# CSC 547 CLOUD ARCHITECTURE

# PROJECT REPORT

**NAME:**

**APURV CHOUDHARI (apchoudh)**
**PRABHUDATTA MISHRA(Pmishra4)**

# 1 Introduction [0%]

## 1.1 Motivation

There is a need for a subject-based social media platform where users can discuss with flexible privacy settings. Most social media platforms focus on the centralized profile page of their users. Since this centralized profile creates a link between real and virtual identities, it restricts users from having an open discussion.

Hence we want to design a social media platform where users can have open discussions on diverse topics without revealing personal information.

## 1.2 Executive summary

The goal is to design a community or group-based social media platform. Registered users submit content to the platform such as links, text posts, images, and videos, which are then voted up or down by other members of the community or group. Submissions with more upvotes appear toward the top of their community.

The platform allows users to create accounts without revealing their real identities. While users can choose to use their real names, the default will be pseudo-names or complete anonymity. This encourages open discussions without the need to present a polished online persona.

## 2 Problem Description [0%]

### 2.1 The problem

In the online community landscape, a need persists for a platform that intuitively supports content creation, sharing, and diverse discussions. Existing platforms lack a specialized approach to fulfill this requirement. Our project aims to bridge this gap by developing an application that empowers users to create, share, and discuss various content, fostering vibrant online communities. Through innovative features, we strive to address existing limitations and meet the evolving needs of modern internet users, creating a unique space for diverse communities to thrive.

## 2.2 Business Requirements

**BR 1: Scale Rapidly**

**BR 2: Guarantee High Availability**

**BR 3: Safeguard Data Backup and Recovery**

**BR 4: Fortify Security Measures**

**BR 5: Enhance Search and Discovery**

**BR 6: Analyze User Behavior**

**BR 7: Uphold Legal Compliance**

**BR 8: Optimize Data Delivery and User Experience**

**BR 9: Secure User Registration and Authentication**

**BR 10: Cost Optimization**

**BR 11: Ensure SDLC Automation**

## 2.3 Technical Requirements

TR 1.1: Prepare for rapid growth by building a scalable architecture accommodating increasing user loads. Based on our application user loads can be:

a)Surge in concurrent users during live discussions
b)Interactions like voting, commenting, and sharing can increase the number of server requests

TR 1.2: Monitor scalability metrics and performance to adapt to growing demands for optimal performance and cost efficiency

TR 2.1: Ensure high availability by replicating data on multiple geographical regions.

TR 2.2: Ensure the system's fault tolerance by having failover mechanisms that enable seamless continuity of service in the event of a failure.

TR 2.3: Continuously monitor system health and response times.

TR 3.1: Establish regular automated backups of generated Data.

TR 3.2: Replicate data across all geographical regions.

TR 3.3: Develop a disaster recovery plan for our data and test it periodically.

TR 3.4: Incorporate data backup insights for better data restoration and compliance.

TR 3.5: Ensure end-to-end data integrity during backup and restore.

TR 4.1: Strengthen data protection for both data in transit and at rest.

TR 4.2: Add protection methods to safeguard against external threats(like DDoS).

TR 4.3: Incorporate real-time threat detection and incident response.

TR 4.4: Have a role-based access control system. Based on our application users can have two roles:

a) Community admin
b) Community member

TR 5.1: Develop robust search methods to improve content discovery based on user interaction.

TR 5.2: Have methods for real-time search results.

TR 5.3: Utilize AI for content relevance ranking based on user's behavior.

TR 5.4: Monitor search and discovery features for effectiveness and user engagement.

TR 6.1: Implement user behavior analytics for insights into content management and community engagement.

TR 6.2: Collect and analyze user data to identify trends and patterns.

TR 6.3: Utilize AI to predict user behavior. In our application, this knowledge will help us to:

a)Build a better recommendation System.
b) Detect Suspicious activities by the users.
c) Violation of community guidelines.

TR 7.1: Monitor content for compliance with data privacy, copyright, and user rights regulations.

TR 7.2: Regularly audit and report on compliance with relevant regulations.

**TR 7.3**: Implement security measures promptly to align with security best practices.

**TR 8.1**: Implement Load Balancing for Efficient Data Delivery.

**TR 8.2**: Data Caching and Acceleration for Improved User Experience.

**TR 9.1**: Implement Multi-factor authentication (MFA) to enhance the security of the application.

**TR 9.2**: Implement tenant identification for billing purposes.

**TR 10.1**: Have a comprehensive system for monitoring and reporting on usage-based costs across various services to identify opportunities for optimization and enforce cost controls.

**TR 10.2**: Cost-Aware Database Scaling and Tiering. Optimize database costs by implementing dynamic scaling and tiering strategies based on usage patterns, ensuring efficient use of resources.

**TR 11.1**: Have a robust CI/CD pipeline to automate the building, testing, and deployment of application code, ensuring rapid and reliable software delivery.

**TR 11.2**: Automate application version rollouts

## 2.4 Tradeoffs

**User Engagement(TR 5.1) vs. Content Moderation(TR 7.1):**
Our platform aims to maximize user engagement by allowing a wide range of information and discussion. However, open space development can present physical restraint challenges. It's a trade-off between controlling inappropriate or harmful content and encouraging user engagement and implementing effective moderation policies. Too strict a restriction can stifle discussion, while a loose restriction can lead to misinformation, bullying, or comments that spread harm.

**Encryption Overhead (TR 4.1) vs. Performance(TR 1.1):**
Implementing strong encryption measures to enhance data security is crucial. However, encryption introduces computational overhead that can impact system performance. The tradeoff here is between the level of encryption applied and the performance of the system. Stronger encryption algorithms contribute to higher security but may result in increased processing demands, affecting response times and overall system performance. Balancing the need for robust security with the potential impact on performance is a key consideration in cloud-based design.

**Data Consistency (TR 3.2) vs. Availability(TR 2.1):**
Data Consistency: Ensuring strict data consistency across all nodes may lead to increased latency and reduced availability, especially in distributed systems. This tradeoff is often referred to as the CAP theorem.
Availability: Prioritizing availability may result in eventual consistency, where all nodes may not have the latest data simultaneously. This can enhance system responsiveness but might lead to temporary inconsistencies.

**Scalability(TR 1.1) vs. Cost(TR 10.1):**
Scalability: Designing for scalability allows your application to handle a growing number of users and increased data loads. This might involve using scalable cloud services, microservices architecture, and distributed databases. However, scaling often comes with increased costs.
Cost: Prioritizing cost efficiency may involve optimizing code, using serverless computing, and carefully choosing cloud service plans. It's essential to find the

right balance between scalability and cost based on your application's growth projections.

## 3 Provider Selection [0%]

You will need a set of services, in order to fulfill the TRs you have listed in Section 2.3. In this section, you must search for the cloud provider that offers most, if not all of these services. You must include at least three providers. At this stage of the game, we do not expect you to know all the details of all the required services.

### 3.1 Criteria for choosing a provider

Supply the criteria you will use for choosing a provider.

Since our application is a social media platform we are considering the below criteria listed as per the priority.

Cost:

- Analyze the pricing structure of the cloud provider, considering both initial setup costs and ongoing operational expenses. Look for transparent pricing and potential discounts based on usage.

Reliability and Uptime:

- Look for a provider with a proven track record of high uptime and reliability. Downtime can have a significant impact on user satisfaction and business operations.

Scalability:

- Ensure the cloud provider can easily scale your infrastructure to accommodate the growing user base of your social media application.

Performance:

- Evaluate the provider's network performance, including latency and data transfer rates, to ensure a smooth and responsive user experience.

Data Security and Compliance:

- Assess the security measures and compliance certifications offered by the cloud provider to ensure the safety of user data and adherence to regulatory requirements.

Data Storage and Retrieval:

- Evaluate the storage solutions provided by the cloud provider and ensure they meet your application's requirements for data storage, retrieval, and backup.

Networking Services:

- Consider the networking services offered, such as content delivery networks (CDN), load balancing, and dedicated private connections, to optimize the performance and reliability of your social media application.

Managed Services and Tools:

- Assess the availability of managed services and development tools provided by the cloud platform to simplify deployment, monitoring, and maintenance tasks.

Integration and Compatibility:

- Ensure that the cloud provider's services seamlessly integrate with the technologies and frameworks you plan to use in your social media application.

Data Center Locations:

- Consider the global distribution of the provider's data centers. Choose a provider with data centers in regions that align with your target audience to minimize latency.

Community and Support:

- Check the availability of community forums, documentation, and support services offered by the cloud provider to ensure that you can easily find help and resources when needed.

Innovation and Future-Proofing:

- Consider the provider's commitment to innovation and the availability of cutting-edge services to ensure that your application can benefit from the latest advancements in cloud technology.

## 3.2 Provider Comparison

Supply a table with the ranking of the providers you considered. Justify the rankings.

Criteria-based comparison of 3 cloud providers AWS, GCP, Azure [1],[2],[3],[4],[5]

| Criteria | AWS | GCP | Azure | Winner | Explanation |
|---|---|---|---|---|---|
| Cost | 1 | 3 | 2 | AWS | We expect the traffic to be steady with less fluctuations. We certainly don't expect frequent burstable instances.<br><br>For both general-purpose and compute-optimized instances AWS offers better discount rates hence we have chosen AWS as the winner in this criterion. |
| Reliability and Uptime | 1 | 2 | 3 | AWS | https://www.clouve.com/blog/aws-vs-gcp-vs-azure-comparing-scalability-availability-and-monitoring-capabilities/ |
| Scalability | 1 | 2 | 3 | AWS | https://www.clouve.com/blog/aws-vs-gcp-vs-azure-comparing-scalability-availability-and-monitoring-capabilities/ |
| Performance | 2 | 1 | 3 | GCP | https://www.thestack.technology/aws-vs-azure-vs-gcp/ |
| Managed Services and Tools | 1 | 2 | 3 | AWS | AWS offers a combination of a comprehensive service offering, global infrastructure, scalability, a large and active community, a rich ecosystem, strong security measures, and proven reliability. These factors make AWS a well-suited and reliable choice for hosting our application |

| | | | | | |
|---|---|---|---|---|---|
| Data Security andCompliance | 2 | 1 | 3 | GCP | https://www.pluralsight.com/resources/blog/cloud/cloud-security-comparison-aws-vs-azure-vs-gcp |
| Data Storage and Retrieval | 1 | 1 | 1 | - | Everyone is good.<br><br>https://www.smikar.com/gcp-azure-and-aws-cloud-storage/ |
| Networking Services | 1 | 1 | 1 | - | Everyone offers everything.<br><br>https://endjin.com/blog/2016/11/aws-vs-azure-vs-google-cloud-platform-networking |
| Integration and Compatibility | 1 | 2 | 3 | AWS | AWS offers a vast array of services covering computing, storage, databases, machine learning, and more, allowing for extensive integration possibilities. |
| Data Center Locations | 3 | 2 | 1 | Azure | We expect to have global connectivity with the least latency for which we require a large array of availability zones which is provided by Azure. |
| Community and Support | 1 | 3 | 2 | AWS | considering the widespread usage, community size, and extensive feature set, AWS would be a strong choice since AWS has the largest and most active community, making it an excellent choice for finding resources, solutions, and community-driven support require to handle the services used for our application. |
| Innovation and Future-Proofing | 2 | 1 | 3 | GCP | GCP's emphasis on open source and multi-cloud solutions provides flexibility for the future.For making our application recommendations Ai driven we would require a provider that offers more flexibility in the innovative fields. |
| Data Security and Compliance | 2 | 1 | 3 | GCP | https://www.pluralsight.com/resources/blog/cloud/cloud-security-comparison-aws-vs-azure-vs-gcp |

## Services offered by each cloud provider.

| Criteria | AWS | GCP | Azure |
|---|---|---|---|
| Cost | AWS Cost Explorer,<br>AWS Budgets,<br>AWS Savings Plans | Google Cloud Billing,<br>Google Cloud Pricing Calculator,<br>Sustained Use Discounts | Azure Cost Management + Billing,<br>Azure Pricing Calculator,<br>Reserved Instances |
| Reliability and Uptime | Amazon EC2 Auto Scaling,<br>Amazon RDS Multi AZ,<br>Amazon Global Accelerator | Compute Engine Autoscalar,<br>Google Cloud, Spanner,<br>Google Cloud Load Balancing | Azure Machine Scale Sets,<br>Azure SQL Database Geo-Replication,<br>Azure Traffic Manager |
| Scalability | Amazon EC2 Instances,<br>Amazon S3,<br>AWS Lambda | Google Compute Engine Instances,<br>Google Cloud Storage,<br>Google Kubernetes Engine (GKE) | Azure Virtual Machines,<br>Azure Blob Storage,<br>Azure Kubernetes Service (AKS) |
| Managed Services and Tools | Amazon RDS,<br>AWS Elastic Beanstalk,<br>AWS CloudFormation | Google Cloud SQL,<br>App Engine,<br>Google Cloud Deployment Manager | Azure SQL Database,<br>Azure App Service,<br>Azure Resource Manager (ARM) |
| Monitoring | Amazon CloudWatch,<br>AWS CloudTrail,<br>AWS X-Ray | Google Cloud Monitoring,<br>Google Cloud Audit Logging,<br>Google Cloud Trace and Debug | Azure Monitor,<br>Azure Activity Log,<br>Azure Application Insights |
| Data Security and Compliance | AWS Key Management Service (KMS),<br>Amazon Macie,<br>AWS Config | Google Cloud Key Management Service (KMS),<br>Google Cloud Data Loss Prevention (DLP),<br>Google Cloud Security Command Center | Azure Key Vault,<br>Azure Information Protection,<br>Azure Policy |
| AI/ML | Amazon SageMaker,<br>Amazon Comprehend,<br>Amazon Rekognition | Google AI Platform,<br>Google Cloud Natural Language API,<br>Google Cloud Vision API | Azure Machine Learning,<br>Azure Cognitive Services,<br>Azure Databricks |

3.3 The final selection

If there is a clear winner, just announce your final selection. If not, mention and justify the tradeoff you made.

Based on the comparison above we have chosen **AWS as the winner**. Its leading serverless computing services, with a comprehensive set of AI/ML services along with most extensive and mature service portfolio, and well-established global infrastructure hence it will help in designing profitable solutions.

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

List here (supplying a URL is also fine) all the services offered by the winner..

- CloudFormation
- Cloud Watch
- Amazon DynamoDB
- Amazon S3 Cross-Region Replication
- AWS cloudendure, AWS backup
- AWS config for compliance, AWS Cost explorer
- Amazon VPC
- AWS Shield
- Amazon CloudFront
- Amazon Route 53
- Amazon GuardDuty
- AWS Lambda
- IAM
- Amazon Elasticsearch Service
- Kendra
- cloudtrail
- KMS
- Audit Manager
- Inspector
- Systems Manager
- Macie
- WAF
- Cloud Trail
- Transcribe
- Rekognition

- Comprehend
- Textract
- SNS
- Amazon Cognito
- Amazon RDS(Data Encryption)
- Amazon Organisations
- AWS Cost and Usage Reports
- AWS CodePipeline + CodeBuild+ CodeDeploy
- AWS CloudFormation
- AWS Auto Scaling

## Describe in detail two services per team member:

**AWS CloudWatch**:

Amazon CloudWatch acts as a crucial tool for observations in AWS environments. It offers real-time monitoring, logging, and alerting, empowering users to keep a close eye on the performance and health of their AWS resources.

**Key Features:**

1. **Metrics and Dashboards:**

CloudWatch collects and visualizes essential performance metrics, providing a customizable dashboard for real-time monitoring and analysis of AWS resources.

2. **Logs and Insights:**

The service facilitates centralized log management, allowing the collection and analysis of log data from various AWS services. CloudWatch Insights offers powerful capabilities for a deeper understanding of log info.

3. **Alarms and Notifications:**

Users can set alarms based on predefined thresholds or custom metrics. CloudWatch Alarms can trigger notifications through Amazon SNS, ensuring a prompt response to any deviations from expected performance.

**AWS CloudFront:**

Amazon CloudFront is a globally distributed Content Delivery Network (CDN) designed to accelerate content delivery by caching data at edge locations across the world.

**Key Features:**

**1.Global Content Delivery:**

CloudFront enhances the distribution of static and dynamic web content globally, utilizing edge locations to cache content and reduce latency for end-users.

**2.Security and DDoS Protection:**

It integrates seamlessly with AWS Web Application Firewall (WAF) to bolster security against common web exploits. Additionally, CloudFront provides DDoS protection, safeguarding applications from distributed denial-of-service attacks.

**3.Customization and Personalization:**

CloudFront offers customization through features like Lambda@Edge, enabling the execution of serverless code at edge locations for tailored content delivery and personalization.

**AWS Lambda:**

AWS Lambda is a serverless computing service that enables developers to run code without the need for server provisioning or management.

**Key Features:**

**1. Event-Driven Computing:**

Lambda operates in response to events, such as changes to data in an S3 bucket, updates to a DynamoDB table, or HTTP requests via Amazon API Gateway, making it highly adaptable to various triggers.

**2. Scalability and Cost Efficiency:**

The service automatically scales based on the number of requests, with users only paying for the actual compute time consumed. Lambda supports multiple programming languages, providing flexibility in development.

**3. Integration with AWS Services:**

Lambda seamlessly integrates with various AWS services, offering developers the ability to build applications using a serverless architecture. It can be part of workflows through services like AWS Step Functions, streamlining application development.

**AWS Cognito:**

Amazon Cognito is a fully managed identity and access management service designed to facilitate secure user sign-up and sign-in for mobile and web applications.

**Key Features:**

**1. User Authentication:**

Cognito supports multiple authentication providers, including social identity providers (Facebook, Google, etc.) and corporate identity providers (SAML, OpenID Connect), offering versatile user authentication options.

**2.User Pools and Identity Pools**:

User Pools manage user identities, providing features such as multi-factor authentication and account recovery. Identity Pools enable secure access to AWS resources for authenticated users, enhancing overall security.

**3.Federated Identities:**

Cognito seamlessly integrates with other AWS services and third-party identity providers, supporting federated identities. This simplifies the management of user identities across different applications, promoting a unified and secure user experience.

## 4 The first design draft [0%]

In this section, you must provide the first draft of your proposed design. It will help if you give this section a try without consulting Chapter 4 (especially Sections 4.4 through 4.6) of the class notes.

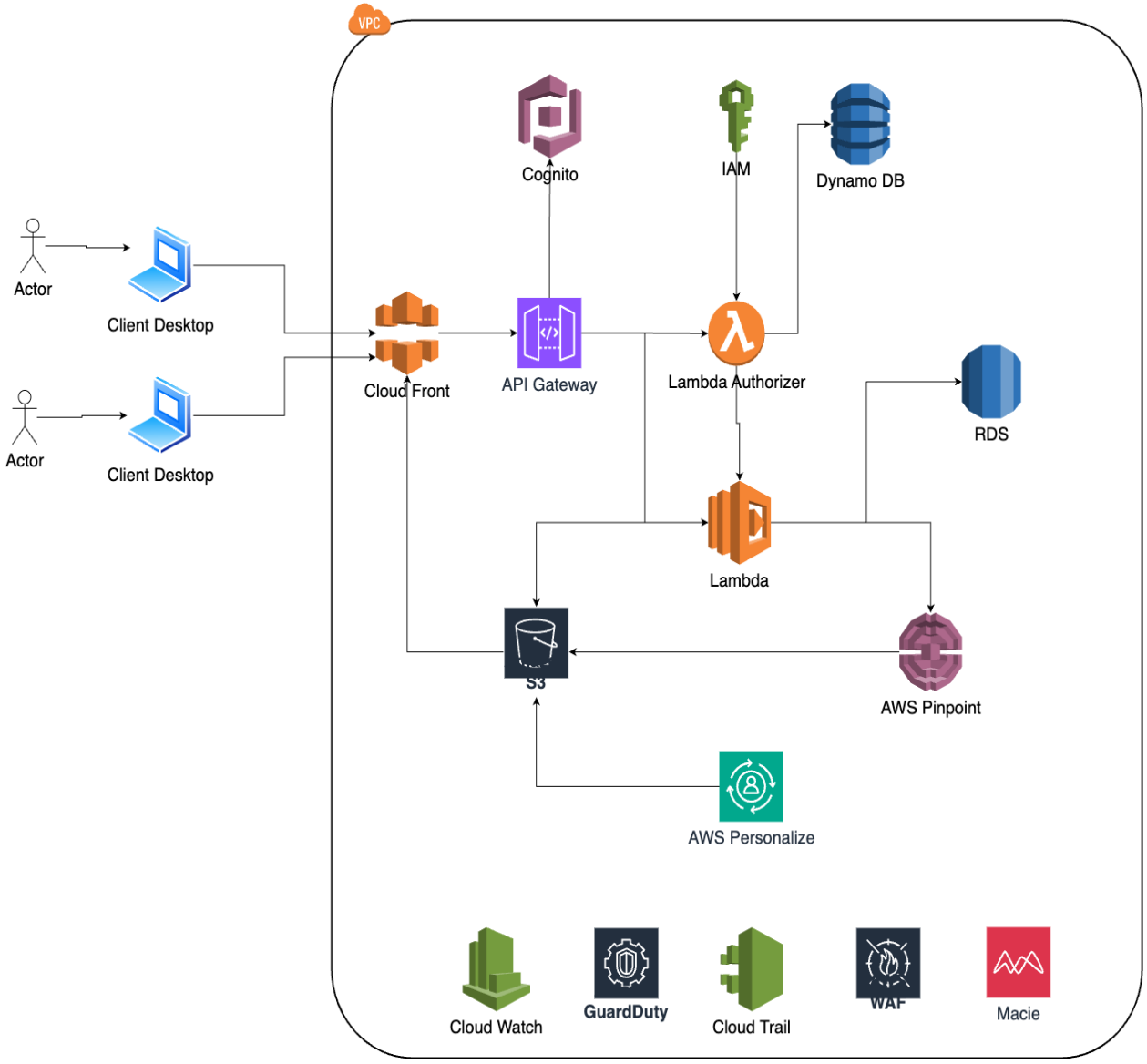
No need for a complete description or structured process, these will come in the next section. The idea here is to present your thoughts on how you plan to solve the problem to the instructors and get early feedback.

**List of TRs for our design:**

Our design using AWS solutions is designed to ensure technical requirements like **Tenant Identification** [TR 9.2]**, Monitoring** [TR 1.2, TR 2.2, TR 5.4, TR 7.1, TR 10.1]**, Security** [TR 3.5, TR 4.1, TR 4.2, TR 4.3, TR 4.4, TR 7.3, TR 9.1]**, Community Engagement** [TR 6.1, TR 6.2, TR 6.3], **load balancing for data delivery** [TR 8.1, TR 8.2], **scalability** [TR 1.1] Cost optimization[10.2]

In our proposed design, we have selected specific Technical requirements that play a pivotal role for our community based social media application.

The below proposed design is our first draft that addresses the mentioned Technical requirements.




4.1 The basic building blocks of the design

Have you identified any distinct elements or features of your solution? List them here.

For our Application, we require to extract user behavior and patterns to build robust AI engines for recommendation of various communities, since our application revolves around community engagement the distinct feature in our application includes:[7]

- **COMMUNITY ENGAGEMENT:**
  **→ AWS Pinpoint**
  **→ AWS S3**

→ **AWS Personalize**

Apart from the community engagement requirement, we have listed all the components for the various requirements:

- **STORAGE:**

  → **AWS S3**
  → **AWS RDS**

- **TENANT IDENTIFICATION AND AUTHENTICATION:**

  → **AWS Cognito**
  → **AWS API Gateway**

- **NETWORKING:**

  → **AWS API Gateway**
  → **AWS Cloudfront**
  → **AWS VPC**
  → **AWS GuardDuty**
  → **AWS WAF**

- **SECURITY:**

  → **AWS KMS**
  → **AWS VPC**
  → **AWS CLOUDFRONT**
  → **AWS guardDuty**
  → **AWS IAM**
  → **AWS WAF**
  → **AWS Lambda Authorizer**

- **MANAGED DATABASE:**
  → **AWS DynamoDB**

- **MONITORING:**
  → **AWS Cloudwatch**
  → **AWS CloudTrail**
  → **AWS Macie**

- **LOAD BALANCING:**
  → **AWS ELB**

- **COMPUTATION:**

  → **AWS Lambda**
  → **AWS cloudfront**

4.2 Top-level, informal validation of the design

Provide arguments that your solution will work; that is, your design will achieve the TRs you came up with in Section 2.3. The instructors (or even yourself) may find the arguments "weak"; that's OK at this stage of the game.

In crafting an efficient and scalable architecture for our application, we have meticulously designed a workflow that leverages various AWS services to ensure optimal performance, security, and scalability. Below is a detailed breakdown of our architecture:

**Tenant Identification and Authorization:**

1. **User and Admin Interaction**: Users and administrators initiate workflow through requests. When a user interacts with the system for the first time, it will be authenticated and will be assigned a security token by Lambda Authorizer. This token then will be stored in DynamoDB and will be used for upcoming authentication and authorization tasks.

2. **Foreground Layer:** Requests directed to Amazon CloudFront for optimized content delivery globally.: Requests to cloudfront and then WAF[11] checks for correct (protection layer)

3. **API Gateway:** Serves as a centralized entry point for backend services, managing and routing requests efficiently. All requests are directed by the API gateway to the corresponding endpoint.

4. **User Authentication:** Integrated with Amazon Cognito for secure identity management, ensuring a user-friendly experience. Each request will have authentication information. Lambda authorizer will authenticate the user using this information with the help of AWS cognito and dynamoDB.

5. **Lambda Authorizer:** Serverless function will be responsible for extracting authentication information from the request and validating user identity either in the form of security tokens or username and password. It will also check access rights with the help of IAM service.

6. **Token Storage:** The generated tokens are securely stored in Amazon DynamoDB, a fully managed NoSQL database service. DynamoDB ensures fast and reliable access to authorization tokens, facilitating seamless user authentication.

**Data Storage**:

**a. Amazon RDS:** Structured data, including user profiles, posts, comments, and votes, are stored in Amazon RDS tables. RDS offers a relational database solution, maintaining data integrity and relationships.

**b. Amazon S3:** Media files, such as images and videos, find their home in Amazon S3 buckets. These buckets are organized by user or post ID, promoting efficient data retrieval. Additionally, S3 can be configured to work in tandem with CloudFront, optimizing media file delivery.

**Security:**

1. **DDoS Protection:**
   Service: Amazon CloudFront
   Description: CloudFront serves as both a CDN and a robust defense against DDoS attacks, ensuring uninterrupted traffic flow to our application.

2. **Data in Transit:**
   Service: Virtual Private Cloud (VPC)
   Description: Amazon VPC provides a private and secure network, allowing control over data flow between services within the isolated VPC boundaries.

3. **Data at Rest:**
   Service: Key Management Service (KMS)
   Description: AWS KMS is employed to create and manage encryption keys, ensuring data stored in services like Amazon RDS and Amazon S3 is encrypted for an added layer of security.

4. **Real-Time Threat Protection:**
   Service: Amazon GuardDuty
   Description: Integrated for continuous threat protection, GuardDuty utilizes machine learning and threat intelligence to swiftly identify and prioritize potential security threats, enhancing the overall security posture of our application.

5. **AWS WAF (Web Application Firewall):** Critical defense layer against web exploits, ensuring content compliance and regulatory requirements are met. Fortifies the application against threats, creating a secure environment for user interactions.

**Monitoring:**

1. **Amazon CloudWatch:** Real-time monitoring solution, collecting metrics from AWS services (e.g., API Gateway, Lambda) for proactive issue resolution and overall system health maintenance.

2. **Amazon CloudTrail:** Integrated with CloudWatch, CloudTrail logs API activity for security auditing, providing a detailed history of actions to enhance visibility and analyze usage patterns.

3. **Amazon Macie:** Utilizes machine learning for content compliance, discovering and protecting sensitive data to ensure adherence to compliance standards and safeguard user information.

**Community Engagement Data Flow:**

1. **User Interaction Events:** Captures clicks, views, and other interactions as crucial indicators of community engagement.

2. **Amazon Pinpoint:** Records and monitors user metrics in real-time, offering insights into user behavior, preferences, and trends for personalized community experiences.

3. **Amazon S3 (Data Lake):** Stores raw and processed data, creating a scalable Data Lake for historical user interaction data. Facilitates easy access for in-depth analysis, promoting a data-driven approach to community engagement.

4**. Amazon Personalize:** Enriches user engagement with personalized content recommendations using machine learning. Analyzes user behavior data in S3 to tailor recommendations for individual community members.

4.3 Action items and rough timeline

Provide a list of action items to complete the project; include a rough timeline on how long each action item will take you to complete it.

The list and timeline are not cast in stone; you will not be penalized if you don't adhere to them.
The idea here is to help you avoid an 11th-hour dash to the finish line.

**SKIPPED.**

5 The second design [0%]

Hopefully, you came to this phase of your design after getting feedback from the instructors. Now you'll follow some structured process for creating the details of your design. Read Sections 4.3 - 4.6 in the class notes.

5.1 Use of the Well-Architected framework

Mention here the distinct steps suggested by the Well-Architected framework. See Section 4.3 in the class notes and the complete descriptions of the framework in the AWS pages for details.

The Well-Architected Framework outlines a set of distinct steps to ensure the creation of reliable, secure, high-performing, and efficient cloud-based architectures. These steps include:[6]

1. **Review the AWS Well-Architected Pillars**:

Understand and review the five key pillars: Operational Excellence, Security, Reliability, Performance Efficiency, and Cost Optimization.

2. **Answer the Well-Architected Questions**:

Go through the set of questions provided for each pillar. These questions act as a guide to assess the architecture against best practices.

3. **Identify Areas for Improvement**:

Analyze the responses to the questions to identify areas where the architecture can be improved in alignment with the Well-Architected best practices.

4**. Implement Best Practices**:

Develop an action plan to address identified areas for improvement. Implement best practices and recommended solutions to enhance the architecture.

5. **Integrate Well-Architected Practices into Workloads**:

Incorporate Well-Architected practices into the ongoing development and operational processes of the workloads.

6.**Monitor and Iterate**:

Continuously monitor the workloads for changes and improvements. Regularly revisit the Well-Architected Framework to ensure ongoing alignment with best practices.

7. **Use Automation to Enforce Best Practices:**

Leverage automation tools and services to enforce and maintain adherence to Well-Architected best practices consistently.

8.**Document Decisions and Architectural Choices**:

Maintain documentation of architectural decisions and choices made in alignment with the Well-Architected Framework. This documentation aids in knowledge transfer and future assessments.

9. **Integrate Well-Architected into Governance Processes**:

Integrate Well-Architected assessments and reviews into governance processes to ensure continuous alignment with best practices across the organization.

10. **Leverage Well-Architected Tool:**

Utilize the Well-Architected Tool provided by AWS to conduct formal reviews, document findings, and track progress in improving workloads.

5.2 Discussion of pillars
Discuss in detail one pillar per team member.

**Reliability Pillar**:[8]

1. **Foundations:**
The Well-Architected Framework suggests building our system on proven and reliable technologies, akin to establishing a sturdy foundation. Use well-tested components and distribute workloads across different areas (Availability Zones) to enhance system resilience.

2. **Change Management:** **[8]**
To address updates seamlessly, the framework recommends an approach of control and automation, envisioning the update process as changing a tire on a moving car. This involves testing changes with a small user group through canary deployments and implementing toggles for feature control, etc.

3. **Failure Recovery:**
The framework advises anticipating system failures and implementing mechanisms for autonomous recovery. This includes self-healing processes and robust backup plans for critical data to ensure business continuity during disruptions.

4. **Scalability:**
Scalability is encouraged by envisioning flexibility as a rubber band, capable of stretching with increased usage and contracting during low-demand periods. The framework suggests choosing scalable components that can adapt to varying workloads.

5. **Performance Efficiency:**
For optimal performance, the framework recommends regular monitoring and addressing potential inefficiencies, envisioning fine-tuning the system to improve efficiency and ensure smooth operation.

6. **Monitoring:**
Monitoring and alerting systems are suggested to detect and respond to issues promptly. The framework envisions this as deploying sensors in our system to ensure readiness through regular testing.

**Security Pillar:**

1. **Data Protection:**
To safeguard data, the framework suggests classifying it based on importance and implementing security measures, including encryption. Specialized keys, like those from AWS Key Management Service, are recommended to secure data.

2. **Identity and Access Management (IAM):**
IAM is crucial for secure access, and the framework suggests envisioning our system as a controlled access club. It advises providing individuals with precisely the permissions needed for their roles and regularly reviewing IAM policies for security.

3. **Infrastructure Protection:**
For infrastructure protection, the framework envisions our system as a fortress. Robust network security measures, such as firewalls and security groups, along with regular software and firmware updates, are recommended to repel potential threats.

4. **Detective Controls:**
The framework recommends implementing detective controls by envisioning security cameras and alarms within our system to monitor and detect suspicious activities. Logging and monitoring tools like AWS CloudTrail are suggested for auditing and tracking changes.

5. **Incident Response:**
Developing and regularly testing incident response plans is suggested to efficiently address security incidents. This is akin to a fire drill for our system, ensuring a coordinated response to emergencies.

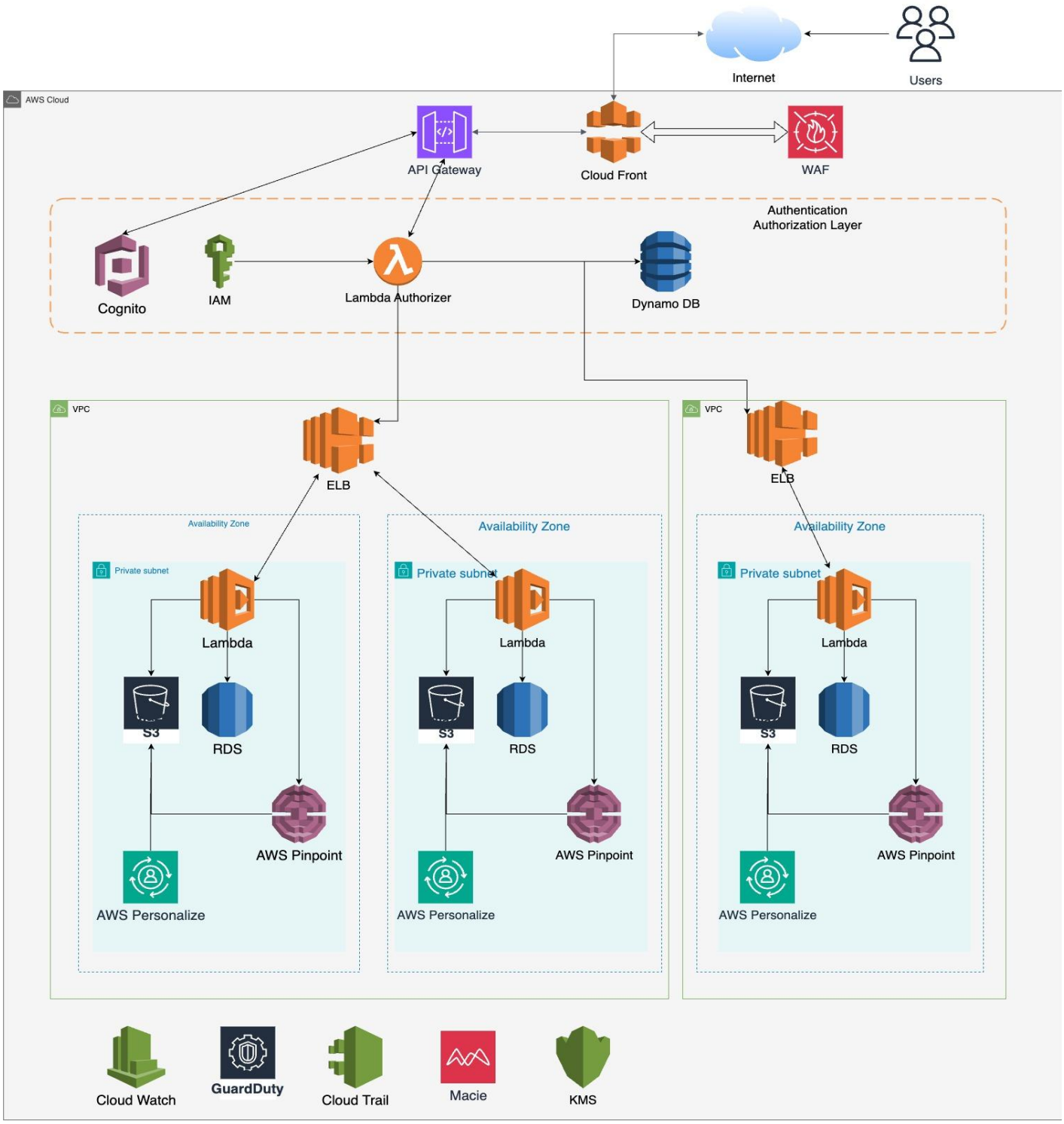
6. **Automation:**
Automation is integral for consistent security practices, and the framework suggests envisioning our system as a robot following rules automatically. Tools like AWS CloudFormation and AWS Config Rules are recommended to automate security configurations and minimize manual errors.

7. **Compliance:**
For adherence to rules and regulations, the framework suggests envisioning our system as compliant. Utilizing AWS tools like AWS Artifact to access compliance reports and certifications provides evidence of regulatory adherence.

In summary, the Well-Architected Framework recommends building our system with a robust foundation, controlled change processes, and enhanced recovery capabilities. Simultaneously, it suggests prioritizing data protection, secure access management, proactive threat detection, and compliance with rules and regulations to ensure a well-architected and secure environment in the AWS cloud.

5.3 Use of Cloudformation diagrams

Present your design using Cloudformation templates.[9]


Internet
Users
AWS Cloud
API Gateway
Cloud Front
WAF
Authentication
Authorization Layer
Cognito
IAM
Lambda Authorizer
Dynamo DB
VPC
ELB
Availability Zone
Private subnet
Lambda
S3
RDS
AWS Pinpoint
AWS Personalize
VPC
ELB
Availability Zone
Private subnet
Lambda
S3
RDS
AWS Pinpoint
AWS Personalize
Availability Zone
Private subnet
Lambda
S3
RDS
AWS Pinpoint
AWS Personalize
Cloud Watch
GuardDuty
Cloud Trail
Macie
KMS

5.4 Validation of the design

Argue about how your design meets the BRs or TRs you listed in Section:

We have listed all the services that we have used below and how the services enable us to cater all the listed business as well as technical requirements:

**Tenant Identification and Authorization:**

1. User and Admin Interaction:

Users and administrators initiate workflows through requests, beginning with authentication and tokenization facilitated by the Lambda Authorizer. The Lambda Authorizer plays a pivotal role in generating security tokens for first-time users, with DynamoDB serving as a secure repository for these tokens. Subsequent interactions leverage these tokens, ensuring seamless authentication and authorization.

2. Foreground Layer:

The foreground layer is fortified by Amazon CloudFront, serving as a global content delivery network (CDN) and a robust defense against Distributed Denial of Service (DDoS) attacks. AWS WAF [11] acts as a gatekeeper, checking requests to CloudFront for correctness and serving as a protection layer against potential threats.

3. API Gateway:

API Gateway stands as the centralized entry point for backend services, efficiently managing and routing requests to their respective endpoints. This architectural choice ensures a streamlined and organized communication flow between different components of the system.

4. User Authentication:

Integrated with Amazon Cognito, the system leverages a secure identity management solution. Each user request carries authentication information, authenticated by the Lambda Authorizer through AWS Cognito and DynamoDB. This multi-layered approach ensures a user-friendly and secure experience.

5. Lambda Authorizer:

This serverless function is responsible for extracting and validating authentication information from user requests. It verifies user identity through security tokens or username/password combinations and employs AWS Identity and Access Management (IAM) services to check access rights. The Lambda Authorizer plays a crucial role in the security and authorization workflow.

6. Token Storage:

The generated security tokens are securely stored in Amazon DynamoDB, a fully managed NoSQL database service. DynamoDB ensures fast and reliable access to authorization tokens, playing a critical role in facilitating seamless and secure user authentication.

**Load Balancing:**

Elastic Load Balancer (ELB):

Following the Lambda Authorizer, our application utilizes Elastic Load Balancers (ELBs) to manage and distribute incoming requests among multiple instances of our application. ELBs play a critical role in load balancing, ensuring that your application can efficiently handle varying traffic levels.

1. **Load Definition:** The load would be concurrent user activity on our platform.
2. **Metric:** The load can be measured by the number of HTTP requests made to the application per second.
3. **Measuring Load:** It should be done for various operations of application. We can measure requests per second in below scenarios.
    a. A burst of new users registering.
    b. A mix of users logging in and out.
    c. Users creating new posts or sharing content.
    d. Users retrieving their feeds.
4. **Load Balance Criterion:** User experience is crucial in social media applications hence we can set average response time as a load balancing criterion. We can balance the load till the application responds to user actions within an acceptable time frame.
5. **Load Balancing Action:** To achieve this load balancing we will need to auto-scale backend compute instances, database instances. Along with that we also need to set up CloudWatch alarms to trigger scaling actions.

**Reliability:**

The decision to deploy our application across multiple Availability Zones (AZs) is rooted in the pursuit of enhanced reliability and fault tolerance. AWS provides Availability Zones as physically isolated data centers within a region, each with its own power, cooling, and networking infrastructure.

**Data Storage:**

a. Amazon RDS:

Structured data, including user profiles, posts, comments, and votes, is stored in Amazon RDS tables. Amazon RDS provides a relational database solution, ensuring data integrity and relationships, crucial for maintaining the core functionalities of the application.

b. Amazon S3:

Media files such as images and videos find their home in Amazon S3 buckets. These buckets are organized by user or post ID, promoting efficient data retrieval. The integration with CloudFront optimizes the delivery of media files, enhancing the overall performance of the system.

**Security:**

1. DDoS Protection:

Amazon CloudFront, with its dual role as a CDN and DDoS protection, ensures uninterrupted traffic flow to the application. This provides a robust defense mechanism against potential disruptions due to malicious attacks.

2. Data in Transit:

Virtual Private Cloud (VPC) establishes a private and secure network, allowing control over data flow between services within isolated VPC boundaries. This ensures the confidentiality and integrity of data in transit.

3. Data at Rest:

Key Management Service (KMS) is employed for creating and managing encryption keys, adding an additional layer of security. Data stored in services like Amazon RDS and Amazon S3 is encrypted, safeguarding sensitive information even when at rest.

4. Real-Time Threat Protection:

Amazon GuardDuty is seamlessly integrated for continuous threat protection. Leveraging machine learning and threat intelligence, GuardDuty swiftly identifies and prioritizes potential security threats, thereby enhancing the overall security posture of the application.

5. AWS WAF (Web Application Firewall):

AWS WAF acts as a critical defense layer against web exploits. It ensures content compliance, meets regulatory requirements, and creates a secure environment for user interactions, safeguarding the application against potential threats.

**Monitoring:**

1. Amazon CloudWatch:

Amazon CloudWatch provides a real-time monitoring solution, collecting metrics from various AWS services (e.g., API Gateway, Lambda) for proactive issue resolution. It ensures the overall health and performance of the system.

2. Amazon CloudTrail:

Integrated with CloudWatch, CloudTrail logs API activity for security auditing. It offers a detailed history of actions, enhancing visibility into system activities and aiding in the analysis of usage patterns for security and optimization purposes.

3. Amazon Macie:

Amazon Macie utilizes machine learning for content compliance, discovering and protecting sensitive data. This ensures adherence to compliance standards and adds an

additional layer of protection to user information, contributing to the overall reliability of the system.

**Community Engagement** :

1. User Interaction Events:

The system captures crucial indicators of community engagement, such as clicks, views, and other interactions. These events serve as essential data points for understanding user behavior and preferences.

2. Amazon Pinpoint:

Amazon Pinpoint records and monitors user metrics in real-time, providing insights into user behavior, preferences, and trends. This real-time monitoring is invaluable for creating personalized community experiences and tailoring content to user preferences.

3. Amazon S3 (Data Lake):

Amazon S3 serves as a scalable Data Lake, storing raw and processed data related to community engagement. This scalable storage solution facilitates historical user interaction data analysis, supporting a data-driven approach to community engagement.

4. Amazon Personalize:

Amazon Personalize enriches user engagement by providing personalized content recommendations using machine learning. By analyzing user behavior data stored in S3, it tailors recommendations for individual community members, enhancing the overall user experience and community engagement.

5.5 Design principles and best practices used

List any specific principle or best practices you used in your design. See Sections 4.5 and 4.6 in the class notes for info.[6]

1. **Security: Implement Robust Identity and Access Management (IAM)**:

We followed the principle of least privilege in IAM, ensuring that users and services have only the permissions necessary for their roles. Regularly reviewing and auditing IAM policies to maintain a secure access environment. This practice will help us prevent unauthorized access and enhance the overall security posture.

2. **Reliability: Leverage Multi-AZ Deployment for High Availability**:

We tried to enhance the reliability of our application by deploying it across multiple Availability Zones (AZs). This ensures that if one AZ experiences issues, the application can seamlessly failover to another, providing high availability and minimizing downtime.

3. **Performance Efficiency: Optimize Database Performance**:

We focused on optimizing the performance of our application's database. We Implement caching mechanisms and consider using managed database services like Amazon RDS to offload administrative tasks and ensure optimal performance.

4. **Cost Optimization: Enhanced Expense Visibility through Cost Allocation Tags**:

We integrated AWS cost allocation tags to meticulously label and categorize resources according to their purpose, ownership, or department within our application. This strategic tagging implementation has provided us with a detailed breakdown of cost distribution across various components. This enhanced visibility allows for precise cost attribution, empowering us to make informed decisions and optimize resources based on specific usage patterns.

5. **Operational Excellence: Implement Logging and Monitoring**:

We established comprehensive logging and monitoring practices for our application. Used services like Amazon CloudWatch, CloudTrail to monitor key performance metrics, set up alerts for critical events, and maintain detailed logs for troubleshooting. This proactive approach to monitoring contributes to operational excellence by enabling timely issue detection and resolution.

5.6 Tradeoffs revisited

Provide more details on the tradeoffs you made so far4. Be as exhaustive as you wish! You must read and use the material in Reference 5.2, ""Even Swaps: A Rational Method for Making Trade-offs".

**User Engagement(TR 5.1) vs. Content Moderation(TR 7.1):**

In our platform, we've strategically chosen to prioritize user engagement (TR 5.1) by maintaining an open space for diverse information and discussions. Our primary goal is to provide users with a platform where they can freely express themselves. This approach inherently involves a trade-off with content moderation (TR 7.1), as we aim to strike a balance between fostering open conversations and controlling inappropriate or harmful content.

Recognizing the potential challenges of an open environment, we've implemented various features and provided comprehensive information to empower users to control their content moderation preferences. By offering users the ability to customize their experience, set preferences, and define personal thresholds for what they find acceptable, we aim to enhance user autonomy while still addressing concerns related to content quality and community well-being.

It's important to acknowledge that this approach comes with its complexities. While we encourage free expression, we understand the importance of safeguarding against misinformation, bullying, and harmful comments.

In essence, our platform recognizes the inherent tension between user engagement and content moderation. By empowering users to control their content experience, we believe we can strike a nuanced balance that fosters a vibrant community without compromising on the safety and well-being of our users.

**Encryption Overhead (TR 4.1) vs. Performance(TR 1.1):**

In our  system, we recognize the critical importance of data security, and as such, we've implemented robust encryption measures, leveraging services such as Key Management Service (KMS) and security features like GuardDuty. However, the implementation of strong encryption introduces a trade-off between Encryption Overhead (TR 4.1) and Performance (TR 1.1).

The use of encryption, particularly strong algorithms facilitated by services like KMS, significantly enhances data protection, ensuring that sensitive information remains confidential and secure. However, this security measure is not without its challenges. The computational overhead associated with encryption processes can impact system performance.

The trade-off involves finding the right balance between the level of encryption applied and maintaining optimal system performance. While stronger encryption algorithms contribute to higher security, they may lead to increased processing demands. This, in turn, can affect response times and overall system efficiency.

Our commitment involves ongoing monitoring, assessment, and optimization of the encryption strategy to minimize performance impacts. By continuously evaluating the trade-off between encryption overhead and system performance, we ensure that our cloud-based design maintains a high standard of security while delivering a responsive and efficient user experience. This approach aligns with best practices in cloud security, where a nuanced understanding of the encryption-performance trade-off is essential for creating a secure and performant system.

**Data Consistency (TR 3.2) vs. Availability(TR 2.1):**

In the context of our system architecture, we face a trade-off between Data Consistency (TR 3.2) and Availability (TR 2.1), a classic dilemma encapsulated by the CAP theorem.

Data Consistency, ensuring that data is strictly consistent across all nodes in a distributed system, can introduce challenges such as increased latency and potentially reduced system availability. The need for synchronous updates and coordination between nodes to maintain a consistent state may impact response times, leading to delays in serving user requests. This trade-off is a fundamental

aspect of distributed systems where achieving absolute consistency across all nodes in real-time may not be feasible.

On the other hand, prioritizing Availability aims to ensure that the system remains responsive to user requests even in the face of network partitions or node failures. This often involves accepting eventual consistency, where all nodes might not have the latest data simultaneously. While this approach enhances system responsiveness, it comes with the trade-off of potential temporary inconsistencies as updates propagate through the system.

In navigating this trade-off, our system architecture is designed to strike a balance that aligns with the specific needs of our application. We recognize that strict data consistency may not always be the top priority in scenarios where rapid response times and high availability are critical. By embracing eventual consistency, we optimize for system responsiveness, acknowledging that temporary discrepancies may occur but are acceptable within the defined parameters of our use case.

**Scalability(TR 1.1) vs. Cost(TR 10.1):**

In our platform, we grapple with the trade-off between scalability (TR 1.1) and cost (TR 10.1). Scalability is crucial for accommodating a growing user base and handling increased data loads effectively. This entails adopting scalable cloud services, leveraging a microservices architecture, and utilizing distributed databases. However, it's important to acknowledge that scaling often correlates with escalated operational costs.

On the flip side, prioritizing cost efficiency is a strategic consideration that involves optimizing code, exploring serverless computing options, and making judicious choices regarding cloud service plans. Striking the right balance

between scalability and cost is paramount, especially when considering the financial implications of accommodating a larger user base.

In our approach, we are mindful of our application's growth projections and aim to find an optimal solution that aligns with both scalability and cost considerations. We invest in scalable infrastructure to ensure our platform can seamlessly expand with the increasing user demand.

5.7 Discussion of an alternate design

For at least one of the objectives, present an alternate design; discuss why you did not pursue it further.

**SKIPPED**

## 6 Kubernetes experimentation [0%]

In this section, you'll validate some aspects of your design experimentally. Which ones is your choice, but you must discuss with the instructors first. The idea is to limit the scope of the experiment runs and required analysis5.

6.1 Experiment Design

List the TR(s) you chose to work with. Describe in detail the experiment that will validate your design. More specifically, define the configuration of the environment, the inputs and the expected outputs.

**TRs:**

- Dynamically balance the load on Log Ingester nodes.
- Handle a large number of user registration requests.

**Design:**

- We developed a Log Ingestor system that can efficiently handle vast volumes of log data. GitHub Repo: https://github.com/apurv-choudhari/LogIngester

- The HTTP server of the log ingester is written in Flask and it listens on port 3000. It accepts JSON data as logs and inserts log data into the MongoDB database. I have used the remote MongoDB database available in the MongoDB Atlas free version.
- We then built the docker image for this application and pushed it into Docker Hub. To scale the application we created deployment in Minikube. We used the image from the docker hub to deploy the pod.
- To accept the requests we needed k8s service. Hence we created NodePort service with External IP same as VM IP. It automatically mapped pod's 3000 port to VM's 30743
- After setting up the application and minikube we set up HPA to scale pods according to load and see the relevant details of autoscaling.
- Our scaling metric is 50% CPU consumption. We tried to add packets-per-second parameter but the metrics server could not get the details properly hence we dropped it.

**Input and Output:** Input is a log entry as JSON. Check the example input below. The output is the return status of the API on the console.

```C/C++
{
 "level": "error",
 "message": "Failed to list collections",
 "resourceId": "server-1234",
 "timestamp": "2023-09-15T08:00:00Z",
 "spanId": "span-456",
 "commit": "5e5342f",
 "metadata": {
   "parentResourceId": "server-0987"
 }
}
```

6.2 Workload generation with Locust

Describe how you intend to use Locust to create (all or some of) the inputs. Discuss with the instructors use of other workload generators.

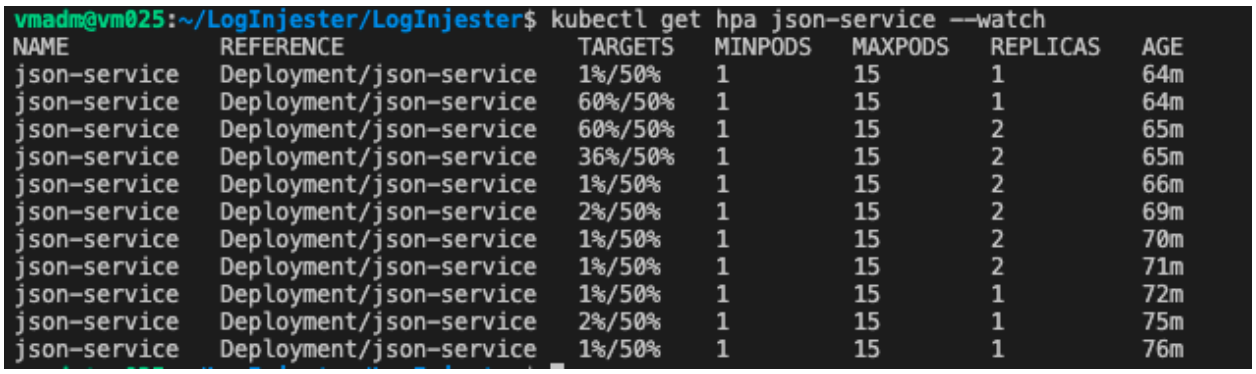
I used Locust to generate a high volume of requests on port 3000. We had to write custom task in locust so that it could use our endpoint and payload to generate load.

```
C/C++
command:
locust -f locust_load.py -u 10 -r 1 -H http://192.168.49.2:30743/json
```

6.3 Analysis of the results

Analyze the outputs you observed. Why do they validate the design?

- When locust increased the load HPA [10] detected high CPU consumption and started more pods. When pods were increased the load got divided into them and CPU consumption on each pod went below 50%.
- When load decreased, pods with 0% load were destroyed.
- Another interesting aspect was observed during scaling. When new pods were deployed, for a very short time span some API requests failed. This happened because new pods were deployed hence k8s service was routing traffic to these new pods but pods were not ready and were in the initialization phase. When pods were up and running all requests were completed successfully. This also explains the small bump in RPS and failure graph.
- Horizontal Pod Auto-Scaling results:



- Locust report

-

# Locust Test Report

**During:** 23/11/2023, 21:55:36 - 23/11/2023, 21:56:56

**Target Host:** http://192.168.49.2:30743/json

**Script:** locust_load.py

## Request Statistics

| Method | Name | # Requests | # Fails | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | RPS | Failures/s |
|---|---|---|---|---|---|---|---|---|---|
| POST | /json | 4805 | 86 | 333 | 0 | 1883 | 20 | 60.6 | 1.1 |
| | **Aggregated** | **4805** | **86** | **333** | **0** | **1883** | **20** | **60.6** | **1.1** |

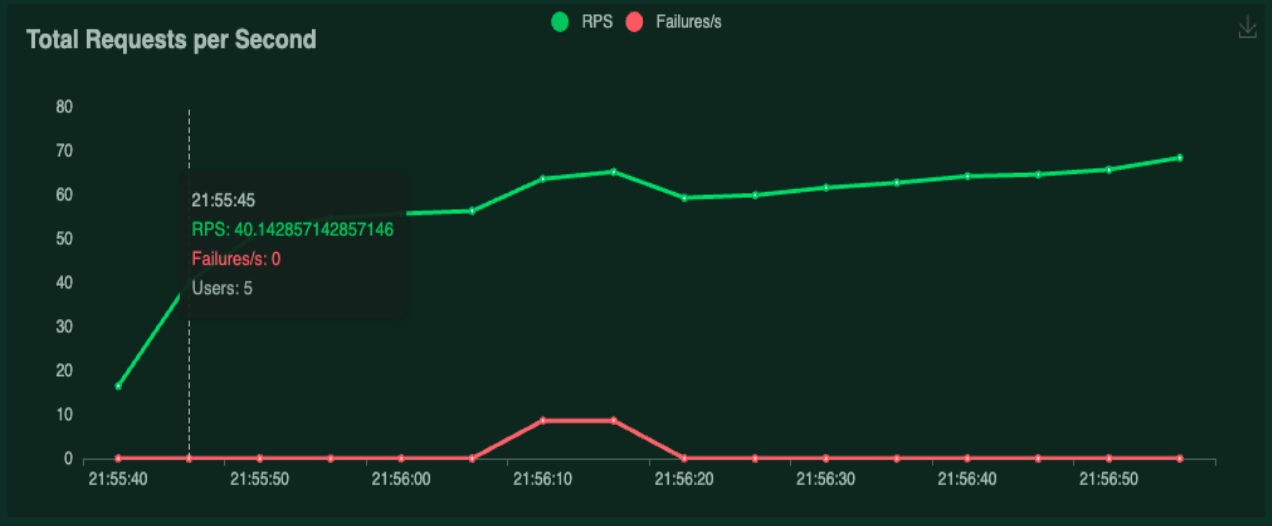
## Response Time Statistics

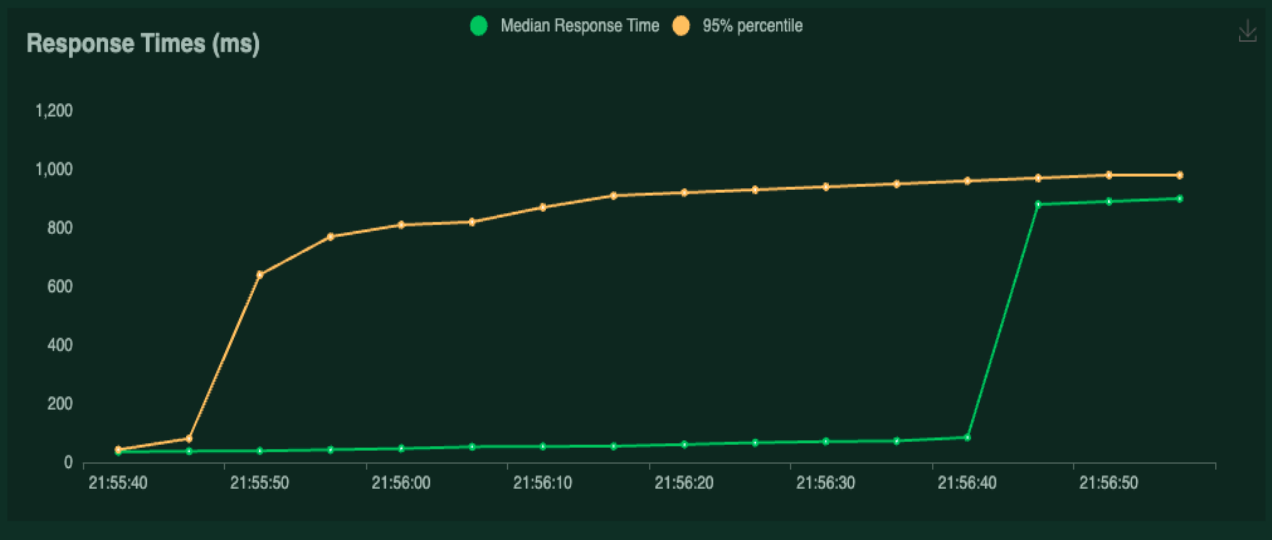
| Method | Name | 50%ile (ms) | 60%ile (ms) | 70%ile (ms) | 80%ile (ms) | 90%ile (ms) | 95%ile (ms) | 99%ile (ms) | 100%ile (ms) |
|---|---|---|---|---|---|---|---|---|---|
| POST | /json | 64 | 76 | 800 | 900 | 930 | 950 | 990 | 1900 |
| | **Aggregated** | **64** | **76** | **800** | **900** | **930** | **950** | **990** | **1900** |

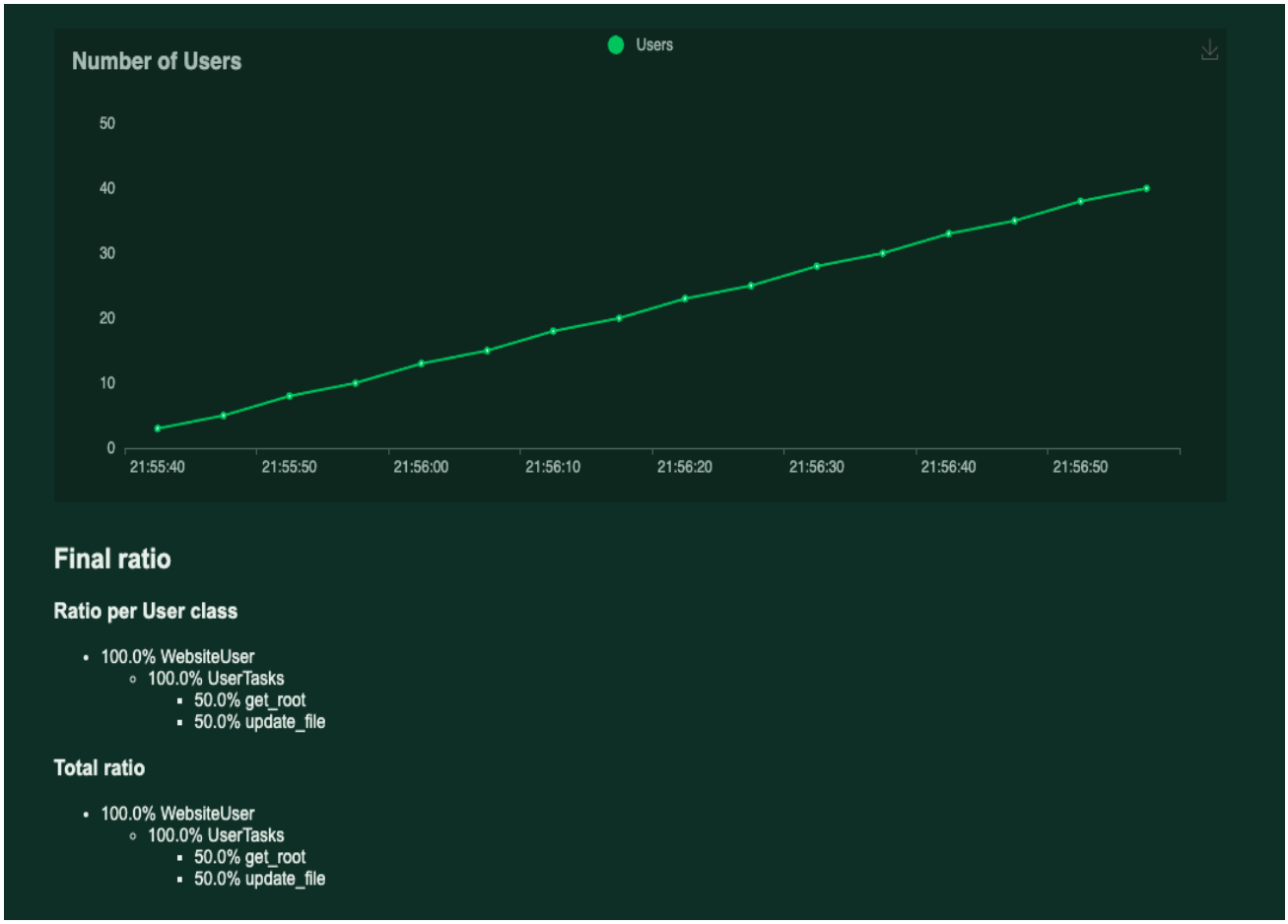
## Failures Statistics

| Method | Name | Error | Occurrences |
|---|---|---|---|
| POST | /json | [Errno 111] Connection refused | 86 |

## Charts


Total Requests per Second
RPS
Failures/s
80
70
60
50
40
30
20
10
0
21:55:45
RPS: 40.142857142857146
Failures/s: 0
Users: 5
21:55:40
21:55:50
21:56:00
21:56:10
21:56:20
21:56:30
21:56:40
21:56:50


Response Times (ms)
Median Response Time
95% percentile
1,200
1,000
800
600
400
200
0
21:55:40
21:55:50
21:56:00
21:56:10
21:56:20
21:56:30
21:56:40
21:56:50




## 7 Ansible playbooks [0%]

**SKIPPED**

In this section, you'll define specific management tasks and write Ansible playbooks to automate them. Since at the beginning of the semester we do not know if we'll have time to cover Ansible, we may skip this section.

### 7.1 Description of management tasks

**SKIPPED**

List the specific tasks. Describe the difficulty involved in executing these tasks manually.

### 7.2 Playbook Design

Write the playbooks for executing the tasks automatically. Comment on the ease/difficulty of producing the playbooks.

### 7.3 Experiment runs

Run the playbooks. Supply verification that the tasks were executed properly.

## 8 Demonstration [0%]

**SKIPPED**

A demo is not required for this project. However, if you want to use any of the (AWS, Azure, IBM, or Google) cloud platforms, you are welcome to do so. Your demo should highlight aspects of your architectural design.

The demo should focus on a few specific TRs. Discuss with the instructors which TRs to use.

## 9 Comparisons [0%]

**SKIPPED**

In this section, you will provide a comparison between two solutions, approaches, tools, current trends, etc. The comparison can be theoretical or experimental.

Here are some suggestions:

1. Facebook’s Katran and another LB algorithm
2. AKS, GKS, EKS and Openshift
3. VxLAN, Geneve, NVGRE virtualization protocols
4. RFC8926: Amazon- and other cloud vendor-specific control channels
5. CloudFormation and another IaC implementation
6. Observability (trend)
7. Microsoft Azure’s Architectural Framework vs AWS vs Google’s Framework 8. Students’ selected topic
9. Others TBD in due time

# 10 Conclusion [0%] 10.1 The lessons learned

Describe, in detail, your experience with this project. What was difficult? interesting? boring? unexpected? Etc.

The project gave us a comprehensive understanding about how we can design a cloud architecture for an application. We chose an application that is a social media platform. The difficulties we faced were if we could order them in terms of magnitude: 1. Tradeoffs between various BRs and TRs. 2. Finding the right kind of services that would encompass all of our TRs. 3. Following the Best practices suggested by AWS.4. Finding the right kind of BRs and TRs.

The most interesting aspects were Implementing a microservices architecture, Exploring and selecting cloud services that align with the project requirements, Setting up load balancing and auto-scaling mechanisms to distribute traffic.

The boring experiences we had were going through tons of documentation from various service providers in order to choose the best.

The unexpected experiences were Performance Bottlenecks and certain failures during the implementation.

## 10.2 Possible continuation of the project

**SKIPPED**

If you plan to take an Independent Study or the advanced course during the spring 2023 semester, provide here some ideas on how to continue the project. Ditto if you are interested in further research/publications.

## 11 References


[1] YV and YP, “ECE547/CSC547 class notes”. [2] Your reference goes here.

1. https://www.clouve.com/blog/aws-vs-gcp-vs-azure-comparing-scalability-availbility-and-monitoring-capabilities/
2. https://cloud.google.com/docs/get-started/aws-azure-gcp-service-comparison
3. https://docs.google.com/spreadsheets/d/1RPpyDOLFmcgxMCpABDzrsBYWpPYCIBuvAoUQLwOGoQw/edit#gid=907731238
4. https://learnk8s.io/kubernetes-autoscaling-strategies#when-autoscaling-pods-goes-wrong
5. https://www.mindee.com/blog/autoscaling-comparison-aws-gcp-azure#:~:text=And%20Microsoft%20Azure-,Autoscaling%20features,managed%20instance%20groups%E2%80%9D%20in%20GCP.
6. https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html
7. https://aws.amazon.com/free/?trk=578b2801-ffbb-4021-a1db-01a0bafb3a4a&sc_channel=ps&ef_id=CjwKCAiAsIGrBhAAEiwAEzMlC-wKj8ERGv08OYQQI2ro-3Ztu7pGIzkfCI0tboGXMjCL4zQOd16OBhoCwFoQAvD_BwE:G:s&s_kwcid=AL!4422!3!507162072467!p!!g!!amazon%20aws!12563449882!121199905084&gclid=CjwKCAiAsIGrBhAAEiwAEzMlC-wKj8ERGv08OYQQI2ro-3Ztu7pGIzkfCI0tboGXMjCL4zQOd16OBhoCwFoQAvD_BwE&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all
8. https://chat.openai.com/
9. https://aws.amazon.com/cloudformation/
10. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#support-for-cooldown-delay
11. https://docs.aws.amazon.com/waf/latest/developerguide/cloudfront-features.html


—----------—----------—----------END----------—----------—----------—----------—----------