Architecture Design - Sports Video App

Table of Contents

AWS Services needed

1. S3 [Video Store]
    a. For Storage of Raw Video Files
    b. Generates Events for AWS Lambda to Initiate Video Segmentation & Transcoding of Videos
    c. Storage of Segmented Videos
    d. Storing Videos in Different Formats on the basis of Different Codecs and Resolutions
2. Cloudfront [CDN]
    a. Amazon's Content Delivery Network tool was designed and optimized for the massive workloads associated with the media & entertainment industry. CloudFront is capable of delivering immense quantities of on-demand content, along with enabling platforms to live stream to millions of viewers.
    b. Helps in the Distribution of Content basis Geographical Distribution of Users.
3. AWS Lambda
    a. Removing the need for server management, this serverless computing service allows platforms to run a vast range of software applications, while Lambda allocates the necessary execution power to serve traffic at any scale.
    b. Handles Events from Various Sources across AWS
4. AWS Media Services
    a. Needed for Handling Media Transcoding Pipelines, DRM Checks (If Needed) for Concent Protection and Media Packaging Service.
5. Various Other Services
    a. Managed Database Services like DynamoDB, RDS (Managing User Accounts)
    b. Managed MQ Services like Kafka/RabbitMQ or something else to Communicate within the Internal Services of the System
    c. API Gateway for Lambda functions to be exposed in the forms of APIs
    d. Cloudwatch for Monitoring
    e. Cognito (We can also develop/use our Custom Authentication/Authorization Service)
    f. SES for sending out Notification Emails
    g. SNS for Push Notifications in case of Mobile App Users
    h. IAM, Certificate Manager, Secrets Manager

Technology Choices

- React/Angular/NextJS (Frontend Web Application)
- NodeJS/Java/Python (Various Microservices for handling Authentication/Content Display/Playback/Search/Analytics)
- Lambda(Serverless) for certain Services can be used
- DynamoDB/MongoDB/Cassandra (Database for Video URLS + Metadata)
- MySQL/Postgres (User Account/Billing Database)
- Apache Spark (Processing User Recommendations)

- Kafka/RabbitMQ (Internal Messaging Queues)
- SES (Notification Emails)
- Elasticsearch/Redis (Search Services)
- Redis (Caching)
- Nginx (Proxy/Ingress)

Questions Asked

1. Will the System have Live Streaming Videos from Sports Grounds?
2. What is the expected number of users for the system?
3. Is there any Constraint on choosing System Components (e.g Open Source or not) if not available in AWS?
4. DRM Content Protection is needed?

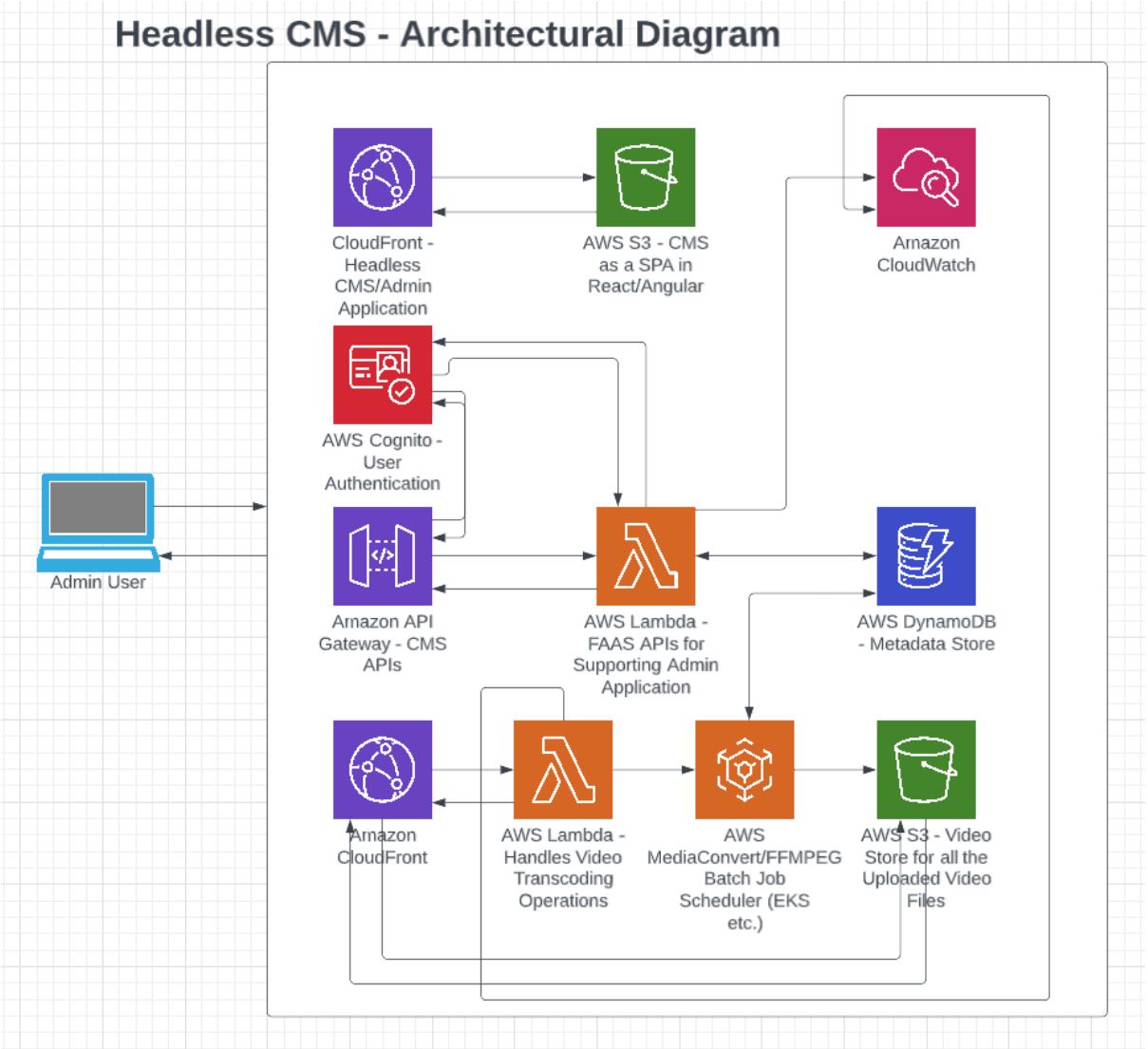
Assumptions Made

- The system uses HTTP APIs for any external communication. We can also use GraphQL, gRPC like Protocols depending upon the fitting areas
- We will be using NoSQL for Video StoreDB, like Metadata, Video CDN Links
- We will be using AWS Managed Services wherever available, we can still avoid them in case of Pricing Constraints or for manual Optimization
- Serverless Architecture to be used wherever feasible
- Protocol for Transmision: RTMP, HLS, MPEG DASH
- Codecs to be Supported: MPEG-2, H.264, VP8, Cinepak
- Resolutions: 240p, 480p, 720p, HD (1020p)
- Using AWS MediaServices for Transcoding, Distribution, Content Protection
- Elasticsearch for Search Engine Implementation
- Apache Spark for Producing Recommendation Results
- Cognito for Authentication with Custom RBAC Design for Role based Access and Control

Headless CMS

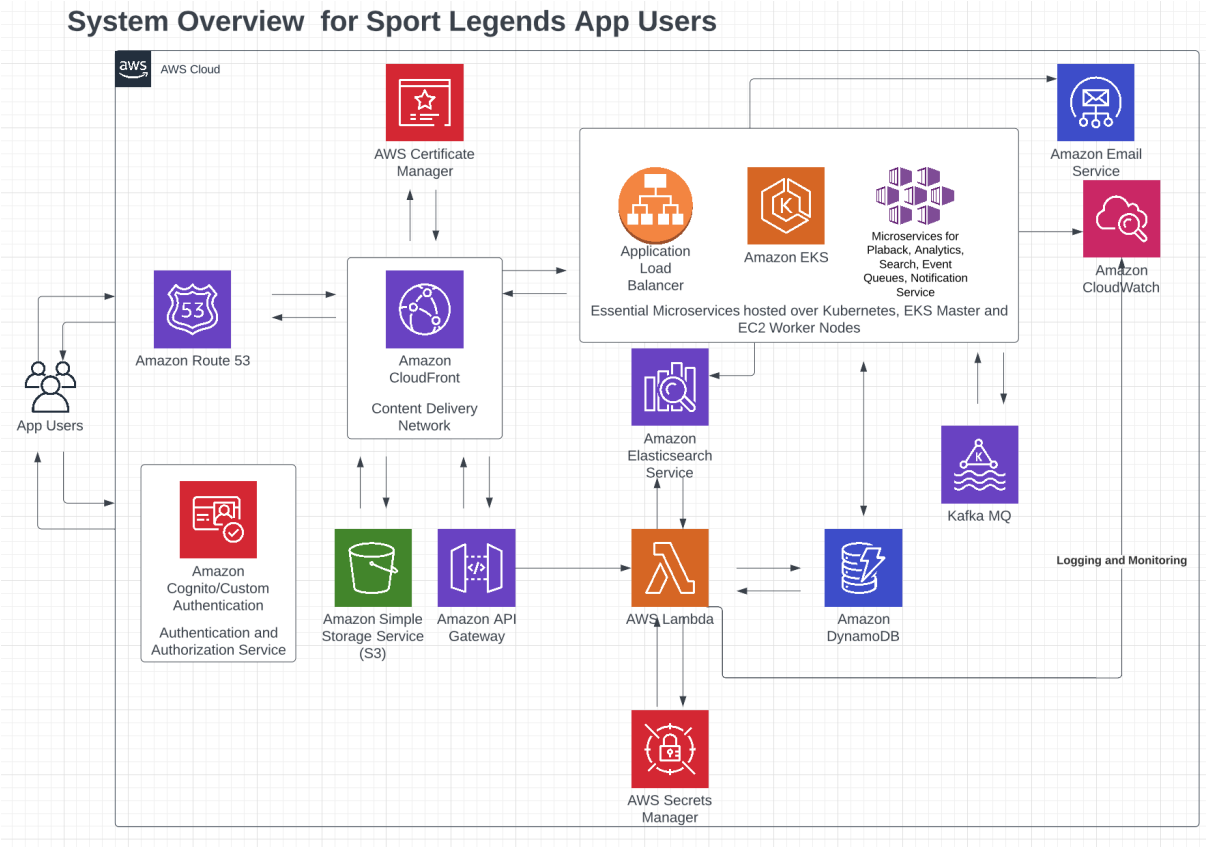
Link:
https://lucid.app/lucidchart/2cb3862e-368c-4ed2-a678-31bcc0457734/edit?invitationId=inv_3156ae4e-d201-48ba-aed9-5a27e21c856b

The Web Application will be needed for Uploading/Managing Video Content and Metadata. The App will have AWS Cognito or we can have our own Custom Authentication Service with Role based Access control to have a safe and secure mechanism to Manage the Video Contents along with the Metadata.



Sport Legends App

The App will be used to server the content to the Users Globally. This app plays a Critical Role and should be able to handle users at Peak Load.
Link:
https://lucid.app/lucidchart/9ce6d989-9d47-4deb-9e33-d241e97f158c/edit?invitationId=inv_ee416d4a-f215-4b80-b7ca-a3a5049d59eb



Search System (For Sports Legends App)

Micro Services

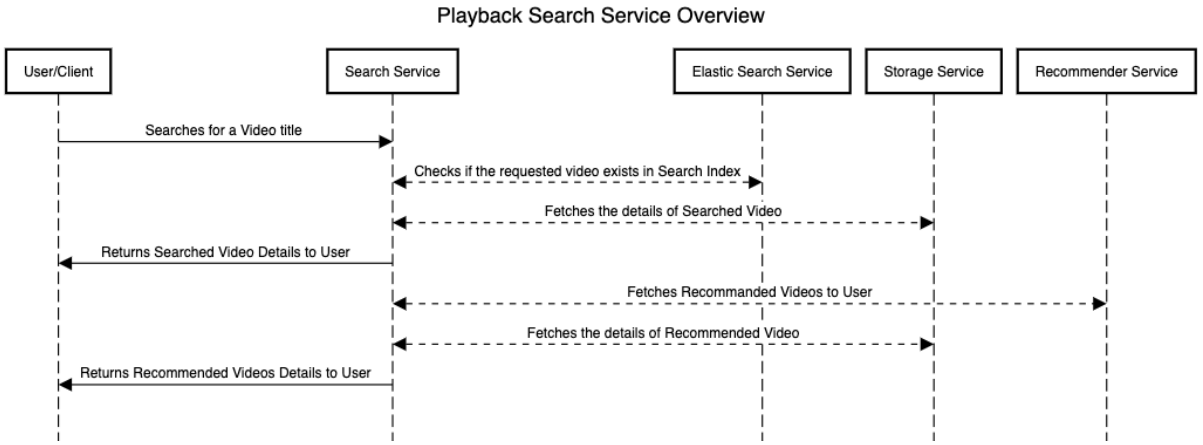
1. Content Discovery Service
2. Recommender Service

1. Responsible for Supporting the Search capability.
2. Creates Indexes of Video Titles in ElasticSearch.
3. Cron Job to Initialise Indexing Service.
4. Get the list of uploaded Videos from Videos MetaStore.
5. Update ElasticSearch Index with new Titles.

Input Parameters

- Video Title
- User-Location

User Flow



1. User Searches for Required Video Content.
2. Discovery Service queries ElasticSearch to check if the video title exists.
3. If found, the details are extracted from Video Store and returned to the User.
4. In case, the Content is not available, similar videos are searched and recommended to the User.

Playback System (For Sports Legends App)

Adaptive Bitrate Streaming

Considering we will have users of various origins and having different devices, but they're not all going to consume our stream in the same manner. Some viewers will have big screens, some will have mobile phones, some will be on amazing internet, and some could be on awful Wi-Fi or even LTE.

That's where adaptive bitrate (ABR) streaming becomes so important. A media server creates ABR 'renditions,' which are like items on a menu consisting of all the different resolutions and bitrates created. Then, the technology playing the video can choose the best rendition for viewing based on the size of the screen and network bandwidth available — while dynamically switching between options as resources fluctuate. In short, ABR ensures our viewers have the best quality live stream for their viewing conditions.

Streaming Video to Devices on Different Connections

Depending on what device viewers are using, they might require a different protocol. While a set-top box connected to our TV may still use RTMP, our iPhone only accepts HLS and our browser player may be running MPEG DASH. Again, our media server can convert your live stream into whatever protocol is required, called transmuxing, and ensure delivery to our viewers no matter on which device they are viewing video.

There are more things a media server can do, too, such as injecting additional information, saving the stream as a video file, and doing similar conversions to the audio. And keep in mind, with one stream coming into a media server (ingest) and multiple renditions coming out (egress) we will need more bandwidth for egress to support a large audience.

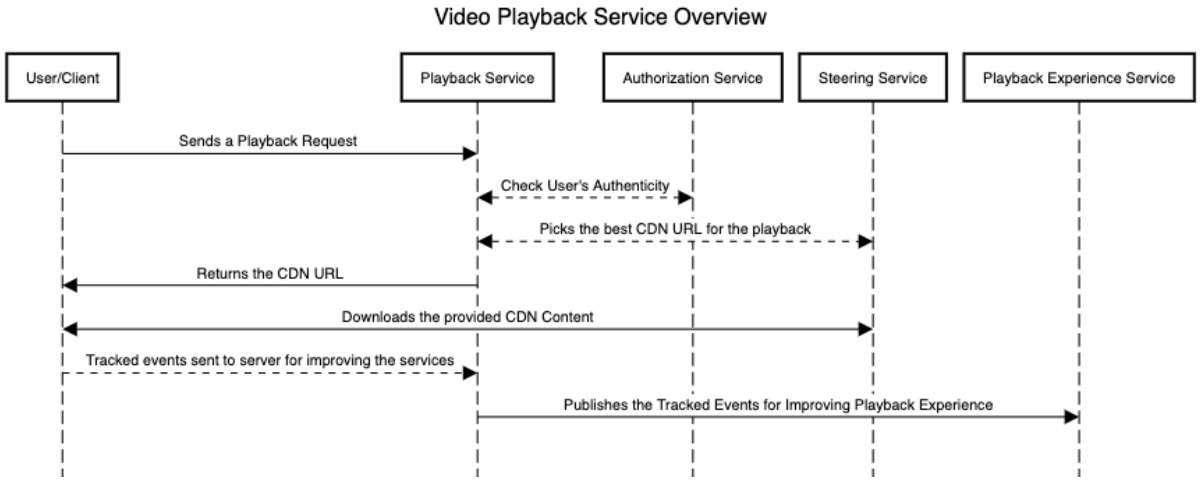
Micro Services

3. Authorization Service
    a. User Authentication, Authorization, and Licensing
4. Steering Service
    a. Deciding the Playback Process
    b. Finds the most Optimal CDN URL based on Input Parameters
5. Playback Experience Service
    a. Tracking Events to measure the Playback Experience

Input Parameters

- Client/User Device
- Bandwidth
- User Query
- User Location

User Flow



1. User Requests for a Media Playback
2. Calls for Authorization Service to authenticate users
3. Calls Steering Service for appropriate CDN URLs
4. CDN URL returned back to User Device
5. Playback Experience/Events get recorded back through Playback Experience Service.

Content Discovery/Recommendation System

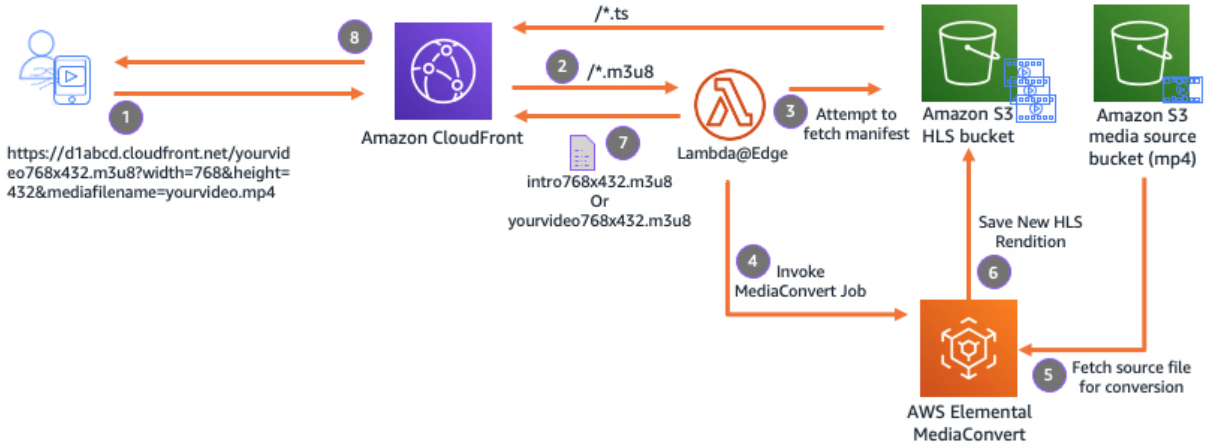
The Recommendation System can be built by processing the Historical patterns of User, Search History, User Location and regional attributes. The data can be used by systems like Apache Spark and our custom algorithms to produce the recommendations for the user.

Video Conversion Workflow

The video content uploaded can be Transcoded with AWS Media Services or our Custom FFMPEG containers running inside EKS or directly inside EC2 Machines. For faster Encoding, we need to ensure GPU access to the FFMPEG VMs or Containers.

Codecs to be Supported: MPEG-2, H.264, VP8, Cinepak - (Question to be asked)
Resolutions: 240p, 480p, 720p, HD (1020p) - (Question to be asked)
Total Videos = Number of Video Resolutions x Number of Supported Codecs
= 4 x 4 = 16

The diagram below shows the Transcoding process run by AWS MediaConvert Service. We can also run the service in Custom Containers over Kubernetes using FFMPEG.



The above illustrates if a format is requested which is not available in the CDN currently. The system then generates the requested format for the requested resolution and the requested codec and makes it available to the user.

General Video Transcoding Steps

1. Source Video file gets uploaded to Media Source S3 bucket.
2. Create your 6-seconds intro HLS video segment in the following resolutions: 240p, 480p, 720p, HD (1020p)
3. Converted segments get uploaded you created to the HLS stream bucket.
4. The Segmented Videos then get pushed to CDN.
5. CDN links are updated in the Database.

Cost

Overall, a cost model for on-the-fly video conversion workflow assumes that video assets can be converted only on request, and at a specific resolution.

For example, converting a video library on the fly based on this request rate:

- 30% of assets are requested in 6x renditions (3xHD, 3xSD)

- 30% of assets are requested in 4x renditions (2xHD, 2xSD)
- 40% of assets are requested in 2x renditions (1xHD, 1xSD)
- Can reduce 36.6% of transcoding time, compared to 100% conversion of the same library:

100% of assets converted in 6x renditions (3xHD, 3xSD)

Considerations

Scaling

1. Caching the data at various levels is really helpful in order to scale. One great example of caching is the OpenConnect CDN by Netflix. It is combination of specially developed hardware and software placed with various ISPs around the globe. It does provide the localised cache of most requested media files with the nearest ISP to the users.
2. Testing the application with simulations load of the predicted users. We can use tools like JMeter/Gatling and flood the system with simulated requests.
3. Multi-Region MediaConvert Endpoint
    a. We recommend that you use MediaConvert in the same Region where the S3 media bucket is located for lower latency, and to save on data transfer costs between Regions.
4. Duplicate requests from different Edge locations
    a. Amazon CloudFront forwards requests from global edge locations to your origin. There may be cases where your viewers send simultaneous requests for the same content rendition. It is recommended to keep a state of the conversion in progress, so only one MediaConvert job is invoked for each content rendition.
        i. Consider using CloudFront Origin Shield to consolidate origin requests for viewers in different geographical regions.
        ii. Consider using a DynamoDB table to store the current conversions-in-progress state. Use the data to determine if a job is in progress, and whether to invoke the MediaConvert job or not.

Cost Calculations

Once we have an estimated/targeted number of users, we can calculate how many compute instances we require. Supporting millions of users will require c4.4xlarge or c4.8xlarge will be capable to handle the load. We will require hundreds of such instances. Apart from that, we will be using EBS, S3 like storage services. The Video Blobs will be stored on to S3 Storage and then pushed to CDNs to provide faster access to the users. The pricing depends on then a lot of components. We can get precise pricing using the AWS Calculator below:
https://calculator.aws/

We can also try to find average usage for one user. And basis that we can estimate the cost of our infrastructure by our total number of users.

Once we know average usage for one user, you can then answer the following questions:

How much CPU per user is needed per month?
C = Number of cycles per second (Hz)
How much Storage per user is needed per month?
S = Quantity of data (MB)
How much Network bandwidth per user is needed per month?
N = Quantity of data per second (MB/s)
Once you have answered all these questions, you can extrapolate the final requirements in term of CPU / Storage / Network. Assuming that you are doing REST then the scalability is linear and you can just multiply the number of users by the values per user.

Reliability

With the AWS in the picture, we can easily take care of the reliability of the System by using:

- **Deployment across Multiple Availability Zones** to prevent failure even in case of Zone failure.
- **Horizontal Scaling** by using Multi Container Orchestration Deployments.
- Using AWS **Managed Services wherever possible** to further reliability.
- With the help of Kubernetes, we can easily handle Users at Scale, while we need to ensure a **Stateless Application Design**.
- **For Transmission reliability**, we need to ensure the **usage of TCP-based Protocols** for providing the best Streaming Experience to the users. Protocols like RTMP, MPEG DASH, and HLS are all based on TCP.
- **Avoid vertically scaled** single points of failure, like relational databases.
- Be lean on your single server and yield results with more room to scale horizontally. Review all application/system configurations as every **system component must be tuned specifically** to the traffic pattern and hardware.
- Sports Legends App can smartly handle broken server connections. **Caching and exponential back-offs** can help to ensure a seamless user experience.

Monitoring and Traceability

For monitoring, we are already using Cloudwatch across our different systems in AWS. This will help us monitor Functional Parts, Important events inside AWS Services to be captured. Monitoring the application can be done by using External monitoring systems like ELK stack, Datadog. Managed AWS Services will be managed by Amazon for us. Kubernetes deployments Monitoring can be done using Prometheus, Graphana. Apart from this, important events can be sent by kubewatch over Slack or any DevOps channel we use for communication. All of the events can be monitored on a single Dashboard.

One Open Source alternative to Datadog Platform for Monitoring and Traceability is SigNoz. It can help us track the faults inside our Deployment/Application and easily resolve them.

Security

Various security considerations we need to take into account for the following kinds of attacks:

- Streaming Content Security
- Application Attacks(Including APIs)
- Denial of Service Attacks
- Fake Applications (Phishing)
- Account hacks/Takeovers

We need to focus on a bunch of things like TLS certificates, authentication, security headers, request logging, rate limiting etc. Encrypting the the Internal Messages passing through the messaging queues is further needed.

Traffice Spikes

We need to be ready for th peak traffic to be handled by the system. Generally we consider 75% of our total registered users to be concurrently active as the peak load.