**How to Design Youtube (Part I)**

One of the most common types of system design interview questions is to design an existing popular system. For example, in the past, we’ve discussed [How to Design Twitter](http://blog.gainlo.co/index.php/2016/02/17/system-design-interview-question-how-to-design-twitter-part-1/), [Design Facebook Chat Function](http://blog.gainlo.co/index.php/2016/04/19/design-facebook-chat-function/) and so on so forth.

Part of the reason is that the question is usually general enough so that there are a lot of areas to discuss. In addition, if candidates are generally curious enough, they are more likely to explore how existing products are designed.

So this week, we’re going to talk about how to design [Youtube](http://www.youtube.com/). It’s a broad question because building Youtube is like building a skyscraper from scratch and there are just too many things to consider. Therefore, we’ll cover most of the “major” components from interviewer’s perspective, including database model, video/image storage, scalability, recommendation, security and so on.

**Overview**

Facing this question, most people’s minds go blank as the question is just too broad and they don’t know where to start. Just the storage itself is non-trivial as serving videos/images seamlessly to billions of users is extremely complicated.

As suggested in [8 Things You Need to Know Before a System Design Interview](http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/), it’s better to start with a high-level overview of the design before digging into all the details. This is true especially for problems like this that has countless things to consider and you’ll never be able to clarify everything.

Basically, we can simplify the system into a couple of major components as follows:

* Storage. How do you design the database schema? What database to use? Videos and images can be a subtopic as they are quite special to store.
* Scalability. When you get millions or even billions of users, how do you scale the storage and the whole system? This can be an extremely complicated problem, but we can at least discuss some high-level ideas.
* Web server. The most common structure is that front ends (both mobile and web) talk to the web server, which handles logics like user authentication, sessions, fetching and updating users’ data, etc.. And then the server connects to multiple backends like video storage, recommendation server and so forth.
* Cache is another important components. [We’ve discussed in details about cache before](http://blog.gainlo.co/index.php/2016/05/17/design-a-cache-system/), but there are still some differences here, e.g. we need cache in multiple layers like web server, video serving, etc..
* There are a couple of other important components like recommendation system, security system and so on. As you can see, just a single feature can be used as a stand-alone interview question.

**Storage and data model**

If you are using a [relational database](https://en.wikipedia.org/wiki/Relational_database) like MySQL, designing the data schema can be straightforward. And in reality, Youtube does use MySQL as its main database from the beginning and it works pretty well.

First and foremost, we need to define the user model, which can be stored in a single table including email, name, registration data, profile information and so on. Another common approach is to keep user data in two tables – one for authentication related information like email, password, name, registration date, etc. and the other for additional profile information like address, age and so forth.

The second major model is video. A video contains a lot of information including meta data (title, description, size, etc.), video file, comments, view counts, like counts and so on. Apparently, basic video information should be kept in separate tables so that we can first have a video table.

The author-video relation will be another table to map user id to video id. And user-like-video relation can also be a separate table. The idea here is [database normalization](https://en.wikipedia.org/wiki/Database_normalization) – organizing the columns and tables to reduce data redundancy and improve data integrity.

**Video and image storage**

It’s recommended to store large static files like videos and images separately as it has better performance and is much easier to organize and scale. It’s quite counterintuitive that Youtube has more images than videos to serve. Imagine that each video has thumbnails of different sizes for different screens and the result is having 4X more images than videos. Therefore we should never ignore the image storage.

One of the most common approaches is to use [CDN (Content delivery network)](https://en.wikipedia.org/wiki/Content_delivery_network). In short, CDN is a globally distributed network of proxy servers deployed in multiple data centers. The goal of a CDN is to serve content to end-users with high availability and high performance. It’s a kind of 3rd party network and many companies are storing static files on CDN today.

The biggest benefit using CDN is that CDN replicates content in multiple places so that there’s a better chance of content being closer to the user, with fewer hops, and content will run over a more friendly network. In addition, CND takes care of issues like scalability and you just need to pay for the service.

**Popular VS long-tailed videos**

If you thought that CDN is the ultimate solution, then you are completely wrong. Given the number of videos Youtube has today ([819,417,600 hours of video](https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=how%20many%20videos%20youtube)), it’ll be extremely costly to host all of them on CDN especially majority of the videos are long-tailed, which are videos have only 1-20 views a day.

However, one of the most interesting things about Internet is that usually, it’s those long-tailed content that attracts the majority of users. The reason is simple – those popular content can be found everywhere and only long-tailed things make the product special.

Coming back to the storage problem. One straightforward approach is to host popular videos in CDN and less popular videos are stored in our own servers by location. This has a couple of advantages:

* Popular videos are viewed by a huge number of audiences in different locations, which is what CND is good at. It replicates the content in multiple places so that it’s more likely to serve the video from a close and friendly network.
* Long-tailed videos are usually consumed by a particular group of people and if you can predict in advance, it’s possible to store those content efficiently.

Note: <https://en.wikipedia.org/wiki/Long_tail>

This is used to describe the retailing strategy of selling a large number of unique items with relatively small quantities sold of each (the "long tail")—usually in addition to selling fewer popular items in large quantities (the "head").

**Summary**

There are just too many topics we’d like to cover for the question “how to design Youtube”. In our next post, we’ll talk more about scalability, cache, server, security and so on.

By the way, if you want to have more guidance from experienced interviewers, you can check [Gainlo](http://www.gainlo.co/?utm_source=blog&utm_campaign=blog&utm_medium=How%20to%20Design%20Youtube%20(Part%20I)) that allows you to have mock interview with engineers from Google, Facebook ,etc..

**How to Design Youtube (Part II)**

This is the second post about how to design Youtube. We’ll continue our discussion from [the first one](http://blog.gainlo.co/index.php/2016/10/22/design-youtube-part/) and please check it if you haven’t read.

In [the last post](http://blog.gainlo.co/index.php/2016/10/22/design-youtube-part/), we mainly talked about database and storage. This week, we’ll cover much more topics, including scalability, web server, cache and security.

**Scale the database**

There are tons of problems to fix once the product has millions or even billions of users. Scalability is one of the most important issues to solve. Basically, storing all the data into a single database is not only inefficient but infeasible. So how would you scale the database for Youtube?

We can follow a lot of general rules when scaling the database. The most common approach is to scale only when you need it. In other words, it’s not recommended to do all the work like partition your database at day one, because it’s almost for sure that at the point you really need to scale, the whole infrastructure and product have been changed dramatically.

So the idea is to start from a single server. Later on, you may go to a single master with multiple read slaves ([master/slave model](https://en.wikipedia.org/wiki/Master/slave_(technology))). And at some point, you’ll have to partition the database and settle on a sharding approach. For instance, you can split the database by users’ location and when a request comes, you’ll route the request to the corresponding database.

For Youtube, we can further optimize it. The most important feature of Youtube is the video. Therefore, we can prioritize traffic by splitting the data into two clusters: a video cluster and a general cluster. We can give a lot of resources to the video cluster and other social network features will be routed to the less capable cluster. A more general idea here is that when solving scalability issue, you should first identify the bottleneck and then optimize it. In this case, the bottleneck is watching videos.

**Cache**

I will not talk much about cache for this topic as we’ve covered this in details in our previous post – [Design a Cache System](http://blog.gainlo.co/index.php/2016/05/17/design-a-cache-system/). But several points are worth to mention here.

First of all, when talking about cache, most people’s reaction is about server cache. In fact, front end cache is equally important. If you want to make your website fast and has low latency, you can’t avoid setting cache for the front end. This is a very common technique when building a website interface.

Secondly, as we briefly discussed in the previous post, caching won’t do a lot of good in terms of serving videos. This is mainly because majority usage of Youtube comes from those long tail videos and it’ll be extremely expensive to set cache for all videos. So the general idea here is that if you are building a long tail product like this, don’t place too much bet on the cache.

**Security**

There are a lot of things that can be discussed security in Youtube. I’d like to cover one interesting topic here – view hacking. Under each Youtube video, it shows the view count, which indicates how popular the video is. People can programmatically send requests to hack the view count, so how should we protect it?

The most straightforward approach is to if a particular IP issues too many requests, just block it. Or we can even restrict the number of view count per IP. The system can also check information like browser agent and user’s past history, which potentially can block a lot of hacks.

People may use services like [Tor](https://en.wikipedia.org/wiki/Tor_(anonymity_network)) to hide IP, and sites like [Mechanical Turk](https://www.mturk.com/mturk/welcome) allows you to pay people to click the video with very low cost. However, hacking the system is harder than most people think.

For instance, a video with high view count but low engagement is very suspicious. With a large number of video Youtube has, it’s not hard to extract patterns of real view count. In order to hack the system, you need to provide reasonable engagement metrics like share count, comment count, view time, etc.. And it’s almost impossible to fake all of them.

**Web server**

Many people overlook web server as it doesn’t have too many things to discuss in terms of system design, However, for large systems like Youtube, there are many things you need to consider. I’d like to share a couple techniques Youtube has used.

* Youtube server was built in Python initially, which allows rapid flexible development and deployment. You might notice that many startups choose Python as their server language as it’s much faster to iterate.
* Python sometimes has the performance issue, but there are many C extensions that allow you to optimize critical section, which is exactly how Youtube works.
* To scale the web server, you can simply have multiple replicas and build a load-balancer on top of them.
* The server is mainly responsible for handling user requests and return response. It should have few heavy logics and everything else should be built into separate servers. For instance, recommendation should be a separate component to let Python server fetches data from.

**Summary**

In this post, we try to cover as many different topics as possible and each of them can be discussed deeper in a separate post. There are still many things I’d like to talk about in the future, but readers can think about them and discuss in the comments.

For instance, Youtube recommendation is a very big topic and it drives user engagement metrics dramatically. How would you build the recommendation system? In addition, how do you identify trending videos of the day and recommend to the relevant audience?

By the way, if you want to have more guidance from experienced interviewers, you can check [Gainlo](http://www.gainlo.co/?utm_source=blog&utm_campaign=blog&utm_medium=How%20to%20Design%20Youtube%20(Part%20II)) that allows you to have mock interview with engineers from Google, Facebook ,etc..