**DESIGN Youtube**

The solution to this question can be applied to other interview questions like designing a video sharing platform such as Netflix and Hulu.

* Total number of monthly active users: 2 billion
* Number of video watched per day: 5 billion
* 73% of US adults use YouTube
* 50 million creators on YouTube
* YouTube’s Ad revenue was $15.1billion for the full year 2019, up 36% from 2018
* YouTube is responsible for 37% of all mobile internet traffic
* Youtube is available in 80 different languages.

### Understanding Problem and Establishing Design Scope:

Quey1: What is the traffic for the system?

-> 5 million DAU

Query2: What clients/devices do we need to support?

-> Mobile apps, web browsers and smart TV

Query3: Which features are important?

-> ability to upload videos and watch videos.

Query4: What is the average time spent by a user?

-> 30 minutes

Query5: Any file size requirement for videos?

-> Our platform focuses on small and medium sized videos. The maximum allowed video size is 1GB.

Query6: **Can we leverage some of the existing cloud infrastructures provided by Amazon, Google, or Microsoft?**

-> Building everything from scratch is unrealistic for most companies, it is recommended to leverage some of the existing cloud services.

Query7: Is encryption required?

-> Yes

Focus on designing video streaming service with following features:

* Ability to upload videos fast
* Smooth video streaming
* Ability to change video quality
* Low infrastructure cost
* High availability, scalability and reliability requirements
* Clients supported: mobile apps, web browser, and smart TV

### Back of the Envelope Estimation:

5\*10^6 DAU.

Users watch 5 videos on an average per day.

10% of users upload 1 video per day.

Assume the average size of video is 300MB.

Total daily storage: 5\*10^6 \* 10^-1 users upload videos = 5\*10^5 \* 3\*10^2 = 1.5\*10^8 MB = 1.5\*10^14 byte = 150TB

CD cost:  
When a cloud CDN serves a video, you are charged for data transferred out of the CDN.

Let us use Amazon’s CDN CloudFront for cost estimation. Assume 100% traffic is served from the United States. The average cost per GB is $0.02. For simplicity, we only calculate the cost of video streaming.

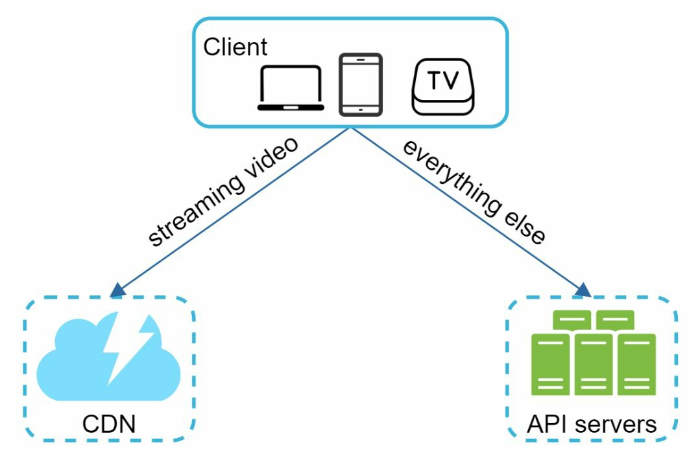
5 million \* 5 video \* 0.3GB\* $0.02 = 25\*10^6\*6\*10^-3 = 6\*25\*10^3 = $ 150000 per day.

### High Level Design Propositions and approaches:

As discussed previously, the interviewer recommended leveraging existing cloud services instead of building everything from scratch. CDN and blob storage are the cloud services we will leverage.

**A Binary Large Object (BLOB) is a collection of binary data stored as a single entity in a database management system**

Netflix leverages Amazon’s cloud services [4], and Facebook uses Akamai’s CDN [5].

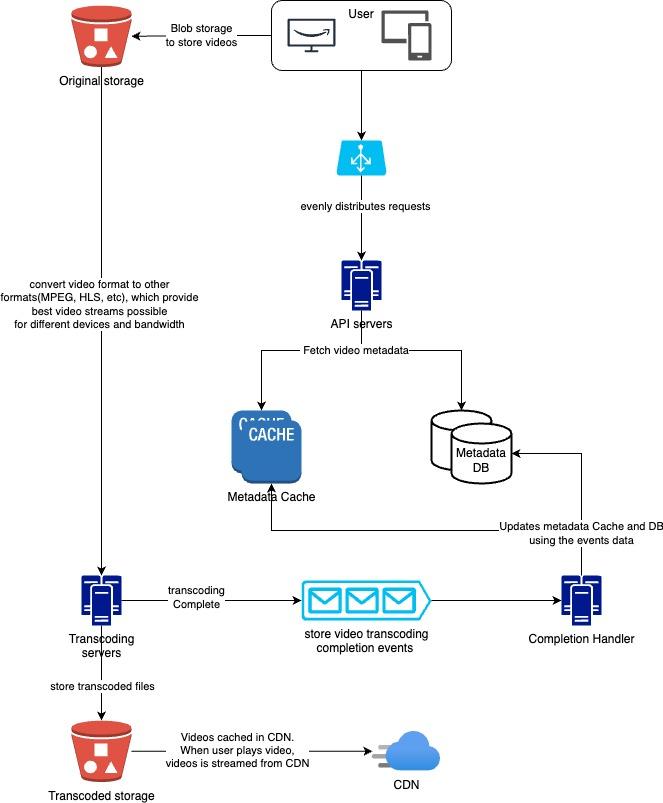


**Client:** You can watch YouTube on your computer, mobile phone and smartTV.

**CDN:** Videos are stored in CDN. When you press play, a video is streamed from the CDN.

**API servers:** Everything else except video streaming goes through API servers. This includes feed recommendation, generating video upload URL, updating metadata database and cache, user sign in, etc.

**Video uploading flow:**



The flow is broken down into two processes running in parallel: **Flow a: Upload the actual video**

1. Videos are uploaded to the original storage
2. Transcoding servers fetch videos from the original storage and start transcoding.
3. Once transcoding is complete, the following two steps are executed in parallel:

3a. Transcoded videos are sent to transcoded storage.

3b. Transcoding completion events are queued in the completion queue.

3a. 1. Transcoded videos are distributed to CDN

3b. 1. Completion handler contains a bunch of workers that continuously pull event data from the queue

3b.1.a and 3b.1.b. Completion handler updates the metadata and cache when video transcoding is complete.

1. API servers inform the client that the video is successfully uploaded and is ready for streaming.

**Flow b: Update the metadata**

While a file is being uploaded to the original storage, the client in parallel sends a request to update the video metadata. The request contains video metadata, including file name, size, format,etc. API servers update the metadata cache and database.

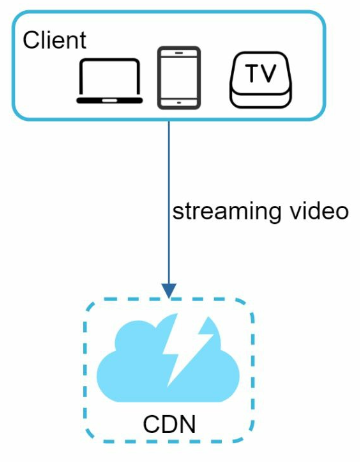
**Video Streaming Flow:**

Whenever you watch a video on YouTube, it usually starts streaming immediately and you do not wait until the whole video is downloaded. Downloading means the whole video is copied to your device, while streaming means your device continuously receives video streams from remote source videos. When you watch streaming video, your client loads a little bit of data at a time so you can watch videos immediately and continuously.

Before we discuss video streaming flow, let us look at an important concept: streaming protocol. This is a standardized way to control data transfer for video streaming. Popular streaming protocols are:

* MPEG-DASH: MPEG stands for “Moving Picture Experts Group” and DASH stands for “Dynamic Adaptive Streaming over HTTP”.
* Apple HLS: HLS stands for “HTTP live Streaming”
* Microsoft Smooth Streaming
* Adobe HTTP Dynamic Streaming(HDS).

The important thing here is to understand different streaming protocols that support different video encoding and playback players. When we design a streaming service, we have to choose the right streaming protocol to support our use case.



### Design Deep Dive:

**Videos transcoding**

When you record a video, the device(usually a phone or camera) gives the video file a certain format. If you want the video to be played smoothly on other devices, the video must be encoded into compatible bitrates and formats. Bitrate is the rate at which bits are processed over time. A higher bitrate generally means higher video quality. High bitrate streams need more processing power and fast internet speed.

Video transcoding is important for following reasons:

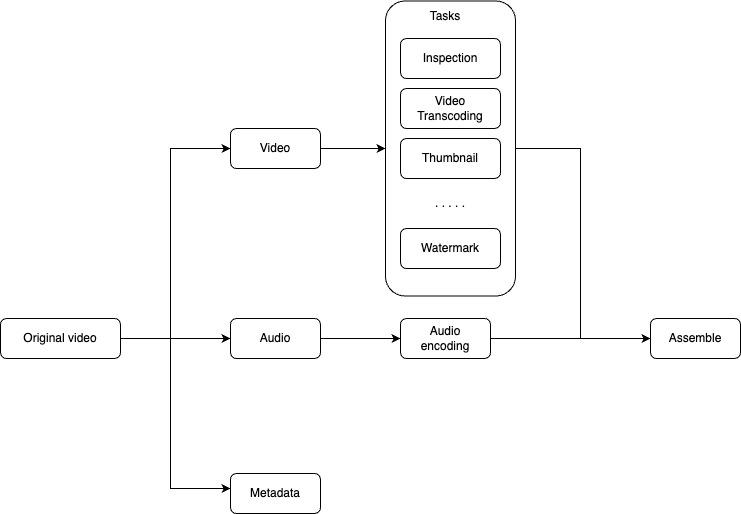
1. Raw video consumes a large amount of storage space. An hour-long high definition video recorded at 60 frames per second can take up a few hundred GB of space.
2. Many devices and browsers only support certain types of video formats. Thus, it is important to encode a video to different formats for compatibility reasons.
3. To ensure users watch high-quality videos while maintaining smooth playback, it is a good idea to deliver higher resolution video to users who have high network bandwidth and lower resolution video to users who have low network bandwidth.
4. Network conditions can change, especially on mobile devices. To ensure a video is played continuously, switching video quality automatically or manually based on network conditions is essential for smooth user experience.

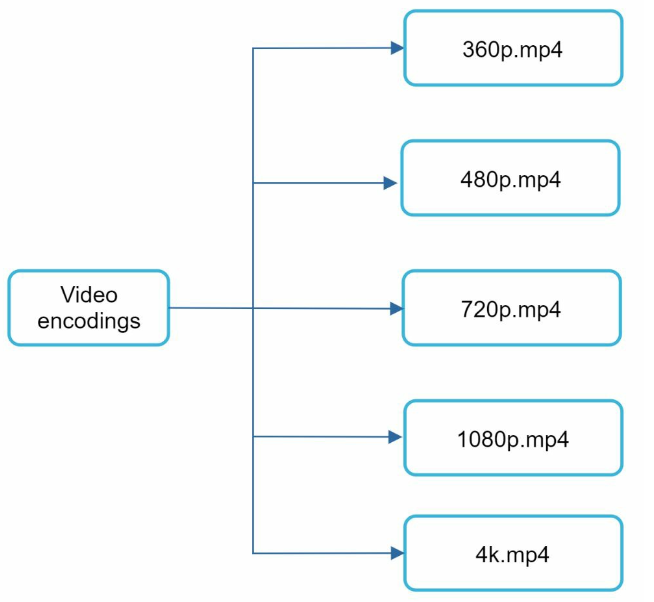
Many types of encoding formats are available; however, most of them contain two parts:

1. Container: This is like a basket that contains the video file, audio and metadata. You can tell the container format by the file extension, such as .avi, .mov or .mp4.
2. Codecs: These compression and decompression algorithms aim to reduce the video size while preserving the video quality. The most used video codecs are H.264, VP9 and HEVC.

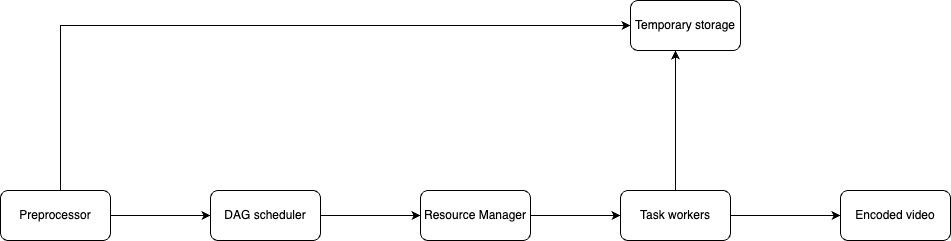
**Directed acyclic graph(DAG) model**

Transcoding a video is computationally expensive and time-consuming. Besides, different content creators may have different video processing requirements.

To support different video processing pipelines and maintain high parallelism, it is important to add some level of abstraction and let client programmers define what tasks to execute. For example, Facebook’s streaming video engine uses a directed acyclic graph(DAG) programming model, which defines tasks in stages so they can be executed sequentially or parallelly. 

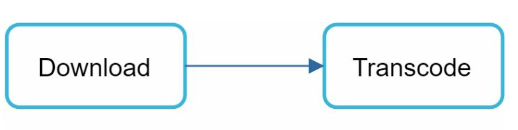
* Inspection: Make sure videos are good quality and are not malformed.
* Video encoding: Videos are converted to support different resolutions, codecs, bitrates, etc
* 
* Thumbnail: Thumbnails can either be uploaded by a user or automatically generated by the system.
* Watermark: An image overlay on top of your video contains identifying information about your video.

**Video Transcoding Architecture:**

****

**Preprocessor:**The preprocessor has 4 responsibilities:

1. Video splitting: Video stream is split or further split into smaller Group of Pictures(GOP) alignment. GOP is a group/chunk of frames arranged in a specific order. Each chunk is an independently playable unit, usually a few seconds in length.
2. Some old mobile devices or browsers might not support video splitting. Preprocessor split videos by GOP alignment for old clients.
3. DAG generation: The Processor generates DAG based on configuration files client programmers write.



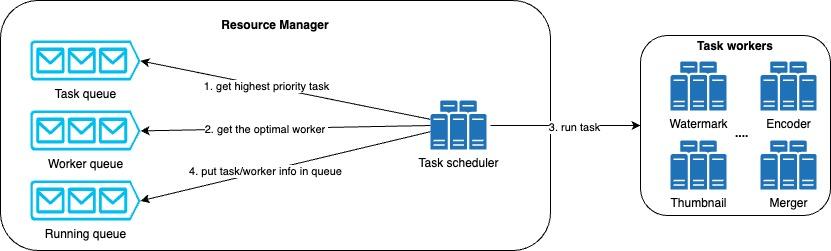
1. Cache data: The preprocessor is a cache for segmented videos. For better reliability, the preprocessor stores GOPs and metadata in temporary storage. If Video encoding fails, the system could use persisted data for retry operations.

**DAG scheduler**

The original video is split into three stages: Stage1: video, audio, and metadata. The video file is further split into two tasks in stage2: video encoding and thumbnail. The audio file requires audio encoding as part of the stage 2 tasks.

**Resource manager:**Responsible for managing the efficiency of resource allocation. It contains 3 queues and a task scheduler.

* Task queue: It is a priority queue that contains tasks to be executed.
* Worker queue: It is a priority queue that contains worker utilization info.
* Running queue: It contains info about the currently running tasks and workers running the tasks.
* Task scheduler: It picks the optimal task/worker, and instructs the chosen task worker to execute the job.

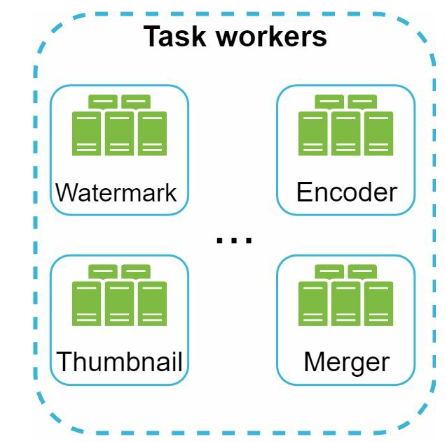


The resource manager works as follows:

* The task scheduler gets the highest priority task from the task queue.
* The task scheduler gets the optimal task worker to run the task from the worker queue.
* The task scheduler instructs the chosen task worker to run the task.
* The task scheduler binds the task/worker info and puts it in the running queue.
* The task scheduler removes the job from the running queue once the job is done.

**Task workers**

Task workers run tasks which are defined in the DAG. Different task workers may run different tasks.



**Temporary storage**

Multiple storage is used. The choice of storage system depends on factors like data type, data size, access frequency, data life span,etc. For instance, metadata is frequently accessed by workers, and the data size is very small. Thus, caching metadata in memory is a good idea. For video or audio data, we put them in blob storage. Data in temporary storage is freed up once the corresponding video processing is complete.

**Encoded video**

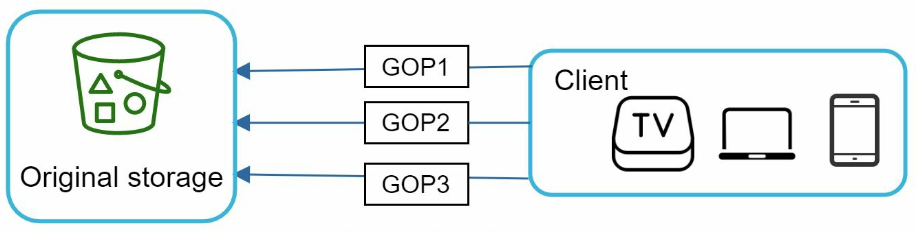
Encoded video is the final output of the encoding pipeline.

**System Optimizations**

**Speed Optimization: parallelize video uploading**

Uploading a video as a whole unit is inefficient. We can split a video into smaller chunks by GOP alignment.





**Speed optimization: place upload centers close to users**

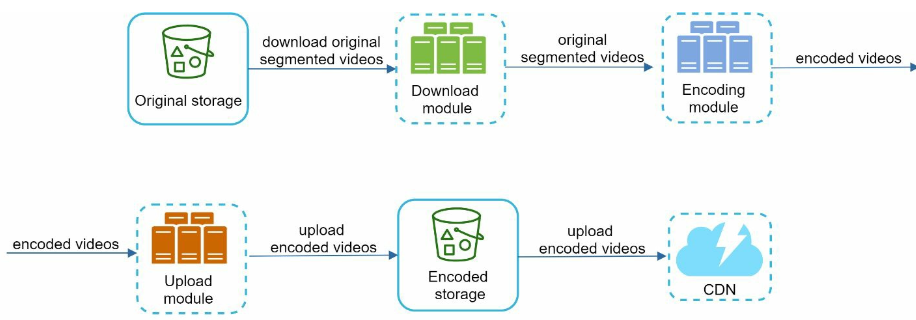
Another way to improve the upload speed is by setting up multiple upload centers across the globe. People in the United States can upload videos to the North America upload center, and people in China can upload videos to the Asian upload center. To achieve this we use CDN as upload centers.

### 

**Speed optimization: parallelism everywhere**

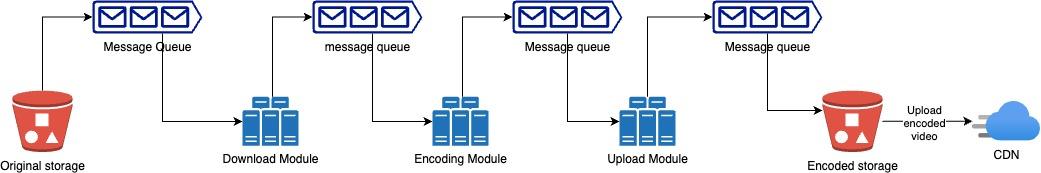
Achieving low latency requires serious efforts. Another optimization is to build a loosely coupled system and enable high parallelism.

Let us zoom in the flow of how a **video is transferred from original storage** to the CDN.



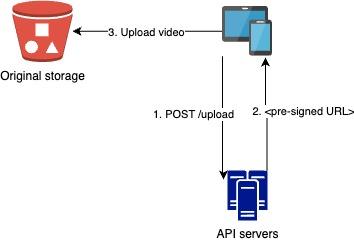
To make the system more loosely coupled, we introduced message queue

* Before the message queue is introduced, the encoding module must wait for the output of the download module.
* After the message queue is introduced, the encoding module does not need to wait for the download module anymore. If there are events in the message queue, the encoding module can execute those jobs in parallel.



**Safety Optimization: pre-signed upload URL**

To ensure only authorized users upload videos to the right location, we introduce pre-signed URLs.

Here the client makes a HTTP request to API servers to fetch the pre-signed URL, which gives the access permission to the object identified in the URL. The term pre-signed URL is used by uploading files to Amazon S3. Other cloud service providers might use a different name. For instance, Microsoft Azure blob storage supports the same feature, but calls it “Shared Access signature”.  
  
Once the client receives the pre-signed URL from the server, it uploads the video using pre-signed URL.

**What is a Pre-signed URL?**A pre-signed URL is a URL that provides temporary access to a private resource without requiring additional authentication. It is typically generated by the server, signed with the server's credentials, and includes an expiration time after which the URL will no longer be valid.

**Safety Optimization: protect your videos**

Many content makers are reluctant to post video online because they fear their original videos will be stolen. To protect copyright videos, we can adopt one of the following three safety options:

* Digital right management(DRM) systems: Three major DRM systems are Apple, FairPlay, Google Widevine and Microsoft Play Ready.
* AES encryption: You can encrypt a video and configure an authorization policy. The encrypted video will be decrypted upon playback. This ensures that only authorized users can watch an encrypted video.
* Visual watermarking: is an image overlay on top of your video that contains identifying information for your video. It can be your company logo or company name.

**Cost saving optimization**

CDN is a crucial component of our system. It ensures fast video delivery on a global scale. However, from the back of the envelope calculation, we know CDN is expensive, especially when data size is large.

**Youtube video steam follows long-trail distribution.** It means a few popular videos are accessed frequently but many others have few or no viewers. Based on this observation, we implement a few optimization:

1. Only serve the most popular videos from CDN and other videos from our high capacity storage video servers.
2. For less popular content, we may not need to store many encoded video versions. Short videos can be encoded on-demand.
3. Some videos are popular only in certain regions. There is no need to distribute these to other regions.
4. Build your own CDN like Netflix and partner with Internet Service Providers (ISPs). Building your CDN is a giant project; however, this could make sense for large streaming companies. An ISP can be Comcast, AT&T, Verizon, or other internet providers. ISPs are located all around the world and are close to users. By partnering with ISPs, you can improve the viewing experience and reduce the bandwidth charges.

**Error Handling**

For a large-scale system, system errors are unavoidable. To build a highly fault-tolerant system, we must handle errors gracefully and recover from them fast. Two types of errors exist:

• Recoverable error. For recoverable errors such as video segment fails to transcode, the general idea is to retry the operation a few times. If the task continues to fail and the system believes it is not recoverable, it returns a proper error code to the client.

• Non-recoverable error. For non-recoverable errors such as malformed video format, the system stops the running tasks associated with the video and returns the proper error code to the client.

Typical errors for each system component are covered by the following playbook:   
• Upload error: retry a few times.

• Split video error: if older versions of clients cannot split videos by GOP alignment, the entire video is passed to the server. The job of splitting videos is done on the server-side.

• Transcoding error: retry.

• Preprocessor error: regenerate DAG diagram.

• DAG scheduler error: reschedule a task.

• Resource manager queue down: use a replica.

• Task worker down: retry the task on a new worker.

• API server down: API servers are stateless so requests will be directed to a different API server.

• Metadata cache server down: data is replicated multiple times. If one node goes down, you can still access other nodes to fetch data. We can bring up a new cache server to replace the dead one.

• Metadata DB server down:

• Master is down. If the master is down, promote one of the slaves to act as the new master.

• Slave is down. If a slave goes down, you can use another slave for reads and bring up another database server to replace the dead one.

### Additional optimization:

here are a few additional points:

• Scale the API tier: Because API servers are stateless, it is easy to scale API tier horizontally.

• Scale the database: You can talk about database replication and sharding.

• Live streaming: It refers to the process of how a video is recorded and broadcasted in real time. Although our system is not designed specifically for live streaming, live streaming and non-live streaming have some similarities: both require uploading, encoding, and streaming. The notable differences are:

• Live streaming has a higher latency requirement, so it might need a different streaming protocol.

• Live streaming has a lower requirement for parallelism because small chunks of data are already processed in real-time.

• Live streaming requires different sets of error handling. Any error handling that takes too much time is not acceptable.

• Video takedowns: Videos that violate copyrights, pornography, or other illegal acts shall be removed. Some can be discovered by the system during the upload process, while others might be discovered through user flagging.