

Finish working on the instructions. I will provide feedback on all of them at that time.

ECE547/CSC547 Cloud Computing



**Fall 2022**

**Viranchee Lotia and Priyanka Arghode**

**August 22, 2022**

<b>1 Introduction [0%]</b>	<b>4</b>
1.1 Motivation	4
<b>2 Problem Description [0%]</b>	<b>5</b>
2.1 The problem	5
2.2 Business Requirements	5
2.3 Technical Requirements	5
2.4 Tradeoffs TR3 conflicts with TR9	7
<b>3 Provider Selection [0%]</b>	<b>9</b>
3.1 Criteria for choosing a provider YV: Do only 5 out of 7	9
3.2 Provider Comparison	9
3.3 The final selection	12
3.3.1 The list of services offered by the winner	12
<b>4 The first design draft [0%]</b>	<b>15</b>
4.1 The basic building blocks of the design	16
4.2 Top-level, informal validation of the design	17
4.3 Action items and rough timeline	18
<b>5 The second design [0%]</b>	<b>18</b>
5.1 Use of the Well-Architected framework	18
5.2 Discussion of pillars	19
5.3 Use of Cloudformation diagrams	19
5.4 Validation of the design	21
5.5 Design principles and best practices used	21
5.6 Tradeoffs revisited	22
5.7 Discussion of an alternate design	22
<b>6 Kubernetes experimentation [0%]</b>	<b>25</b>
6.1 Experiment Design	25

<b>6.2 Workload generation with Locust</b>	<b>25</b>
<b>6.3 Analysis of the results</b>	<b>25</b>
<b>7 Ansible playbooks [0%]</b>	<b>26</b>
<b>8 Demonstration [0%]</b>	<b>27</b>
<b>9 Comparisons [0%]</b>	<b>27</b>
<b>10 Conclusion [0%]</b>	<b>28</b>
<b>10.1 The lessons learned</b>	<b>28</b>
<b>10.2 Possible continuation of the project</b>	<b>28</b>
<b>11 References</b>	<b>29</b>

# 1 Introduction [0%]

## 1.1 Motivation

Our motivation is to create *online learning and development platform* to up-skill an individual's career in cutting-edge technologies and instructors who want to advance in their teaching career can refer to our model. Starting up with this small-scale project we want to gain experience as a designer and implement our learnings on how we reduce the pain of hosting and managing services with optimized cost options.

To achieve this, we are referring to <https://www.pointfree.co> [18].

## 1.2 Executive summary

Two Software Developers working at "Kickstarter," quit their full-time jobs to start a subscription-based video series explaining Functional Programming in the Swift programming language. They named it pointfree. Co. They drive 8k+ Twitter followers (<https://twitter.com/pointfreeco>)[19].

They architected the whole infrastructure and deployed their websites. The fascinating fact is that they made it using Swift.

We bring motivation by looking at individuals who transitioned full-time to content production like

1. RayWenderlich.com [20]
2. Objc.io [21]
3. HackingWithSwift.com [22]

With the basis of the established project "pointfree.co [18]," we will expose ourselves to the design complexities involved.

Applying our learnings from the cloud computing class, We plan to design an app that has hosted services supporting several users by deploying a cloud solution. After comparing cloud providers and their services that meet our criteria, we will declare the cloud provider winner that we have selected to deploy our app.

## 2 Problem Description [0%]

### 2.1 The problem

The subscription model that we are planning to provide will allow instructors to create educational content and earn money on subscriptions. With this proposal, we plan to address a High-Level Problem of *educating knowledge seekers via pre-recorded Videos and Blogs*.

### 2.2 Business Requirements

BR 1. Scalable Storage

BR 2. High-performance

BR 3. High availability: Service up-time 99%

BR 4. Cost optimization

BR 5. High security and data integrity

BR 6. Automation

BR 7. Ease of administration

BR 8. Multi-Factor Authentication

BR 9. High reliability

### 2.3 Technical Requirements

**Mandatory TR1:** Tenant Identification for **security**: Identify the tenant who is requesting the data from the application. Depending on the tenant we want to restrict certain functionalities of our application, a subscribed user or a free user, which subscribed user. The user pays a fixed monthly cost for having access to the videos. This TR helps Identify patterns among tenants. We intend to fulfill [BR5] [BR8] with this and follow the AWS pillar of Security [23].

**Mandatory TR2: Monitor** resources for CPU utilization, and application processes, setting up alerts on events. Timely monitoring resources can remediate degradation issues and deploying applications in different regions can help in workload balancing. Identifying if a tenant is gaming the system. Identifying how many simultaneous users are accessing the resources at a given time, and noticing patterns. That's how we accommodate Automation[BR6], Availability [BR3] [33], Reliability[BR9] [27].

**TR3:** For BR1 **Scalable Storage** [34]- (increasing 3.2 GB/month) Storage and access to stored files is a crucial requirement. For our model, we see that our app will require:

- a. Static site Web hosting (500 kB static site + increasing 100 kB/month for blogs)
- b. Image hosting (increasing 1 MB / month or 16 images/month)
- c. As we see the growing data, we would like it to be secure and scalable.
- d. Video conversion of 400 MB 1080p to 720p 480p and 240p formats
- e. Video hosting (increasing 3.2 GB/month or 240 minutes/month)
- f. Bandwidth to serve up to 1000 streams of a single video for subscribers

**TR4:** For BR2 **High performance** [26]- High performance and user quality experience can be maintained by:

- a. Frequent deployments of static websites for updating bug fixes, promotions, and adding content.
- b. Static site hosted over the edge for faster response times for consumers.
- c. Static site deployed over CDN
- d. Managing server workload balance to ensure it's below the threshold.
- e. Network monitoring to determine the best bandwidth supported by customers and serve them the highest quality video format.
- f. Auto upscaling/downscaling video quality when the user's network fluctuates.
- g. Video distribution to multiple simultaneous users.

We refer to AWS pillar Performance Efficiency[26] to meet this.

**TR5:** For BR3: **High availability** [33] - We plan to achieve service availability of 99.95% by following DR or failover capacity. Also, monitoring is required to ensure service is accessible.

**TR6:** For BR4: **Cost Optimization**- Reduced cost for traffic increase, by discount or tiered pricing which decreases with traffic. CDN and edge deployment for faster response times to consumers. We plan to ensure AWS pillar Cost Optimization [28] is followed when implementing.

**TR 7: For BR5: High Security and Data Integrity-**

- a. We plan to have the video encrypted during transit.
  - b. Video unavailable with direct link for a user, only via streams to the website.
  - c. Automated Authentication key rotation per user per periodic cycle
  - d. GDPR and Government compliance: Encrypt and secure user information in transit and storage in the cloud.
  - e. DDoS Mitigation on attacks
  - f. Multi-Factor Authentication
- We plan to follow AWS pillar Security [23] to achieve this.

**TR 8: For BR6: Automation:** Service providers actively mitigating DDoS and the cost increase due to the same. Trace video traffic to users, classifying subscribers to video-watching behavior. We plan to follow the AWS pillar of Security [23] and Performance Efficiency[26] to achieve this.

**TR 9: For BR7: Ease of Administration and BR4 Cost Optimization [28]:** We plan to have a configurable application to incorporate the business need in case of a growing subscriber count/needs.

**TR 10: For BR9: High Reliability:** We expect our model to be 99.99% reliable by ensuring a workload balance by distributing it across multiple resources. With the design principles outlined under AWS pillar Reliability [27], we plan to operate and test the application workload through.

**TR 11: Low Latency[26] :** The website should be fast to load on a client. The time to first-byte metric should be low, less than 200 milliseconds. We cover BR2 under this.

## 2.4 Tradeoffs TR3 conflicts with TR9

TR5 (High Availability) conflicts with TR2 (Monitoring): The data analysis offered takes 60 seconds time on the cloud service provider and is not real-time.

TR9 (Ease of Administration) conflicts with TR7 (High Security and Data Integrity) : We use AWS Cognito and AWS Secrets Manager service specifically for High Security and Data integrity. Every increase in service increases the complexity and administration efforts of an architecture.

TR6 (Operation costs) conflicts with TR1 (Tenant Identification): Cloudflare costs are 4.5x cheaper than AWS, however, Cloudflare does not yet offer Tenant Identification support for their services.

Limit scaling to 1k simultaneous users The assumption is, there are limited iOS developers interested in functional programming, and paying to learn from our service—high quality, limited quantity user-base, limited concurrent viewership, of the same video. Scale up needed only at the time of the new video release.



TR9 (Ease of Administration) conflicts with TR10 (High Reliability): Reducing variables inside the cloud reduces administration requirements. However, to increase the proper functioning and reliability of the service, we need to add services like Route 53.

TR4 (High Performance) conflicts with TR 5 (High Availability): Utilizing all the services at all times, AWS provides better availability by load balancing and scalability.

### 3 Provider Selection [0%]

We analyzed and studied the cloud services provided by: Amazon Web Services (AWS) [16] , Cloudflare [17] and Microsoft Azure, and Google Cloud.

#### 3.1 Criteria for choosing a provider YV: Do only 5 out of 7

1. Cost: static web hosting, video hosting, Outbound streaming traffic, CDN / Global distribution, cost distribution analysis, video conversion
2. Scalability
3. DDoS Protection, Mitigation, and cost for if under DDoS
4. Security- Signed Private Video URL, limiting piracy on the user (tenants)
5. Performance- Video bitrate adaptation support

#### 3.2 Provider Comparison

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

Analysis for 5000 subscribers and 260 videos:

Criteria for choosing	AWS	CloudFlare	Azure	GCP
Cost	\$1,750	\$375	\$1,500	> \$1700
Scalability	✓	🔥	✓	
DDoS Protection, Web Security	🔥	🔥	🔥	
Signed Private Video URL / tenant	✓	✓	✓	
Tenant Usage Monitoring	✓	✗	✓	
Video bitrate adaptation support	✓	✓	✓	

- **Scalability comparison:**  
Solutions: CDN, Horizontal scaling of servers across the globe.  
AWS, Azure, and GCP allow for providing Kubernetes scripts that handle the scaling of servers.  
Cloudflare handles and scales the application without any Kubernetes requirements.
- **DDoS Protection and Web security:** AWS, Cloudflare, and Azure offer free DDoS mitigation and don't charge users for extra load when they are attacked.
- **Signed Private URL per Tenant:** All service providers offer to create a tokened URL that can be used/shared by clients
- **Tenant Usage Monitoring:** AWS and Azure allow monitoring of network usage per tenant. Cloudflare does not yet support per-tenant monitoring

- **Video bitrate adaptation support:** The video bitrate adaptation can be configured by developers
- **Cost Comparison:** To compare various providers, the TRs were evaluated to actual numbers for 1 day, 1 month, 1 year, 2 years, and 5 years. Overall, the distribution cost eclipsed other costs.

Constants	
Video Minutes	60
Size per video (MB)	400
Size per video in all formats	800
Video watch / sub / month	10
1 GB to MB	1024

Quantity	1 day	1 month	1 year	2 years	5 years
Videos	1	4	52	104	260
Subscribers	1	10	100	1000	5000
Traffic (Tb/month)	0	0.04	0.4	4	20

AWS requires using 11 services at least for Video-on-demand applications. Operational costs would be roughly \$1700 at 5000 subscribers and 260 videos (which is the current subscriber and video count for the real service)

<b>Amazon using given architecture</b>					
<b>Amazon Cognito</b>	\$0	\$0	\$0	\$0	\$0
<b>AWS AppSync</b>	\$4.09	\$4.36	\$7.69	\$11.38	\$22.36
<b>AWS S3</b>	\$0.02	\$0.05	\$0.48	\$0.94	\$2.35
<b>AWS Lambda</b>	\$0.00	\$0.00	\$0.03	\$0.07	\$0.17
<b>AWS DynamoDB</b>	\$1.00	\$1.00	\$4.00	\$8.00	\$13.00
<b>AWS Secrets Manager</b>	\$0.01	\$0.13	\$1.33	\$13.33	\$66.67
<b>AWS Elemental MediaConvert</b>	\$10	\$10	\$10	\$10	\$10
<b>Amazon CloudFront</b>	\$0.00	\$0.00	\$0.00	\$244.00	\$1,526.00
<b>AWS Amplify</b>	\$65	\$65	\$65	\$65	\$65
<b>AWS CloudFormation</b>	\$5	\$5	\$5	\$5	\$5
<b>Total AWS Costs</b>	\$85.13	\$85.54	\$93.54	\$357.72	\$1,710.54

Calculating for Cloudflare using the Stream Creator Bundle service results in costs of 375\$ for the same 5000 subscribers and 260 videos. Note: For 1 day and 1 month, Cloudflare costs can go 10\$ as well.

<b>Cloudflare Stream Stream</b>	0	0	1	55	295
<b>Cloudflare Stream Storage</b>	\$50	\$50	\$50	\$50	\$80
<b>Total Cloudflare Costs</b>	50	50	51	105	375

That is a >4.5x difference in cost. And AWS's cost model could increase if more services are added.

Azure total cost for deploying 5000 subscribers and 260 videos is ~\$1500.

<b>Azure</b>					
<b>CDN</b>					\$1,469
<b>Storage</b>					\$5.20
<b>Encoding</b>					4.5
<b>Licensing</b>					10
<b>Azure total costs</b>					\$1,489

90+% of the costs are in distribution via CDN. When the service is small, there are fixed and low costs, but as the service consumption and network bandwidth increase, the service cost rises.

Since CDN cost is the elephant in the room, we will compare Google Cloud (GCP) CDN services only. GCP charges for 2 services, GCP Storage -> GCP CDN (cache refill) and GCP CDN -> Consumers (CDN cache egress). We only took into consideration GCP Egress bandwidth for a quick number. Just GCP CDN Egress with 5000 subscribers costs roughly similar to the Full AWS Deploy cost.

<b>GCP</b>					1689+
------------	--	--	--	--	-------

### Cloudflare Pros:

DDoS Protection, Faster network request processing, Early Hints, Virtualization with V8 engine instead of Docker (faster start times with similar security), Global CDN coverage, Security (Web application firewall)

### Cloudflare Cons:

No individual tenant monitoring service. The support is per video monitoring

### Benefits of using AWS:

Allows custom configuration inside the architecture, Kubernetes deployment

## 3.3 The final selection

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

Upon Cloud Comparison with respect to providers AWS, Cloudflare, and Azure, **AWS** is the **winner**.

We incline towards AWS mainly for Geographically dispersed data center options. (For disaster recovery, availability, and performance); Scalability for increased baseline workload, Offered Security and Reliability.



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

Technical Requirements listed under Section 2.3 can be fulfilled using below AWS services to ensure scalable, resilient and highly available architecture:

- **Amazon Cognito [32]:** This is an AWS webservice used for authentication and authorization using cognito user pool and cognito identity pool.
- **Amazon Route 53 [1]:** This is a highly available and scalable DNS web service that connects user requests to internet applications running on AWS or on-premises.
- **AWS Shield [6] :** It is a managed service that provides protection against Distributed Denial of Service (DDoS) attacks for applications running on AWS. All AWS customers get AWS Shield Standard automatically enabled to all AWS customers at no additional cost. Full benefits of AWS Shield Standard can be availed by following the best practices of DDoS resiliency on AWS.
- **AWS API Gateway:** It is an AWS service used for creating, publishing, maintaining, monitoring, and securing API's. For RESTful API's that are HTTP based. this service enables stateless client-server communication and also can implement standard HTTP methods. WebSocket APIs, enable stateful, full-duplex communication between client and server. Also, this service routes incoming messages based on message content. [3]
- **Amazon CloudFront [5]:** This is a CDN service that caches and serves dynamic content. It can help our app in delivering high performance with low latency and high reliability.

- **AWS Cloudwatch [2]:** Cloudwatch service is used for proactive application monitoring so that one can gain system-wide visibility into resource utilization, application performance, and operational health.
- **AWS CloudTrail [7]:** This service can be integrated into the application using API's that help to enable operational and risk auditing, governance, and compliance of AWS accounts. All the actions are recorded as events in CloudTrail.
- **Amazon S3 [8] :** It's an object storage service that will help in offering scalability, data availability, security, and performance. Also, the offered management features can help in optimizing, organizing, and configuring data access to meet specific business requirements.
- **AWS VPC [10]:** VPC enables the launch of AWS resources into a virtual network that includes features like VPC, Once VPC is created, one can add subnets followed by assigning IPv4 and IPv6 IP addresses. Other features that can help in configuring VPC are routing, configuring gateways and endpoints, peering connections, traffic mirroring, transit gateways, VPC flow logs, and VPN connections.
- All the payments should be done securely so **AWS Security Hub [12]** service performs security best practice checks, aggregates alerts, and enables remediation automatically.
- **Amazon GuardDuty [11] :** When the GuardDuty service integration is enabled, it sends all the findings to Security Hub using ASFF (AWS security Finding Format).
- **Amazon Inspector [14]:** Manages vulnerability by scanning AWS workload continuously.
- **Amazon DynamoDB [29]:** It's a managed, serverless, and NoSQL database designed to run high-performance applications at any scale. DynamoDB provides built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data import and export tools.
- **AWS LAMBDA [30]:** It is a computing service used to run code without provisioning or managing servers. Lambda runs a code on high-availability compute infrastructure and performs the administration of the compute resources.
- **AWS WAF[4]:** It is a web application firewall service that can monitor the HTTP(S) requests that are forwarded to the protected web application resources. It also provides control access to application content.
- Services like **Amazon CloudFront [5]** and **AWS shield [6]** can bring efficient, low latency, and highly secured payment transactions.
- **AWS Elemental MediaConvert [31]:** It is used for file-based video processing and allows to easy create video-on-demand content for broadcast and multiscreen delivery at scale.

## 4 The first design draft [0%]

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

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

### **(Short list of TRs against which the design is going to be). DESIGN CF TRADEOFF AND DESIGN**

Our design using AWS scalable solutions provisions the AWS services automatically. is designed to ensure the Tenant-Identification [TR1], Monitoring [TR2], Auto-Scalable [TR3], high performance [TR4], Availability [TR5], Network Security [TR7], and Cost Efficient [TR6].

The user and Admin will use Amazon Cognito [32] which will authenticate through the Cognito User pool. Route 53 [1] obtains the website's DNS name, and all the website traffic will get routed via Route 53. The website is accessed through an API gateway.

Admin will be able to store educational videos in an S3 bucket via serverless Lambda.

CloudFront will cache the videos from the S3 bucket and will be delivered securely and quickly to the users.

The website is accessed using an API gateway. Admin will be able to store educational videos in an S3 bucket via serverless Lambda.

AWS Elemental MediaConvert takes a video file input from an S3 bucket, then transcodes file-based content into a live stream quickly and reliably and makes it available to watch on varying resolutions. These videos will be stored in an S3 bucket.

Thus our design takes source videos and processes them on a wide range of devices, stores the transcoded files, and delivers them to end users through CloudFront.

We will see this in detail in section 4.2.

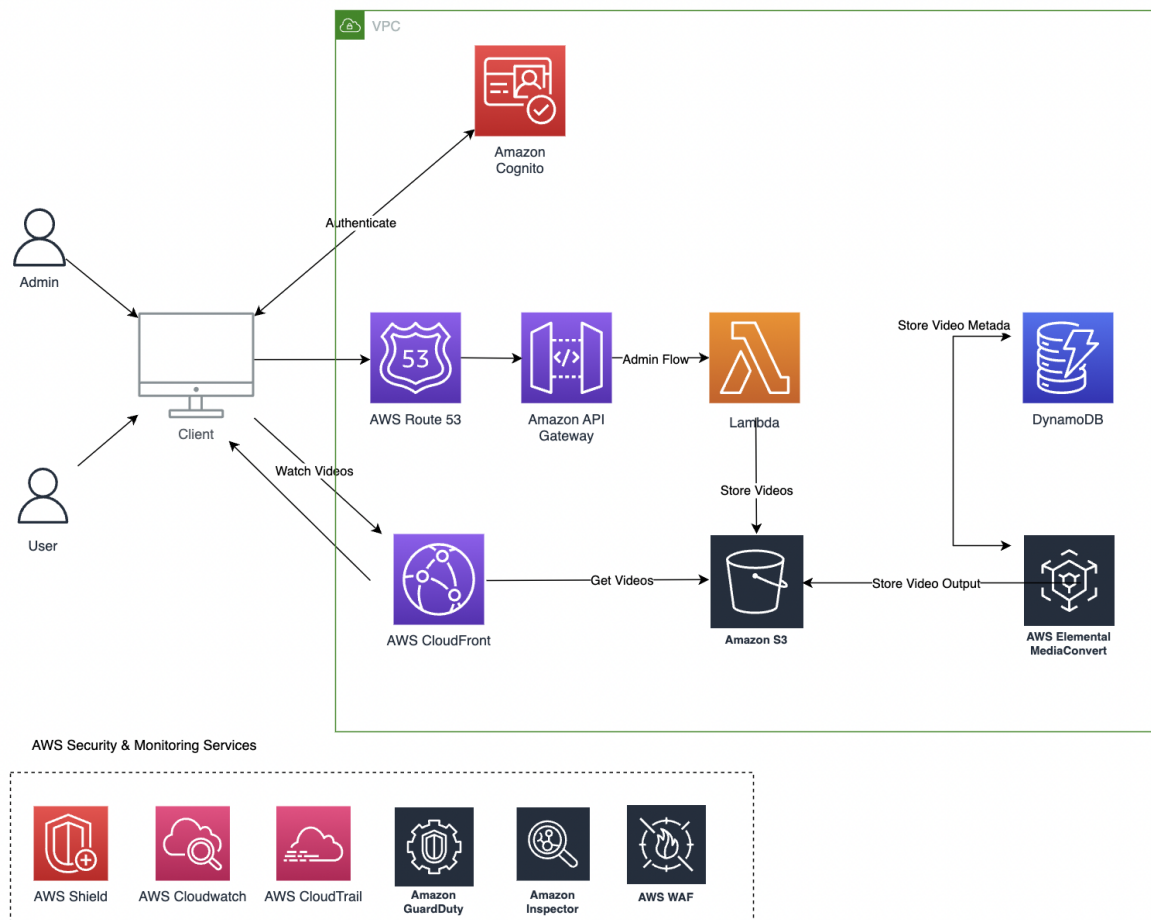


Figure 1: Architecture diagram

## 4.1 The basic building blocks of the design

Have you identified any distinct elements or features of your solution? [List them here.](#)

- Storage
  - AWS S3
- Networking
  - Amazon Route53
  - AWS IAM
  - AWS API Gateway
  - AWS Shield
  - AWS VPC
  - Amazon Cloudfront



- Amazon GuardDuty
- Amazon Inspector
- Amazon WAF
- Amazon Cognito
- Managed Database
  - Amazon Dynamodb
- Computation
  - Amazon CloudFront
  - AWS Lambda
  - AWS Elemental MediaConvert
  - Monitoring
    - AWS CloudWatch
    - AWS CloudTrail

## 4.2 Top-level, informal validation of the design

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

**Tenant Identification [TR1]:** The user and Admin will use **Amazon Cognito** which will authenticate through the Cognito User pool. Amazon Cognito user pool will take care of user authentication. **MFA** adding to the user pool will **secure** the authentication process and the Cognito Identity pool will ensure access control by providing single-sign-on access to AWS resources like DynamoDB, S3 Bucket, and AWS Lambda serverless solution. This is how our solution meets the mandatory Technical requirement of tenant Identification.

Our design fulfills the mandatory technical requirement i.e. **monitoring [TR2]** by using AWS services like **AWS CloudWatch** and **CloudTrail**. As we propose to ensure 99.95% **availability [TR5]** and **high performance [TR4]**, CloudWatch will monitor application and infrastructure performance in the AWS environment. We propose to monitor bandwidth utilization, CPU, and traffic parameters and setting alerts and alarms on the threshold. Whereas AWS CloudTrail will alert in case of **security** is compromised by an attacker with the help of CloudTrail data log.

**Route 53** obtains the website's DNS name, and all the website traffic will get routed via Route 53. This is a highly scalable service and can be used for **load balancing [TR4]** based on geographical topology and latency. Route 53 uses geolocation and routing policies. The website is accessed through an **API gateway** which manages and balances load based on network traffic.

Admin will be able to store educational videos in an **S3 bucket** via serverless Lambda. CloudFront will cache the videos from the S3 bucket and will be delivered securely and quickly **[TR4- performance]** to the users. Amazon S3 provides strong encryption features and access management tools that are **highly secure** from unauthorized access. Also, the S3 bucket will grow in storage size automatically as the number of objects within the s3 bucket increase. Thus meeting **Auto-scalable [TR3]** Technical Requirements.

**AWS Elemental MediaConvert** takes a video file input from an S3 bucket, then transcodes file-based content into a live stream **quickly and reliably** and makes it available to watch on varying resolutions. These videos will be stored in an S3 bucket.

**Network Security [TR7]** is being ensured that using **AWS Shield** will protect against Distributed Denial of Service (DDoS) attacks for AWS resources at the network and transport layers. **Amazon GuardDuty** automatically updates AWS WAF to block suspicious activities and restrict application access.

Internal resource access and management is taken care by **AWS IAM [34]**

## 4.3 Action items and rough timeline

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

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

SKIPPED

## 5 The second design [0%]

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

### 5.1 Use of the Well-Architected framework

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

The AWS framework is based on six pillars:

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

## 5.2 Discussion of pillars

Discuss in detail one pillar per team member.

As the application workload increases, the system should balance efficiently by auto-scaling which is one of the AWS framework pillars i.e. **Performance efficiency**. [26]

Performance efficiency includes using computing resources efficiently to meet system requirements and maintaining that efficiency as demand changes and technologies evolve. Low application latency at a lower cost with a better experience is another measure of Performance efficiency.

Our design achieves performance efficiency with the use of AWS Cloudfront and AWS Lambda, S3 bucket for scalability.

**Security**[23]:

Tenant Identification and mapping of views to tenants. DDoS protection and mitigation.

Encrypted data at rest and data during transit using authenticated tokens. Our solution using IAM with Cognito consumer and identity pool achieves tenant identification and security while network security is taken care of by AWS shield, AWS inspector, and AWS guard duty.

## 5.3 Use of Cloudformation diagrams

Present your design using Cloudformation templates. (WHY THIS CF FORMATION

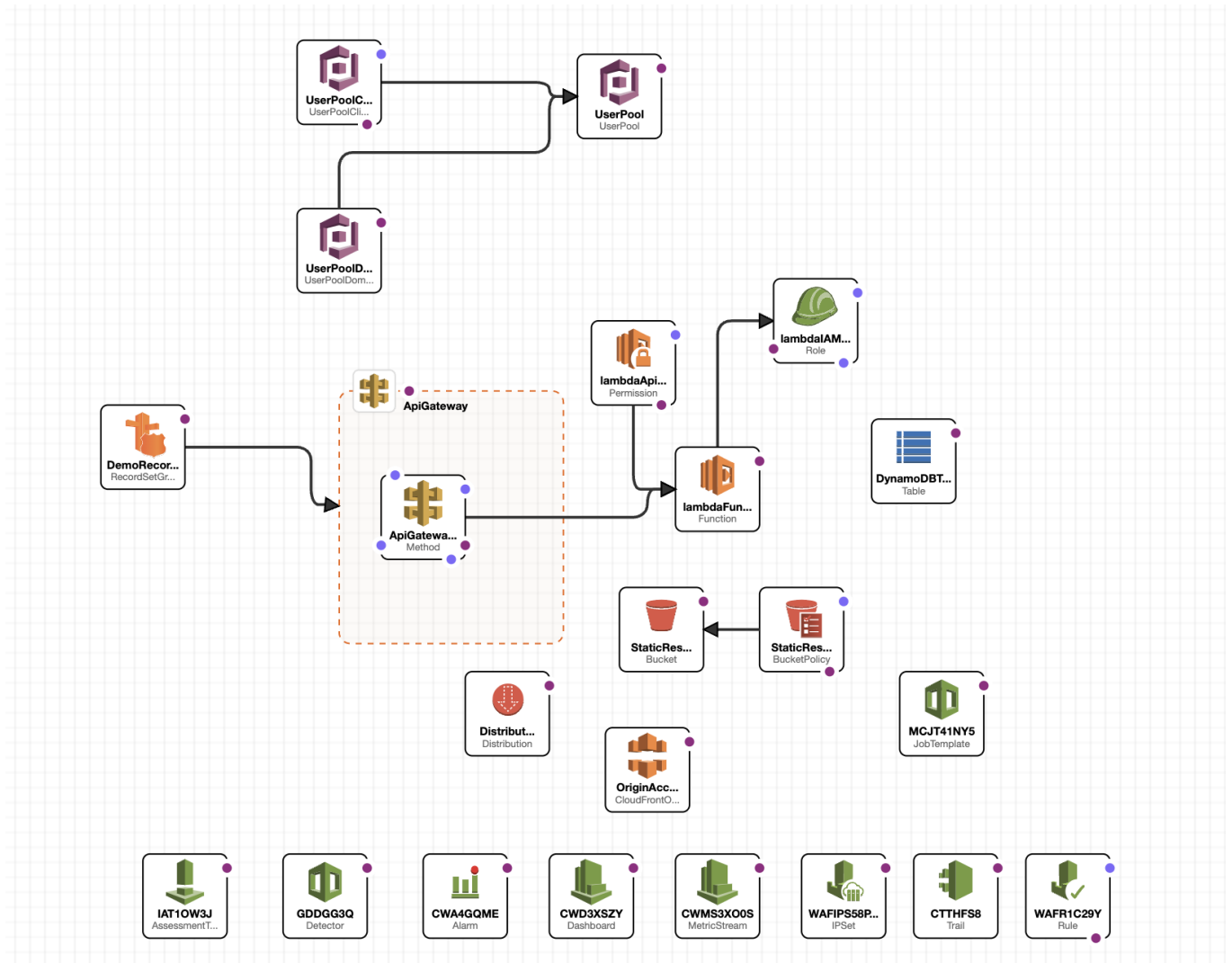


Figure 2: Cloudformation Diagram using AWS designer

The AWS CloudFormation template deploys the earlier described architecture.

Some of the high level components are explained as below:

1. An Amazon S3 bucket for source and destination media files according to the Amazon S3 lifecycle bucket policies.
2. AWS Elemental MediaConvert to transcode media files from their source format.
3. An Amazon CloudFront distribution to cache and deliver video content to end users.
4. AWS Lambda functions to compute the work and process error messages
5. An Amazon DynamoDB table stores data captured through the workflow.
6. Amazon CloudWatch for logging and Events rules for AWS Elemental MediaConvert notifications. CloudTrail for risk auditing, governance and compliance.

## 5.4 Validation of the design

Argue about how your design meets the BRs or TRs you listed in Section 2.

We have followed the AWS standard design principles enlisted under the AWS Framework pillars [25].

Our Solution is designed to ensure the Tenant-Identification [TR1] [TR7] to fulfill High security and data integrity [BR 5] using Amazon Cognito to authorize and authenticate users and admin to access the application.

We fulfill the business requirement for High Performance [BR2] and High availability [BR3] [TR5] by using AWS services like **AWS CloudWatch** and **CloudTrail** that ensures **system** monitoring [TR2] and alerts.

Amazon **Route 53** is a highly scalable service and we are using it for **load balancing** [TR4] based on geographical topology and latency thus providing high performance [BR2].

Auto-Scalable [TR3] [BR1], high performance [TR4][BR2] is taken care of by the S3 bucket using strong encryption and auto-scaling feature.

**Network Security** [TR7] [BR5] is being ensured using **AWS Shield** will protect against Distributed Denial of Service (DDoS) attacks.

## 5.5 Design principles and best practices used

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

We followed the AWS best practices and design principles [25] while designing our project.

- 1) Scalable storage for capacity needs, we used an S3 bucket.
- 2) Operational excellence by providing ease of administration, workload balance, and deploying alerts and monitoring services. This way we ensure that our application has low to no latency and auto-remediation because of the proactive monitoring. Thus we follow prepare, operate, and evolve practices.
- 3) Security to implement stringent tenant identification (IAM), secured data storage(Data Protection), and secured network (Infrastructure Protection). This way we apply security at all layers and enable traceability.
- 4) Reliability by ensuring workload architecture and failover management.
- 5) Performance efficiency by following serverless architecture and by deploying applications in multiple AWS regions.

## 5.6 Tradeoffs revisited

Provide more details on the tradeoffs you made so far<sup>2</sup>. Be as exhaustive as you wish!

<sup>2</sup>In the humble opinion of the course instructors, the greatness of an architect is judged by the tradeoffs s/he masters.

TR5 (High Availability) vs TR2 (Monitoring)  
TR4 (High Performance) vs TR5 (High Availability)  
TR6 (Cost Optimization) vs TR3 (Scalability)

To determine the workload impact, we plan to collect and evaluate metrics and end-user impact. Using load testing, we plan to improve system performance. During design, to ensure high performance we trade off consistency and durability.

Data store increases system performance, To ensure low to no customer impact, implementing data caching to improve system performance. We plan to use monitoring and system metrics to decide tradeoffs.

We also revisited the trade-off between scalability and cost; the more resources, the more will be the cost. In real-time streaming applications, variability makes a difference in how the application behaves to alerts.

The services used in section 3.3.1 ensures that the BR's and TR's are fulfilled.

## 5.7 Discussion of an alternate design

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

Alternate Design for Cost Optimization and Performance Excellence:

This design is centered around Cloudflare Stream.

Admin Flow: Upload videos to CF Stream, images to CF Images, Website to CF Pages(JAMStack) / Workers (Dynamic, FullStack)

User Flow: Request website → CF CDN → CF Pages, CF Images, CF Stream

Services:

CF Stream: Videos uploaded for the stream are converted to multiple formats and provided at CDN. Charged based on minutes of video streamed, and minutes of video uploaded. Provides a

3-5x cheaper solution than competitors. No charges for encoding and bandwidth.

CF Images: Image conversion and hosting for CDN. No charges for conversion and bandwidth.

CF Pages: Static Website hosting

CF VK: Key-Value Storage close to the edge. Useful for storing User: Tokens for Video streams

CF Workers: For generating Authentication tokens, communicating between CF Services and the user. Similar to AWS Lambda

CF CDN: Globally distributed 275 data centers, connection 50ms to 95% of the world

CF Queues: Batching logs and writing them to D1 in a single go instead of 1 per API request.

CF R2: Object Storage, similar to S3. For storing data sets

CF D1: Scalable Database to store metadata and logs.

CF DDoS: Free DDoS protection and mitigation

SSL Encryption

WebAnalytics

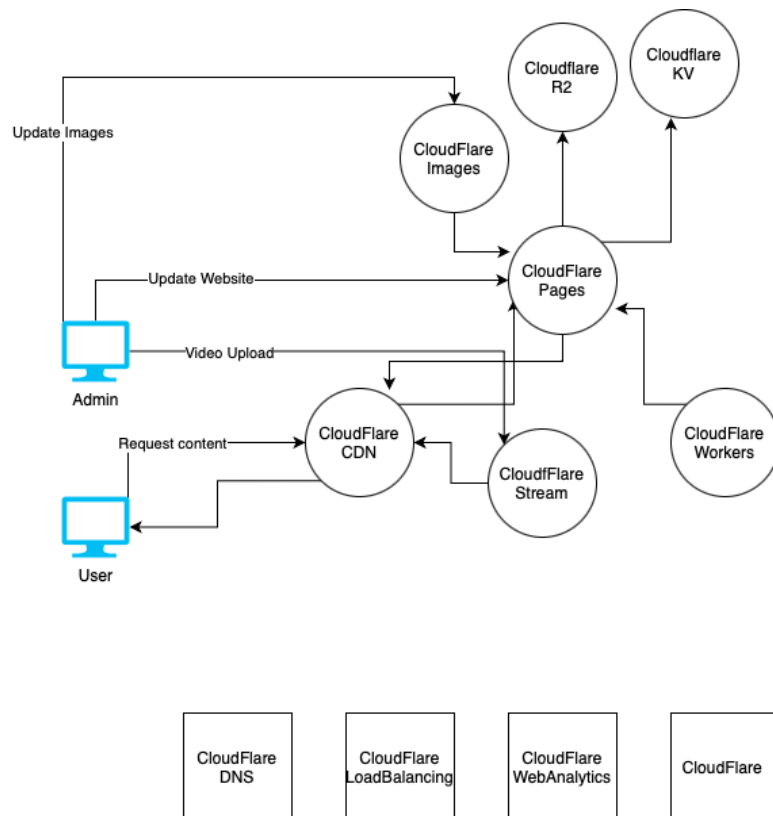
No Data Egress charges

**(NEED TO MAP TRs)**

How is it possible for Cloudflare to have competitive price, performance, and stability?

Network design at Hardware level

Compute platform: Workers run on a V8 virtualization engine. Not Docker virtualization. This takes less space, and is fast and scalable. Faster time-to-first-byte than docker.



Why not pursued:

Cloudflare (CF) based bottlenecks:

1. Tenant monitoring not yet supported (the feature is being worked on)



## 6 Kubernetes experimentation [0%]

In this section, **you'll validate some aspects of your design experimentally**. Which ones is your choice, but you must discuss with the instructors first. The idea is to limit the scope of the experiment runs and require analysis<sup>3</sup>.

### 6.1 Experiment Design

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

We test the AutoScaling aspect of Kubernetes as it applies to Horizontal Pod Autoscaling. On demand change the number of currently running clusters.

This is our Kube file:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: virwebapp
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: virwebapp
  targetCPUUtilizationPercentage: 50 # target CPU utilization
```

Repo: <https://github.com/Viranchee/ece547>

### 6.2 Workload generation with Locust

**Describe how you intend to use Locust to create (all or some of) the inputs.**

Locust creates API calls which use up the server in calculation for approximately 1-10 seconds.  
Fibonacci for values 39 - 43 work good  
Make many same calls which consume CPU resources, increasing CPU utilization.  
Example Locust File:

```
from locust import HttpUser, between, task

class WebsiteUser(HttpUser):
    wait_time = between(1, 3)

    def on_start(self):
        self.client.post("/43", { })

    @task
    def fib(self):
        self.client.get("/43")

    @task
    def fib(self):
        self.client.get("/43")
```

> locust -f locustfile.py

## 6.3 Analysis of the results

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

On start:

```
> kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
virwebapp	Deployment/virwebapp	0% / 50%	1	10		

On Load testing:

After 20 seconds

```
> kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
virwebapp	Deployment/virwebapp	50% / 50%	1	10	10	

<sup>3</sup>This part can easily turn into a full-fledged project. If that's the case, you may want to extend this to an Independent Study or conference/journal publication.

## 7 Ansible playbooks [0%]

*Skipped*

In this section, you'll define specific management tasks and write Ansible playbooks to automate them.

## 7.1 Description of management tasks

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

## 7.2 Playbook Design

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

## 7.3 Experiment runs

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

## 8 Demonstration [0%]



*Skipped*

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

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

## 9 Comparisons [0%]

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

Operations		 <b>CLOUDFLARE</b>
Latency	Low (~70ms)	Lower (~50 ms)
Administration	More number of services required	Easy integration of services
Streaming Capabilities	Provides a lot of control over data, network organization, data movement	Upto 1080p streaming without tenant identification. Has it's own solution for streaming content
Business	Precision, observability, monitoring	Better for small to medium enterprises
Security	Secured as well as rapid service distribution	Mainly focussed on network security and not on tenant monitoring
Cost	Pay as you go	Different costings as per business requirements

## 10 Conclusion [0%]

### 10.1 The lessons learned

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

The project was a great experience in terms of gaining cloud computing basic building blocks. We learned how to define business requirements and respective technical requirements considering cloud computing pillars. This exercise gave us an opportunity to learn AWS WAF. We applied all the design principles mentioned in AWS WAF. We gained knowledge on dealing with trade-offs while delivering better customer experiences. The great learning was how to select cloud providers by comparing the operations and services provided where cost plays a vital role. Overall with our no to low background in cloud computing technology, this project was a great learning experience and we are in a position to update our profile so that we have a good knowledge to work on cloud applications.

### 10.2 Possible continuation of the project

*Skipped*

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

# 11 References

Include here your references.

- [1] <https://aws.amazon.com/route53/>
- [2] [https://docs.aws.amazon.com/cloudwatch/?icmpid=docs\\_homepage\\_mgmtgov](https://docs.aws.amazon.com/cloudwatch/?icmpid=docs_homepage_mgmtgov)
- [3] [https://docs.aws.amazon.com/apigateway/?icmpid=docs\\_homepage\\_serverless](https://docs.aws.amazon.com/apigateway/?icmpid=docs_homepage_serverless)
- [4] [https://docs.aws.amazon.com/waf/?icmpid=docs\\_homepage\\_security](https://docs.aws.amazon.com/waf/?icmpid=docs_homepage_security)
- [5] <https://aws.amazon.com/cloudfront/>
- [6] [https://docs.aws.amazon.com/shield/?icmpid=docs\\_homepage\\_security](https://docs.aws.amazon.com/shield/?icmpid=docs_homepage_security)
- [7] [https://docs.aws.amazon.com/cloudtrail/?icmpid=docs\\_homepage\\_mgmtgov](https://docs.aws.amazon.com/cloudtrail/?icmpid=docs_homepage_mgmtgov)
- [8] [https://docs.aws.amazon.com/s3/?icmpid=docs\\_homepage\\_serverless](https://docs.aws.amazon.com/s3/?icmpid=docs_homepage_serverless)
- [9] [https://docs.aws.amazon.com/ec2/?icmpid=docs\\_homepage\\_featuredsvcs](https://docs.aws.amazon.com/ec2/?icmpid=docs_homepage_featuredsvcs)
- [10] [https://docs.aws.amazon.com/vpc/?icmpid=docs\\_homepage\\_featuredsvcs](https://docs.aws.amazon.com/vpc/?icmpid=docs_homepage_featuredsvcs)
- [11] [https://docs.aws.amazon.com/guardduty/?icmpid=docs\\_homepage\\_security](https://docs.aws.amazon.com/guardduty/?icmpid=docs_homepage_security)
- [12] [https://docs.aws.amazon.com/securityhub/?icmpid=docs\\_homepage\\_security](https://docs.aws.amazon.com/securityhub/?icmpid=docs_homepage_security)
- [13] <https://aws.amazon.com/glue/features/databrew/>
- [14] [https://docs.aws.amazon.com/inspector/?icmpid=docs\\_homepage\\_security](https://docs.aws.amazon.com/inspector/?icmpid=docs_homepage_security)
- [15] <https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802st=sb>
- [16] <https://docs.aws.amazon.com>
- [17] [https://www.cloudflare.com/searchresults/q=services&f\[customer\\_facing\\_source\]=Product](https://www.cloudflare.com/searchresults/q=services&f[customer_facing_source]=Product)
- [18] <https://www.pointfree.co/>
- [19] <https://twitter.com/pointfreeco>
- [20] <https://www.kodeco.com/>
- [21] <https://www.objc.io/>
- [22] <https://www.hackingwithswift.com/>
- [23] <https://docs.aws.amazon.com/wellarchitected/latest/framework/security.html>
- [24] <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.security.en.html>
- [25] <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillars.wa-pillars.en.html>
- [26] <https://docs.aws.amazon.com/wellarchitected/latest/performance-efficiency-pillar/performance-efficiency.html>
- [27] <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.reliability.en.html>
- [28] <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.pillar.costOptimization.en.html>
- [29] [https://aws.amazon.com/dynamodb/?trk=94bf4df1-96e1-4046-a020-b07a2be0d712&sc\\_channel=ps&s\\_kwcid=AL!4422!3!610000101513!e!!g!!amazon%20dynamodb&ef\\_id=CjwKCAiA7IGcBhA8EiwAFfUDsdpHKu1HOyAXFFWWJvQQfBp\\_KLT\\_XN0KjcZ1cz4Z9Svi6x62YfH0bRoCY6QQAvD\\_BwE:G:s&s\\_kwcid=AL!4422!3!610000101513!e!!g!!amazon%20dynamodb](https://aws.amazon.com/dynamodb/?trk=94bf4df1-96e1-4046-a020-b07a2be0d712&sc_channel=ps&s_kwcid=AL!4422!3!610000101513!e!!g!!amazon%20dynamodb&ef_id=CjwKCAiA7IGcBhA8EiwAFfUDsdpHKu1HOyAXFFWWJvQQfBp_KLT_XN0KjcZ1cz4Z9Svi6x62YfH0bRoCY6QQAvD_BwE:G:s&s_kwcid=AL!4422!3!610000101513!e!!g!!amazon%20dynamodb)

- [30] <https://aws.amazon.com/lambda/>
- [31] <https://docs.aws.amazon.com/mediaconvert/latest/ug/what-is.html>
- [32] <https://docs.aws.amazon.com/cognitoidentity/latest/APIReference/Welcome.html>
- [33] <https://docs.aws.amazon.com/whitepapers/latest/real-time-communication-on-aws/high-availability-and-scalability-on-aws.html>
- [34] <https://aws.amazon.com/products/storage/>
- [35] <https://aws.amazon.com/iam/>