ECE547/CSC547 Cloud Architecture Project Fall 2023

Harish Rohan Kambhampaty (hkambha), Soham Niranjan Sawant (ssawant)

1. Introduction

This document presents the conceptualization and features of "CommunitySphere," a dynamic cloud application inspired by Reddit, with enhanced functionalities such as live user chat rooms and live streaming of audio and video within communities. CommunitySphere aims to provide users with an immersive online community experience, combining the freedom of user-generated content with robust community management tools and secure communication features. This document outlines the motivation behind the application, an executive summary of its key components, a description of the prevailing issues in the online community sphere, and comprehensive business and technical requirements for the successful development and deployment of CommunitySphere.

1.1. Motivation

CommunitySphere is driven by the growing demand for interactive and engaging online platforms that foster real-time communication and content sharing among community members. Recognizing the limitations of existing social networking platforms, CommunitySphere seeks to fill the void by providing a feature-rich environment that encourages live interactions, collaborative discussions, and multimedia content sharing, thereby fostering a stronger sense of community and connection among users.

1.2. Executive Summary

CommunitySphere is an innovative cloud-based application designed to facilitate the creation and management of user-generated communities. It offers an array of interactive features, including live user chat rooms and live streaming of audio and video content within communities. Building upon the framework of Reddit, CommunitySphere aims to provide users with a seamless and immersive online experience, promoting real-time engagement, content sharing, and community building through a secure and user-friendly interface.

2. Problem Description

2.1. The Problem

Existing social networking platforms often lack the functionality to facilitate live, real-time interactions and multimedia content sharing within communities. Users are often constrained by the static nature of the platforms, leading to limited engagement and reduced opportunities for meaningful interactions. Moreover, the absence of secure and integrated communication tools poses a significant challenge, impeding the establishment of a safe and vibrant online community environment.

2.2. Business Requirements

BR 1. Scalable Cloud Infrastructure

The ability of the application to efficiently accommodate increased user demand by dynamically adjusting computing resources and storage capacity within the cloud environment.

BR 2. Data Security & Compliance (Tenant Identification)

Ensuring the protection and confidentiality of user data through robust encryption and adherence to industry-specific data protection regulations and compliance standards.

BR 3. Cost-Effective Resource Management

Optimizing the allocation and utilization of cloud resources to minimize operational costs while maintaining optimal performance and efficiency within the cloud infrastructure.

BR 4. High-Availability

Ensuring uninterrupted access and minimal downtime for the application, enabling users to access the platform and its features consistently without service disruptions.

BR 5. High-Performance

Ensuring that the application operates efficiently, delivering quick response times and minimal latency, providing users with a seamless and responsive experience.

BR 6. High-Reliability

Establishing a dependable and resilient application that consistently operates without failures or errors, instilling user trust and confidence in the platform's stability and consistency.

BR 7. Disaster Recovery

Implementing comprehensive strategies and protocols to restore the application and its data in the event of a system failure or unforeseen disaster, ensuring minimal data loss and quick system recovery.

BR 8. Third-Party Login

Enabling users to log in to the application using their existing accounts from third-party platforms, providing a convenient and streamlined authentication process for users.

BR 9. Continuous Integration/Continuous Delivery and Deployment

Automating the software development and deployment processes to ensure frequent and reliable updates to the application and its resources, allowing for efficient delivery of new features and improvements without disrupting the user experience.

BR 10. Notifications & Alerts

Providing users with updates and notifications regarding community activities, events, and important announcements, enhancing user engagement and promoting active participation within the platform.

2.3. Technical Requirements

BR#	TR#s	Justification
1	3.1, 3.2, 3.3, 3.4	**Scalability** of the "cloud" requires defining the types of infrastructure and the types of cloud services we would need in the design to ensure the scalability requirement is addressed. TRs 3.1, 3.2, 3.3, and 3.4 clarify this information.
2	1.1, 1.2, 2.1, 2.2, 2.3, 2.4,	Data Security involves tenant identification (authentication), access control (authorization), and confidentiality. These requirements are specified explicitly in TRs 1.1 and 1.2. Compliance with standards involves monitoring through auditing logs at various levels (user actions and admin actions). These are defined by TRs 2.1, 2.2, 2.3, and 2.4.
3	2.1, 2.2, 2.3, 2.4, 6.1, 6.2	Cost-Effective Resource Management involves analyzing how resources are being utilized, which can be done through monitoring. TRs 2.1, 2.2, 2.3, and 2.4 elaborate what needs to be monitored for performing the analysis. Using the analysis, automated resource management needs to be performed to make the application cost-effective. These automations are defined in TRs 6.1 and 6.2.
4	5.1, 5.2, 5.3, 5.4, 6.1, 6.2	High-Availability implies minimization of service failure/disruption. The definition of how much the service can be disrupted is provided by TRs 5.1, 5.2, and 5.3. Recovery from failure/disruption should be automated, and these are further elaborated by TRs 6.1, and 6.2.
5	2.1, 2.2, 2.3, 2.4, 4.1, 4.2, 4.3, 4.4	Performance of an application is determined by how the application is operating. The expected operating ranges are defined in TRs 4.1, 4.2, 4.3, and 4.4. These conditions are gathered through monitoring the application. The metrics to be monitored are elaborated by TRs 2.1, 2.2, 2.3, and 2.4.
6	2.1, 2.2,	Reliability in an application means that the application is resistant to failure. TRs 5.3, and 5.4 define how resistant to

	2.3, 2.4, 5.3, 5.4	failure the application should be. Monitoring the application will enable us to identify the time and cause of failure to try and prevent them or to make recovery from failure smoother. This is elaborated in TRs 2.1, 2.2, 2.3, and 2.4.
7	2.1, 2.2, 2.3, 2.4, 5.1, 5.2, 5.3, 5.4	Disaster Recovery is planning for when failure occurs and operations have to be restored. TRs 5.1, 5.2, 5.3, and 5.4 define what should be recovered and how long can the system take to recover from failure. Monitoring the applications will help us gain insights into the nature of failures and the affected parts of the applications. TRs 2.1, 2.2, 2.3, and 2.4 elaborate on what should be monitored.
8	1.1	Third-Party Login enables users to use credentials of a third-party authentication authority. TR 1.1 provides more concrete details for this requirement.
9	6.1, 6.2	CI/CD ensures frequent and reliable delivery and deployment of updates to the application without disrupting user experience. TRs 6.1 and 6.2 describe what automation would be necessary to facilitate the CI/CD requirement.
10	2.3	Notifications & Alerts involve sending push, email, or SMS notifications to users regarding various activities. TR 2.3 describes this requirement.

TR 1. Security

Addressing *BR2 and BR8*, security requirements are:

TR 1.1. **User authentication and authorization** through a federated single-sign-on (SSO) providing Role-Based Access control to users

TR 1.2. **User Data Privacy and Confidentiality** through Data Encryption

TR 2. Monitoring

Addressing *BR2, BR3, BR5, BR6,* and *BR7*, monitoring requirements are:

TR 2.1. **Tracking performance metrics** from various microservices such as resource utilization

TR 2.2. **Tracking user activity** to detect unauthorized actions

TR 2.3. **Alerts and Notifications** on detecting an abnormality or breach, or user activity

TR 2.4. **Detecting Application/Resource Failure** and perform Disaster Recovery

TR 3. Scalability

Addressing *BR1*, scalability requirements are:

TR 3.1. **Auto-scaled compute resources** to dynamically adjust provisioned resources to optimize resource utilization

TR 3.2. **Load-Balancing** to evenly distribute traffic across compute resources

TR 3.3. **Content Delivery Networks** (CDNs) to cache static content

TR 3.4. **Scalable Storage** for the following data:

- **10-20 MB of Object Storage** for static web hosted content with CDN hosting
- **5-10GB of NoSQL Data Store** for User Identity and Profile Data (assuming 1kB per user and 5-10 million users exist)
- **100 TB of Object Storage** for User Generated Content (assuming 1,000,000 posts each 100MB in size on average)
- **10-20 GB of Object Storage** for log data

TR 4. Performance

Addressing *BR5*, performance requirements are:

TR 4.1. **Max Response times**: 10 secs

TR 4.2. **Availability**: 99.95%

TR 4.3. **Throughput**: 20 Gbps

TR 4.4. **Error rate**: Less than 0.1%

TR 5. Disaster Recovery & Availability

Addressing *BR4*, *BR6 and* *BR7*, disaster recovery requirements are:

TR 5.1. **Recovery Point Objective** (RPO): 5 mins

TR 5.2. **Recovery Time Objective** (RTO): 2 mins

TR 5.3. **Data backup**: all data critical to restoration of operations

TR 5.4. **Standby Resources**: to quickly switch over when failure occurs

TR 6. Automation

Addressing *BR3*, *BR4*, and *BR9*, automation requirements are:

TR 6.1. **Infrastructure-as-Code** (IaC) for Automated Resource Deployment

TR 6.2. **Auto-Scaling** to provision more resources when demand increases and release when demand falls

2.4. Trade Offs

2.4.1. Scalability vs. Security:

- **Scalability Requirement (TR3.1)**: Auto-scaled compute resources to dynamically adjust provisioned resources to optimize resource utilization.
- **Security Requirement (TR1.1)**: User authentication and authorization through a federated single-sign-on (SSO) providing Role-Based Access Control to users.

Conflict: While auto-scaling is crucial for adapting to varying workloads and optimizing resource utilization, federated SSO with Role-Based Access Control often involves additional authentication steps that could potentially impact the seamless scalability of resources.

2.4.2. Performance vs. Scalable Storage:

- **Performance Requirement (TR4.3)**: Throughput - Download 100 posts/sec, Upload 50 posts/sec.
- **Scalability Requirement (TR3.4)**: Scalable Storage for various data types, including 100 TB of Object Storage for User-Generated Content.

Conflict: The high throughput requirement implies quick data access and retrieval, which may conflict with the need for scalable storage, especially for user-generated content that can accumulate rapidly. Achieving high throughput might require careful management of data storage to maintain performance standards.

3. Provider Selection

3.1. Criteria for choosing a provider

1. Security (TR1) And Monitoring (TR2): User authentication, Data privacy, Tracking user activity, performance metrics, real time updates
2. Performance(TR4): Max Response Time, Availability, error rate
3. Cost Efficiency: cost analysis,
4. Scalability (TR3) And Automation (TR6)
5. Disaster Recovery (TR5)

3.2. Provider Comparison

Feature	AWS	GCP	Azure
Security And Monitoring			
User authentication	Amazon Cognito, IAM	Google Identity Platform, Firebase Authentication	Azure Active Directory, Azure AD B2C
Data privacy	Amazon S3 encryption, AWS KMS, AWS Secrets Manager	Google Cloud DLP, Google Cloud IAM, Google Cloud Data Loss Prevention	Azure Information Protection, Azure Sentinel
Tracking user activity	Amazon CloudWatch, Amazon Kinesis	Google Analytics 360, Google Cloud Dataproc	Azure Monitor, Azure Sentinel
Performance metrics	Amazon CloudWatch Logs, Amazon CloudWatch Metrics	Google Cloud Monitoring, Google Cloud Logging	Azure Monitor, Azure Log Analytics
Real-time updates	Amazon SNS, Amazon SQS	Google Cloud Pub/Sub	Azure Event Hubs, Azure IoT Hub
Performance and Speed			
Max Response Time	Amazon API Gateway,	Google Cloud CDN, Google Cloud Load Balancing	Azure CDN, Azure Front Door
Availability	AWS Elastic Beanstalk, AWS CloudFormation	Google App Engine, Google Kubernetes Engine	Azure Load Balancer, Azure Site Recovery
Error rate	AWS CloudWatch Alarms, AWS CloudWatch Events	Google Cloud Monitoring Alerts, Google Cloud Logging Errors	Azure Monitor, Azure Log Analytics
Cost Efficiency			
CDN / Global distribution	Amazon CloudFront, Amazon Global Accelerator	Google Cloud CDN, Google Cloud Load Balancing	Azure CDN

Cost distribution analysis	AWS Cost Explorer, Amazon CloudWatch Metrics	Google Cloud Billing, Google Cloud Cost Management	Azure Cost Management, Azure Reservations
Scalability And Automation			
Scalability	Amazon EC2, AWS Lambda,Amazon S3, Amazon S3 Glacier.	Google Compute Engine, Google Cloud Functions	Azure Auto-Scaling, Azure Cosmos DB
Automation	AWS CloudFormation, AWS CodePipeline	Google Cloud Deployment Manager, Google Cloud Build	Azure Automation, Azure Logic Apps
Disaster Recovery			
Disaster Recovery	AWS Backup, AWS Disaster Recovery	Google Cloud Backup and Restore, Google Cloud Disaster Recovery	Azure Site Recovery, Azure Backup

For cost comparison we are using the following parameters:

Parameters	**Values**
No of Users	7.5 million
CDN Hosting (web hosted content)	10-20 MB
User Identity and Profile Data	5-10 GB
User Generated Content	100 TB
Log Data	10-20 GB

Cost Estimate For AWS:

Service	**Estimated Monthly Cost**
S3	$4,000 - $8,000
CloudFront	$2,000 - $4,000
CloudTrail	$75 - $150
API Gateway	$250 - $500
Route 53	$75 - $150
AWS Lambda	$25 - $50
Amazon RDS	$600 - $1,200
Amazon DynamoDB	$750 - $1,500
Amazon Cognito	$75 - $150
Amazon SNS	$75 - $150
Amazon CloudWatch	$75 - $150
Amazon CloudWatch Logs	$75 - $150
Amazon VPC	$75 - $150
Total:	**$14,725 - $32,350**

AWS requires using 13 services at least for CommunitySphere. Operational costs would be roughly $20,000+ for 7.5 million users. When we look at Azure and GCP we will get similar estimates.

1. **Azure**: Azure: Azure offers the **performance** and **scalability** for CommunitySphere. It has a large global network and a wide range of compute options, including virtual machines, containers, and serverless computing. Azure also has a strong suite of security and monitoring features, including Azure Active Directory, Azure Sentinel, and Azure Monitor.

2. **AWS**: AWS is a good choice for CommunitySphere if you need a **high degree of flexibilit**y and **customization**. AWS has a wide range of services, including compute, storage, networking, database, and analytics. AWS also has a strong ecosystem of third-party tools and services. However, AWS can be **more expensive** than Azure, and it can be **more complex** to manage.

3. **GCP**: GCP is a good choice for CommunitySphere if you are looking for a **cost-effective** solution. GCP is generally less expensive than AWS and Azure, and it has a number of features that can help you save money, such as its preemptible virtual machines and its committed use discounts. However, GCP may **not be as performant** as Azure or as flexible as AWS

3.3. Final Selection

Upon comparison AWS is winner because of the following reasons:

- **Maturity and Breadth**: AWS is the most mature and widely adopted cloud provider, offering a comprehensive range of services, including compute, storage, networking, databases, and analytics.

- **Pricing**: AWS is generally considered to be the most cost-effective cloud provider, offering a variety of pricing options to suit different needs.

- **Ecosystem**: AWS has a vast ecosystem of partners and third-party tools, making it easy to find solutions for specific use cases

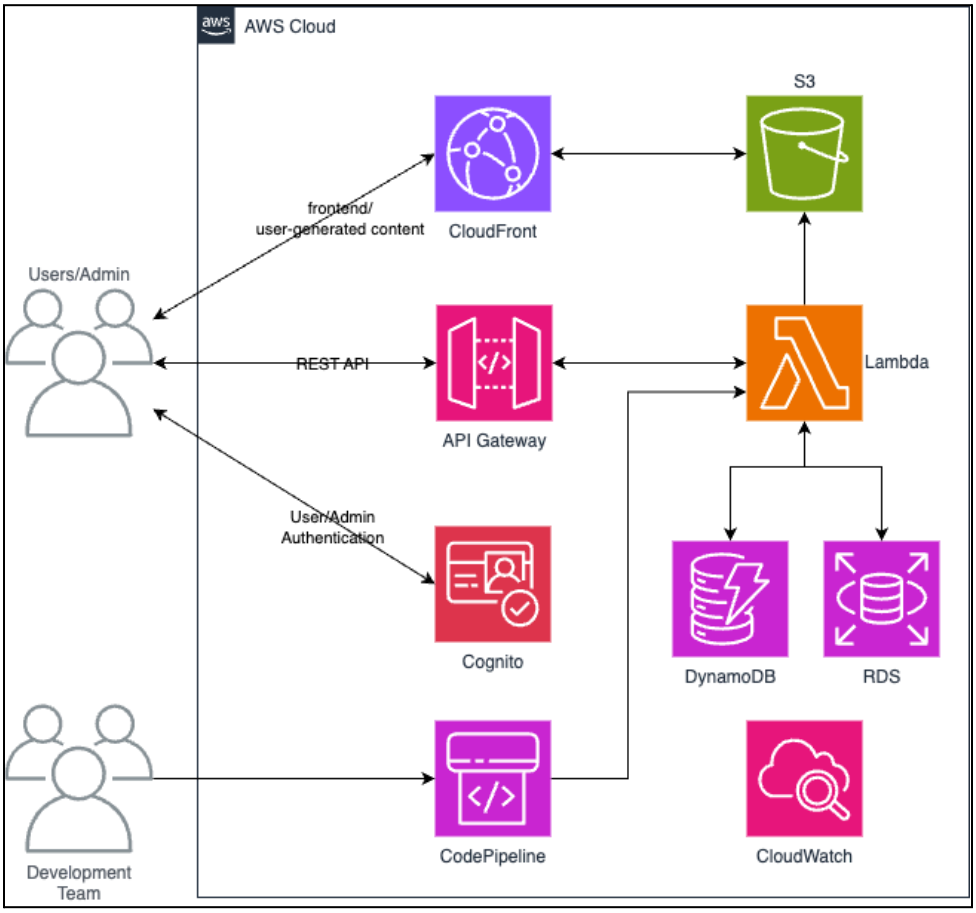
3.3.2. The list of services offered by the winner

- **Lambda** is a serverless compute service that lets you run code without provisioning or managing servers. It executes code in response to events, such as changes to data in Amazon S3 or HTTP requests.

- **Amazon S3** (Simple Storage Service) is a scalable, object storage service for the cloud. It provides secure, durable, and inexpensive storage for a wide range of data, including static websites, images, videos, and backups.
- **Amazon S3 Glacier** is a low-cost object storage service designed for long-term data archiving. It offers secure, durable storage for infrequently accessed data at a lower cost than Amazon S3.
- **Amazon RDS** (Relational Database Service) is a managed relational database service that makes it easy to set up, operate, and scale a relational database in the cloud. It supports a variety of database engines, including MySQL, PostgreSQL, Oracle Database, and SQL Server.
- **Amazon CloudFront** is a content delivery network (CDN) service that accelerates the delivery of static and dynamic web content. It caches content at edge locations around the world, ensuring low latency and high availability for users worldwide.
- **Amazon Route 53** is a cloud-based DNS service that provides domain name registration, traffic management, and health checks. It enables you to route traffic to your websites and applications with high availability and low latency.
- **Amazon CloudTrail** is a cloud service that records API calls made to AWS services. It provides a history of your AWS account activity, enabling you to track changes, troubleshoot issues, and comply with regulatory requirements.
- **Amazon Web Application Firewall (WAF)** is a web application firewall that protects your web applications from common web vulnerabilities, such as SQL injection and cross-site scripting (XSS).
- **Amazon DynamoDB** is a fully managed NoSQL database service that provides fast and scalable storage for a wide range of applications. It is a key-value and document database that offers high availability, low latency, and pay-per-request pricing.
- **Amazon Cognito** is a user identity and access management service that provides secure authentication, authorization, and management of users and groups for web and mobile applications. It simplifies the process of managing user identities, eliminating the need to build and manage custom authentication and authorization systems.
- **Amazon CloudWatch** is a monitoring and observability service that provides detailed monitoring and logging for AWS resources and applications. It collects metrics, logs, and events from various AWS services, enabling you to visualize and analyze data to identify trends, troubleshoot issues, and optimize resource utilization.

- **Amazon Simple Notification Service (SNS)** is a highly scalable, fully managed messaging service that enables you to decouple and scale microservices, distributed systems, and web applications. It provides a reliable and cost-effective way to send notifications to clients, such as mobile devices, email addresses, or other SNS topics.
- **Amazon CodePipeline** is a continuous delivery service that automates the process of building, testing, and deploying your applications. It allows you to define pipelines that specify the steps required to build, test, and deploy your code, ensuring consistent and reliable deployments.
- **Amazon CodeCommit** is a fully managed Git repository hosting service that makes it easy to store, manage, and collaborate on Git repositories. It provides a secure and scalable repository solution for your code.
- **Amazon CodeBuild** is a fully managed build service that lets you build and test your code without provisioning or managing servers. It supports various build environments, including Linux, macOS, and Windows, and integrates with CodePipeline to automate your build process.
- **Amazon CodeDeploy** is a deployment service that automates the process of deploying applications to a variety of compute platforms, including Amazon EC2, AWS Lambda, and Amazon Elastic Beanstalk. It provides a consistent and reliable way to deploy your code to production environments.
- **Amazon CloudFormation** is a declarative cloud infrastructure management service that lets you define and manage AWS resources in a template file. It provides a consistent and repeatable way to provision and manage your cloud infrastructure, ensuring that your environment is always in the desired state.

4. The first draft



4.1. The basic building blocks of the design

- Compute
  - Lambda
- Storage
  - Simple Storage Service (S3)
  - Simple Storage Service (S3) Glacier
- Networking
  - Route53
  - CloudFront
- Database
  - DynamoDB
  - Relational Data Store (RDS)
- Identity and Access Management
  - Cognito
  - WAF
  - Shield

- Monitoring
    - CloudWatch
    - CloudTrail
    - AWS SNS
- DevOps
    - CodePipeline
    - CodeCommit
    - CodeBuild
    - CodeDeploy
    - CloudFormation

4.2. Top-level, informal validation of the design

To fulfill the crucial technical requirement of Security TR 1.1, our solution employs Amazon Cognito, a comprehensive authentication and access control service. Amazon Cognito User Pools manage user authentication, while Cognito Identity Pools provide single-sign-on access to AWS resources.Additionally, the incorporation of WAF and AWS Shield will satisfy the TR 1.2 by safeguarding data.

Our design complies with the mandatory technical requirement for monitoring TR 2.1 and TR 2.2 by leveraging AWS services such as AWS CloudWatch and CloudTrail. To ensure the proposed 99.95% availability TR 4.2 and Max Response Time TR 4.1, CloudWatch will continuously monitor application and infrastructure performance within the AWS environment. We propose monitoring bandwidth usage, CPU utilization, and traffic parameters, establishing alerts and alarms based on predefined thresholds. Additionally, AWS SSN will provide real-time notifications in the event of a security breach, enabling prompt response and mitigation efforts thus satisfying TR 2.3 and TR 2.4.

Route 53 plays a crucial role in managing website traffic. It acts as a DNS resolver, translating website domain names into IP addresses, ensuring that visitors are directed to the correct website location. This service is highly scalable, capable of handling massive amounts of traffic and enabling effective load balancing based on geographical location and latency. Route 53 utilizes geolocation and routing policies to intelligently distribute traffic across multiple servers, ensuring optimal performance and availability for users worldwide. This satisfies our TR 3.1, TR 3.2 , TR 3.3 and TR 4.3

Amazon S3 will be used to store static web content,user data and user log. S3 is a highly scalable and reliable object storage service. Amazon DynamoDB will be used to store user identity and profile data. DynamoDB is a highly scalable and reliable NoSQL database service. This satisfies our scalable storage TR 3.4. Also, the S3 bucket will grow in storage size automatically as the number of

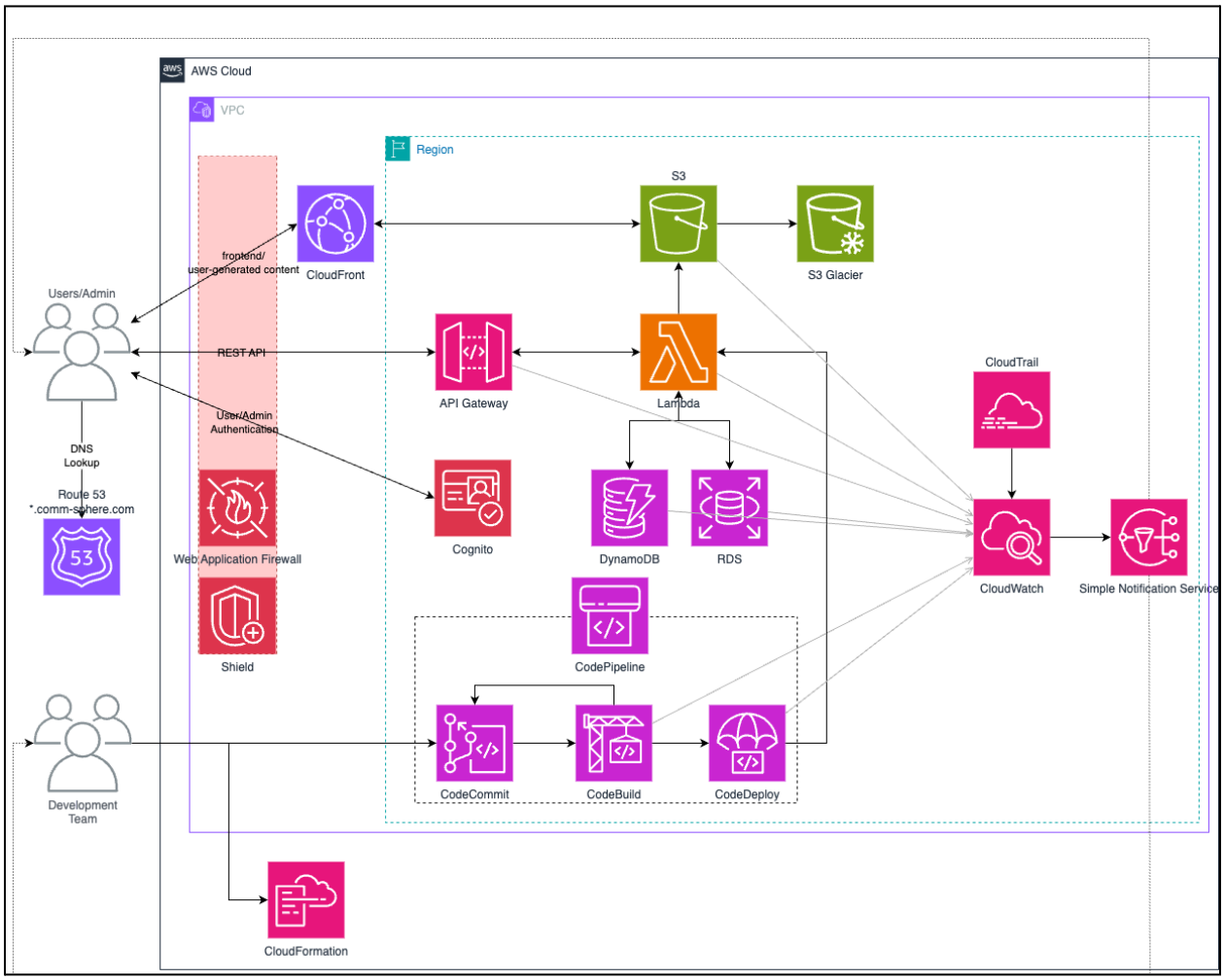
objects within the s3 bucket increase. Thus meeting Auto-scalable TR 3.1 and availability TR 6 Technical Requirements.

AWS Cloudfront will be used to cache static content hence satisfy TR 3.3. AWS CloudFormation is a service for modeling and provisioning AWS and third-party resources in an automated and secure way. It allows you to define and provision AWS infrastructure resources using a JSON or YAML formatted Infrastructure as Code (IaC) template hence satisfying TR 6.1

4.3. Action items and rough timeline

SKIPPED

5. The second design



5.1. Use of the Well-Architected framework

The AWS framework is based on six pillars:

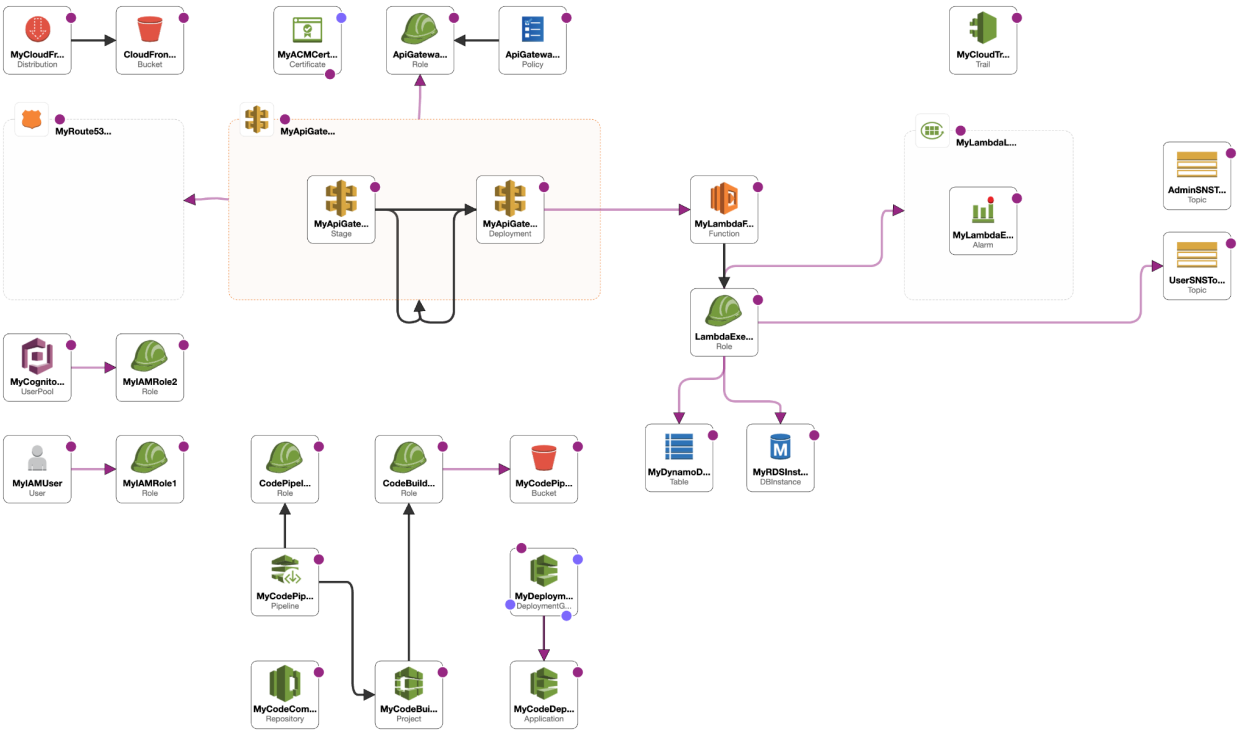
- Operational Excellence
- Security
- Reliability
- Performance Efficiency
- Cost Optimization
- Sustainability

5.2. Discussion of pillars

As the application workload increases, the system should auto-scale efficiently to maintain optimal performance and meet fluctuating demands. This aligns with the AWS Well-Architected Framework pillar of **Performance Efficiency**. Performance efficiency encompasses the efficient utilization of computing resources to fulfill system requirements, while adapting to changing demands and technological advancements. Low application latency at a lower cost, resulting in a superior user experience, is another measure of Performance Efficiency. Our design achieves Performance Efficiency through the implementation of AWS CloudFront, AWS Lambda, AWS Route 53 and an S3 bucket for scalability.

To ensure **Security** such as user authentication, authorization, and data privacy, our system employs a combination of encryption and access control mechanisms. Both data at rest and data in transit are encrypted using authenticated tokens. Our solution leverages IAM (Identity and Access Management) with Cognito consumer and identity pools to achieve tenant identification and security. Additionally, network security is safeguarded by AWS Shield and AWS WAF (Web Application Firewall).

5.3. Use of Cloudformation diagrams



The above CloudFormation diagram for the CommunitySphere application in AWS depicts a **well-architected and scalable infrastructure** leveraging various AWS services. At its core, AWS Lambda functions serve as the backbone for **serverless compute**, facilitating efficient and event-driven execution of specific tasks. Data storage is managed through a combination of Amazon S3 for **general-purpose object storage**, S3 Glacier for cost-effective **archival of infrequently accessed data**, and DynamoDB as a **NoSQL database** for fast and scalable access to structured data. Additionally, RDS is utilized for **relational data storage**, providing a managed database service for structured data requirements. The application's content delivery is optimized through the integration of CloudFront, an AWS **Content Delivery Network** (CDN), and Route 53, AWS's highly available and scalable domain name system (DNS) web service, enhancing global accessibility and performance.

Security measures are robustly implemented through AWS Cognito for **user authentication**, Web Application Firewall (WAF) for **protecting against common web exploits**, and AWS Shield for **advanced Distributed Denial of Service (DDoS) protection**. **Monitoring and logging** are addressed through CloudWatch, ensuring real-time visibility into application performance and resource utilization, while CloudTrail offers **comprehensive auditing of API calls and account activity**. AWS SNS handles **notifications**, allowing the application to proactively alert stakeholders about critical events. The DevOps

pipeline, orchestrated by AWS CodePipeline, seamlessly integrates with CodeCommit for version control, CodeBuild for **continuous integration**, and CodeDeploy for **automated application deployment**, providing an **end-to-end solution for development and release management**. Lastly, the entire infrastructure is provisioned and managed through AWS CloudFormation, enabling the creation and updating of AWS resources in a consistent and automated manner, promoting **infrastructure as code** and ensuring reproducibility across environments.

5.4. Validation of the design

5.4.1. Our solution prioritizes **user authentication** to achieve **high security** and **data integrity** TR 1 by employing **Amazon Cognito** for user and administrator authorization and authentication and **Amazon WAF** and **Amazon Shield** for data privacy within the application.

5.4.2. We fulfill the technical requirement TR 2, TR 4 and TR 5 by using services like **AWS Cloudtrail, AWS Cloudwatch and AWS SNS** that's system monitoring. AWS CloudWatch continuously monitors system health and performance metrics, providing real-time insights into resource utilization, **application performance**, and **infrastructure stability**. This enables us to identify potential issues early on and take corrective actions before they impact user experience or disrupt business operations. AWS CloudTrail, on the other hand, acts as an audit trail for all AWS account activity, logging every API call and user action. This data is invaluable for troubleshooting, **security analysis**, and compliance auditing. By analyzing CloudTrail logs, we can **detect unauthorized access, identify anomalous activity,** and maintain a comprehensive record of system events. **Amazon SNS** gives us **real time notifications** about the system By employing these powerful AWS services, our solution ensures that the system remains **highly performant**, **available**, and also easily **recovered i**n case of failure hence meeting the technical requirements.

5.4.3. To achieve **high performance** TR 4 , our solution utilizes **Amazon Route 53**, a highly scalable DNS service, for **load balancing** TR 3.2 based on geographical topology and latency. This intelligent routing mechanism ensures optimal performance for users worldwide.

5.4.4. Our system's **auto-scalability** TR 3.1, TR 6 and **high performance** TR4 are further enhanced by the combination of an **S3 bucket** and **AWS Lambda** The S3 bucket provides auto scalable storage for static content, while AWS Lambda handles dynamic requests efficiently. Additionally, robust encryption safeguards sensitive data at rest and in transit.

5.4.5. For **scalable storage** TR 3.4, our solution employs RDS, DynamoDB for NoSQL data, and S3. This combination caters to diverse storage requirements while ensuring scalability and performance

5.4.6. By employing **CloudFormation**, it simplifies AWS resource deployment, allowing infrastructure to be defined and managed as code, streamlining provisioning, updates, and deletions for consistent and scalable cloud environments hence fulfilling TR 6.1 and also a content delivery network **(CDN)** TR 3.3 **Cloudfront** that delivers web content with low latency and high availability worldwide.

5.5. Design principles and best practices used

The following AWS-specific design principles and best practices were used:

5.5.1. **Scalability**: The use of S3 buckets and CloudFront distribution ensures that the application can scale to meet any demand.

5.5.2. **Operational excellence**: The use of CloudTrail, CloudWatch, and Simple Notification Service (SNS) enables proactive monitoring and auto-remediation, which helps to reduce latency and improve reliability.

5.5.3. **Security**: The use of IAM, Cognito,WAF and AWS Shield ensures that the application is secure and that data is protected.

5.5.4. **Reliability**: The use of CodePipeline, CodeBuild, and CodeDeploy enables continuous integration and continuous delivery (CI/CD), which helps to improve the reliability of the application.

5.5.5. **Performance efficiency**: The use of a serverless architecture helps to improve performance and reduce costs and deploying in multiple AWS regions.

5.6. Tradeoffs revisited

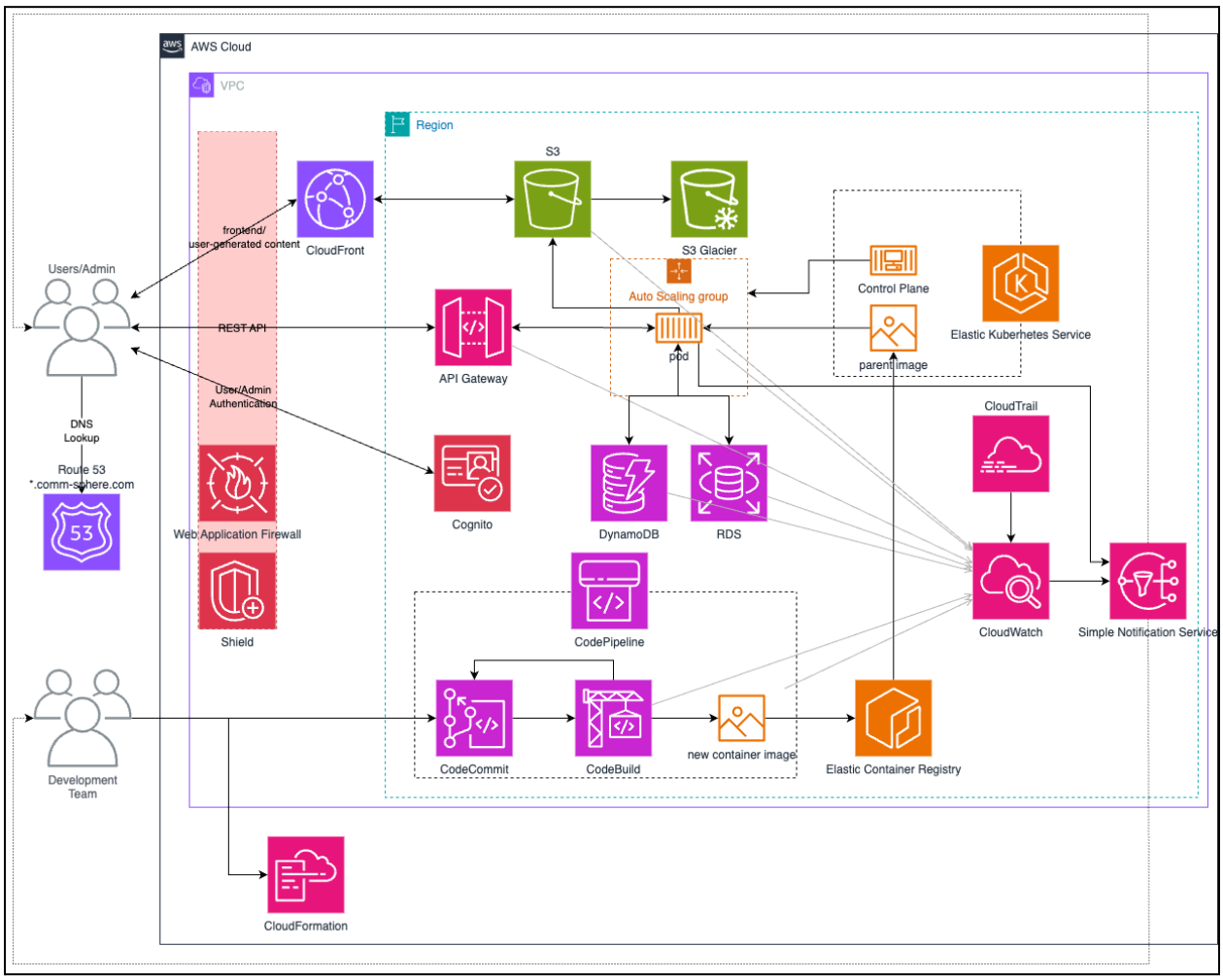
5.6.1. Scalability vs. Security

- **Decision**: Prioritized Scalability
- **Reasoning**: Prioritizing scalability over security in a Community Sphere application could be justified, as during periods of rapid growth or when dealing with an overwhelming influx of users. In these scenarios, the ability to handle the increased demand without significant downtime or performance degradation may take precedence over implementing the most stringent security measures.

5.6.2. Performance Vs Scalable Storage

- **Decision**: Prioritized Scalability
- **Reason**: Community Sphere, Reddit-like applications handle massive amounts of user-generated content, requiring extensive storage capacity. Scalable storage solutions like object storage are preferred over high-performance storage options like SAN due to their ability to seamlessly scale with increasing data volumes. These solutions offer cost-effectiveness, flexibility, and reliability, making them ideal for managing vast amounts of unstructured data.

5.7. Discussion of an alternate design



The decision against opting for Amazon Elastic Kubernetes Service (EKS) and Amazon Elastic Container Registry (ECR) over AWS Lambda for the CommunitySphere application stems from the application's event-driven, lightweight nature. Lambda's serverless model, offering automatic scaling and

reduced infrastructure management, aligns well with the sporadic task execution in CommunitySphere. This approach simplifies operational overhead, allowing the development team to prioritize application logic. Additionally, Lambda's pay-as-you-go pricing, based on actual code execution, is cost-effective for unpredictable workloads. While EKS and ECR offer container orchestration capabilities, the simpler and more cost-efficient serverless architecture of Lambda better suits the CommunitySphere application's requirements, avoiding unnecessary complexity in infrastructure management.

6. Kubernetes experimentation

All code related to this experiment is available here [10] on GitHub and the container image is available here [11] on Docker Hub.

6.1. Experiment Design

The experiment will involve deploying the application with a specified number of replicas and dynamically adjusting the number of replicas using Horizontal Pod Autoscaling (HPA).

- **Objective**: Dynamic Scaling for Responsiveness
- **Design Consideration**: Utilize Kubernetes Horizontal Pod Autoscaling (HPA) to automatically adjust the number of application replicas based on CPU utilization, ensuring responsiveness during varying workloads.
- **Kubernetes Experiment**: We'll simulate a pod failure and observe Kubernetes automatically recovering by deploying a new pod.

The following code is the **minikube deployment configuration file**:

```
# mk-depl.yml
# minikube deployment configuration .yml file
apiVersion: apps/v1
kind: Deployment
metadata:
 name: community-sphere
spec:
 replicas: 1  # Initial number of replicas
 selector:
   matchLabels:
     app: community-sphere
 template:
   metadata:
     labels:
       app: community-sphere
```

```
   spec:
     containers:
       - name: community-sphere-container
         image:
docker.io/untitlederror09/community-sphere-image
         resources:
           requests:
             cpu: 100m
         ports:
           - containerPort: 80
     imagePullSecrets:
       - name: regcred
```

The following code is the **minikube HPA configuration file**:

```
# mk-hpa.yml
# minikube HPA configuration file
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: community-sphere-hpa
spec:
 scaleTargetRef:
   apiVersion: apps/v1
   kind: Deployment
   name: community-sphere
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Resource
   resource:
     name: cpu
     target:
       type: Utilization
       averageUtilization: 50
```

The following code is the **web-server file**:

```
# app.py
# web-server
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    i = 0
    while i < 99999999:
        i+=1
    return 'Welcome to CommunitySphere!'
```

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=80)
```

6.2. Workload generation with Locust

Locust was used to simulate user load on the CommunitySphere application. For simulating computational delay, a counting program was used.

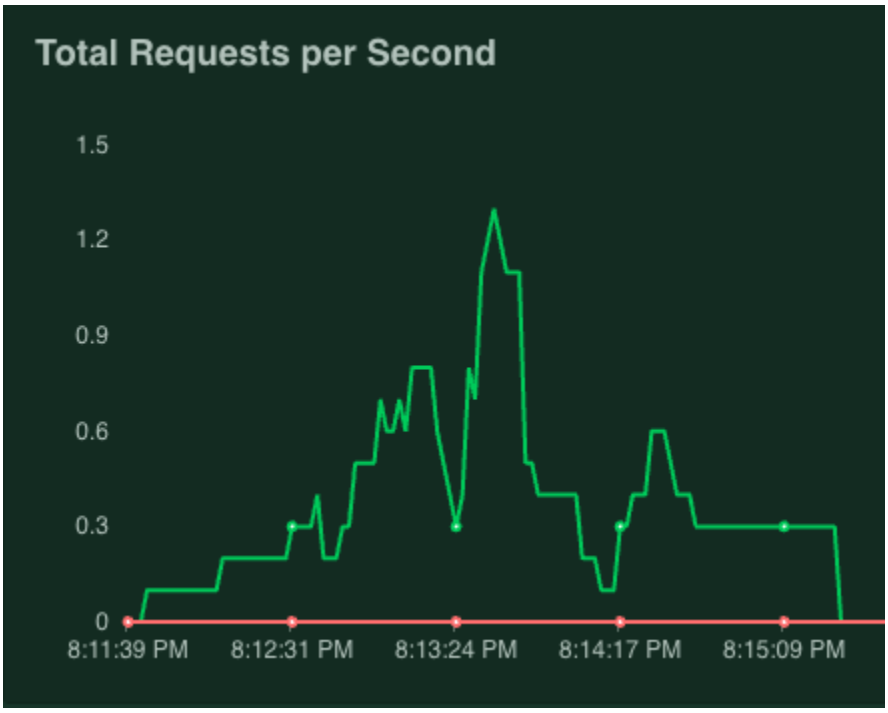
The following code shows the **locust configuration**:

```
# locustfile.py
# locust configuration file
from locust import HttpUser, task, between


class CommunitySphereUser(HttpUser):
    wait_time = between(5, 15)

    @task
    def visit_home_page(self):
        self.client.get("/")
```

The test includes an increasing load phase and a decreasing load phase. The test characteristics are:

- Peak Users: 100
- Spawn/Kill Rate (per minute): 5

Results of load testing:



The above image shows the **total requests per second** generated by the above defined load profile.

The following table contains the **HPA event log**:

Type	Reason	Age	From	Message
Normal	Successful Rescale	51m	horizontal-pod-autoscaler	New size: **1**; reason: All metrics below target
Normal	Successful Rescale	24m	horizontal-pod-autoscaler	New size: **2**; reason: All metrics below target
Normal	Successful Rescale	21m	horizontal-pod-autoscaler	New size: **4**; reason: All metrics below target
Normal	Successful Rescale	18m	horizontal-pod-autoscaler	New size: **8**; reason: All metrics below target
Normal	Successful Rescale	16m	horizontal-pod-autoscaler	New size: **10**; reason: All metrics below target
Normal	Successful Rescale	10m	horizontal-pod-autoscaler	New size: **3**; reason: All metrics below target
Normal	Successful Rescale	9m	horizontal-pod-autoscaler	New size: **1**; reason: All metrics below target

6.3. Analysis of the results

The provided events log indicates successful rescaling events triggered by the Horizontal Pod Autoscaler (HPA) in a Kubernetes cluster, validating the scalability features of Kubernetes.

The event log shows an **initial gradual increase** in number of replicas (from 1 to 10) due to the increasing load, followed by limiting replicas to maximum threshold, finally a **gradual decrease** in the number of replicas (from 10 to 1) due to decreasing load.

In summary, the observed events showcase the dynamic and responsive nature of Kubernetes scalability through the Horizontal Pod Autoscaler. The autoscaler effectively adjusts the number of pods in the deployment based on observed metrics, ensuring optimal resource utilization and responsiveness to fluctuating workloads. This validation demonstrates Kubernetes' ability to efficiently scale applications in response to varying demand, a key feature for maintaining performance and reliability in dynamic environments.

7. Ansible playbooks

SKIPPED

8. Demonstration

SKIPPED

9. Comparisons

SKIPPED

10. Conclusion

10.1. The lessons learned

The cloud computing project provided a valuable learning experience in mastering the fundamental building blocks of cloud computing. We gained insights into defining business requirements and corresponding technical requirements aligned with the key pillars of cloud computing. This exercise also introduced us to AWS Well Architected Framework, enabling us to apply its design principles effectively. Additionally, we developed an understanding of balancing trade-offs while prioritizing customer satisfaction. A significant takeaway was the process of selecting cloud providers based on a comprehensive evaluation of their services and operations, with cost being a crucial factor. Overall, despite our limited prior knowledge in cloud computing, this project proved to be an enriching learning experience, positioning us to confidently work on cloud-based applications.

10.2. Possible continuation of the project

SKIPPED

References:

[1] https://aws.amazon.com/route53/

[2] https://docs.aws.amazon.com/cloudwatch/?icmpid=docs homepage mgmtgov

[3] https://docs.aws.amazon.com/apigateway/?icmpid=docs homepage serverless

[4] https://docs.aws.amazon.com/waf/?icmpid=docs homepage security

[5] https://aws.amazon.com/cloudfront/

[6] https://docs.aws.amazon.com/wellarchitected/latest/framework/security.html

[7] https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.security.en.html

[8] https://bard.google.com/

[9] https://aws.amazon.com/architecture/

[10] https://github.com/UntitledError-09/CommunitySphere

[11] https://hub.docker.com/repository/docker/untitlederror09/community-sphere-image/general