Design a live streaming service

Requirements:
1. Publishers should be able to stream live events
2. Users should be able to watch them live
3. Store video and metadata after the event to be viewed as video on demand(VOD)
4. Collect analystics on viewing, stream quality etc

Usages and data
10 concurrent events a day
8 Million concurrent users (average), 20 Million (peak)
avg size of video - 1 GB

Non Functional Requirements
1. Service should have very low latency (in the order of milliseconds)
2. Service should support various devices/resolution/bitrate
3. Service should be reliable.

****Out of scope****
1. video catalogue
2. search
3. authentication

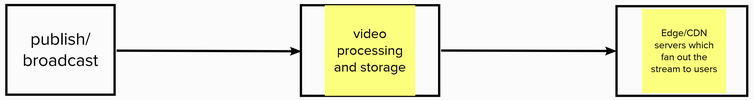
Data Storage requirements:
Storage per day = 10 * 1 GB = 10GB
Let's say we create 20 different encodings of roughly the same size, therefore
total storage per day = 20 *10GB = 200 GB

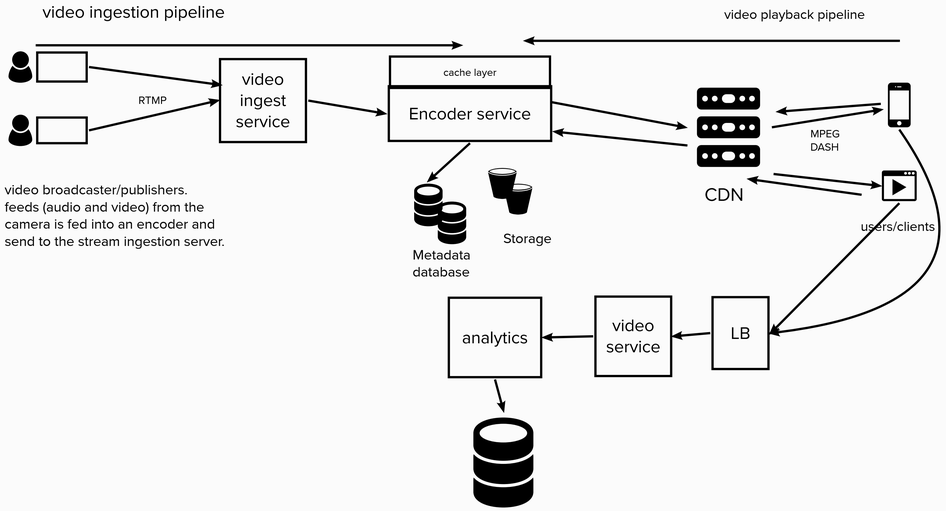
If we keep videos for 10 years, total storage required
= 200 * 365 * 10 = 730,000 GB = 730 TB

TPS we need support = 20 M users /sec . Since we are using CDN to server the streams, request for streams will not reach the BE servers.Content is served by CDN and Encoder BE server process and owns the stream and metadata.

Please note, areas like database design, logging and analytics are not covered in details here as the focus is on streaming live events.

High Level Design





Video ingestion pipeline:

Broadcasters starts uploading stream by establishing a connecton with one of the nearest Video Ingest Service and initiating a stream. The stream is identified by streamId.

Ingest Service do basic validity checks and forwards the stream to one of the backend encorder service. Consistent hashing can be used to forward to one of the server. This will help to increase reliability as a stream will always be mapped to the same server.

Encoder server now owns the stream and process it - process streams into 1 sec segments and generates combinations of encoding, resolution, bitrates to meet various client specifications.

These chunks/segments can then be stored in the storage (s3) and the video metadata is stored in the database. We can use Cassandra db which will scale and good for storing large amout of data.

Video playback pipeline

When client request for a stream, it first looks in the edge/cdn server for the streams.
If it results in a cache miss, so the request will be send to the Encoder server which owns the stream. It gets the media files to the user's device populating the CDN cache too, thereby future requests can be served from the cache.
Once the streaming has started, it can then periodically push the stream/segments to the CDN/Edge servers so that clients can be served directly from the CDN.

We can proactively push the video segments to CDN /Edge servers on the locations where we have high possibility of traffic (somewhat similiar to prefetching). This will reduce the latency as the initial request for stream will be delivered from CDN and not BE encoder servers.

Client Login and accessing the service.
Client access the video service through LB. Video service authenticates the user using an authentication service (like OKTA, which is out of scope of this design).
Video service gets the details of the stream and other metadata (description, title etc) from Encoder Server and gets the stream URL to the user, User initiates the stream URL by requesting for the stream (request to CDN).

Non Functional Requirements
Scalability - All components are horizontally scalable. We can add additional nodes to scale up. On the ingestion side, we can have a handle of servers to handle the load. These incoming connections will be persistent connections. For video processing and metadata processing, we can horizontally scale the encoders.
On the playback side, CDN video streaming servers accelerate delivery of video by caching content on multiple servers that are widely dispersed across a certain geography. When a viewer submits a request for a stream, the CDN streaming server nearest to user's location provides the content, speeding delivery and improving quality of the viewing experience.
User's access video service through Load Balancer to authenticate and request the stream prior to streaming starts.

Reliability /Fault Tolerance- Streams are ingested using RTMP protocol and for playback MPEG DASH protocol can be used. Streams are forwarded to the same server at the time as StreamId is used for consistent hashing. Metadata database runs in active -passive mode and video storage is replicated for redundancy.

Latency - kept as low by processing the streams into 1 sec segment and using MPEG DASH protocol and EDGE/CDN srvers to serve the content.

High Availability - All components are horizontally scalable and there is no Single Point of Failure. Database and storage can be partitioned using stream Id and replicated for redundancy. We have a cache layer (Redis or Memcache should suffice in this caase) at the backend where we can store user details and stream metadata as these are less likely ro change frequently and heavily read.