CSC/ECE547 - Fall 2023

Cloud Architecture

Project Report - University-Made-Easy

Submitted By

Priya Andurkar (@pandurk), Akruti Sinha (@asinha6)

Index

S.No.	Section Name	Page Number
1.1	Motivation	4
1.2	Executive Summary	4
2.1	Problem Statement	5
2.2	Business Requirements	5
2.3	Technical Requirements	5
2.4	Trade Offs and Conflicting Requirements	12
3.1	Criteria for choosing a provider	14
3.2	Provider Comparison	15
3.3	The final selection	19
3.3.1	The list of services offered by the winner	19
4.	The first design draft	23
4.1	The basic building blocks of the design	24
4.2	Top-level, informal validation of the design	25
5 -5.1	Second design draft - Use of the Well-Architected Framework	27
5.2	Discussion of pillars	28
5.3	Use of Cloudformation diagrams	29
5.4	Validation of the design	30
5.5	Design principles and best practices used	33
5.6	Tradeoffs Revisited	34
5.7	Discussion of an alternate design	35
6 - 6.1	Kubernetes experimentation - Experiment Design	36
6.2	Workload generation	37
6.3	Analysis of the results	38
10-10.1	Conclusion -The lessons learned	42

10.2	Possible continuation of the project	42
11	References	44

1.1 Motivation

In the current digital age, traditional educational boundaries are evolving. Universities are increasingly seeking innovative ways to interact, share content, and enhance the learning experiences among students and teachers. As students, our motivation behind developing this specialized platform stems from the growing need for easy collaboration, knowledge sharing, and efficient access to educational resources.

1.2 Executive Summary

For students, it's crucial to effectively manage their academic responsibilities, such as keeping up with activities, events, projects, and discussions, all while juggling coursework. Although there are multiple tools available for tracking these elements, the need to maintain consistency and check different platforms for essential information can be burdensome for both students and teachers. This problem seeks to streamline the process by combining the advantages of Piazza, Moodle, and Zoom into a single, convenient platform, offering a unified solution.

To address this challenge, we propose the development of an integrated Software as a Service (SaaS) solution designed specifically for academic environments. This platform will amalgamate the key features of multiple platforms, creating a one-stop-shop for academic activities, resource sharing, and collaboration.

2.1 Problem Statement

We aim to create a specialized SaaS platform for academic purposes, providing streaming, content sharing, and collaboration features. This platform is designed to benefit students by enabling content sharing and teachers by allowing them to upload educational materials, such as lecture recordings. The problem focuses on developing a user-friendly solution tailored to academic needs while ensuring scalability, security, and accessibility.

2.2 Business Requirements

BR1: Application should be highly available and avoid single points of failure.
BR2: Application should be secure and accessible to authenticated users only.
BR3: Application should have a low latency.
BR4: Application should scale flexibly based on varying loads.
BR5: Application should be compliant with data protection policies and regulations.
BR6: Application should have a lower cost.
BR7: Application must be compatible with multiple devices and browsers.
BR8: Application should have minimum buffering interruptions while streaming.
BR9: The application should log the user's activities, and other health metrics for analysis.
BR10: The application should be accessible globally.
BR11: The application should efficiently manage, upload, and organize content.
BR12: The application should be sustainable.
BR13: The application should enable efficient data-backups.
BR14: The application should stream a variety of content formats like videos, audios, and presentations.
BR15: The application should have an intuitive and simple interface.

BRs from solution architect perspective:
BR16: Live stream should support real-time encoding and interactive features like polls.
BR17: The application should have good customer support.
BR18: The application should be accessible to users with disabilities.

2.3 Technical Requirements

- **TR1.1:**
  - System should be available for 99% of the time, 1% downtime allowed in cases of deployment failures, hardware failure, global outage.
- **TR1.2:**
  - Implement CI/CD for streamlining application development and deployment processes.
- **TR1.3:**
  - Data should be replicated for recovery from db failures/faulty transactions in one copy.
- **TR1.4:**
  - Mechanism to automatically detect and recover from system/node failures.

- **TR1.5:**
    - CPU capacity and Memory should scale up when load goes over the set threshold by 30% and and should scale down when load decreases below the set threshold by 10%.
- **TR1.6:**
    - Resources should be deployed across multiple geographical locations to avoid regional outage.
- **TR1.7:**
    - Implement logging and monitoring the health metrics of all resources to detect anomalies in behaviors and failures.
- **TR1.8:**
    - Conduct periodic disaster recovery testing of high availability strategies.
- **TR1.9:**
    - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR1.10:**
    - Maintain detailed documentation for troubleshooting and recovery.
- **TR2.1:**
    - Implement logging and monitoring the health metrics of all resources to detect anomalies in behaviors and failures.
- **TR2.2:**
    - Maintain detailed documentation for troubleshooting and recovery.
- **TR2.3:**
    - Implement identity access management to enforce roles and permissions.
- **TR2.4:**
    - Data in transit over the network should be encrypted for secure transmission.
- **TR2.5:**
    - Perform vulnerability assessment scans and recognise weakness in systems to prevent data leaks and malicious use.
- **TR3.1:**
    - Implement logging and monitoring the health metrics of all resources to detect anomalies in behaviors and failures.
- **TR3.2:**
    - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR3.3:**
    - Implement CDNs to cache data from edge servers at different geographic locations.
- **TR3.4:**
    - Implement efficient data compression for optimized data transfer.
- **TR3.5:**
    - Introduce in-memory caching and integrate with third party caching tools for frequently accessed data from the database.

- **TR3.6:**
  - Implement stateless architecture to speed up requests.
- **TR3.7:**
  - Identify tenants to identify data intensive workloads e.g. PHD students frequently access huge datasets for analysis and experiments.
- **TR4.1:**
  - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR4.2:**
  - Perform vulnerability assessment scans and recognise weakness in systems to prevent data leaks and malicious use.
- **TR4.3:**
  - Implement CDNs to cache data from edge servers at different geographic locations.
- **TR4.4:**
  - Implement stateless architecture to speed up requests.
- **TR4.5:**
  - Use container orchestration to scale containers automatically.
- **TR5.1:**
  - Implement identity access management to enforce roles and permissions.
- **TR5.2:**
  - Data in transit over network should be encrypted for secure transmission.
- **TR5.3:**
  - Use secure and efficient database service to store, retrieve, and update data of different formats.
- **TR6.1:**
  - Implement CI/CD for streamlining application development and deployment processes.
- **TR6.2:**
  - CPU capacity and Memory should scale up when load goes over the set threshold by 30% and and should scale down when load decreases below the set threshold by 10%.
- **TR6.3:**
  - Implement logging and monitoring the health metrics of all resources to detect anomalies in behaviors and failures.
- **TR6.4:**
  - Implement CDNs to cache data from edge servers at different geographic locations.
- **TR6.5:**
  - Identify tenants for allocating resource quotas.
- **TR6.6:**
  - Choose the most appropriate pricing models.
- **TR6.7:**
  - Implement data cleanup strategies and use tiered storage options.

- **TR6.8:**
  - Implement policies to prevent long-running and unnecessary instances.
- **TR7.1:**
  - Maintain detailed documentation for troubleshooting and recovery.
- **TR7.2:**
  - Implement push notification service to prompt updates to users to avoid incompatibility.
- **TR7.3:**
  - Implement responsive design for adapting on multiple devices.
- **TR8.1:**
  - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR8.2:**
  - Implement a dynamic bitrate adaptation algorithm to adjust the video quality based on the user's network bandwidth.
- **TR8.3:**
  - The application should use a pre-buffering mechanism to download a portion of the video before it begins playback.
- **TR8.4:**
  - The application should implement error correction and resilience techniques to minimize the impact of network errors on the video stream.
- **TR9.1:**
  - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR9.2:**
  - Store log in a secure and durable location.
- **TR9.3:**
  - The application should provide a way to easily analyze the logs to identify trends and patterns.
- **TR10.1:**
  - Resources should be deployed across multiple geographical locations to avoid regional outage.
- **TR10.2:**
  - Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- **TR10.3:**
  - Implement CDNs to cache data from edge servers at different geographic locations.
- **TR11.1:**
  - The application should provide a user-friendly interface for uploading and organizing content.
- **TR11.2:**
  - The application should implement metadata tagging, versioning, and permissions.

- **TR11.3:**
  - The application should provide a way to integrate with other content management systems.
- **TR12.1:**
  - Mechanism to automatically detect and recover from system/node failures.
- **TR12.2:**
  - The application should be designed to be energy-efficient and scalable to accommodate future growth.
- **TR12.3:**
  - The application should use cloud-based resources to reduce infrastructure costs and improve agility.
- **TR13.1:**
  - Data should be replicated for recovery from db failures/faulty transactions in one copy.
- **TR13.2:**
  - The application should implement a regular backup schedule to ensure that all data is regularly backed up.
- **TR13.3:**
  - The backups should be stored in a secure and durable location to avoid malicious use.
- **TR14.1:**
  - Implement responsive design for adapting on multiple devices.
- **TR14.2:**
  - The application should support a wide range of content formats and codecs.
- **TR14.3:**
  - The application should be able to transcode content to different formats and resolutions on the fly.
- **TR15.1:**
  - The application should provide recommendations to help users discover new content.
- **TR15.2:**
  - The application should provide concise and efficient search functionality.

BRs	Associated TRs	Justification
BR1	TR1.1, TR1.3, TR1.4, TR1.8	• To ensure high availability and avoid single points of failure. • High availability also helps to improve the customer experience by reducing the likelihood of downtime and outages. • Avoiding single points of failure can help to protect businesses from financial losses and reputational damage.

BR2	TR2.1, TR2.3, TR2.4, TR2.5	• To ensure the security of the application and the data it stores. • Data security is especially important for businesses that handle sensitive customer or financial data. • A secure application can help to protect businesses from costly data breaches and cyberattacks.
BR3	TR3.1, TR3.2, TR3.3, TR3.4, TR3.5, TR3.6, TR3.7	• To reduce latency and improve the performance of the application. • Low latency is important for businesses that need to provide a responsive and user-friendly experience to their customers. • Improved application performance can also help businesses to increase productivity and reduce costs.
BR4	TR1.5, TR4.1, TR4.5, TR6.2	• To ensure that the application can scale flexibly to meet varying loads. • This is important for businesses that experience seasonal or unpredictable traffic spikes. • A scalable application can help businesses to avoid performance bottlenecks and ensure that their customers have a good experience even during peak periods.
BR5	TR5.1, TR5.2, TR5.3	• To ensure that the application complies with data protection policies and regulations. • This is important for businesses that handle personal or sensitive data. • Failure to comply with data protection laws and regulations can result in hefty fines and reputational damage.
BR6	TR6.3, TR6.6, TR6.7, TR6.8	• To reduce the cost of the application. • Reducing the cost of the application can help businesses to improve their bottom line and become more competitive.
BR7	TR7.1, TR7.2, TR7.3	• To ensure that the application is compatible with multiple devices and browsers. • This is important for businesses that want to reach a wider audience and provide a seamless user experience.

		• A cross-platform compatible application can help businesses to increase customer engagement and grow their business.
BR8	TR8.1, TR8.2, TR8.3, TR8.4	• To minimize buffering interruptions while streaming. • This is important for businesses that provide streaming services such as video or audio on demand. • A smooth and uninterrupted streaming experience can help businesses to improve customer satisfaction and retention.
BR9	TR9.1, TR9.2, TR9.3	• To log user activities and other health metrics for analysis. • This information can be used to improve the application, identify and resolve problems, and gain insights into how users are interacting with the application. • By analyzing user activity and health metrics, businesses can make informed decisions about how to improve their products and services.
BR10	TR10.1, TR10.2, TR10.3	• To ensure that the application is accessible globally. • This is important for businesses that want to reach a global audience. • A globally accessible application can help businesses to expand into new markets and increase their revenue.
BR11	TR11.1, TR11.2, TR11.3	• To efficiently manage, upload, and organize content. • It will help businesses to save time and money by reducing the amount of manual effort required to manage, upload, and organize content. • A well-organized and easy-to-use content management system can also help businesses to improve the customer experience by making it easier for users to find the content they are looking for.
BR12	TR12.1, TR12.2, TR12.3	• To make the application sustainable and scalable.

		• A sustainable application is one that can be maintained and updated without incurring excessive costs. • A scalable application is one that can be scaled up or down to meet the changing needs of the business.
BR13	TR13.1, TR13.2, TR13.3	• To enable efficient data-backups. • Data backups are important for protecting businesses from data loss and corruption. • Efficient data backups can help businesses to minimize the downtime and costs associated with data loss.
BR14	TR14.1, TR14.2, TR14.3	• To stream a variety of content formats like videos, audios, and presentations. • This can be useful for businesses that provide streaming services or that need to share content with employees or customers who are using different devices and browsers. • A variety of supported content formats can help businesses to reach a wider audience and provide a more engaging user experience.
BR15	TR15.1, TR15.2	• To provide an intuitive and simple interface. • This is important for making the application easy to use for users of all skill levels (professors and students alike) • A user-friendly interface can help businesses to improve the customer experience and increase adoption of the application.

2.4 Trade Offs and Conflicting Requirements:

1. **TR1.1 (System availability) vs. TR1.2 (CI/CD):** CI/CD can lead to more frequent deployments, which can increase the risk of downtime. To mitigate this risk, it is important to have a robust deployment process and to test deployments thoroughly before they are released to production.
2. **TR1.3 (Data replication) vs. TR3.4 (Data compression):** Data compression can reduce the amount of data that needs to be replicated, but it can also increase the processing overhead required to compress and decompress the data. It is important to choose a data compression algorithm that balances the need for performance with the need to reduce the amount of data that needs to be replicated.

3. **TR1.4 (Automatic failure recovery) vs. TR3.6 (Stateless architecture)**: Stateless architectures are easier to scale and recover from failures, but they may not be suitable for all workloads. For example, a stateless architecture may not be suitable for a workload that requires maintaining state information about users.
4. **TR1.5 (CPU and memory scaling) vs. TR6.6 (Pricing models)**: Some pricing models charge for CPU and memory usage, so scaling up CPU and memory can increase costs. It is important to choose a pricing model that is aligned with our application's needs.
5. **TR1.6 (Multi-regional deployment) vs. TR6.7 (Data cleanup strategies)**: Data cleanup strategies can help to reduce the amount of data that needs to be stored, but they can also lead to data loss. It is important to carefully consider the risks and benefits of different data cleanup strategies before deploying them.
6. **TR1.9 (Load balancing) vs. TR11.3 (Integration with other content management systems):** Load balancing can distribute traffic evenly across multiple instances of an application, but it can also make it more difficult to integrate with other content management systems. It is important to choose a load balancing solution that supports integration with other content management systems. Load balancing can make it more difficult to integrate with other content management systems because it can introduce complexity and additional overhead to the integration process.
7. **TR1.5 (Controlled Scaling) and TR3.6 (Stateless Architecture):** The tradeoff lies in balancing the benefits of controlling scalability using EC2 and it's ability to add flexible configurations with the inherent statelessness of Lambda. Making the right choice for the optimal performance and scalability in mind to achieve our desired requirement.

AWS WAF:

We employed the AWS Well-Architected Framework (WAF) [9] as a tool to help us with the requirement-creation process. This is a list of TRs arranged according to the WAF's pillars:

Operational Excellence: TR1.2, TR1.7, TR1.9, TR1.10, TR3.7, TR6.7
Security: TR2.3, TR2.4, TR2.5, TR13.3
Reliability: TR1.1, TR1.3, TR1.4, TR1.6, TR1.8, TR4.5, TR6.5
Performance Efficiency: TR1.5, TR3.3, TR3.4, TR3.6, TR3.7
Cost Optimization: TR6.6, TR6.7
Sustainability: TR12.2, TR12.3

Section 3:

3.1 Criteria for choosing a provider

We develop a number of important standards to direct our process of choosing cloud providers. When selecting a provider, the following factors are crucial for our application:

1. **Reputation**: The provider's reputation in the IT world is of paramount importance to avoid issues related to outages, data leaks, and other negative incidents. It also helps reassure the customers.
2. **Reliability**: Ensuring that the application is available 24/7 without interruptions is crucial for its reliability and effectiveness. Taking it into consideration is important considering multiple students, teachers are likely to use the application throughout multiple different hours of the day.
3. **Data Security**: Security of data and applications is a top priority, and a provider's track record in this area is essential. Additionally since the data of any university is proprietary, it is essential to make sure that their resources are protected.
4. **Service Availability**: The provider's service catalog and its alignment with our technical requirements are fundamental.
5. **Performance Enhancement**: High-performance solutions that can accelerate computational speed are valuable for our application. The users often don't enjoy higher wait times to be served with the response to their requests, so it's impertinent that the application has lower latency.
6. **Scalability and Flexibility**: The ability to adapt to changing requirements and scale, including the selection of suitable programming languages and services, is vital. There could be times of the day when the application sees a surge of requests and it's essential to scale up the instances to keep the performance up.
7. **Service Familiarity**: Familiarity with the cloud provider's services among team members can streamline implementation and reduce retraining needs.
8. **Pricing Models:** Selecting the most cost-effective pricing model for optimizing our application is a significant factor as it is essential to provide good service without burning a hole in the pockets.
9. **Disaster Recovery**: A cloud provider offering disaster recovery services is valuable for quick recovery from single point failures, hardware issues, regional outages and more. It is an essential component of high availability and hence a crucial consideration.
10. **Service Variety:** A broad range of services from the cloud provider is advantageous for implementing various business and technical requirements.
11. **Regulatory Compliance**: Compliance with industry regulations and standards is important. It often helps gain the trust of customers and hence needs to be considered strongly.
12. **API Integration**: Offering a comprehensive set of APIs for easy integration with existing systems is desirable since it makes the design easy and more loosely coupled. It helps in effective debugging in case of faults and failures.
13. **Global Presence:** While beneficial, the global presence of the cloud provider is of slightly lower priority in this context. However, it helps to keep this consideration from the

perspective of planning for the future if the customer base becomes more global and diverse.

14. **Ecosystem**: A thriving partner ecosystem is beneficial but ranks lower in importance compared to other criteria. Having a strong community support is always helpful for easier troubleshooting and good design patterns.

3.2 Provider Comparison

In the below comparison, we are scoring our services out of 3, as there are three main contenders. Here, 1 signifies the worst score and 3 signifies the best score. These scores are in relevance to the requirements of this application only.

Parameter	Associated TRs	AWS		GCP		Azure	
		Justification	Score	Justification	Score	Justification	Score
Reputation	All TRs would be justified by choosing a well reputed provider	AWS has a very good reputation in the IT world. It is one of the most popular and trusted cloud providers, and it has a long track record of reliability and security.	3	GCP is a top contender in cloud providers globally. It has been preferred due to its reliability, security, and innovation. Highly reputed for AI, ML.	2	Azure is one of the leading cloud providers globally, known for its strong reputation and presence in the enterprise sector. Favored by organizations with existing Microsoft infrastructure	2
Reliability	TR1.1-TR1.10, TR3.2, TR3.3	AWS is highly reliable. It offers a 99.9% uptime guarantee for most of its services. AWS also has a number of features in place to ensure high availability, such as load balancing, auto scaling, and disaster recovery.	3	GCP offers SLA for high uptime for at least 99.95% over a given month for various services.	3	Azure provides a Service Level Agreement (SLA) for high uptime, which typically guarantees at least 99.9% availability for its various services. It has a reliable infrastructure which is distributed across the globe.	3
Data Security	TR2.3, TR2.4, TR2.5, TR5.1, TR5.2, TR5.3	AWS takes data security very seriously. It has a number of security measures in place to protect customer data, such as encryption, access control, and auditing. AWS also complies with a variety of industry regulations and standards.	3	GCP is highly secure and provides features like encryption, IAM, DDos protection. Also adheres to the security standards and certifications.	3	Azure offers a range of features such as encryption, IAM, and threat detection. Also compliant with standards and data protection regulations.	3
Service Availability	All TRs would be justified by	AWS has a very broad service catalog. It offers a wide range of services to meet the	3	GCP provides a large range of services to meet multiple kinds of business	2	Azure provides a large number of services which will help with all our technical requirements.	2

	choosing a cloud provider the widest range of services available which can be used for solving this problem	needs of businesses of all sizes. AWS services are also well-aligned with our technical requirements		requirements. These provisions also align with our technical requirements.			
Performance Enhancement	TR3.2, TR3.3, TR3.4, TR3.5, TR3.7	AWS offers a variety of high-performance solutions, such as high-performance computing (HPC) instances, graphics processing units (GPUs), CloudFront, and machine learning (ML) accelerators. These solutions can help to accelerate computational speed	3	They have various services like CDN for reduced latency, Bigtable and BigQuery for high-performance storage and analytics. Such services can help with the better performance of the applications.	3	Azure offers performance optimization services, such as Azure Content Delivery Network (CDN) and Azure SQL Database, which enhance content delivery and database performance.	3
Scalability and Flexibility	TR1.5, TR1.1, TR1.9	AWS is very scalable and flexible.	3	Kubernetes Engine enables efficient container orchestration for scaling applications and ensures flexibility in handling varying workloads.	3	Services like Azure Kubernetes Service (AKS) enable efficient container orchestration for scaling applications.	3
Service Familiarity	All TRs would be justified as familiarity would	Highest familiarity and experience	3	Moderate familiarity and no experience	2	No familiarity and no experience	1

	make design decisions easier						
Pricing Models	TR6.1-TR6.8 to ensure that the lower costs are incurred	AWS offers a variety of pricing models, so we can choose the one that best meets our needs. For example, we can pay for resources on demand or we can reserve resources in advance to save money.	3	GCP provides flexible pricing models, including pay-as-you-go, sustained use discounts, and custom pricing options.It also offers a pricing calculator to estimate costs based on resource usage.	3	Azure offers flexible pricing models, including pay-as-you-go, reserved instances, and hybrid licensing options. It also helps estimate costs based on resource usage.	3
Data Center Proximity	TR1.6, TR3.3, TR10.1, TR10.2	AWS has a global network of data centers in over 200 countries and territories.	3	GCP's global network of data centers allows users to choose the region closest to their target audience. Currently has 39 regions in operation.	3	Azure currently has 300+ data centers arranged into multiple regions which ensures data resilience and quick recovery in case of disruptions.	3
Disaster Recovery	TR1.1, TR1.3, TR1.4	AWS offers a variety of disaster recovery services, such as Amazon Aurora Global Database and Amazon Disaster Recovery.	3	GCP provides disaster recovery solutions like data replication, snapshots, and backup services to ensure data resilience and recovery.	3	Azure offers disaster recovery solutions like Azure Site Recovery, which ensures data resilience and quick recovery in case of disruptions.	3
Service Variety	All TRs are justified because a diversified catalog would help us	AWS offers a very broad range of services, including computing, storage, networking, databases, analytics, machine learning, and artificial intelligence. This means that we	3	GCP offers a variety of services, like compute, storage, databases, ML, IoT. This also caters to a diverse set of business needs.	3	Azure provides a broad range of services, from virtual machines and databases to AI and machine learning tools, catering to diverse business needs.	3

	choose the right services	can use AWS to implement a wide range of business and technical requirements.					
Regulatory Compliance	TR5.1, TR5.2, TR5.3	AWS complies with a variety of industry regulations and standards, such as PCI DSS, HIPAA, and SOC 2.	3	GCP complies with a range of industry standards and regulations such as GDPR, CJIS, NIST, and more.	3	Azure complies with a range of industry standards and regulations such as PCI DSS, GDPR, NIST, and more.	3
API Integration	TR14.1, TR14.2, TR15.2	AWS offers a comprehensive set of APIs for easy integration with existing systems. This means that we can easily integrate the AWS applications with our on-premises systems.	3	GCP offers robust APIs for integration with its services and third-party applications, enabling seamless data exchange and automation.	3	Azure offers robust APIs for integration with its services and third-party applications, enabling seamless data exchange and automation.	3
Ecosystem	All TRs are justified with this criteria because we may need third-party tools for certain functionalities	AWS has a thriving partner ecosystem. There are many third-party tools and services that are available to help us use AWS more effectively.	3	GCP has a vast ecosystem of partners, third-party tools, and a developer community, making it easier for businesses to build and manage their cloud solutions efficiently.	3	Azure boasts a robust ecosystem with a variety of partners, third-party tools, and a large developer community, facilitating efficient cloud solution development and management.	3

3.3 The final selection

The services provided by the main three are largely comparable. However, because AWS offers attractive pricing models, a big catalog of services, and we are more familiar with AWS than the other cloud providers, we will choose AWS. As a result, it also accounts for the greatest total score.

3.3.1 The list of services offered by the winner

TR1.1:
AWS Elastic Load Balancing (ELB) for distributing traffic and ensuring high availability.
AWS CloudWatch for monitoring and creating alarms for resource health and performance.

TR1.2:
AWS CodePipeline for building, testing, and deploying applications.
AWS CodeBuild for building and testing code.

TR1.3:
Amazon RDS (Relational Database Service) Multi-AZ deployments for database replication and failover.

TR1.4:
AWS Auto Scaling for automatically adjusting the number of instances based on workload.

TR1.6:
AWS Global Accelerator and Amazon Route 53 for global load balancing and routing.

TR1.7:
AWS CloudWatch for monitoring resource health and setting alarms.
AWS CloudTrail for auditing and tracking user activity.

TR1.8:
AWS Disaster Recovery services, such as AWS Disaster Recovery and AWS Backup.

TR1.9:
AWS Elastic Load Balancing (ELB) for distributing traffic.

TR2.2:
AWS Systems Manager for managing and documenting resources.

TR2.3:
AWS Identity and Access Management (IAM) for controlling access to resources.

TR2.4:
AWS Virtual Private Cloud (VPC) for creating secure network environments.

TR2.5:
AWS Inspector and AWS Security Hub for security assessment and monitoring.

TR3.4:
AWS Glue, AWS Lambda

TR3.5:
Use in-memory caching solutions like Amazon ElastiCache.

TR3.7:
AWS Resource Groups and tagging for resource organization and identification. AWS Organizations can be used for better IAM. AWS CloudTrail and AWS Config can be used allowing you to monitor and track actions performed on resources by different tenants, aiding in tenant identification and management.

TR4.2:
AWS Inspector and AWS Security Hub for security assessment and monitoring.

TR5.2:
Use SSL/TLS encryption for data in transit.

TR5.3:
Amazon RDS, Amazon DynamoDB, or other AWS database services.

TR6.6:
Consider AWS Cost Explorer and use AWS Pricing Calculator for cost optimization.

TR6.7:
AWS S3 Object Lifecycle policies for data cleanup and S3 Storage Classes for tiered storage.

TR6.8:
AWS Config for setting policies and rules for resource management.

TR7.2:
Use Amazon SNS (Simple Notification Service) for push notifications.

TR7.3:
Amazon CloudFront: A content delivery network (CDN) that delivers content to users with low latency and high availability.

TR8.1:
AWS Elastic Load Balancing (ELB): A load balancer that distributes incoming traffic across multiple Amazon EC2 instances.
Amazon CloudWatch: A monitoring and alerting service that helps you monitor and manage your AWS resources.

TR8.2:
Amazon Elastic Transcoder: A video transcoding service that allows you to convert your videos into different formats and bitrates.
Amazon CloudWatch: A monitoring and alerting service that helps you monitor and manage your AWS resources.

TR8.3:
Amazon S3: A highly scalable object storage service that can store your video files.
Amazon CloudFront: A content delivery network (CDN) that delivers content to users with low latency and high availability.

TR8.4:
Amazon Elastic Transcoder: A video transcoding service that allows you to add error correction to your videos.
Amazon CloudFront: A content delivery network (CDN) that can handle network errors and deliver content to users with high availability.

TR9.1:
AWS Elastic Load Balancing (ELB): A load balancer that distributes incoming traffic across multiple Amazon EC2 instances.
Amazon CloudWatch: A monitoring and alerting service that helps you monitor and manage your AWS resources.

TR9.2:
Amazon CloudWatch Logs: A log management service that allows you to collect, store, and analyze logs from your Amazon EC2 instances and other AWS resources.
Amazon S3: A highly scalable object storage service that can store your log files.

TR9.3:
Amazon CloudWatch Logs Insights: A log analytics service that allows you to easily analyze your logs with predefined metrics and dashboards.
Amazon QuickSight: A business intelligence (BI) service that allows you to visualize and analyze your data.

TR10.1:
Amazon EC2: A compute service that allows you to deploy your applications across multiple AWS Regions.
Amazon CloudFront: A content delivery network (CDN) that can deliver content to users from multiple edge servers around the world.

TR10.2:
AWS Elastic Load Balancing (ELB): A load balancer that distributes incoming traffic across multiple Amazon EC2 instances.
Amazon CloudWatch: A monitoring and alerting service that helps you monitor and manage your AWS resources.

TR10.3:
Amazon CloudFront: A content delivery network (CDN) that can cache your content from edge servers around the world, reducing latency and improving performance for users.

TR11.1:
Amazon S3: A highly scalable object storage service that allows you to upload and organize your content.

Amazon CloudFront: A content delivery network (CDN) that can deliver your content to users with low latency and high availability.

TR11.2:
Amazon S3: A highly scalable object storage service that allows you to add metadata tags, version your content, and control access with permissions.

TR11.3:
Amazon S3: A highly scalable object storage service that can be integrated with other content management systems through APIs and SDKs.

TR12.1:
Amazon EC2 Auto Scaling: A service that automatically scales your Amazon EC2 instances up or down based on predefined metrics, such as CPU utilization.
Amazon S3: A highly scalable object storage service that can store your application data and protect it from

4. The first design draft

Shortlisting certain TRs provided in the earlier section:

Tenant Identification TRs:

- TR3.7: Identify tenants to identify data intensive workloads e.g. PHD students frequently access huge datasets for analysis and experiments.
- TR6.5: Identify tenants for allocating resource quotas.

Monitoring TRs:

- TR1.7: Implement logging and monitoring the health metrics of all resources to detect anomalies in behaviors and failures.

Elasticity TRs:

- TR1.9: Implement load balancing should be able to distribute traffic evenly across multiple instances and reduce load on a single server.
- TR4.2: Perform vulnerability assessment scans and recognise weakness in systems to prevent data leaks and malicious use.
- TR4.3: Implement CDNs to cache data from edge servers at different geographic locations.
- TR4.4: Implement stateless architecture to speed up requests.
- TR4.5: Use container orchestration to scale containers automatically.

Reliability TRs:

- TR1.1: System should be available for 99% of the time, 1% downtime allowed in cases of deployment failures, hardware failure, global outage.
- TR1.4: Mechanism to automatically detect and recover from system/node failures.

Conflicting TRs:

TR1.5 vs Optimal Costs

- TR1.5: CPU capacity and Memory should scale up when load goes over the set threshold by 30% and and should scale down when load decreases below the set threshold by 10%.

TRs from AWS WAF Pillars:

Security:

- TR2.3: Implement identity access management to enforce roles and permissions.
- TR2.4: Data in transit over the network should be encrypted for secure transmission.
- TR13.3: The backups should be stored in a secure and durable location to avoid malicious use.

Reliability:

- TR1.6: Resources should be deployed across multiple geographical locations to avoid regional outage.

Performance Efficiency:

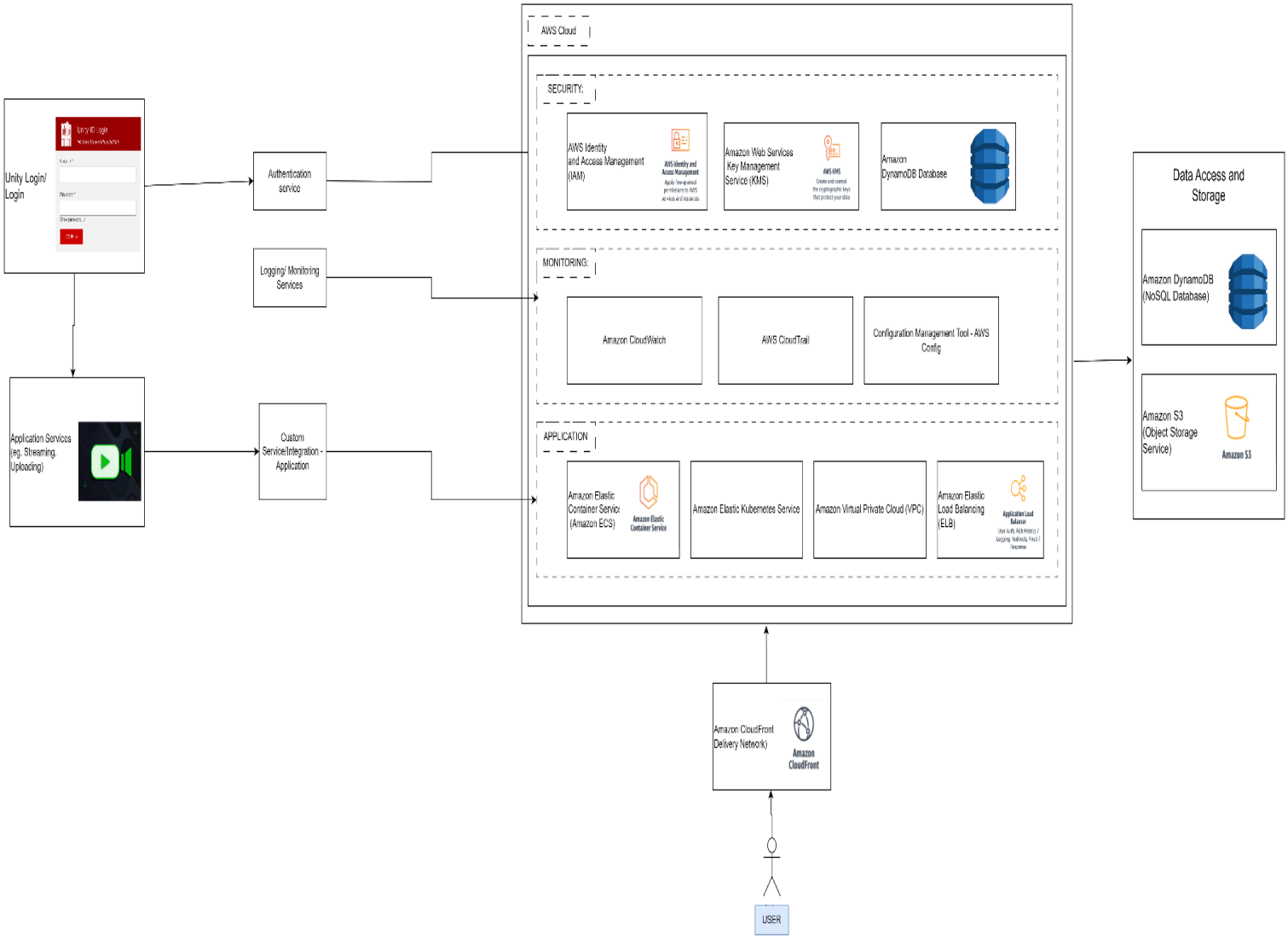
- TR3.4: Implement efficient data compression for optimized data transfer.

Below is a preliminary version of the architecture:

4.1 The basic building blocks of the design

1. Infrastructure
    a. Amazon Elastic Compute Cloud (EC2): EC2 provides scalable compute capacity in the cloud. It allows us to launch and manage virtual machines (VMs) on demand. It would help satisfying TR1.1, TR1.3, TR1.5,TR1.7, TR1.9, this helps achieve elasticity, fault tolerance, autoscaling, logging and monitoring, and load balancing.
    b. Amazon Elastic Container Service (ECS): ECS is a fully managed container orchestration service that makes it easy to deploy, manage, and scale containerized applications. It allows us to run and scale containerized applications on a cluster of Amazon EC2 instances. Using ECS would help satisfy the TR1.5, TR4.5 which helps achieve autoscaling and container orchestration.
2. Storage:
    a. Amazon S3: Object storage for images and other static content. S3 is highly scalable, durable, and cost-effective, making it ideal for storing large amounts of data.
    b. Amazon DynamoDB: NoSQL database for storing user details, image manipulation logs, and other metadata. DynamoDB is a fast, flexible, and scalable database that is well-suited for storing key-value pairs and other non-relational data. It would aid TR1.7, TR13.3.
3. Networking
    a. Amazon Elastic Load Balancing (ELB): ELB distributes incoming traffic across multiple EC2 instances or containers, ensuring high availability and low latency for our application. It can help achieve TR1.7, TR1.9, TR1.4, and help achieve logging and monitoring, load balancing, and fault tolerance.
    b. Amazon CloudFront: CloudFront is a content delivery network (CDN) that delivers content to users with low latency and high availability. It will help us cache static content, such as images and videos, closer to users, reducing load on our origin servers. Helps with TR4.3.
4. Security:
    a. Amazon Identity and Access Management (IAM): IAM provides secure access control for AWS resources. It allows us to create and manage users, groups, and roles, and define granular permissions for each resource. This would help achieve TR2.3.

b. Amazon Web Services Key Management Service (KMS): KMS is a fully managed service that makes it easy to manage cryptographic keys and use them to protect your data. We will use KMS to encrypt data at rest and in transit. This would help us achieve TR2.4
c. Amazon Virtual Private Cloud(VPC): AWS VPC (Virtual Private Cloud) is a service that enables you to create a private network in the cloud, allowing you to isolate and secure your application resources by defining network configurations, subnets, routing, and access control policies to prevent unauthorized access from the internet and maintain a secure environment for your applications and data. This would help us achieve TR2.4.



4.2 Top-level, informal validation of the design

1. Scalability: Our proposed design is highly scalable, meaning that it can be easily adapted to accommodate increasing demands without compromising performance or availability. This is achieved through the following mechanisms:
    a. Autoscaling: The system can automatically scale up or down based on real-time usage patterns using Amazon EC2 Auto Scaling. This ensures that we always have enough resources to handle peak demand, while also avoiding unnecessary costs during periods of low usage.

    b. Elasticity: The cloud-based infrastructure provides elastic resources that can be provisioned and de-provisioned on demand using Amazon EC2 instances and Amazon Elastic Container Service (ECS). This allows us to quickly add or remove resources as needed, without having to worry about upfront hardware purchases or long-term commitments.
    c. Horizontal Partitioning: The system can be horizontally partitioned by adding more instances or containers to distribute the workload across multiple compute nodes using Amazon EC2 instances and ECS. This helps to improve performance and responsiveness, especially for demanding tasks such as high-definition video streaming and real-time collaboration.
2. Availability: Our proposed design ensures high availability, meaning that the system is always accessible to users, even in the event of hardware failures or network disruptions. This is achieved through the following strategies:
    a. Redundancy: We use multiple availability zones (AZs) to distribute our resources across multiple physical locations using Amazon Elastic Compute Cloud (EC2) Multi-AZ Deployments. This ensures that if one AZ becomes unavailable, the system can continue to operate using resources in other AZs.
    b. Load Balancing: We use Amazon Elastic Load Balancing (ELB) to distribute incoming traffic across multiple instances or containers, preventing any single instance from becoming overloaded. This helps to maintain consistent performance and availability even under heavy load.
    c. Failover: We use automatic failover mechanisms to detect and recover from failures using Amazon EC2 Auto Scaling and Amazon ECS. If an instance or container fails, the system will automatically switch over to a backup instance or container to ensure that users are not affected.
3. Security: Our proposed design incorporates robust security measures to protect data and prevent unauthorized access. This is achieved through the following practices:
    a. Identity and Access Management (IAM): We use IAM to control who can access our resources and what actions they can perform. This ensures that only authorized users can access sensitive data or make changes to the system.
    b. Encryption: We use encryption at rest and in transit to protect data from unauthorized access. At rest, data is encrypted using industry-standard algorithms, such as AES-256. In transit, data is encrypted using TLS/SSL protocols to secure network connections.
    c. Vulnerability Management: We implement a regular vulnerability scanning and patching process to identify and remediate security weaknesses in our system using Amazon Inspector and AWS Security Hub. This helps to protect against known vulnerabilities and zero-day attacks.
    d. Access Logging and Monitoring: e continuously log and monitor access to our system to detect and investigate suspicious activity using Amazon CloudWatch Logs and Amazon CloudWatch Metrics. This helps to identify potential breaches and take corrective action promptly.
4. In addition to these specific measures, we also adhere to industry best practices for secure cloud computing, such as the AWS Well-Architected Framework. This helps us to maintain a high level of security throughout the development and operation of our system.

4.3 Action items and rough timeline

Skipped

5 Second design draft

5.1 Use of the Well-Architected Framework

The AWS Well-Architected Framework supports cloud system design and operation, assuring security, dependability, efficiency, cost-effectiveness, and long-term viability. It provides continual review and development as a collaborative endeavor rather than as an audit. It offers cloud best practices and tactics based on AWS experience. It includes core questions to align designs with cloud best practices and offers hands-on learning through laboratories and collaboration with the AWS Partner Network.
The primary terms of the Well-Architected Framework are as follows:

- **Component**: Represents the code, configuration, and AWS Resources that are linked together to suit specific needs; it is the architecture's autonomous unit of technical ownership.
- **Workload**: A group of components that work together to provide business value; it is the level at which business and technology leaders discuss architecture.
- **Architecture**: The study of how components within a workload interact and communicate; architecture diagrams are frequently used to illustrate this.
- **Milestones**: Significant architectural modifications that occur along the product life cycle (from design to production).
- **Technology Portfolio**: A collection of tasks required for the business's operations.

Trade-offs between pillars are made while creating workloads depending on factors such as business context and guiding engineering priorities. Depending on the solution's criticality or special needs (e.g., performance influencing income in e-commerce), decisions may prioritize sustainability and cost reduction over reliability, or vice versa. Through automated inspections, mechanisms, processes, and expert oversight ensure standards compliance. Amazon's culture encourages processes that are aligned with customer objectives, with a focus on the utilization of senior engineers to teach AWS best practices. The Well-Architected Framework embodies AWS's internal review process, codifying engineering principles and encouraging a distributed, customer-driven enterprise architecture. Performing Well-Through procedures, training, or shared knowledge sessions led by principle engineers, designed reviews assist technology leaders in identifying risks and addressing corporate problems.

The WAF wants us to adhere to the following general design principles:

1. Make no assumptions about the application's capabilities or requirements. It is critical to comprehend the magnitude of tasks.
2. Production scale testing should be enabled at all times, and such situations should be simulated.
3. Avoid manual workload replication, auditing impacts, and infrastructure experimentation. Implement automation whenever possible.
4. Avoid making architectural decisions in a one-time occurrence. The risk of the impact of design changes is reduced when the cloud is used properly.
5. Monitoring and logging to keep track of how your workload behaves in order to help refine the system by making fact-based judgments.
6. Game days should be mimicked on a regular basis to identify areas for development and to help generate organizational expertise in dealing with events.

5.2 Discussion of pillars

The most important component of the AWS WAF is their well defined pillars for achieving architectural excellence. We would like to discuss the following two pillars:

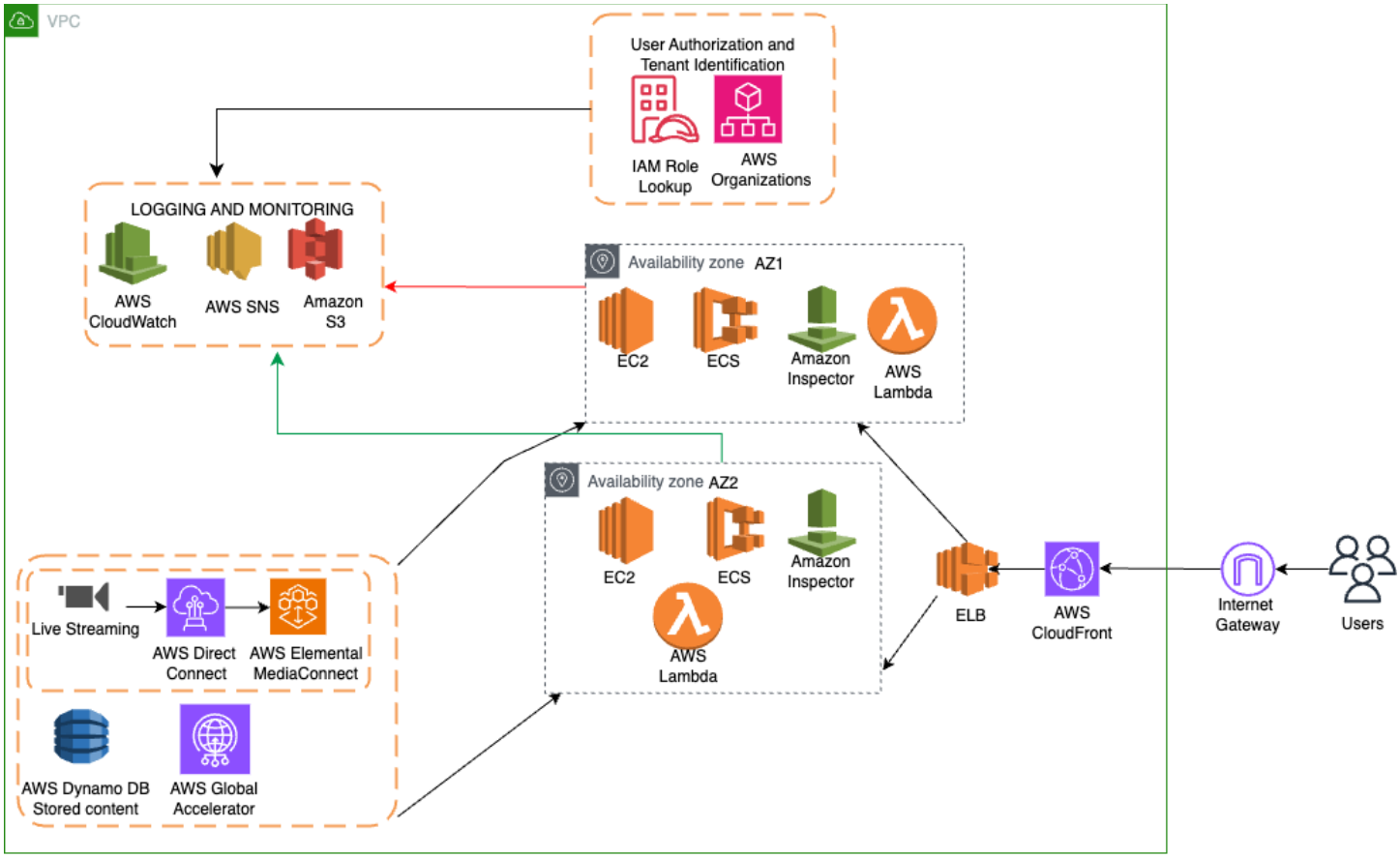
1. **Operational Excellence**
   This pillar supports development and running workloads effectively. It also entails supporting continuous improvements by gaining operational insights and delivering business value.
   The operational excellence principles for cloud operations consist of five core principles:
   Execute Operations as Code: Apply the same engineering approach used for applications to manage the entire environment by defining and updating workloads and operations as code, reducing errors, and ensuring consistent responses.
   Implement Small, Frequent, Reversible Changes: Design workloads for regular small updates that can be reversed if they fail, minimizing customer impact during failures.
   Regularly Enhance Operations Procedures: Continuously improve procedures alongside workload changes, conducting frequent reviews and exercises like "game days" to validate their effectiveness.
   Predict Failures: Identify potential failure sources through exercises like pre-mortems, test failure scenarios, validate response procedures, and simulate events to improve preparedness.
   Learn from Operational Failures: Drive improvement by learning from operational events, sharing insights across teams, and applying lessons learned across the organization.
   Aligned with four key best practice areas:
   Organization: Teams need a deep understanding of workload, shared goals, customer needs, governance requirements, and risks to make informed decisions and regularly update priorities.
   Preparation: Assess threats, manage risks, establish ownership for components, processes, and procedures, empower teams with support and leadership guidance.
   Operations: Encourage experimentation, skill development, and diversity within teams while leveraging AWS tools for learning, improvement, and architectural assessment.
   Evolution: Use AWS tools for architectural review, prioritize actions based on AWS Trusted Advisor insights, and continuously educate and support teams through AWS resources and training programs. Utilize tools like AWS Organizations and Managed Services for centralized governance and cloud operations expertise.

2. **Performance Efficiency**
   The capacity to use computational resources efficiently while meeting system requirements is included in the performance efficiency pillar. It emphasizes workload optimization for cost-effective, scalable, and efficient operations.
   The following are the primary design principles for cloud performance efficiency:
   Simplify Advanced Technology Integration: Simplify complex activities by employing cloud vendor services for specialist technologies such as NoSQL databases, machine learning, and video transcoding. This method lets your team to focus on product development rather than managing complex resources.
   Rapidly expand your global reach: Extending workloads across different AWS Regions improves customer experience by reducing latency while incurring minimum additional costs.

Accept Serverless Architectures: Use serverless architectures to avoid the headache of managing real servers. Serverless storage and event hosting, for example, expedite processes, save expenses, and relieve the strain of server maintenance.
Allow for Frequent Experimentation: Use virtual and automatable resources to run quick comparison experiments with different instance kinds, storage options, or configurations.
Adopt a Strategic Technology Selection Approach: Understand how cloud services work and choose technology approaches that best correspond with workload objectives. Consider data access patterns, for example, while selecting database or storage options to enhance overall performance.
Key best practices are:
Select Right Resources: One should choose appropriate AWS resources based on their respective workload requirements to ensure optimal performance and cost-effectiveness. These resources include correct selection of Compute, Storage, Database, and Network.
Monitoring: It is impertinent to implement robust monitoring systems to track performance metrics, detect bottlenecks at system points, and to optimize resource utilization continuously. Use tools to collect data, analyze trends emerging in it, and ensure workload efficiency. It is essential to enable quick responses to potential issues.
Trade Offs: Acknowledge and manage tradeoffs between various performance parameters. These parameters could be cost, speed, reliability, and scalability. Optimization of workloads by making informed decisions that balance these factors according to business priorities.
Review: Regularly assess and review workload performance against predefined performance objectives. Conduct comprehensive evaluations to identify areas for improvement, adjust strategies, and incorporate evolving best practices to enhance overall efficiency.

By adopting these principles, organizations can take the right steps and can build and manage workloads efficiently. It helps with ensuring optimal performance, scalability, and cost-effectiveness, which are the basic needs of all products out there.

5.3 Use of Cloudformation diagrams



5.4 Validation of the design

We would now attempt to validate our design selections using the TRs chosen in Section 4:

Tenant Identification TRs:

1. **Tenant Identification (TR3.7):**

- IAM enables fine-grained control over resource access based on tags or attributes associated with individual tenants.
- Resource tagging and IAM regulations allow for customized access to data-intensive tasks such as PHD student datasets.

2. **Tenant Resource Quotas (TR6.5):**

- Amazon Organizations assists in grouping accounts by tenants or OUs, allowing resource quotas to be defined per tenant.
- Organizational Service Control standards (SCPs) assist enforce restrictions and standards across tenant accounts, guaranteeing resource consumption compliance.

In essence, IAMs access controls, when combined with Organizations' account structuring and policy management, facilitate tenant identification, personalized resource access, and allocation of specific quotas, thereby effectively managing and securing tenant-related activities within AWS environments.

3. **Monitoring TRs:**

- TR1.7 is fully met by AWS CloudWatch, in conjunction with Amazon SNS (Simple Notification Service) and Amazon S3, which provide centralized log management and monitoring capabilities.
- CloudWatch collects logs from multiple AWS services and allows for custom metric tracking, which aids in anomaly identification via customisable alarms.
- When abnormalities or failures are observed, CloudWatch sends warnings via SNS to predefined recipients or systems, allowing for rapid response and issue resolution.
- Furthermore, CloudWatch allows archiving logs to Amazon S3, allowing for long-term storage and analysis of historical data for deeper insights and compliance purposes, boosting monitoring capabilities and assisting in the detection of behavioral anomalies or failures across multiple AWS services.

4. **Elasticity TRs:**

- Elastic Load Balancing (ELB) contributes to TR1.9 compliance by effectively distributing incoming application traffic across numerous instances or resources in an AWS environment. It ensures even distribution, preventing any single server from becoming overburdened with traffic, lowering the danger of performance deterioration or server overload. By controlling traffic distribution across numerous instances, ELB efficiently balances the load among available resources, improving system dependability, scalability, and overall application performance.
- Amazon Inspector, CloudWatch, and SNS collaborate successfully to achieve TR4.2 system security within AWS resources. Amazon Inspector performs automated security evaluations to find vulnerabilities, whereas CloudWatch continuously monitors system health to detect anomalies or potential security issues. When vulnerabilities are discovered by Inspector or anomalies are detected by CloudWatch, SNS immediately sends notifications, allowing for quick action to remedy any data leaks or malicious activity and enhancing the overall security of AWS services.
- Amazon CloudFront supports TR4.3 by caching and distributing data across many geographic regions via a global network of edge servers. By providing material from adjacent edge sites, this caching strategy provides faster content delivery to end users, lowers latency, and enhances performance.
- AWS Lambda contributes to TR4.4 compliance by natively enabling a stateless design, resulting in speedier request processing. Lambda functions run independently and statelessly, processing individual events or requests with no persistent state between invocations. Because each invocation acts in isolation, each invocation handles requests effectively without the need to keep session data or state information. As a result, Lambda's stateless nature naturally contributes to faster request processing, coinciding with the goal of designing a stateless architecture for better performance.
- Amazon ECS (Elastic Container Service) contributes to TR4.5 compliance by offering comprehensive container orchestration tools that automate container scaling. ECS makes containerized application management easier by providing automated scaling options. Users can establish scaling policies using ECS based on metrics such as CPU use or request rates. ECS dynamically adjusts the number of containers as demand changes, guaranteeing optimal resource usage without the need for manual intervention. This container orchestration effectively scales containers in response to changing workloads, satisfying TR4.5's criterion for automated container scaling.

5. **Reliability TRs:**

- For TR1.1, we rely on the SLAs promised to us by AWS services. The services used in our designs have an SLA of 99.99%. As per the AWS WAF, the 99.99% SLA would ensure that for a year, a maximum of 52 minutes of unavailability would be observed.

- AWS Inspector can be considered a justifiable choice for TR1.4 because it offers automated security assessments and vulnerability detection within the AWS environment. While not explicitly focused on detecting system or node failures, AWS Inspector aids in automatically identifying security vulnerabilities, potential weaknesses, and misconfigurations within EC2 instances, thereby helping to prevent security-related failures. Although its primary function is security assessment, the automated nature of Inspector's scans indirectly contributes to system stability by proactively identifying and rectifying security flaws, reducing the risk of system compromises or failures due to vulnerabilities. However, for directly handling system or node failures, AWS Inspector's focus lies more on preemptive security measures rather than immediate system recovery mechanisms.

6. **Conflicting TRs:**

When weighing the decision between using a hybrid of EC2 and Lambda, the tradeoff primarily revolves around cost and functionality:

- Cost:
  - EC2 provides flexibility but may incur higher operational expenses due to manual management and potential underutilization.
  - Lambda's pay-per-use model can be cost-effective for event-driven workloads but might become expensive for prolonged or continuously active tasks.
- Functionality:
  - EC2 offers more control and customization, suitable for specific configurations or legacy software.
  - Lambda excels in rapid scaling and responsiveness, making it ideal for event-driven or bursty workloads but may lack flexibility for certain applications.
  - The tradeoff involves balancing cost-efficiency with operational control and adaptability to workload requirements, considering factors like workload characteristics, anticipated usage, and performance needs.

TRs from AWS WAF Pillars:

Security:

- IAM accomplishes this by providing a centralized platform within AWS to manage users, groups, roles, and their associated permissions. It enforces the principle of least privilege, allowing administrators to define and allocate granular permissions to users or systems. Through policies, IAM controls access to AWS resources by defining who (identities) can do what (actions) on which resources. By leveraging IAM, organizations can effectively manage and enforce access control policies, ensuring that users and services only possess the necessary permissions required to perform their tasks or access specific resources, thus adhering to defined roles and permissions.

- To align with TR2.4, secure data transmission is ensured over the network by using ELB and CloudFront. ELB encrypts data by supporting SSL/TLS termination, securing communication between

clients and the load balancer. CloudFront enables SSL/TLS encryption, ensuring secure data transmission between CloudFront distributions and end-users' devices or browsers.

- Amazon S3 (Simple Storage Service) addresses TR13.3 by serving as a secure and durable repository for backups. Through robust security measures like encryption at rest and in transit, along with access controls, S3 ensures that backups are securely stored and accessible only by authorized users, reducing the risk of unauthorized access or tampering. Its high durability, backed by redundant storage across multiple locations, offers 99.999999999% durability, safeguarding backups from data loss or corruption. These features collectively establish S3 as a resilient and protected environment for storing backups, meeting the requirement to securely store data and prevent malicious use as per TR13.3.

Reliability:

- Deploying resources across multiple Availability Zones (AZs) across diverse geographical areas mitigates the risk of regional outages, aligning with TR1.6. This strategy ensures high availability by providing redundancy and fault tolerance; if one zone or geographic region experiences issues, resources in other zones remain operational, preventing disruptions. Geographic diversity helps minimize the impact of localized incidents or regional failures on services, reducing dependency on a single location and enhancing overall resilience against widespread downtime.

Performance Efficiency:

- AWS Global Accelerator, AWS Direct Connect, and S3 Transfer Acceleration contribute to achieving TR3.4 by optimizing data transfer through various mechanisms. Global Accelerator uses the AWS global network infrastructure to improve the performance of internet traffic to and from various AWS regions, ensuring efficient and low-latency data transfer. AWS Direct Connect establishes a dedicated network connection between on-premises infrastructure and AWS, allowing for faster and more consistent data transfer rates, thereby optimizing data movement efficiency. S3 Transfer Acceleration employs edge locations to accelerate uploads to Amazon S3 by optimizing data transfer through faster network paths, suiting scenarios where data compression could enhance the overall efficiency of transfer mechanisms. Each service enhances data transfer speed and efficiency, aligning with TR3.4 by offering optimized data transmission capabilities, reducing transfer times, and improving overall performance.

5.5 Design principles and best practices used

As per the AWS Well Architected Framework, we have followed the below mentioned best practices:

1. Automatic recovery from failures: as a part of continuous improvement and monitoring for faults and vulnerabilities. We also took into account the periodic disaster testing for ensuring availability strategies are correctly implemented.
2. Do not guesstimate the capacities: we have implemented load balancing, monitoring, and tenant identification for setting appropriate quotas of resources. We also ensure automating.
3. We also keep emphasis on making frequent, small, reversible changes: This can be easily achieved by implementing CI/CD.
4. Identity and Access Management: is also achieved by enforcing roles and authorized access to the application resources.

5. Data protection: we have emphasized the protection of data in our design by making use of secured databases such as DynamoDB and S3.
6. Monitoring and Logging: for fault detection, vulnerability detection, self-recovery, usage patterns, security, and more are all well thought of as a part of this design.

5.6 Tradeoffs revisited

With help of the initial TRs mentioned in section 2.4 and refined decisions we took in the subsequent sections in the design of our system, we have summarized the following arguments to resolve our trade-offs.

1. **TR1.1 (System availability) vs. TR1.2 (CI/CD):** The design places emphasis on system availability (TR1.1) by relying on AWS services with a 99.99% SLA. CI/CD (TR1.2) is still essential for continuous improvements, but the risk of downtime is mitigated by a robust deployment process and thorough testing before production releases. The tradeoff involves ensuring a balance between frequent deployments and maintaining high system availability. We believe that we can rely on Amazon SLAs and hence it is required to focus on CI/CD to enable continuous improvements and evolution of our system.
2. **TR1.3 (Data replication) vs. TR3.4 (Data compression):** Data compression (TR3.4) is leveraged to reduce the amount of data needing replication. The tradeoff is carefully choosing a compression algorithm that balances performance needs with the goal of minimizing the data to be replicated. Hammond's advice on making trade-offs by assessing benefits and risks aligns with the consideration of processing overhead against data replication benefits.
3. **TR1.4 (Automatic failure recovery) vs. TR3.6 (Stateless architecture):** The design incorporates a stateless architecture (TR3.6) for easier scalability and failure recovery. While automatic failure recovery (TR1.4) is crucial, the focus is on preemptive security measures through tools like AWS Inspector. The trade off involves choosing a balance between immediate system recovery mechanisms and proactive security measures. Since our system needs stateless architecture for the efficiency of performance, we have chosen statelessness. However, we have incorporated robust reactive ways to deal with failure and mitigating risks. Monitoring and Logging can also help to pre-empt many such situations and has been effectively considered in our design
4. **TR1.5 (CPU and memory scaling) vs. TR6.6 (Pricing models):** The design acknowledges the tradeoff between CPU and memory scaling (TR1.5) and pricing models (TR6.6). It's important to align scaling decisions with the chosen pricing model to manage costs effectively. Hammond's approach of considering alternatives and their trade-offs aligns with the need to choose a pricing model aligned with the application's requirements. A flexible pricing model along with our hybrid use of services helps us achieve cost efficiency. Ranging from configurable instances to serverless instances, we have leveraged the division of responsibilities and reaching an optimal cost while still achieving scaling to satisfy our requirements.
5. **TR1.6 (Multi-regional deployment) vs. TR6.7 (Data cleanup strategies):** Multi-regional deployment (TR1.6) is considered for reliability, while data cleanup strategies (TR6.7) are implemented to reduce storage requirements. The tradeoff involves carefully considering the risks and benefits of data cleanup strategies to avoid potential data loss, aligning with Hammond's recommendation to weigh the pros and cons of alternatives. Our design emphasizes heavily on reliability of the application to build customer

trust. As per the WAF best practices, it would be a good practice to implement multi-regional deployment to avoid single points of failure and increasing performance efficiency by being available at multiple locations. Hence, we choose to go forward with it and implement a rigorous clean up strategy to overcome any overhead that may be caused by our choice.

6. **TR1.9 (Load balancing) vs. TR11.3 (Integration with other content management systems)**: Load balancing (TR1.9) is implemented to distribute traffic evenly, and the tradeoff involves choosing a solution that supports integration with other content management systems (TR11.3). Hammond's concept of "even swaps" aligns with the need to balance the benefits of load balancing with the integration complexities it might introduce. The emphasis is on selecting a load balancing solution that aligns with integration needs.
7. **TR1.5 (Controlled Scaling) and TR3.6 (Stateless Architecture):** As we know both these functionalities add a unique value to our system. The choice between Lambda and EC2 depends on multiple factors. These factors are characterizing workloads, scalability needs, control, and considerations of costs. Hence we decided to go forward with a hybrid approach leveraging both services. This is the optimal strategy, using Lambda for event-driven tasks and EC2 for more traditional or predictable workloads.
8. **TR4.2 (Vulnerability assessment) vs. TR4.3 (CDNs):** While CDNs enhance performance, vulnerability assessments are crucial for security. The tradeoff involves finding a balance between security measures and performance optimization. Hammond's recommendation to find alternatives with comparable advantages aligns with this tradeoff. Performance efficiency is impertinent to our application and the time taken to deliver content is important. Hence, we choose to use CDNs.

5.7 Discussion of an alternate design

Skipped

6 Kubernetes experimentation
6.1 Experiment Design

Experiment Overview:

Our experiment strives to test the correctness of our flexible scalability TR, i.e. TR1.5, of the application. Scalability is an essential component of our design because it helps with accommodating time varying workloads, and eliminating idle resources. This results in achieving high availability, gaining user trust, and cost optimisations.

We make use of Kubeternetes' autoscaling feature, facilitated by the Horizontal Pod Autoscaler (HPA) to help drive this experiment. Kubernetes serves as an open-source platform for managing containerized applications, automating their deployment, scaling, and administration processes. Additionally it's ability to dynamically regulate the number of running pods based on observed resource usage metrics, such as CPU utilization, helps in achieving flexible scalability.

We also make use of Locust for load generation to our sample application. It is an open-source, Python-based load testing tool. It's widely used to evaluate the performance and scalability of web applications. It helps us simulate concurrent user loads and analyze how our system behaves under various levels of stress. It also provides a user-friendly interface to define tasks, set up test cases, and generate detailed reports displaying performance metrics such as response times, error rates, and throughput.

Experiment Setup:

Overview:
For the execution of this experiment we created a simple application with an exposed endpoint to fetch the result of a student's query to retrieve a text based post by a professor. The application is dockerized and deployed on a local Kubernetes cluster using Minikube. We set up the HPA with necessary target CPU utilization. Once the target CPU utilization across the pods is reached, the number of replicas increases up to the specified maximum. When the load is less and no longer needs the CPU resources of the multiple replicas it drops down to the specified minimum value. A local locust server is used to generate load for the application to test the application's ability to handle the incoming traffic and increase the number of replicas as needed.

Specifications:

- For this experiment, we created a simple Python FastAPI application with a single endpoint which is responsible for returning text based information to simulate a student's query to retrieve some post made by the professor.
- After creating the docker image of the application, we make use of Minikube to build a Deployment and Service resource in our local k8s cluster.
- We then configure the Horizontal Pod Scaler and set the target CPU usage as 70%.
- The HPA is further configured to scale up to a maximum of 10 replicas and scale down to a minimum of 1 replica based on the target consumption.
- The pod running on the local cluster can be accessed using the associated URL

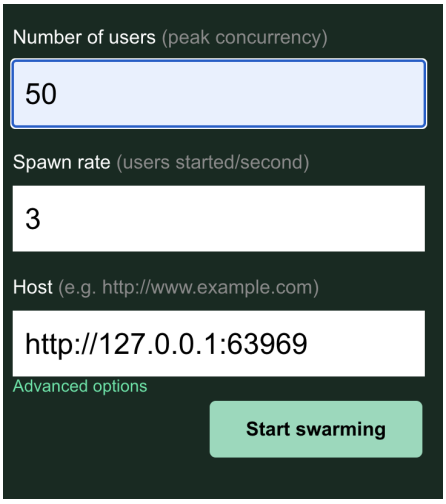
```
priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl apply -f config.yml
deployment.apps/uni-made-easy created
service/uni-made-easy created
priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl autoscale deployment uni-made-easy --cpu-percent=70 --min=1 --max=10
horizontalpodautoscaler.autoscaling/uni-made-easy autoscaled
priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl get service
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        20d
uni-made-easy   NodePort    10.98.111.176   <none>        80:30401/TCP   2m23s
priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
uni-made-easy   1/1     1            1           2m29s
priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl get hpa
NAME            REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache      Deployment/php-apache      0%/80%    1         10        1          19d
uni-made-easy   Deployment/uni-made-easy   0%/70%    1         10        1          2m24s
priyaandurkar@Priyas-Air-4 Cloud Workspace %
```

```
priyaandurkar@Priyas-Air-4 ~ % minikube service uni-made-easy --url
http://127.0.0.1:63969
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

6.2 Workload generation

For this experiment, as mentioned above, we have made use of the Locust library to generate load on our application.[13] To do so, we need to create a locust test file where we define our test scenarios. Within this file we define tasks which are used to emulate the simulated user's behavior. Since our use case is simple, we only need to navigate to the root address of our running application to hit the dummy API endpoint.

Once our simple test file is ready and run, we navigate to the interface and start swarming our application. To do this we enter the following details:



These inputs specify the number of users, spawn rate and URL of our locally deployed application. When we start the load generation, following are some of the observations:

Total number of requests: 82
Requests/second - 0.5 requests/second
Failure/sec - 0.0 /second

6.3 Analysis of the results

The main aim of the experiment is to validate whether Kubernetes HPA is able to scale our application based on increasing load. With above given inputs to generate the load we observed the following statistics in the Locust dashboard. Additionally, the effect of the HPA increasing and decreasing the scaled replicas is also observed in the subsequent screenshot

Locust Test Report

During: 11/24/2023, 8:56:49 PM - 11/24/2023, 8:59:32 PM

Target Host: http://127.0.0.1:63969

Script: locustTest.py

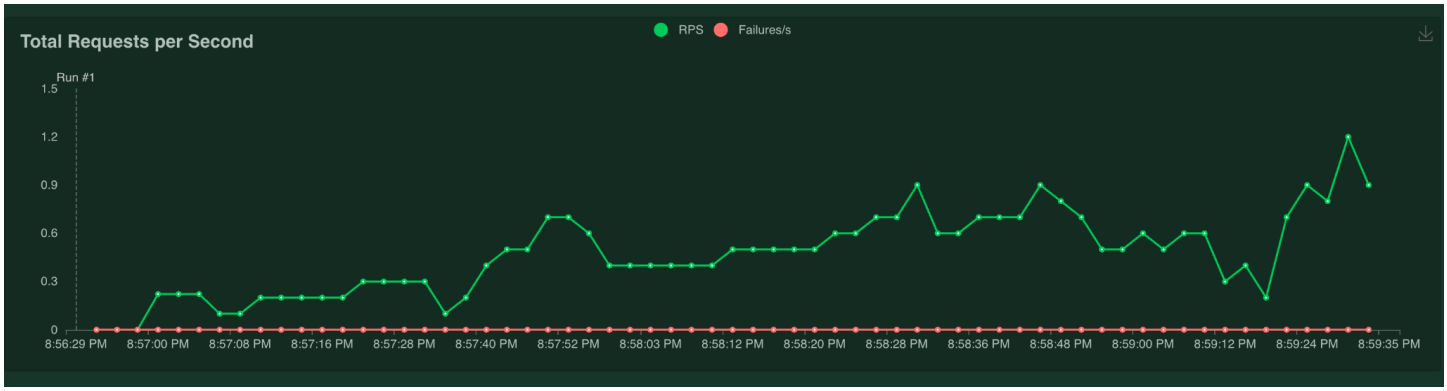
Download the Report

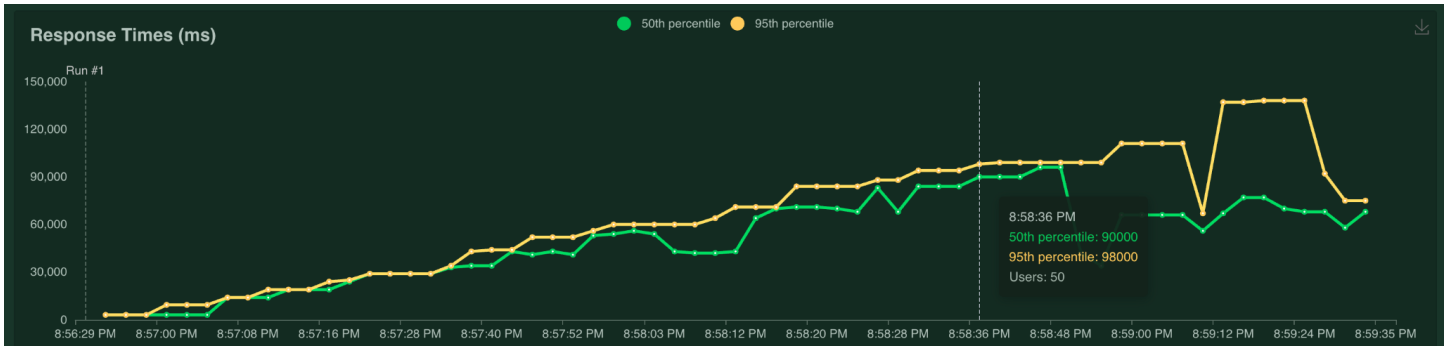
Request Statistics

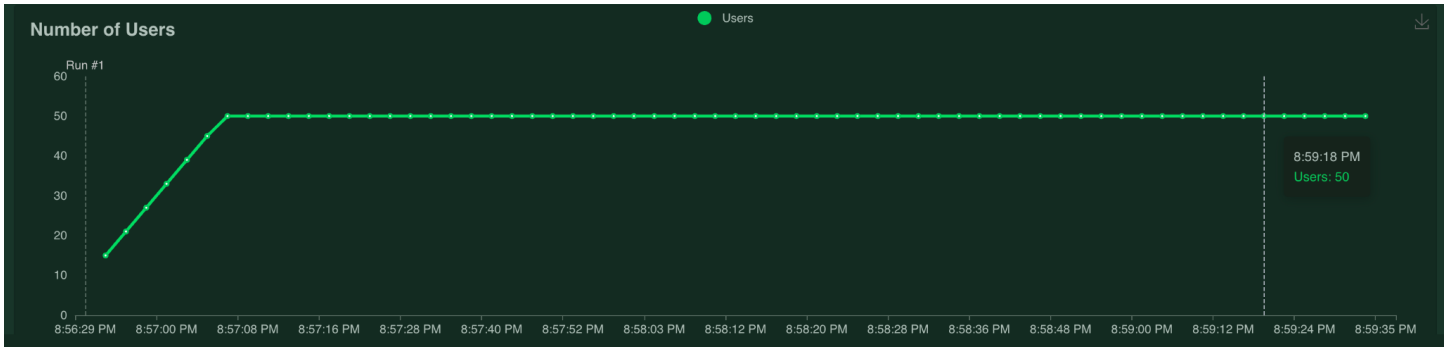
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	82	0	55051	2884	138119	3	0.5	0.0
	Aggregated	**82**	**0**	**55051**	**2884**	**138119**	**3**	**0.5**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	50000	56000	68000	76000	92000	99000	138000	138000
	Aggregated	**50000**	**56000**	**68000**	**76000**	**92000**	**99000**	**138000**	**138000**








```
[priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
uni-made-easy-6c89d9c9f7-h9xd5   1/1     Running   0          4s
[priyaandurkar@Priyas-Air-4 Cloud Workspace % kubectl get hpa uni-made-easy  --watch
NAME            REFERENCE                     TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
uni-made-easy   Deployment/uni-made-easy   <unknown>/70%   1         10        1          7m41s
uni-made-easy   Deployment/uni-made-easy   <unknown>/70%   1         10        1          8m1s
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        1          9m1s
uni-made-easy   Deployment/uni-made-easy   165%/70%        1         10        1          13m
uni-made-easy   Deployment/uni-made-easy   165%/70%        1         10        3          13m
uni-made-easy   Deployment/uni-made-easy   250%/70%        1         10        3          14m
uni-made-easy   Deployment/uni-made-easy   250%/70%        1         10        4          14m
uni-made-easy   Deployment/uni-made-easy   250%/70%        1         10        4          15m
uni-made-easy   Deployment/uni-made-easy   250%/70%        1         10        8          15m
uni-made-easy   Deployment/uni-made-easy   250%/70%        1         10        10         15m
uni-made-easy   Deployment/uni-made-easy   187%/70%        1         10        10         16m
uni-made-easy   Deployment/uni-made-easy   70%/70%         1         10        10         17m
uni-made-easy   Deployment/uni-made-easy   18%/70%         1         10        10         18m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        10         19m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        10         20m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        10         22m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        3          23m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        3          23m
uni-made-easy   Deployment/uni-made-easy   0%/70%          1         10        1          24m
priyaandurkar@Priyas-Air-4 Cloud Workspace %
```

From the above screenshot we can observe the behavior of the autoscaler. Initially when there is no load directed towards our application the number of replicas is 1. But as soon as Locust begins swarming, the rise in the percentage of CPU usage crosses 70% and results in an increase of the number of replicas. Once we reached the maximum number of replicas, the load generation from locust was stopped. As the load drops, the number of replicas also drops to the minimum number over the subsequent time period

Additionally The 0 failure rate in the requests statistics helps us understand that the scaled requests are well handled by our application otherwise we would have run into failures due to the load and our application not being able to handle it.

Hence we can conclude that our experiment verifies our TR1.5 successfully. Our application would flexibly scale up and down and react appropriately to time varying workloads.

7. Ansible playbooks
[skipped]

8. Demonstration
[skipped]

9. Comparisons
[skipped]

10. Conclusion

10.1 The lessons learned

In conclusion, our project journey highlighted crucial elements in building cloud solutions and the challenges faced by the Cloud Solutions Architect. Following are some lessons we learnt:

- The lessons learned emphasized understanding each component's role, outlining business and technical needs, considering tradeoffs, and utilizing the AWS Well-Architected Framework.
- Comparing cloud providers aided in making informed service selections. It also helped us dive deeper into all the options that are readily available to us today.
- The project also stressed the importance of comprehending Technical Requirements (TRs) and utilizing available services effectively. It is impertinent to not try to reinvent the wheel, rather use the best practices associated with the plethora of offerings available to us.
- Identifying conflicting requirements proved both rewarding and time-consuming. The AWS Well Application Firewall (WAF) was an extremely valuable resource. It provided us with ready-made technical requirements and essential documentation.
- Kubernetes experimentation offered practical insights, while researching solution options revealed diverse perspectives vital in cloud solution design.
- Altogether, this project has shown us the depth of research and considerations pivotal in creating robust cloud solutions.

10.2 Possible continuation of the project

There are quite a few things we can focus on to continue this project:

1. **CI/CD Pipeline Optimization:** By working together, we may investigate advanced strategies for optimizing our Continuous Integration and Continuous Deployment (CI/CD) pipelines. Let us look at incorporating new technologies or approaches to streamline our development and deployment processes, making them more efficient and lowering deployment risks.
2. **Strategies for Dynamic Resource Scaling**: We can look at more advanced strategies for dynamically scaling CPU and memory. Working together, we can investigate the use of machine learning techniques or predictive analytics to anticipate workload changes and modify resources proactively, improving both performance and cost.
3. **Advanced Security methods:** Research and execute advanced security methods to improve our security posture. We may collaborate to investigate new tools and approaches for detecting and mitigating emerging security threats in our AWS environment.
4. **Serverless Architecture Optimization**: By working together, we can improve our serverless architecture (AWS Lambda). Let's look at some more features, best practices, and future upgrades, such as using upcoming serverless technologies or frameworks to increase overall performance and scalability.
5. **AI-Driven Anomaly Detection:** We may incorporate artificial intelligence (AI) and machine learning models into our monitoring systems for more advanced anomaly detection. We can create models that not only detect anomalies but also predict possible problems, allowing us to manage our system more effectively and reduce downtime.

6. **Strategies for Cost Reduction**: Working together, we can investigate more cost-cutting options, taking into account improvements in AWS pricing models and services. We can look into approaches to improve resource allocation, reduce underutilization, and increase overall cost-effectiveness while maintaining performance and dependability.
7. **Enhanced Disaster Recovery Testing**: By working together, we can create and deploy more complex disaster recovery testing procedures. We will simulate difficult scenarios, test failover mechanisms, and guarantee the efficacy of our high availability techniques under a variety of demanding settings.
8. **Advances in Container Orchestration**: We can keep up to date on improvements in container orchestration tools and processes. Working together, we can investigate new features or services that could improve the automation and scalability of our containerized applications, while also exploring emerging technologies that are not now supported by our solutions.
9. **User Experience and Content Recommendations**: To improve our users' experiences, we can perform user surveys and provide new features. We can look into personalized content recommendations based on user behavior and preferences, with the possibility of using machine learning algorithms for more accurate recommendations.
10. **Edge Computing Integration**: By working together, we may look at integrating edge computing technologies to bring computation closer to our end customers. We can also look at how deploying resources at the edge can increase the speed of our application, minimize latency, and improve the overall experience for our geographically dispersed user base.

11. References

In this project, the ChatGPT language model developed by OpenAI, and Bard language model developed by Google was utilized as a resource for generating content related to material like AWS WAF, Kubernetes, and AWS Services.

[1] https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/headend-in-the-cloud-ra.pdf?did=wp_card&trk=wp_card
[2] https://aws.amazon.com/solutions/implementations/live-streaming-on-aws/?did=sl_card&trk=sl_card
[3] https://aws.amazon.com/solutions/implementations/video-on-demand-on-aws/?did=sl_card&trk=sl_card
[4] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/manage-aws-iam-identity-center-permission-sets-as-code-by-using-aws-codepipeline.html?did=pg_card&trk=pg_card
[5] https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc
[6] https://www.datamation.com/cloud/cloud-costs/
[7] https://cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/
[8] https://aws.amazon.com/vpc/features/
[9] https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=ite
[10]m.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitep
[11] apers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc
[12] https://cloud.google.com/learn/what-is-kubernetes
[13] https://medium.com/nerd-for-tech/load-testing-using-locust-io-f3e6e247c74e
[14] 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&aws-sla-cards.q=EC2&aws-sla-cards.q_operator=AND
[15] https://aws.amazon.com/inspector/
[16] https://aws.amazon.com/s3/transfer-acceleration/
[17] https://aws.amazon.com/global-accelerator/
[18] https://aws.amazon.com/organizations/
[19] https://locust.io/