ECE547/CSC547 Cloud Architecture

Bhavesh Ittadwar & Shardul Khare
buittadw, srkhare

Contents

1. Introduction

1.1 Motivation

As avid gamers who've spent our teenage years immersed in the world of video game streaming, our project is a reflection of our deep passion for this medium. We aim to leverage cloud computing technologies to enhance the streaming experience, connecting viewers and content creators in new and exciting ways. Our motivation stems from the belief that technology can elevate the gaming and streaming community to new heights.

1.2 Executive Summary

Firstly, it aims to connect people worldwide, fostering cultural exchange and inclusivity. Empowering content creators is central, as the platform offers high-quality streaming capabilities that enable streamers to share their skills and creativity with a global audience. The real-time chat feature is a pivotal aspect, enhancing the viewer experience by promoting interactivity and connection. The goal is to create a holistic ecosystem where streamers can engage with their viewers, thereby fostering audience growth. In essence, this project is driven by a vision of global connectivity, empowerment, and immersive content consumption, all facilitated through cutting-edge technology.

2. Problem Description

2.1 The Problem

Imagine a scenario where a startup intends to utilize apt Cloud Provider's infrastructure and services to develop a new web platform for creating a seamless live streaming experience as a PAAS. This platform will empower content creators to stream their playable video content in various playback qualities and enable viewers to engage with streamers through a live chat system. What are the essential considerations that a company must address before deploying a multi-faceted and large-scale project like this?

2.2 Business Requirements

BR1: **Reliability & Performance**: Attain 24x7 availability
BR2: **Performance:** Efficient auto-scaling
BR3: **Performance:** Attain minimal delay in response
BR4: **Cost Optimization:** Analyze Expenditure
BR5: **Performance**: Reduce content load times
BR6: **Operational Excellence:**Enhance robust authorization and authentication
BR7: **Reliability:** Minimize downtime due to unforeseen accidents and make it fault tolerant
BR8: **Operational Excellence:** Implement tenant identification, to provide apt services to customers based on their roles and subscription tiers.
BR9: **Operational Excellence:** Release processes should be highly automated
BR10: **Operational Excellence:** Make creator's live content accessible to a large audience at the same time in high quality
BR11: **Reliability:** Enhance intellectual property protection through automated compliance, content tracking, and real-time monitoring
BR12: **Operational Excellence:** Implement Scalable Storage Solutions
BR13: **Security:** Implement reliable testing environment
BR14: **Reliability:** Define clear SLAs(Service Level Agreements) for legal risk mitigation
BR15: **Reliability:** Choose the most suitable cloud resources for needs and establish a routine for regular monitoring
BR16: **Monitoring:** Monitoring view times, channel statistics, live viewers etc

BR17: Operational Excellence: Building a live chat functionality and managing a multitude of messages coming in at the same time and reflecting on both server and client side

BR18: Cost Optimization: Minimize Cost

BR19: Sustainability : Cater region specific resources and services to locations based on user activity and load to minimize idle time & unnecessary energy consumption

BR20: Security: Proactively identify and respond to security threats

BR21: Operational Excellence: Implementing payment features, empowering streamers to effortlessly monetize content through user donations and optimizing platform efficiency

BR22: Reliability: Content Moderation

2.3 Technical Requirements

TR1.1: Implement Redundant Systems - Implement backup duplicate systems that can be used if the currently active systems fail to ensure 99.99% availability. Necessary because if the system fails the backup system will take over and keep the service available 99.99% of the time.

TR1.2: Distribute Load, strategize a way to distribute load evenly across several servers to stay safe from server overloads. It is necessary because not having any downtime would result in 99.99% availability.

TR1.3: Deploy duplicates of the application in different locations.

Together all **TR1.1, TR1.2** & **TR1.3** ensure continuous seamless operation even if some part fails(redundancy) and distribute the workload evenly(load balancing), ensuring uninterrupted experience and service and close to 24x7 availability(as it is not realistically possible).

TR2.1: Implement efficient auto-scaling to automatically adjust computing resources according to real-time demand. It’s necessary because doing so will enable efficient use of resources and minimal idle time. At the same time preventing under or over provisioning, leading to reduced costs.

TR2.2: Set robust monitoring systems to continuously track performance metrics and trigger alerts when predefined thresholds are exceeded. Necessary to proactively gauge the status of the system and ensure close to instant, if not instant, scaling. This makes the scaling up/down more efficient.

Together **TR2.1** & **TR2.2** i.e. dynamic auto scaling and robust performance and metrics monitoring ensure that the resources are being used efficiently by adjusting the resources allocated dynamically as the demand fluctuates. Monitoring enables quicker and more efficient reaction to changes.

TR3.1, TR5.1: Achieve maximum latency of 0.5s. Necessary as it reduces physical data travel directly reducing latency by ensuring faster response times for the users.

TR3.2: Implement an efficient server distribution strategy as well to minimize network latency. Necessary as it reduces the response time for a request as it is received from a nearby server.

Together, a **TR3.1:** CDN and **TR3.2:** distributed servers work synergistically to minimize response delays. CDN reduces distance related delays while distributed servers reduce network latency.

TR4.1: Implement a system to track and categorize expenses in real time. It's necessary to get and organize spending data to understand where funds are allocated and mitigate any superfluous expenditure.

TR4.2: Build comprehensive cost and usage reports leveraging an expense monitoring service. It should provide detailed insights into spending patterns, cost trends, and usage metrics. This tool should allow for customizable, detailed analysis of expenditure data.

TR6.1: Implement MFA(Multi-factor authentication) to enhance authentication by requiring users to provide two or more forms of verification before granting access. This is necessary as it adds an extra layer of security beyond just passwords, significantly reducing the risk of unauthorized access in the cloud environment.

TR6.2: Utilize RBAC(Role-based access control) to enforce robust authorization by defining and managing user permissions based on their roles within the organization. This is essential as it ensures that users have access only to the resources and functionalities required for their specific roles, minimizing the risk of unauthorized actions.

TR7.1: Deploy a high-availability and disaster recovery (HA/DR) strategy to reduce downtime from unexpected incidents. Analyze and deduce "Recovery Time Objective (RTO) & Recovery Point Objective (RPO)"[1].
Implementing an HA/DR strategy is crucial to minimize downtime from unforeseen incidents, ensuring consistent performance. Analyzing RTO & RPO establishes recovery goals, reducing disruptions significantly.

TR7.2: Enhance resource creation and deletion process so as to reinitiate malfunctioned resource quickly

TR8.1: Set up tenant identification to identify users and authorize them to access resources

TR9.1: Implement a comprehensive CI/CD pipeline for swift releases. Automate testing, staging, and deployment to ensure seamless content delivery

TR9.2: Configure tools to automate the provisioning and management of infrastructure components. This is essential to standardize infrastructure deployment, increase scalability, and enhance operational efficiency by automating resource creation and configuration

TR10.1: Utilize adaptive bitrate streaming, low-latency protocols(WebRTC, RTMP, HLS, and DASH), and load balancing for high-quality, simultaneous access to creator's live content by a large audience.

TR10.2: Use advanced video encoding techniques to deliver high-quality video at different bitrates.

TR10.3: Employ monitoring tools to track server performance, user experience, and other relevant metrics in real-time.

TR11.1: Integrate digital watermarking or fingerprinting methods into the content to embed distinctive identifiers, enabling the tracking of duplicated material sources for deterrence and unauthorized copy identification.

TR11.2: Automate the Digital Millennium Copyright Act (DMCA) process.

TR11.3: Implement a comprehensive reporting system to allow the owners to report their content theft.

TR12.1: Implement an elastic storage system by assessing data requirements and continuously monitoring storage capacity, while making adjustments to scale it up or down as necessary.

TR13.1: Establish a testing environment at a production scale that accurately mimics the live environment, in order to assess the application's reliability.

TR14.1: Define Service Level Agreements(SLA) for application uptime, availability and application performance.

TR14.2 : Update service level agreements(SLAs) such as availability or data retention periods to minimize the number of resources required to support your workload while continuing to meet business requirements.

TR15.1: Establish usage alerts for every cloud provider service and promptly respond when the threshold is reached to maintain cost-effectiveness and efficient scaling.

TR16.1: Create a system to gather data from user sessions and save it in a database, streamlining the process for subsequent data analysis.

TR16.2: Monitoring real-time statistics on viewing duration and the number of viewers, incorporating both current and historical data to enrich the experiences of viewers and streamers along with the resources being used for services.

TR17.1: Implement a communication protocol to implement delivery

TR17.2: Create an automated message filtering algorithm designed to identify and eliminate negative content, including spam, sexual material, and scam links, in order to safeguard the platform from potential legal issues.

TR17.3: Utilize WebSockets for the implementation of real-time messaging with minimal latency.

TR17.4: Implement a message queue to prioritize messages.

TR18.1: Frequently monitor resource prices and be ready to respond if there are significant price fluctuations
TR18.2: Minimize cost while maintaining sufficient performance levels
TR19.1: Optimize geographic placement of workloads for user locations: Analyze network access patterns to identify where your customers are connecting from geographically.
TR19.2: Select Regions and services that reduce the distance that network traffic must travel to decrease the total network resources required to support your workload.
TR19.3: Maintain a comprehensive user registration to know beforehand, where we should put our resources.
TR20.1: Conduct regular incident response drills to ensure that the team is well-prepared to respond effectively to security events, and continuously refine response processes based on lessons learned.
TR20.2: "Protect data in transit and at rest" [2]
TR20.3: All Logs, metrics, and telemetry should be collected centrally, and automatically analyzed to detect anomalies and indicators of unauthorized activity.
TR20.4: Employ comprehensive encryption measures to secure data during transmission and storage.Since securing data during transit and storage is vital for maintaining integrity and confidentiality. Robust encryption ensures protection, meets compliance standards, and lowers the risk of unauthorized access, building trust in data confidentiality
TR22.1: Implement a real-time monitoring system to track content and ensure prompt identification and filtering of potential harmful or violating content.

2.4 Tradeoffs

1. **Trade-off Pair 1:**
   TR3.1: Achieve maximum latency of 0.5s
   TR1.3: Deploy duplicates of the application in different locations

   Conflict Justification:

Deploying duplicates across different locations can introduce network latency due to data synchronization between these locations, potentially impacting the achievement of a 0.5s latency goal. (*Rephrased using [23]*)

2. **Trade-off Pair 2:**
   TR1.1: Implement Redundant Systems for 99.99% availability
   TR18.2: Minimize cost while maintaining sufficient performance levels

   Conflict Justification:
   TR1.1 emphasizes redundancy and high availability to ensure continuous service in case of failures, which may lead to increased infrastructure costs to maintain redundancy.
   TR18.2 aims to minimize costs while maintaining performance, potentially conflicting with the cost implications of implementing redundant systems, as it might increase infrastructure expenses.(*Rephrased using [23]*)

3. **Trade-off Pair 3:**
   TR2.1: Implement efficient auto-scaling based on real-time demand
   TR12.1: Implement an elastic storage system by continuously monitoring and adjusting storage capacity

   Conflict Justification:
   TR2.1 emphasizes efficient auto-scaling to dynamically adjust computing resources based on demand to ensure optimized resource utilization.
   TR12.1 focuses on an elastic storage system that continuously monitors and adjusts storage capacity, potentially leading to increased resource allocation to maintain scalability, which might conflict with the goal of optimizing resource utilization. (*Rephrased using [23]*)
4. **Trade-off Pair 4:**
   TR9.1: Implement a comprehensive CI/CD pipeline for swift releases
   TR20.4: Employ comprehensive encryption measures to secure data during transmission and storage

   Conflict Justification:
   TR9.1 emphasizes swift software releases through a comprehensive CI/CD pipeline, focusing on rapid deployment and delivery.
   TR20.4 involves implementing comprehensive encryption measures for securing data during transmission and storage, potentially introducing additional steps that could affect the speed of deployment and delivery due to encryption processes. (*Rephrased using [23]*)

3. Provider Selection

3.1 Criteria for choosing a provider

When selecting among AWS, Azure, and GCP, crucial criteria include ensuring reliable availability, efficient auto-scaling for performance optimization, and proactive security measures. We decided to focus on the following factors as our criteria to decide a provider of choice:

CR1: Market Share : Which candidate owns what portion of the market share?
CR2: Years of expertise: Since how long has been the provider functioning and evolving?
CR3: Number of services they offer: How extensive is the service inventory of the provider?
CR4: Services for live streaming and content moderation: Which provider offers best live video streaming and content moderation services?
CR5: Global Coverage: Which service has a good worldwide coverage?

3.2 Provider Comparison

Below, you'll find a table for each criteria and our observations about what we inferred from them.

Criteria	**Amazon Web Services**	**Google Cloud Provider**	**Microsoft Azure**
CR1:Market Share	32% in Q2 2023	11% in Q2 2023	22% in Q2 2023 [3]
CR2: Years of expertise	Founded on March 2006	Founded on April 7, 2008	Founded on February 1, 2010
CR3: Number of services they offer	AWS has over 200 fully featured services for a wide range of technologies, industries, and use cases. [4]	Over 100 products [5]	More than 200 products and cloud services [6]
CR4: Services for	Amazon Kinesis	Google Cloud	Azure Media

live streaming and content moderation	Video Streams, AWS Elemental MediaLive, AWS Elemental MediaPackage, AWS Elemental MediaStore, AWS Elemental MediaConvert, Amazon Rekognition Video	Video Intelligence API, Google Cloud Pub/Sub, Google Cloud CDN, Google Cloud Video Transcoding, Google Cloud Storage	Services, Azure Video Analyzer for Media, Azure Media Player, Azure Content Moderator, Azure Stream Analytics
CR5: Global Coverage	32 geographic regions, 102 Availability Zones, 15 more availability zones announced, 5 more regions announced [7]	39 Regions, 118 Zones, 187 Network Edge Locations, 200+ Countries [8]	52 regions, 6 more planned, 60+ announced regions [9]

Below you'll find a table with a **ranking** as 1 for best and 3 for the worst in that criteria.

Criteria	AWS	GCP	Azure	Justification
CR1:Market Share	1	3	2	Amazon is a clear dominator in the market share with 32% holding
CR2: Years of expertise	1	2	3	AWS has been in the market for the longest and has the most data and insights about how to function as a provider. This increases their reliability
CR3: Number of services they offer	1	2	1	Both AWS and Azure offer a similar number of services. While GCP offers fewer of them, they might have done a great job of packaging multiple solutions in fewer services.
CR4: Services for live streaming and content moderation	1	1	1	All of them offer pretty similar services for managing live streamed videos. But the workflow of some services such as rekognition, AWS IVF, AWS Elemental Media services we discovered to be more streamlined

CR5: Global Coverage	3	2	1	Azure is the leader in this aspect. But availability is not just dependent on physical presence but also on how the networks are setup

3.3 The Final Selection

We decided to go with AWS as our provider as we are more familiar with how it functions. Moreover, after surveying similar applications we found that plenty of credible ones are using the same provider. (For example: Twitch. Please find the case study in reference [10])

3.3.1 The list of services offered by the winner

List of services provided by AWS corresponding to the technical requirements:

TRs	**Short Description**	**Corresponding AWS Service/Tools**
TR1.1	Redundant Systems	AWS Auto Scaling, AWS Backup
TR1.2	Load Balancing	AWS Elastic Load Balancing (ELB)
TR2.1	Auto-scaling	AWS Auto-scaling
TR2.2	Monitoring & Alerts	Amazon CloudWatch
TR3.1, TR5.1	Content Delivery Network	Amazon CloudFront
TR3.2	Efficient Server Distribution	AWS Global Accelerator
TR4.1	Real-time Expense Tracking	AWS Cost Explorer, AWS Budgets
TR4.2	Cost and Usage Reports	AWS Cost and Usage Report
TR6.1	Multi-factor Authentication	AWS Identity and Access Management (IAM)
TR6.2	Role-based Access Control	AWS Identity and Access Management (IAM)
TR7.1	High Availability/Disaster Recovery	Amazon Elastic Block Store (Amazon EBS) snapshot, Amazon DynamoDB

		backup, Amazon S3 Cross-Region Replication (CRR)
TR7.2	High Availability/Disaster Recovery	AWS EKS
TR8.1	Tenant Identification	AWS Organisation, AWS Cognito, AWS IAM
TR9.1	CI/CD Pipeline	AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy
TR9.2	Infrastructure Automation	AWS CloudFormation, AWS OpsWorks
TR10.1	Video Content Delivery	Amazon Elastic Transcoder, Amazon Kinesis Video Streams
TR10.2	Advanced Video Encoding	Amazon Kinesis Video Streams, AWS Elemental MediaLive, AWS Elemental MediaPackage, AWS Elemental MediaStore, AWS Elemental MediaConvert,
TR10.3	Real-time Monitoring	Amazon CloudWatch
TR11.1	Content Identification	AWS Elemental MediaTailor, Amazon Rekognition
TR11.2	DMCA Automation	AWS Elemental MediaTailor
TR11.3	Automated Content Monitoring	Amazon Rekognition
TR12.1	Elastic Storage	Amazon Elastic File System (EFS), Amazon S3
TR13.1	Production-scale Testing Environment	AWS CloudFormation, AWS Cloud9
TR15.1	Usage Alerts	Amazon CloudWatch, AWS Budgets
TR16.1	User Session Data Collection	Amazon Kinesis, Amazon Redshift
TR16.2	Real-time Statistics Monitoring	Amazon Kinesis, Amazon CloudWatch
TR17.1	Communication Protocol	Amazon Simple Queue Service (SQS)
TR17.2	Message Filtering	Amazon Simple Notification Service

		(SNS), Amazon Simple Queue Service (SQS)
TR17.3	Real-time Messaging	Amazon API Gateway, Amazon MQ
TR17.4	Message Queue	Amazon Simple Queue Service (SQS)
TR18.1	Price Monitoring	AWS Cost Explorer, AWS Price List API
TR19.1	Workload Geographic Placement	AWS Global Accelerator, AWS Local Zones
TR19.2	Regional Selection	AWS Global Accelerator, AWS Regions
TR19.3	User Registration	Amazon Cognito
TR20.1	Security at All Levels	AWS Security Hub, AWS Identity and Access Management (IAM)
TR20.2	Incident Response Drills	AWS Security Hub, AWS Incident Manager
TR20.3	Log Analysis for Anomalies	Amazon CloudWatch, AWS Security Hub
TR22.1	Real Time Content Moderation	Amazon Rekognition Video

Explanations:

1. **AWS ELB(Elastic Load Balancing):**

   Elastic Load Balancing distributes the incoming traffic to multiple targets such as EC2 instances, containers, across multiple availability zones. It also considers the health of these targets and only forwards traffic to targets which are healthy. Elastic Load Balancer reacts when the incoming traffic changes and thereby making sure our resources are properly utilized.[11]

   AWS ELB works with following services:
   - Amazon EC2 : EC2 instances are virtual servers that run our application in the cloud. We can configure our Load Balancer such that all the incoming traffic is routed across these instances.
   - Amazon CloudWatch : We can use CloudWatch to monitor incoming traffic.
   - Amazon ECS : We can configure our Load Balancer such that the traffic is routed to the containers.

2. **AWS EC2(Amazon Elastic Compute Cloud):**
   It offers resizing compute capabilities in the cloud. The main function is to launch virtual servers and scale capacity as needed. The instances have various types which are optimized for different needs such as compute, memory, storage and networking capabilities.[12]
   a. Features -
      i. Elasticity: Scale resources up & down according to demand
      ii. OS Variety: Multi Operating System support
      iii. Security: Offers security groups, VPC, and storage options like EBS
   b. Components -
      i. Instances, AMIs(Amazon Machine Images), EBS(Elastic Block Store), Security Groups, Key Pairs.
      ii. AMIs: Templates providing the information required to launch an instance, including the operating system, applications, and configurations
      iii. EBS: Provides persistent block-level storage volumes for EC2 instances.
      iv. Key Pairs: Consists of a public key and a private key used for secure access to instances
   c. Practical Applications:
      i. Hosting applications and websites
      ii. Development and testing environments
      iii. High-performance computing (HPC) applications
      iv. Machine learning and artificial intelligence workloads
   d. Usage:
      i. Accessible via Management Console, CLI, SDKs for management.
      ii. Utilize ELB, Auto Scaling for load balancing and scalability.

3. **Amazon CloudWatch:**
   This service is used for real time monitoring of the AWS resources and applications and setting up alerts for them. It collects and tracks variables/metrics for them. [13]
   a. Metrics like the ones listed below are monitored:
      i. CPU Utilization
      ii. Network Traffic
      iii. Disk Metrics
      iv. Memory Utilization
      v. Latency

b. Features of CloudWatch: Offers system-wide visibility into resource utilization, application performance, and operational health
c. Dashboards: Create custom dashboards to display metrics and set alarms for thresholds. They provide a consolidated view of metrics and alarms, aiding in health assessment, incident response, and team collaboration. Cross-account observability and favorites list enhance dashboard functionality. Access requires specific permissions or policies.
d. Related Services:
    i. Amazon SNS: Sends messages when alarm thresholds are reached
    ii. Amazon EC2 Auto Scaling: Scales EC2 instances based on CloudWatch alarms.
    iii. AWS CloudTrail: Monitors CloudWatch API calls for your account.
    iv. AM: Controls access to AWS resources for users.
e. Amazon CloudWatch facilitates real-time monitoring, metric tracking, alarms setup, and access to related AWS services while offering cost analysis features for better resource management

4. **AWS Rekognition :** AWS Rekognition is a service provided by Amazon Web Services. It is used for image and video analysis [14]. The tenant who uses AWS Rekognition just has to provide an image or an video and the service can:
    - Offer facial recognition services
    - Identify objects in the media provided
    - Detect inappropriate content (texts, images, scenes)
    - Recognize celebrities from various fields in the media
    - Provide scalable image analysis

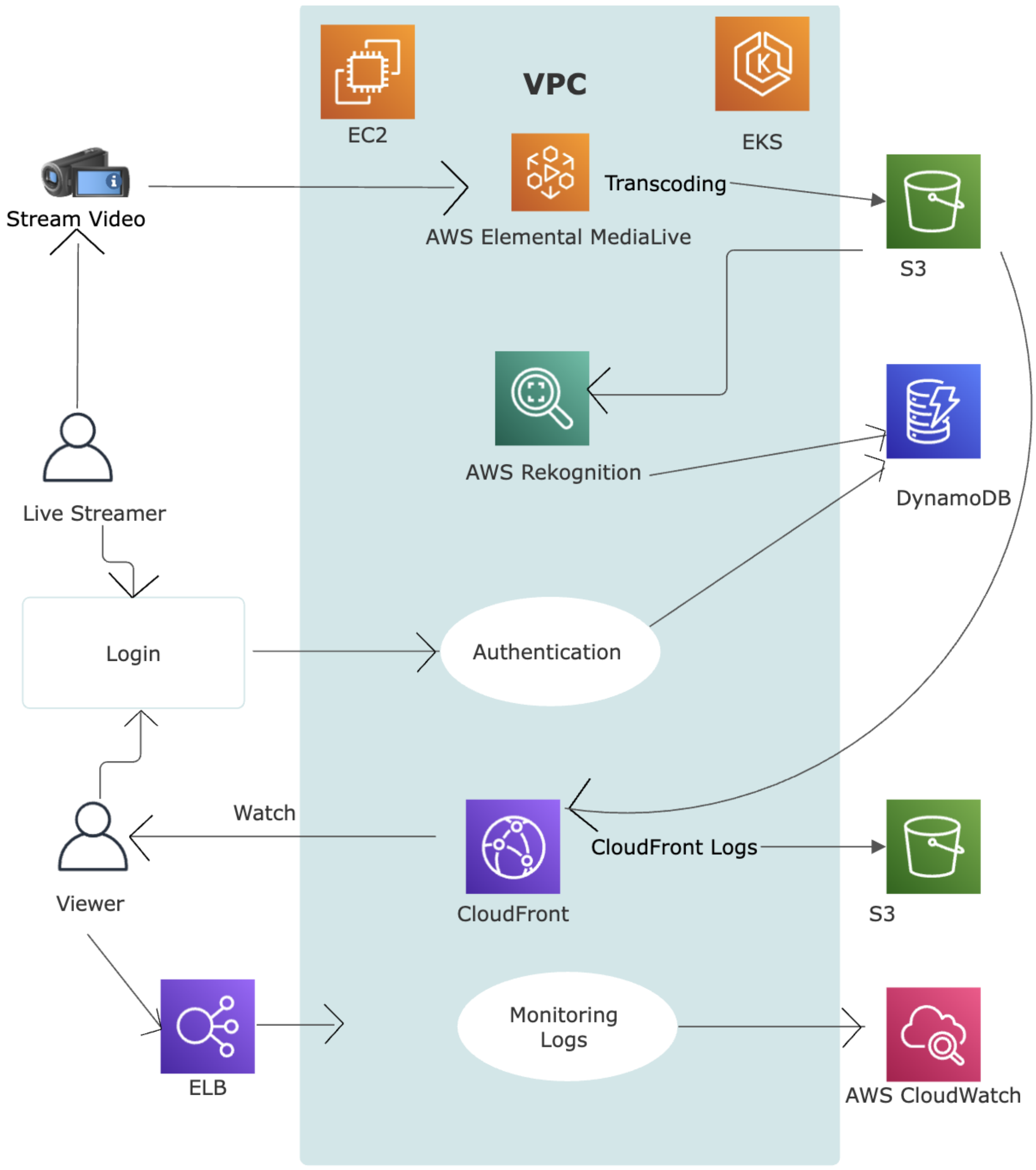
    How does it work?
    - AWS Rekognition has been built upon extensive research conducted by Amazon's Computer Vision scientists over the course of several years.
    - It provides two API sets : Amazon Rekognition Image for analyzing images, and Amazon Rekognition Video for analyzing videos
    - It gives responses in the format of JSON. Following is an example of a response from AWS Rekognition API.

```
"ModerationLabels": [
 {
        "Confidence": 99.24723052978516,
        "ParentName": "",
        "Name": "Explicit Nudity"
  }
]
```

In this response, the "Name" field indicates the type of violation detected, such as "Explicit Nudity," and the "Confidence" field provides a measure of the system's confidence in the accuracy of the detection.

4. The First Design Draft

Below is our introductory architecture idea:



We choose the below mentioned TRs from section 2:

WAF Reliability Pillar TR:
TR1.3: Deploy duplicates of the application in different locations.

WAF Performance Pillar TR:
TR1.2: Distribute Load, strategize a way to distribute load evenly across several servers to stay safe from server overloads
TR2.1: Implement efficient auto-scaling to automatically adjust computing resources according to real-time demand

WAF Security Pillar TR:
TR20.2: Protect data in transit and at rest

Tenant Identification TR:
TR8.1: Set up tenant identification to identify users and authorize them to access resources.

Monitoring TR:
TR2.2: Set robust monitoring systems to continuously track performance metrics and trigger alerts when predefined thresholds are exceeded.
TR16.1: Create a system to gather data from user sessions and save it in a database, streamlining the process for subsequent data analysis.

Conflicting TRs:
Pair I:
TR3.1: Achieve maximum latency of 0.5s
TR7.2: Enhance resource creation and deletion process so as to reinitiate malfunctioned resource quickly
Pair II:
TR1.1: Implement Redundant Systems - Implement backup duplicate systems that can be used if the currently active systems fail to ensure 99.99% availability
TR18.2: Minimize cost while maintaining sufficient performance levels

4.1. The basic building blocks of the design

1. **Compute Services:**

- **AWS EC2:** Works in conjunction with EKS, facilitating the execution of Kubernetes worker nodes for processing live stream data and managing real-time application demands.
- **AWS EKS:** Manages the Kubernetes Control Plane, orchestrating and scaling live stream application components within an AWS Virtual Private Cloud (VPC).

2. **Storage Services:**
    - **AWS S3:** Utilized as object storage for streaming content such as videos, thumbnails, and other media assets.
    - **AWS DynamoDB:** Stores user profiles, session details, and metadata related to live streams
3. **Networking Services:**
    - **Amazon Elastic Load Balancer:** Distributes incoming live streaming traffic among multiple instances or Kubernetes pods, ensuring scalability and availability.
    - **Amazon VPC:** Establishes public and private subnets, manages traffic routing, and provides secure internal communication between streaming components in different availability zones.
4. **Monitoring Services:**
    - **AWS CloudWatch:** Monitors and manages the performance metrics, logs, and resource utilization of the live streaming platform, ensuring optimal operation and swift detection of issues.
5. **Tenant Identification Services:**
    - **AWS Cognito:** Manages user authentication, access control, and session management for viewers accessing the live streaming platform.
6. **Video Handling Services:**
    - **AWS Elemental**: AWS Elemental is a comprehensive suite of cloud-based services tailored for efficient video processing, transcoding, packaging, and delivery across diverse platforms and devices.(Services like: AWS Elemental MediaLive, AWS Elemental MediaPackage, AWS Elemental MediaStore, AWS Elemental MediaConvert)
    - **Amazon Kinesis Video Streams :** Amazon Kinesis Video Streams is an AWS service for streaming, processing, and storing real-time video data, making it suitable for applications requiring analytics, machine learning, and real-time processing**.**
7. **Other Crucial Services:**
    - **AWS Rekognition:** Provides real-time analysis for content moderation, identifying objects, scenes, and faces within the live stream content.

4.2 Top level, informal validation

As per the selected TRs above and the building blocks, i.e. the services. The justifications are as follows.

WAF Security Pillar TRs:

1. The data which is stored in the database and is not moving through the network is the data at rest is encrypted as both DynamoDB and S3 have this provision taken care of by default.
2. The data transiting via the network is also encrypted by default in AWS. [15][16]

WAF Reliability Pillar TRs: We use S3 Multi-region access points service to replicate S3 databases over regions.

WAF Performance Pillar TRs:

1. **Load Balancing:** AWS ELB efficiently distributes incoming streaming traffic among multiple instances or Kubernetes pods, ensuring uninterrupted streaming experiences.
2. **Autoscaling(Elasticity):** AWS EKS automatically adjusts Kubernetes worker nodes based on varying demands, ensuring scalability.

Tenant Identification TRs:

1. With VPC, we can leverage VPC flow logs to retrieve IPs of nodes accessing the network attempting to view the stream.
2. AWS Cognito & IAM enables to authorize and authenticate the users

Monitoring TRs:

1. CloudWatch fulfills the requirements by enabling continuous tracking of performance metrics, setting predefined alarms for threshold breaches
2. They generate comprehensive cost and usage reports for expense monitoring within AWS services.

Other TRs:

1. AWS EC2 enables creating duplicate instances across different regions, providing redundancy and facilitating failover to backup systems in case of primary system failure
2. AWS S3 offers milliseconds latency S3 Glacier Instant Retrieva class onwards. Moreover, they are designed for 99%+ for availability. This helps us minimize latency and improve reliability. [17]
3. AWS DynamoDB claims single-digit millisecond performance and up to 99.999% availability, it is in reach to attain 0.5s max latency[18]

4.3 Action items and rough timeline

Skipped

5. The Second Design

5.1. Use of the Well-Architected Framework

5.1.1. Operational excellence:

“The ability to support development and run workloads effectively, gain insight into their operations, and to continuously improve supporting processes and procedures to deliver business value.”[19]

Design Principles [20]:

1. Perform Operations as Code
2. Make Frequent, Small, Reversible Changes
3. Refine Operations Procedures Frequently
4. Anticipate Failure
5. Learn from Operational Failures

Best Practices [20]: Organisation, preparation, operation and evolution

5.1.2. Security:

“The security pillar describes how to take advantage of cloud technologies to protect data, systems, and assets in a way that can improve your security posture.”[19]

Design Principles [2]:

1. Implement a Strong Identity Foundation
2. Enable Traceability
3. Apply Security at All Layers
4. Protect Data in Transit and at Rest
5. Keep People Away from Data

Best Practices [2]: Identity and Access Management, Detection, Infrastructure Protection, Data Protection, Incident Response

5.1.3. Reliability:

“The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it’s expected to. This includes the ability to operate and test the workload through its total lifecycle.”[19]

Design Principles [20]:

1. Automatically Recover from Failure

2. Test Recovery Procedures
3. Scale Horizontally
4. Stop Guessing Capacity
5. Manage Change in Automation

Best Practices [20]: Foundations, Workload Architecture, Change Management, Failure Management

5.1.4. Performance Efficiency:

“The ability to use computing resources efficiently to meet sys- tem requirements, and to maintain that efficiency as demand changes and technologies evolve.”[19]

Design Principles [20]:

1. Democratize Advanced Technologies
2. Go Global in Minutes
3. Use Serverless Architectures
4. Experiment More Often
5. Consider Mechanical Sympathy

Best Practices [20]: Selection, Compute, Storage, Database, Network, Review, Monitoring, Trade Offs

5.1.5. Cost Optimization:

“The ability to run systems to deliver business value at the lowest price point.” [19]

Design Principles [20]:

1. Implement Cloud Financial Management
2. Adopt a consumption model
3. Measure overall efficiency
4. Stop spending money on undifferentiated heavy lifting
5. Analyze and attribute expenditure

Best Practices [20]: Practice Cloud Financial Management, Expenditure and usage awareness, Cost-effective resources, Manage demand and supply resources, Optimize over time

5.1.5. Sustainability:

“The ability to continually improve sustainability impacts by reducing energy consumption and increasing efficiency across all components of a workload by maximizing the benefits from the provisioned resources and minimizing the total resources required” [19]

Design Principles [20]:

1. Understand Your Impact
2. Establish Sustainability Goals
3. Maximize Utilization
4. Anticipate and Adopt New Efficient Technologies
5. Use Managed Services
6. Reduce Downstream Impact

Best Practices [20]: Region Selection, User Behavior Patterns, Software and Architecture Patterns, Data Patterns, Hardware Patterns, Development and Deployment Patterns

5.2. Discussion of pillars

1. Reliability Pillar in AWS Well-Architected Framework: [21]
    a. The Reliability pillar ensures consistent workload performance and functionality in AWS. It comprises design principles and best practices [19]:
    b. Design Principles:
        i. Automatic recovery from failures using KPI-triggered automation.
        ii. Testing and validating recovery procedures in cloud environments.
        iii. Horizontal scaling for improved workload availability and resilience.
        iv. Dynamic resource allocation based on demand to prevent over/under-provisioning.
        v. Automating infrastructure changes to track and review modifications.
    c. Best Practice Areas:
        i. Foundations: Establishing an environment accommodating workload needs.
        ii. Workload Architecture: Designing systems to prevent/mitigate failures.
        iii. Change Management: Accommodating workload changes efficiently.
        iv. Failure Management: Ensuring workloads withstand and recover from failures.
    d. Key Considerations:
        i. Manage service quotas and network topology for reliable cloud-based architectures.
        ii. Design scalable and reliable workloads using service-oriented or microservices architecture.
        iii. Monitor and automate workload changes based on demand for adaptability.
        iv. Implement controlled changes to maintain known software states and predict outcomes.
    e. Failure Management:
        i. Employ automation to react to monitoring data and automate remedial actions.
        ii. Implement fault isolation boundaries and backup strategies for workload protection.
        iii. Conduct regular testing and plan for disaster recovery to ensure workload resilience.
        This framework emphasizes automation, scalable architectures, proactive testing, and effective failure management to ensure reliable workload operations on AWS. (*Rephrased using [23]*)

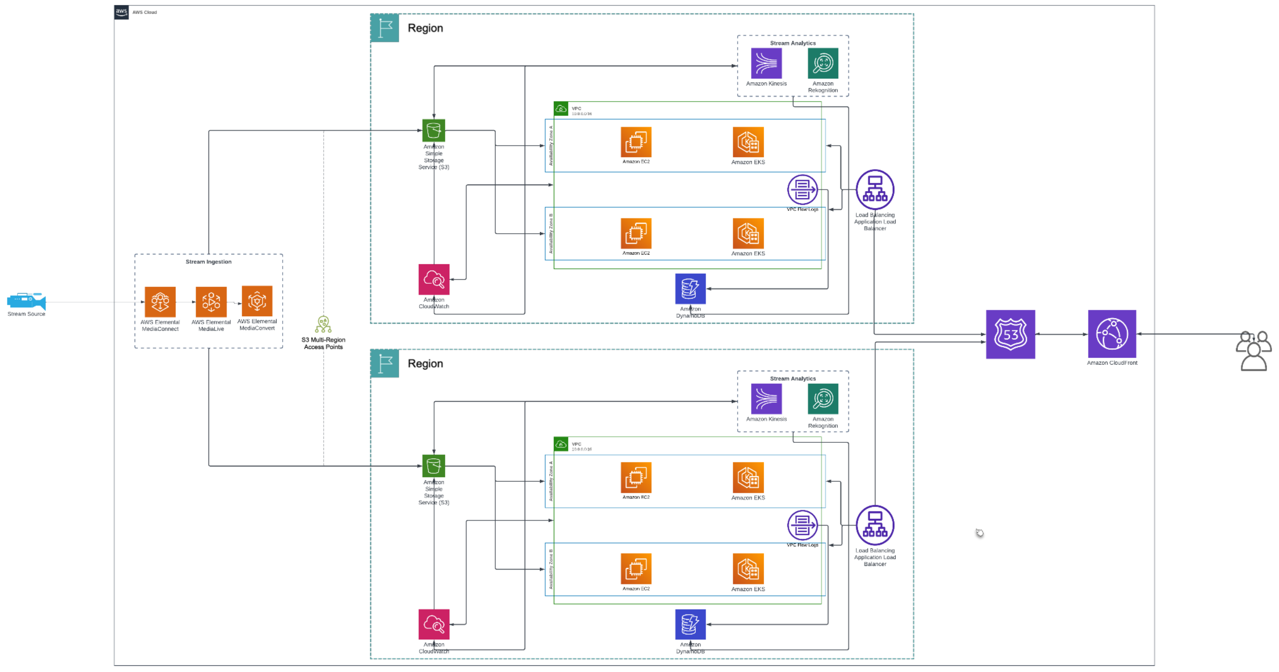
2. Sustainability in AWS Well-Architected Framework: [22]

   The Sustainability pillar emphasizes environmental impact reduction and energy efficiency in cloud workloads. [19]

   a. **Design Principles:**
      It comprises six design principles and best practice areas. Measure and model the impact of workloads, establish sustainability goals, maximize utilization, adopt new efficient technologies, use managed services, and reduce downstream impact.
   b. **Best Practice Areas:**
      Region selection, user behavior patterns, software and architecture patterns, data patterns, hardware patterns, and development and deployment patterns.
   c. **Key Considerations:**
      Select Regions based on sustainability goals and user behavior. Optimize infrastructure to match user load and minimize resources. Utilize efficient software, manage data effectively, and optimize hardware usage.
   d. **Region Selection:**
      Choose regions based on business needs and sustainability goals, favoring areas near renewable energy projects.
   e. **User Behavior Patterns:**
      Scale infrastructure to match user load efficiently. Align service levels with user needs and remove unused assets.
   f. **Software and Architecture Patterns:**
      Optimize for consistent high utilization, remove underutilized components, and optimize code and device compatibility.
   g. **Data Patterns:**
      Efficiently manage data storage, classify data, and optimize data access and storage patterns.
   h. **Hardware Patterns:**
      Minimize hardware usage, leverage efficient instance types, use managed services, and optimize GPU usage.
   i. **Development and Deployment Patterns:**
      Test and validate sustainability improvements, keep software updated, increase build environment utilization, and use managed device farms for testing.

This framework aims to reduce environmental impact and improve energy efficiency across all aspects of cloud workload management and promotes ongoing resource efficiency and reduced energy consumption in workload management. It advises integrating sustainability goals into operations for long-term environmental benefits and optimized cloud infrastructure. (*Rephrased using [23]*)

5.3. Cloud Architecture Diagram



5.4. Validation of the Design

Justification of how our design meets the Technical Requirements (TRs) we mentioned in section 4.

1. In our system architecture, tenant identification is vital. [TR8.1] Utilizing a Virtual Private Cloud (VPC) and AWS Flow Logs, we monitor individual IP addresses accessing the website and store them into Amazon DynamoDB, ensuring precise tenant identification. [24]
2. We have mentioned 99% percent availability, to achieve this our design has following components:
    - We host Amazon EKS worker nodes on EC2 instances, monitored by AWS CloudWatch for CPU utilization. Our YAML configuration sets a 70% threshold. Using Horizontal Pod Autoscaler, new pods dynamically spawn when CPU usage exceeds 70%, calculated using the formula to determine desired replicas. Also, enabling autoscaling on ec2 helps us increase ec2 if the load needs it.
    - Our application is deployed across multiple regions and availability zones for enhanced resilience and availability.
    - You utilize AWS CloudWatch alongside AWS Kinesis to monitor key metrics such as throughput capacity, iterator age, error rates, latency, and

buffer usage. This proactive monitoring strategy allows you to identify and address potential delays in data consumption for your real-time video streaming services, ensuring the overall health and quality of the streams.[25]
    - We used S3 Glacier Flexible Retrieval which provides ideal backup and disaster recovery solutions[26]
3. To fulfill our stream analytics and content moderation needs, we have opted for a design that leverages AWS Kinesis and AWS Rekognition services[TR16.1]. This architecture involves fetching stored media from Amazon S3, processing the media using the selected services, and then depositing the generated reports back into DynamoDB.
4. To meet our security requirements[TR20.2], we've chosen **S3 Glacier Flexible Retrieval** as our storage option, as previously mentioned in point 2.
    - S3 supports SSL for data in transit
    - S3 supports built-in encryption for data in rest
    - The use of a Virtual Private Cloud (VPC) further enhances our security posture by encrypting all network traffic within the VPC.
5. We used Layer 7 Application Load Balancer to distribute the load coming from users in our application which distributes the load evenly across the EC2 instances.[TR1.2] We use AWS CloudWatch to monitor Load Balancer metrics such as request count, healthy host count, unhealthy host count, latency, response time etc.
6. We've implemented an Elastic Load Balancer to evenly distribute the incoming user traffic across our application's EC2 instances . CloudFront first checks its cache for the requested content. If the content is not present in the cache or has expired, CloudFront proceeds to send the request to the configured load balancer.
7. In order to meet our need for availability in different regions, we have deployed our application in two distinct geographical locations.[TR1.3]
8. Optimized latency to 0.5 seconds by integrating CloudFront caching and Auto Scaling with AWS CloudWatch. Monitoring key metrics, including Cache Hit Rate, latency, group size, and CPU utilization for agile scaling and operational success. [TR3.1]
9. AWS CloudWatch plays a key role in monitoring intricate metrics across our architecture [TR2.2]. We've set up alarms to alert us when predefined thresholds for multiple metrics are exceeded. Metrics of following components are monitored by CloudWatch:
    a. EC2:
        - **CPUUtilization** : We have set 80% as the threshold for CPU utilization. If the usage exceeds 80% we would raise an alarm.

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/viewing_metrics_with_cloudwatch.html

b. S3:
   - **ReplicationLatency** : If the replication latency exceeds 2 seconds, we will raise an alarm.
   - **BucketSizeBytes** : If the bucket size exceeds 80% of its capacity, we will raise the alert. https://docs.aws.amazon.com/AmazonS3/latest/userguide/cloudwatch-monitoring.html

c. EKS
   - **node_cpu_limit** : If cpu utilization of a node exceeds 70%, we will raise an alarm.
   - **node_memory_utilization** : If memory utilization of a node exceeds 70%, we will raise an alarm. https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Container-Insights-metrics-EKS.html

d. AWS Kinesis:
   - **GetRecords.Latency :** We target latency for this operation to be 2 seconds.
   - **IteratorAgeMilliseconds :** We will raise an alarm if the value for this metric goes above 6000 milliseconds. https://docs.aws.amazon.com/streams/latest/dev/monitoring-with-cloudwatch.html

e. AWS Rekognition:
   - **Throttled Count** : We've established a throttled count threshold equivalent to 90% of our total count. If the number of requests surpasses this threshold, we trigger an alarm.
   - **DetectedFaceCount** : We've established a 90% threshold for DetectedFaceCount to ensure effective classification of all our media. https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-monitoring.html

f. AWS CloudFront:
   - **Total Error Rate :** We have set a threshold of 1% for this metric, which combines 4xx error rate and 5xx error rate. This helps us get updates about failures quickly. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/monitoring-using-cloudwatch.html

10. We've set up AWS EKS clusters and EC2 instances as the hosting environment for our Kubernetes pods. In this configuration, Horizontal Pod Scaling (HPA) is

employed—a mechanism that automatically adjusts pod counts based on metrics such as memory and CPU usage [TR2.1]. AWS CloudWatch is utilized to monitor these metrics, ensuring our scaling strategy is responsive to real-time demand.

11. We've established redundancy by deploying our systems across multiple availability zones [TR1.1] in different regions, ensuring high availability and fault tolerance.
12. We have used multiple services to effectively address our storage requirements
    - AWS DynamoDB
    - AWS S3
    - AWS CloudWatch

5.5. Design Principles and Best Practices Used

1. **Scalability** [27]: AWS EKS enables autoscaling of pods which handle the application container that runs our services. Also, it can size up and down with minimal delays as containers are faster to set up than virtual machines.
2. **Disposable Resources Instead of Fixed Servers** [27]: We use EKS to manage pods on our EC2 instances. Managing microservices like these don't have much overhead of deletion and creation and can be done quickly compared to VMs.
3. **Caching** [27]: CloudFront is a CDN service that minimizes the number of requests the origin server needs to handle by caching the responses to them in the closest edge server near the location of the request.
4. **Security** [27]:
5. **Make frequent, small, reversible changes**: One of the crucial TR is to implement CI/CD pipelines for swift releases and also we used the microservices architecture to implement our functionality. This software development practice and architecture is the best suited for making frequent reversible changes.
6. **Keep it simple and stupid** [28]: Instead of implementing services for video on our servers, we leveraged tailored solutions like elemental media live and elemental media connect for our use case.
7. **Implement a Strong Identity Foundation** [29]: We leveraged VPC for this. Have a look at 5.4. Validation of the Design pt 1.
8. **Protect Data in Transit and at Res**t [29]: Choosing AWS & its database services like S3 and DynamoDB ensures this by default. As discussed in 4.2 Top level, informal validation, WAF Security Pillar TRs.

5.6. Tradeoffs Revisited

Below we briefed the additional tradeoffs we faced after our first draft of the design

5.6.1 Multi-region duplicacy and latency

Multi - region duplicacy allows us to have redundancy in our systems and allows us to have wider coverage area. While Latency allows us to have faster response time and better user experience. To achieve duplicacy, we have deployed our application across two regions. We even have a multi region access point for our S3 storage. For minimizing latency we have used Cloudfront.

Identifying Objectives :

a) Redundant systems, greater availability and greater
b) Minimizing Latency

Identifying Alternatives :

a) Multi-Region Redundancy
b) Single Region Architecture
c) Selective Duplicacy

Creating Consequence Table and eliminating dominated alternatives:

Alternative	Redundancy & Fault Tolerance	Latency	Evaluation	Dominated
Multi-Region Redundancy	High	Moderate	Balanced	No
Single Region	Low	Very High	Latency Based	Yes
Selective Duplicacy	Moderate	Moderate	Balanced	No

We eliminate the option of going with Single Region Architecture, as it is dominating.

Ranking Table :

Ranking	Alternative	Redundancy & Fault Tolerance	Latency	Evaluation	Dominated
1	Multi-Region Redundancy	High	Moderate	Balanced	No

3	Selective Duplicacy	Moderate	Moderate	Balanced	No
2	Single Region	Low	Very High	Latency Based	Yes

We used an even swaps method to come to a decision that Multi-region redundancy is in fact the most suitable option for us which meets our requirements. Single region architecture gives us very minimal latency but we are willing to sacrifice latency to a certain extent to get our desired redundancy. We considered selective duplicacy where we don't duplicate all our services, but it's still not better than complete multi-region architecture in our view.

5.6.2. Autoscaling and latency

We have evaluated this tradeoff using the even swaps method [30]. Auto-Scaling allows us to dynamically change our resource allocation. This helps in cost optimization. But if we scale-up, then latency will be affected. Even while scaling down, latency is affected.

Identify Objectives :

a) Minimize Latency
b) Efficient AutoScaling

Identifying Alternatives :

a) Very high level auto scaling
b) Extremely fast systems, min latency
c) Balance between auto scaling and latency

Creating Consequence Table and eliminating dominated alternatives:

Alternative	Auto Scaling Efficiency	Latency	Evaluation	Dominated
Performance	Very High	Low	Auto Scaling based	Yes
Cost	Moderate	Very High	Latency Based	No
Scalability	Moderate	Moderate	Balanced	No

We eliminate Performance , as it does not provide us with the latency as per requirements.

Ranking Table :

Ranking	Alternative	Auto Scaling Efficiency	Latency	Evaluation	Dominated
1	Scalability	Moderate	Moderate	Balanced	No
2	Cost	Moderate	Very High	Latency Based	No
3	Performance	Very High	Low	Auto Scaling based	Yes

We used the even swaps method, and came to a decision that implementing a system where the focus is equally distributed among auto scaling efficiency and latency would be the most balanced and suitable decision.

5.6.3 Complexity for performance but against security

We want our live streaming platform to be extremely secure. We also want our operation to be less complex. To achieving both of them at the same time, we need to make few considerations:

Identifying Objectives :

a) Extremely secure operations
b) Minimal complexity

Identifying Alternatives :

a) Implement security measures manually - Very robust system, added complexity
b) Use simple service with less granularity for control of configuration - Minimal Complexity, loose security
c) Use a hybrid model where we configure some services and use pre-configured services with them - Balanced complexity and security

Creating Consequence Table and eliminating dominated alternatives:

Alternative	Security	Complexity	Evaluation	Dominated
Security Measures Manually	High	High	Balanced	No
Use Simple configurations	Moderate	Low	Complexity based, low security	No
Hybrid	Moderate	Moderate	Balanced	No

Ranking Table :

Ranking	Alternative	Security	Complexity	Evaluation	Dominated
1	Configure Manually	High	High	Balanced	No
2	Hybrid	Moderate	Moderate	Balanced	No
3	Use Simple configurati ons	Low	Low	Complexity based, low security	No

We need strict security measures. We are willing to sacrifice complexity if that means our system is safe.

5.7. Discussion of an alternate design

Skipped

6. Kubernetes Experimentation

6.1. Experiment Design

1. **The workload:** We exposed a microservice which hosted on node js using express and mongo db. It inserts a json object into the db when a user hits that endpoint. We made a deployment for this service in kubernetes. This deployment is our primary workload for this experiment. In our case the endpoint is present at '/insertDocument' POST route and the data is passed through the response body. It is a common use case when let's say a client user sends a chat or presses the like button on a live video. In this design we are testing how we can scale server load if multiple users hit this endpoint with some defined restrictions. Also, we'll be able to see how a single database handles requests from several users.
2. **Objective:** To understand how horizontal auto scaling of pods work for a kubernetes based workload and the limits set on the extent of autoscaling and observe its behavior.
3. **Tools used:**
    a. Locust: For load generation
    b. Kubernetes: Autoscaling, pod generation and management
    c. minikube: Facilitates local kubernetes deployments
    d. Docker: To deploy our application container within pods
    e. MongoDB Atlas: Our primary database where all the json objects are stored
4. **Design Choices:**
    a. We chose to test this workload on just one kubernetes node. All the requests will be redirected through this into multiple pods.
    b. We built a deployment for this workload with the following limits mentioned in the deployment's yaml file

```
limits:
   memory: "64Mi"
   cpu: "150m"
requests:
   cpu: "50m"
   memory: "32Mi"
```

    Which means
    The limits are: CPU usage to a maximum of 150 milliCPU (mCPU) and memory usage to a maximum of 64 megabytes (MiB)

Minimum requirements are: 50 milliCPU and 32 megabytes for CPU and memory respectively

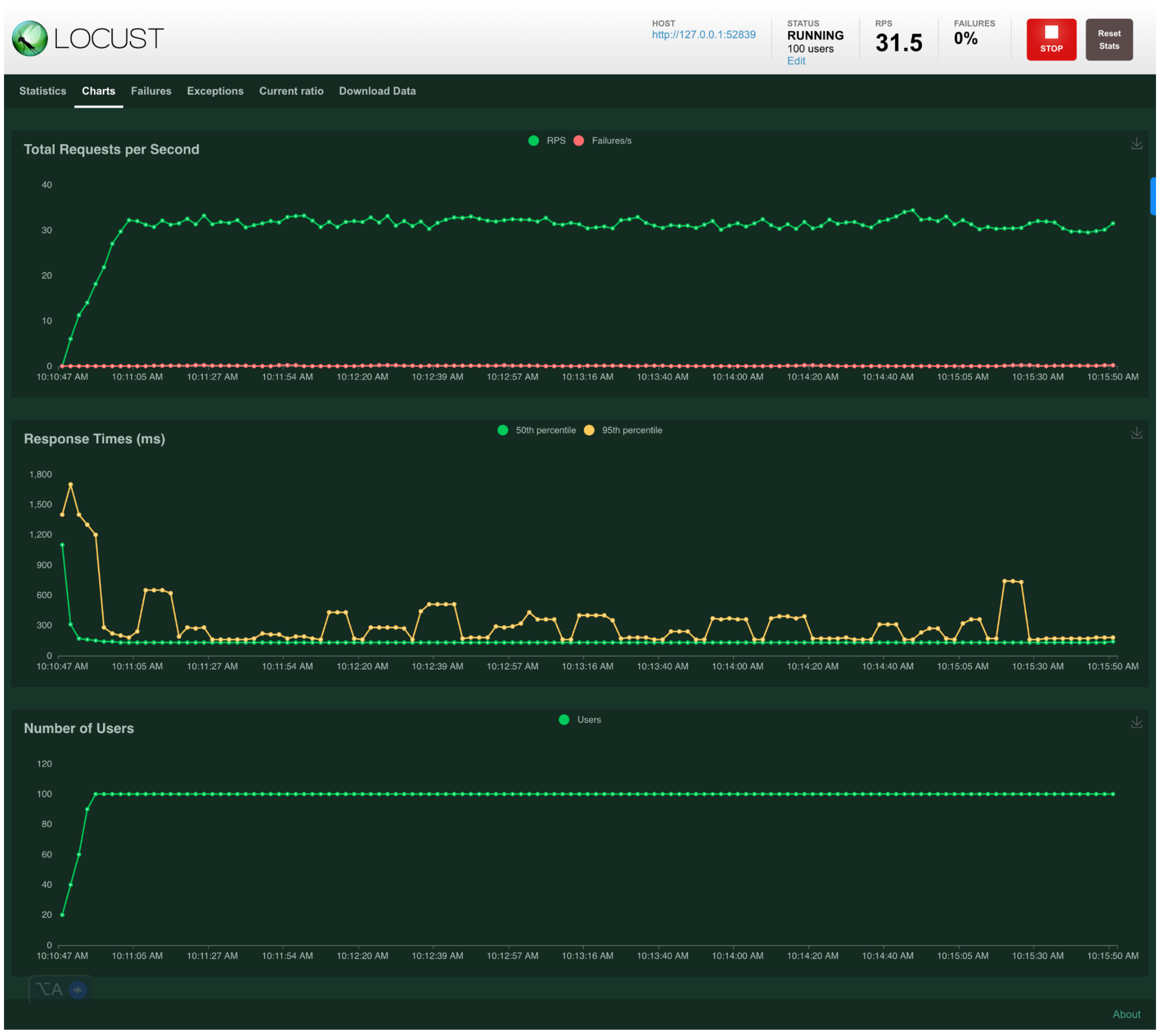
c. For load generation we decided to make use of locust, an open-source load testing tool. We limited the wait time between 1 to 5 seconds. The upper limit of number of users was 100 with a spawn rate of 10 users started per seconds. All these considerations were done keeping in mind that our MongoDB Atlas cluster had a limitation of 100 operations per second in the tier-of-choice. We aimed at reaching no more than 50 operations per second.

   The locust task would stub a json object consisting of dummy data to the database.
d. For auto scaling we used the horizontal auto scalar by kubernetes. We chose the following criteria for the pods to scale accordingly.
   i. minReplicas: 1, to ensure responsiveness if there's not much load but a spike could occur anytime.
   ii. maxReplicas: 5, Considering database limitations
   iii. Resource type being assessed - cpu, as we want to observer how the server handles requests
   iv. Target CPU Percentage: 50% utilization, as it strikes a balance between responsiveness to increased demand and resource efficiency for many workloads

5. **The workflow:**
   a. We leveraged the default internal load balancer that comes with the service that has `type: ClusterIP` specification that handles the incoming load to the node which redirects it to a pod which can handle it.
   b. Initially there will just be minimum number of replicas as there's no significant load
   c. We will use load generation using locust to build traffic on our service.
   d. We expect that the pods will increase gradually to meet the load
   e. We will cut off locust and stop generating load to see how the pods autoscale down back to its initial state.

6.2. Workload Generation with Locust

We intend to use "Locust" to generate load on a post route end point and see how the pods hosting this service autoscale by using the kubernetes horizontal autoscaler . Refer to the design choices above to see what design decisions we made for locust. For the design above we started the load generation and this is how the requests were handled:



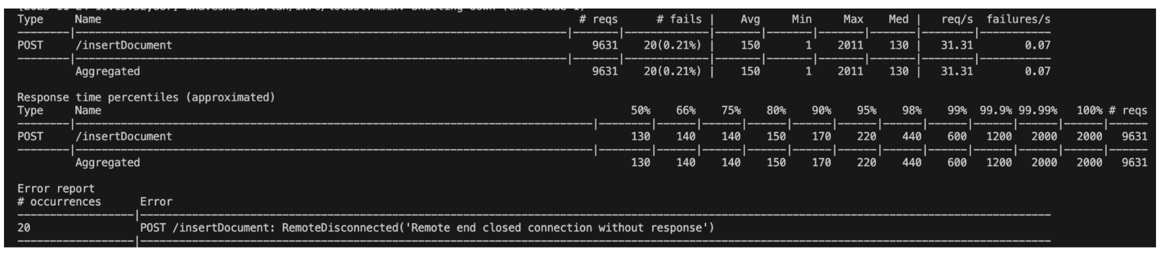
6.3. Analysis of the results

Locust:

Referring to the locust chart.
Observations:

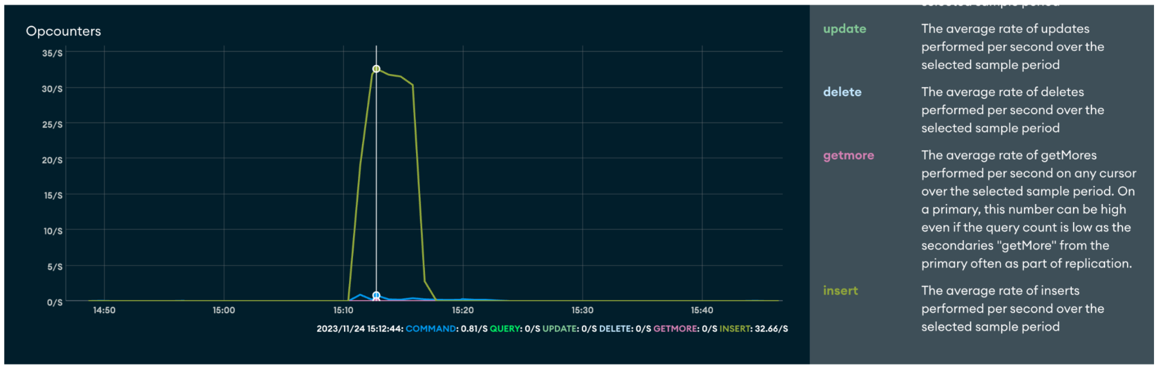
1. As per the specification of spawning 10 users per second and a limit of 100 users the number of users grew to 100.
2. Initially the response time took a hit as there was a large inflow of requests and not sufficient pods. But once the pods got established and stabilized the response time stabilized as well.
3. The requests per second rate reaches a 31 requests per second figure.

4. We stop the load generation after a while.
5. You can find the metrics report below.



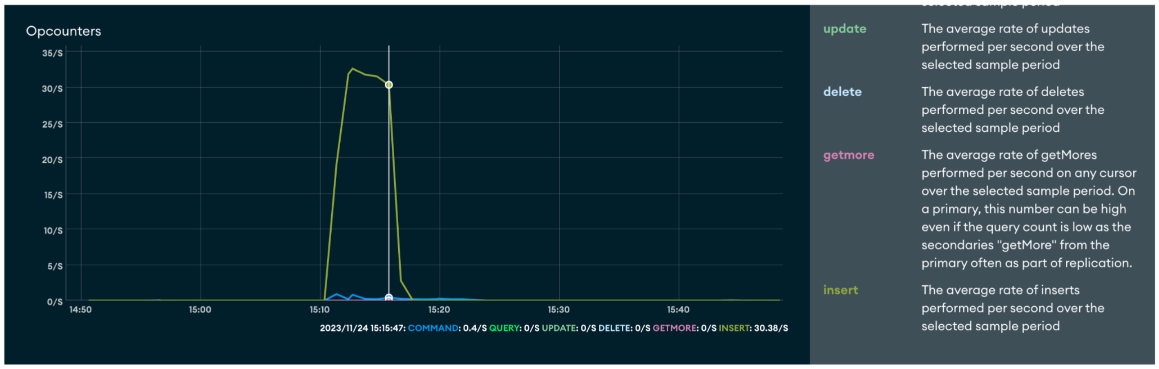
```
Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
POST     /insertDocument                                                                 9631   20(0.21%) |    150       1    2011    130 |   31.31        0.07
--------|----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                                                      9631   20(0.21%) |    150       1    2011    130 |   31.31        0.07

Response time percentiles (approximated)
Type     Name                                                                                  50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|--------------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
POST     /insertDocument                                                                       130    140    140    150    170    220    440    600   1200   2000   2000   9631
--------|--------------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
         Aggregated                                                                            130    140    140    150    170    220    440    600   1200   2000   2000   9631

Error report
# occurrences      Error
------------------|-------------------------------------------------------------------------------------------------------------------------------------------
20                 POST /insertDocument: RemoteDisconnected('Remote end closed connection without response')
------------------|-------------------------------------------------------------------------------------------------------------------------------------------
```

Database:



2023/11/24 15:12:44: COMMAND: 0.81/S QUERY: 0/S UPDATE: 0/S DELETE: 0/S GETMORE: 0/S INSERT: 32.66/S

update	The average rate of updates performed per second over the selected sample period
delete	The average rate of deletes performed per second over the selected sample period
getmore	The average rate of getMores performed per second on any cursor over the selected sample period. On a primary, this number can be high even if the query count is low as the secondaries "getMore" from the primary often as part of replication.
insert	The average rate of inserts performed per second over the selected sample period



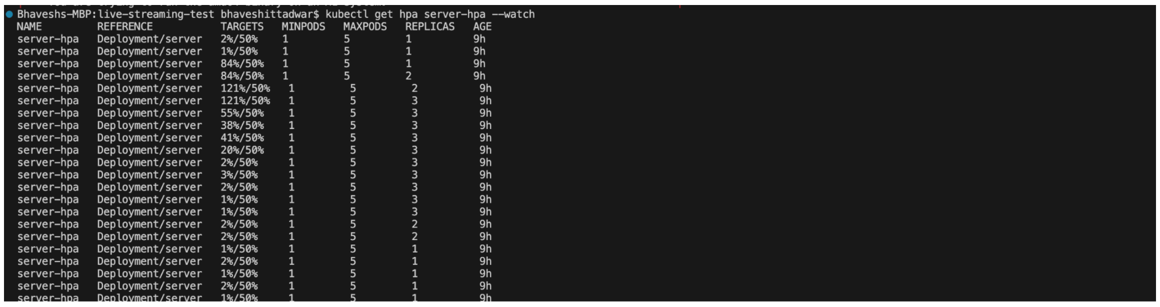
2023/11/24 15:15:47: COMMAND: 0.4/S QUERY: 0/S UPDATE: 0/S DELETE: 0/S GETMORE: 0/S INSERT: 30.38/S

update	The average rate of updates performed per second over the selected sample period
delete	The average rate of deletes performed per second over the selected sample period
getmore	The average rate of getMores performed per second on any cursor over the selected sample period. On a primary, this number can be high even if the query count is low as the secondaries "getMore" from the primary often as part of replication.
insert	The average rate of inserts performed per second over the selected sample period

Observations:

1. The database gets to approximately 32 insert operations per second. This is equitable with the locust's requests per second value indicating that the requests were successful. This is right below our constraint of 50 operations per second.
2. When we stop the locust load generation, we can observe the down spike in insertion operations in the second diagram.

Horizontal AutoScaler:



```
Bhaveshs-MBP:live-streaming-test bhaveshittadwar$ kubectl get hpa server-hpa --watch
NAME         REFERENCE           TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
server-hpa   Deployment/server   2%/50%     1         5         1          9h
server-hpa   Deployment/server   1%/50%     1         5         1          9h
server-hpa   Deployment/server   84%/50%    1         5         1          9h
server-hpa   Deployment/server   84%/50%    1         5         2          9h
server-hpa   Deployment/server   121%/50%    1         5         2          9h
server-hpa   Deployment/server   121%/50%    1         5         3          9h
server-hpa   Deployment/server   55%/50%     1         5         3          9h
server-hpa   Deployment/server   38%/50%     1         5         3          9h
server-hpa   Deployment/server   41%/50%     1         5         3          9h
server-hpa   Deployment/server   20%/50%     1         5         3          9h
server-hpa   Deployment/server   2%/50%      1         5         3          9h
server-hpa   Deployment/server   3%/50%      1         5         3          9h
server-hpa   Deployment/server   2%/50%      1         5         3          9h
server-hpa   Deployment/server   1%/50%      1         5         3          9h
server-hpa   Deployment/server   1%/50%      1         5         3          9h
server-hpa   Deployment/server   2%/50%      1         5         2          9h
server-hpa   Deployment/server   2%/50%      1         5         2          9h
server-hpa   Deployment/server   1%/50%      1         5         1          9h
server-hpa   Deployment/server   2%/50%      1         5         1          9h
server-hpa   Deployment/server   1%/50%      1         5         1          9h
server-hpa   Deployment/server   2%/50%      1         5         1          9h
server-hpa   Deployment/server   1%/50%      1         5         1          9h
```

Observations:

1. It is evident from the watch logs of the horizontal autoscaler that initially the number of replicas were 1 and cpu utilization was minimal(1-2%).
2. When we generated load using locust. You can observe a spike in the number of replicas being spawned. It went from 1 replica to 2 and finally settled at 3. We noticed that spawning new pods takes some time and affirmed that it'd be better to have at least one replica running to handle initial spikes. Not having any replicas might result in failed requests due to the delays in pod set up.
3. A similar trend could be observed with CPU utilization. This, in fact, is rather not the effect of change in number of replicas but a causation of it. As the CPU utilization crosses the value of 50%, the kubernetes HPA starts spinning up new pods to handle the load.
4. Finally, when the locust load generation is halted, eventually first the CPU utilization drops down and subsequently the number of replicas settle down to the minimum number of replicas that we had set i.e. 1.

7. Ansible Playbooks

Skipped

8. Demonstration

Skipped

9. Comparisons

Skipped

10. Conclusion

10.1. Lessons Learned

This project helped us understand the importance of fundamentals and how they can be our crutch while deciding between nuanced choices where there's no clear winner. We realized that cloud services are just tools & options and a good architect brings life to these by making the best choice they can make for the scenario they are presented with. With methodologies and best practices explored right from design principles to best practices and pillars from the amazon well architected framework to diving deep into services documentation to understand if they are the right fit or not. We explored a vast spectrum of decision-making anyone working with the cloud wearing any hat might have to make while utilizing methods for analyzing tradeoffs and making informed decisions. It was a great mix of brainstorming and applying the concepts practically to assess if the decisions we made held true. Overall, it was a great learning experience.

10.2. Possible continuation of the project

Unfortunately, we don't plan to take an independent study or the advanced course offered in the next semester. So we will not be able to continue this project ahead.

11. References

[1] AWS Well-Architected Framework, Disaster Recovery (DR) objectives, https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/disaster-recovery-dr-objectives.html

[2] Amazon Well-Architected Framework, Publication date: December 2, 2021 (Document Revisions (p. 46)), Security Section, https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html

[3] Amazon Maintains Lead in the Cloud Market By Felix Richter https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

[4] Amazon Web Services, What we do https://www.aboutamazon.com/what-we-do/amazon-web-services

[5] Google Cloud products, https://cloud.google.com/products

[6] What is Azure? https://azure.microsoft.com/en-us/resources/cloud-co

[7] AWS Global Infrastructure, https://aws.amazon.com/about-aws/global-infrastructure/

[8] Google Cloud, Cloud Locations https://cloud.google.com/about/locations

[9] Microsoft Azure, Products available by region https://azure.microsoft.com/en-us/explore/global-infrastructure/products-by-region/

[10] How Twitch built the global live streaming network that powers Amazon IVS, AWS for M&E Blog, https://aws.amazon.com/blogs/media/how-twitch-built-the-global-live-streaming-network-that-powers-amazon-ivs/

[11] Elastic Load Balancing Documentation, https://docs.aws.amazon.com/elasticloadbalancing/

[12] Amazon Elastic Compute Cloud Documentation, https://docs.aws.amazon.com/ec2/

[13] Amazon CloudWatch Documentation, https://docs.aws.amazon.com/cloudwatch/

[14] What is Amazon Rekognition?, https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html

[15] DynamoDB encryption at rest, https://aws.amazon.com/s3/security/?nc=sn&loc=5 , https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html

[16] Data protection in Amazon EC2, Encryption in transit https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/data-protection.html#encryption-transit

[17] Amazon S3 Storage Classes, Performance across the S3 Storage Classes https://aws.amazon.com/s3/storage-classes/

[18] Amazon DynamoDB features, https://aws.amazon.com/dynamodb/features/

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

[20] Amazon Well-Architected Framework, Publication date: December 2, 2021 (Document Revisions)

[21] Reliability Pillar - AWS Well-Architected Framework,
https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html
[22] Sustainability Pillar - AWS Well-Architected Framework,
https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html
[23] OpenAI. (2023). ChatGPT [Large language model]. https://chat.openai.com
[24] Logging IP traffic using VPC Flow Logs
https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html
[25] Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch
https://docs.aws.amazon.com/streams/latest/dev/monitoring-with-cloudwatch.html
[26] Amazon S3 Glacier
https://docs.aws.amazon.com/amazonglacier/latest/dev/introduction.html
[27] YV and YP, "ECE547/CSC547 class notes", Example 4.18. 11 Design Principles for AWS Cloud Architecture. Core Principles.
[28] YV and YP, "ECE547/CSC547 class notes", Example 4.15. 11 Design Principles for AWS Cloud Architecture
[29] AWS Well-Architected Framework, Design Principles
https://docs.aws.amazon.com/wellarchitected/latest/framework/oe-design-principles.html
[30] Even Swaps: A Rational Method for Making Trade-offs
https://hbr.org/1998/03/even-swaps-a-rational-method-for-making-trade-offs