amazon.com/elasticloadbalancing/features/?nc=sn&loc=2&dn=1>
<https://aws.amazon.com/route53/what-is-dns/>
- [33] AWS Auto Scaling - <https://docs.aws.amazon.com/autoscaling/plans/userguide/what-is-a-scaling-plan.html>
- [34] AWS CloudFormation-<https://docs.aws.amazon.com/cloudformation/>
- [35] Amazon S3 - <https://docs.aws.amazon.com/AmazonS3/latest/userguide//Welcome.html>
AWS Glue - <https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html>
- [36] AWS Cost Explorer - <https://aws.amazon.com/aws-cost-management/aws-cost-explorer/>
AWS Pricing Calculator - <https://calculator.aws/#/addService>
- [37] <https://aws.amazon.com/architecture/icons/>
- [38] <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html>
- [39] <https://hbr.org/1998/03/even-swaps-a-rational-method-for-making-trade-offs>
- [40] <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>

- [41] https://github.ncsu.edu/EB2-2027/CSC547_DEMO3_FIBONACCI_ON_AWS
- [42] <https://github.ncsu.edu/EB2-2027/Demo05-Monitoring-LoadBalancer>
- [43] <https://www.freecodecamp.org/news/understanding-vpc-architecture-securing-aws-ec2-instances/>
- [44] <https://aws.amazon.com/solutions/implementations/vpc/>
- [45] <https://aws.amazon.com/solutions/implementations/vpc/>
- [46] <https://www.simplilearn.com/tutorials/aws-tutorial/aws-vpc>
- [47] <https://creately.com/diagram/example/ip83dl0h1/vpc-with-public-and-private-subnets>
- [48] <https://cloudacademy.com/blog/aws-vpc-with-high-availability-and-scalability-for-your-cms/>
- [49] https://www.softnas.com/docs/softnas/v3/snapha-html/aws_vpc_architecture__virtual_ip.html
- [50] <https://k21academy.com/amazon-web-services/aws-vpc-virtual-private-cloud/>
- [51] <https://grapeup.com/blog/the-path-towards-enterprise-level-aws-infrastructure-architecture-scaffolding/#>
- [52] <https://itnext.io/high-available-vpc-architecture-in-cloudformation-2f4d8a86f4d2>
- [53] <https://medium.com/@mrdevsecops/vpcs-components-213c4977f7da>
- [54] <https://devopsrealtime.com/deploy-scalable-vpc-architecture-on-aws-cloud/>