Netflix system design

Problem Statement: Design A video streaming platform like netflix

Netflix is a subscription-based streaming service that allows its members to watch TV shows and movies on an internet-connected device. It is available on platforms such as the Web, iOS, Android, TV, etc.
We will be focusing more on the high level side of the netflix system design.

So the features that we need to understand today are the video or content upload, search, view and recommendations for the users.

Functional requirements

- The content team should be able to upload new videos (movies, tv shows episodes).
- Users should be able to stream and interact (like, dislike,share) with videos.
- Users should get personalized video recommendations

Non-Functional requirements

- High availability with minimal latency - low latency devices and regions.
- Scalability and efficiency - stream to millions of users without fail.

System design

Systems design implies a systematic approach to the design of a system. System Design is the process of designing the architecture, components, and interfaces for a system so that it meets the end-user requirements.

Approaching a Design Problem

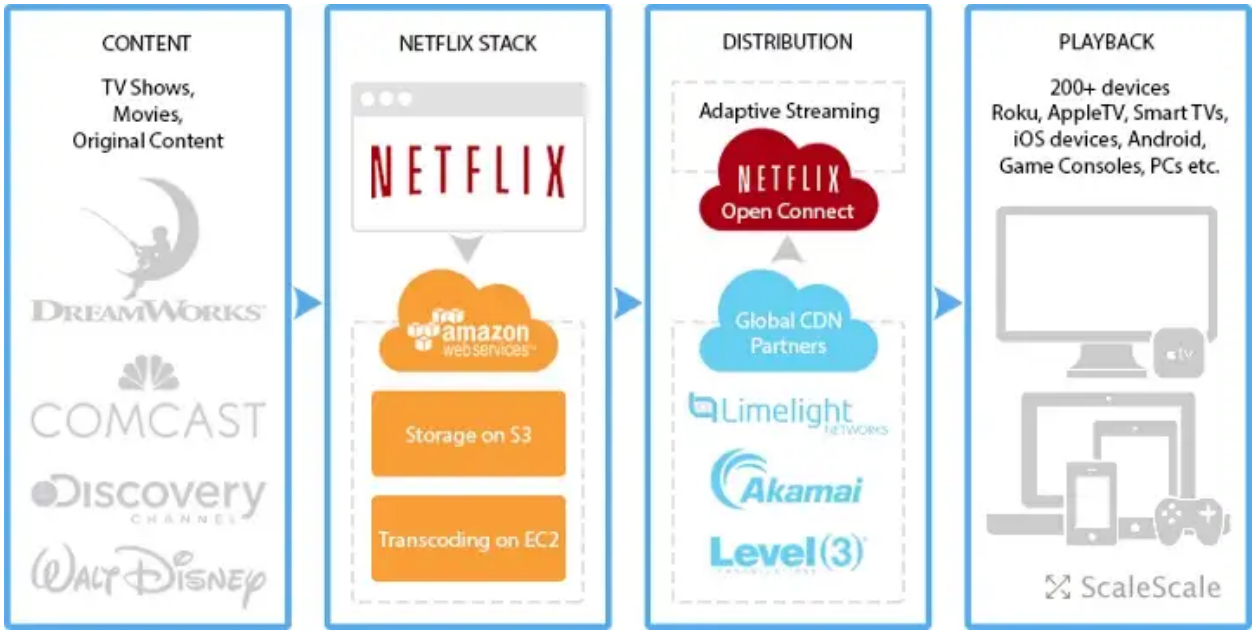
- Breaking Down the Problem: When you are given a Design Problem start breaking it down to small components. These components can be **Services or Features** which you need to implement in the System.
  We need to **clarify the goal** of the system. If we don't narrow it down to a specific goal, it will become complicated to design the system

- Estimation/ assumptions of important parts. One of the important points of the system design is to know about the scale of the system. Make some reasonable assumptions while you are designing the System. **Suppose you have to assume the number of requests the system will be processing per day, the number of database calls made in a month**

- Data and Flow. We need to define the **system's data model and how data will flow between different system components.** The kind of data we need to store like user data, inventory data, shipping data, product data etc. We need to figure out the entities of the system and different aspects of data management. How do these connect, how do we need to access them and what operations will we be performing on them. Database system selection is part of this section. **NoSQL or SQL database selection is a common scenario. On the other hand, we may need to decide on what kind of storage needs to be chosen for photos and videos.**

- High-level Component design. If we try to design the system in one go, it is a tough task. So, it's better to break them as high-level components. Then, break those components into detailed design. High-Level Design (HLD) provides a comprehensive overview of the software development process along with the system architecture, applications, database management, and complete flowchart of the system and navigation. It's a **blueprint that consolidates the various steps and modules**, their objectives, variable components, results, architecture and how they link.

- Low-Level Design (LLD) **deals with the planning, coding, and execution of the various components, modules, and steps in the HLD, at an individual level**. Each module in an HLD has a unique LLD document that provides comprehensive details about how the module will be coded, executed, tested for quality, and integrated into the larger program. LLD provides actionable plans by deconstructing HLD components into working solutions.



Assumptions

Before we begin our design we need to get some estimates or make some assumptions about the scale and requirements of the system. We list few of them below

- 1 billion total users with 200 million daily active users (DAU)
- 1 million videos on the platform
- 1 thousand new uploads everyday
- 1 billion requests per day or approx 12k per second

Some numbers assumed here are actually way higher than the actual number, but we want our system to scale and be ready for the maximum need

Q- What data do we need to solve this problem?

Q- list sample data needed for recommendation system

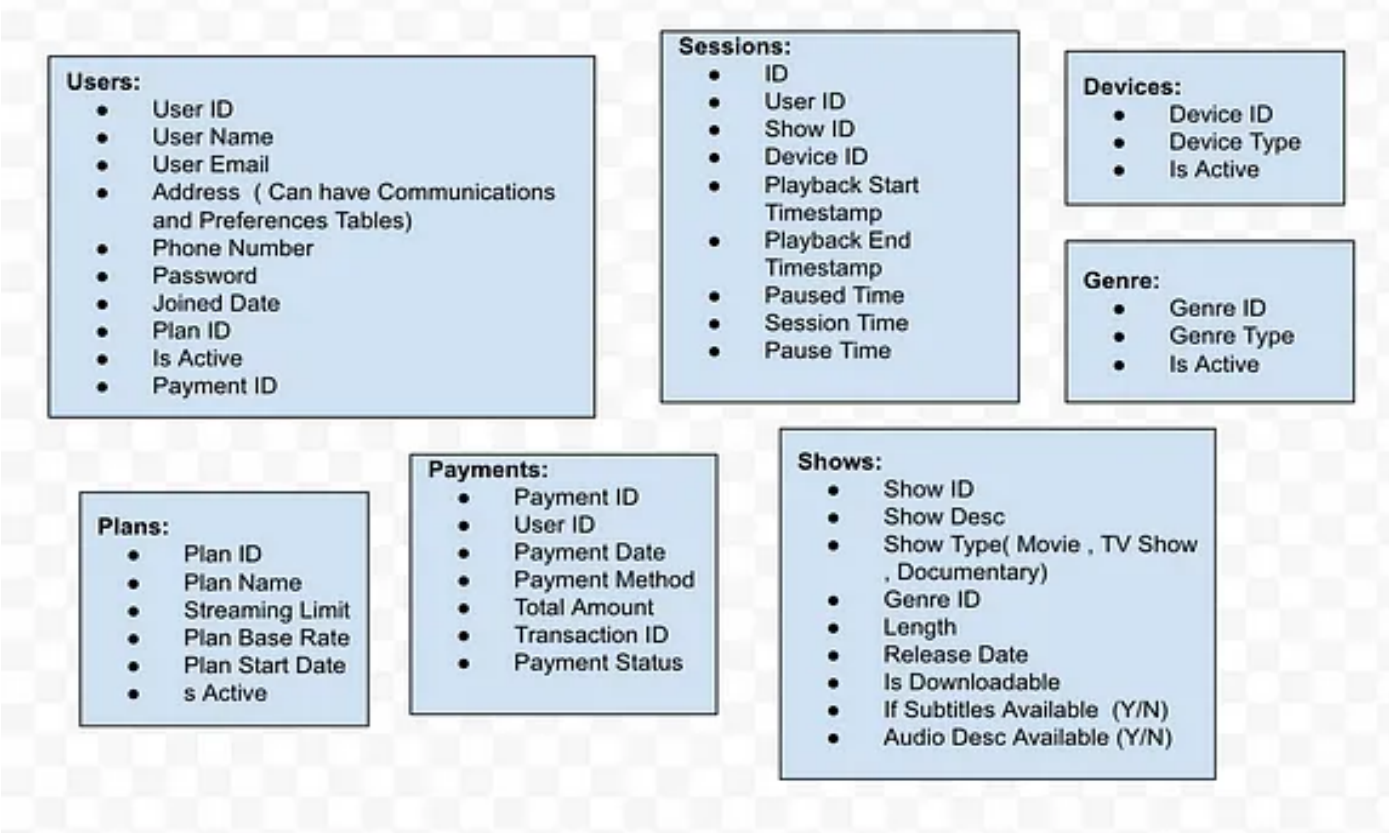
Now that we know what is the goal for our system and now we have the different features we need, we can start by listing out the data and the content for our system and how it needs to be handled.

1. We have user data:
    - Name
    - Email
    - Location
    - Languages
    - Subscription
    - …..
2. User interaction data:

- Video_ids (view history)
- Ratings
- Time of day
- Duration
- …

3. Video data:
    - Video_ids
    - Genre / categories
    - Cast
    - Release year
    - Avg rating
    - Stream url
    - …

There may be a lot of tables and data at netflix like plans/subscriptions, payments, video tags, descriptions, and a lot more data in each table than we have listed, for example the user interaction table might store the last viewed video or till the duration the video is watched for resuming it and so on. But for our features that we are implementing today we will consider only part of it.



Q- How do you store all these data

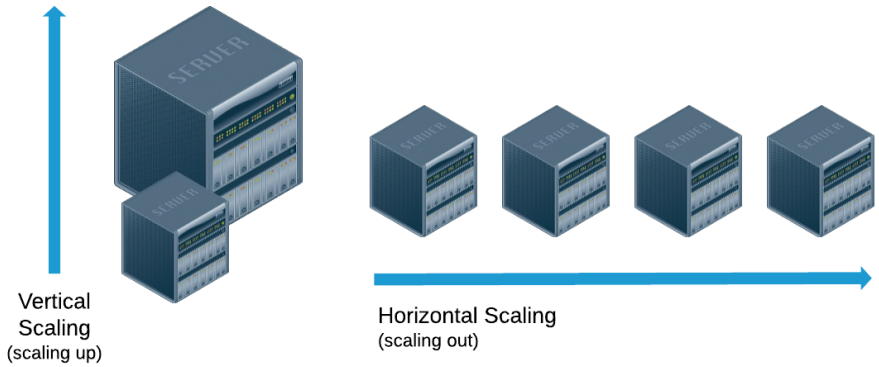
Q- What considerations for choosing the correct data storage

For this there are various options. Netflix uses a combination of **Postgre SQL and NoSQL database (Apache cassandra)**

- **PostgreSQL**:
    - PostgreSQL is highly reliable, scalable, stable and secure.
    - Support for clustering , redundancy and replication
    - The **video data and the user data data** are stored in sql format because they are of **fixed size and the schema is fixed.**
    - The amount of data that is stored there can be easily handled by vertical scaling and would still be pretty fast and work the best.
    - **billing information, transactions** are also stored in sql tables
    - This data can also be cached easily that would help in retrieving it fast.
- **NoSQL database**
    - The **user interaction** part is stored in the no sql database because not everyone will have the same interaction data, some people will watch more content and interact more, so it is better to have a **flexible schema** and this data will definitely be a lot more than the user data or video data.
    - If on average each person has viewed 20 videos the data will be almost 20 times the user data and this needs horizontal scaling of the hardware for best usage.

Scaling can be categorized into 2 types:

- **Vertical Scaling**: When new resources are added in the existing system to meet the expectation, it is known as vertical scaling.
  The vertical scaling approach increases the capacity of **a single machine** by increasing the resources in the same logical server. This involves adding resources like memory, storage, and processing power to existing software, enhancing its performance.
  Vertical scaling is not only easy but also cheaper than Horizontal Scaling. It also requires less time to be fixed.



- **Horizontal Scaling**: When **new server racks are added** in the existing system to meet the higher expectation, it is known as horizontal scaling.
  Consider a rack of servers and resources that comprises the existing system. (as shown in the figure). Now when the existing system fails to meet the expected needs, and the expected needs cannot be met by just adding resources, we need to add completely new servers. This is considered as horizontal scaling.
  Horizontal scaling is based on the idea of **adding more machines** into our pool of resources.

Horizontal Scaling	Vertical Scaling
When additional machines are added to the existing system to meet the higher expectation, it is known as horizontal scaling.	When new resources are added in the existing system (increasing RAM, CPU) to meet the expectation, it is known as vertical scaling
It is easier to upgrade.	It is harder to upgrade and may involve downtime.
It is difficult to implement	It is easy to implement
It is costlier, as new server racks comprise a lot of resources	It is cheaper as we need to just add new resources
Cassandra, MongoDB, Google Cloud Spanner	MySQL and Amazon RDS

Netflix **S3** serves as a data lake as well as data warehouse

- A data warehouse stores data in a structured format. It is a central repository of preprocessed data for analytics and business intelligence.
- On the other hand, a data lake is a central repository for raw data and unstructured data
- It is the repository of everything from different parquet files. This has resulted in allowing them to decouple storage from compute
    - Allowing to experiment and choose the right querying engine and also move as the industry moves.
- This also helped on simplifying the process of getting or connecting the data because S3 is compatible with most of the current services and easy to manage versions and which cluster has access to which data

https://docs.aws.amazon.com/athena/latest/ug/creating-tables.html
https://aws.amazon.com/s3/storage-classes/?nc=sn&loc=3

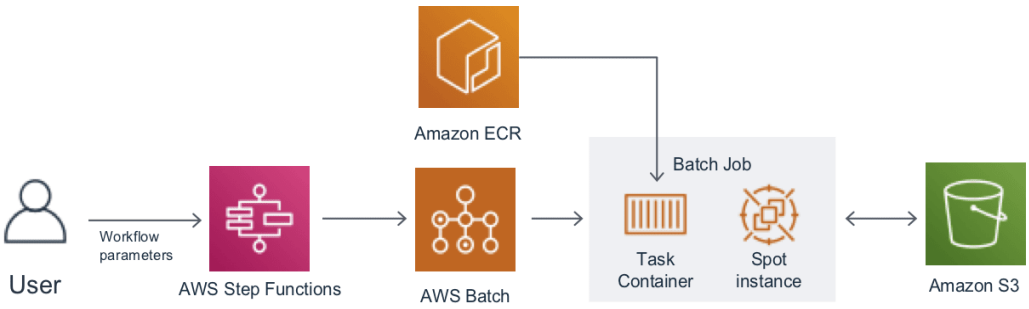
Compute And Orchestration

In 2008 Netflix went all-in on cloud migration and began moving all of its internally hosted infrastructure to AWS (Amazon Web Services). Today almost all of Netflix runs on VMs (virtual machines) in AWS. A customer's catalog browsing experience, content recommendation calculations, and payments are all served from AWS.

Compute environment

While EC2 supported advanced scheduling for services, this didn't help batch users (users that had to run multiple job or multiple variations simultaneously). **At Netflix, there is a significant set of users that run jobs on a time or event based trigger that need to analyze data, perform computations and then emit results to Netflix services, users and reports.**

At netflix there are **thousands of experiments running for every country** and every language that netflix operates in, this leads to **thousands of recommendation models** alone, so this can **require huge resources and can easily starve out tasks that are more important for the system to run.**



AWS Step Functions is a **workflow automation service** that can simplify orchestrating other AWS services. AWS Batch is a **fully managed batch processing service** that can dynamically scale to address computationally intensive workloads. Together, these services can orchestrate and run demanding workloads.

- Has support for long running workflows that take sometimes days to complete
- Netflix runs hundreds of experiments regularly (eg A/B test parallelly)
- Scale as required
- Work according to priority

AWS Batch is a set of batch management capabilities that enable developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS. **AWS Batch dynamically provisions the optimal quantity and different types of computing resources, such as CPU or memory-optimized compute resources, based on the volume and specific resource requirements of the batch jobs submitted.**

With AWS Batch, there is no need to install and manage batch computing software or server clusters, instead, you focus on analyzing results and solving problems. AWS Batch plans, schedules, and executes your batch computing workloads using Amazon EC2, available with spot instance, and AWS compute resources with AWS Fargate

AWS ECR holds the docker image that has the job definitions or what needs to run in the form of a docker container. It holds everything required for that job making it independent on the machine and making scheduling and changing between jobs easier.

-

Notebook instances

Netflix uses **sagemaker notebooks** for experimentation, because everyone is familiar with them and easy to work with customizable flexible and compatible with all sort of tools and frameworks, so teams can decide what frameworks and tools to use for their tasks and visualizations

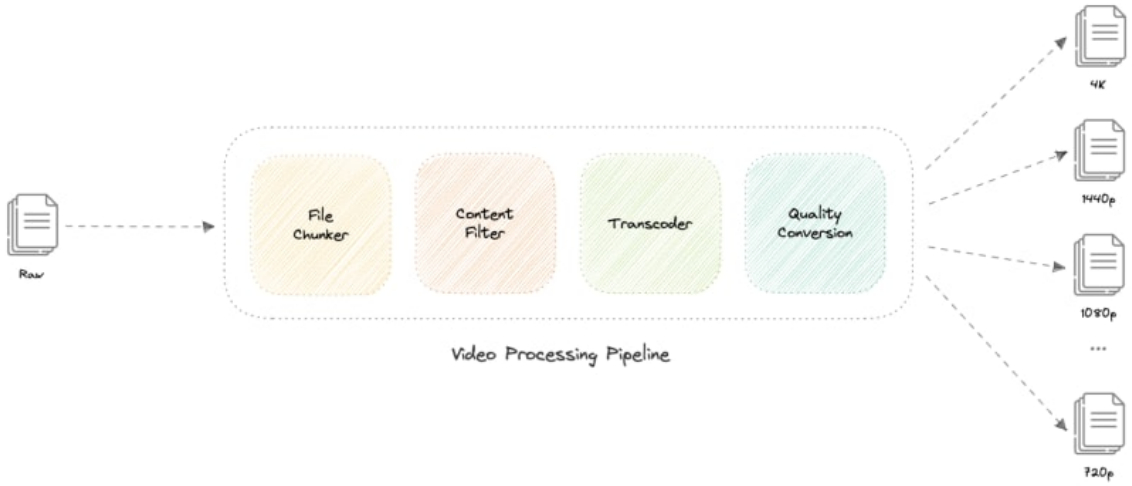
Q- How will you store the actual video content

Apart from all of this, Netflix has the major chunk of storage for actual video content that is available on the platform.
Netflix stores all these processed video data on **Amazon S3**, Which is a highly scalable and available storage platform for storing static data. Because it is theoretically infinitely scalable we don't have to calculate or provision storage before it is needed.

Q- How will you handle different resolutions and formats needed

All the data stored on the s3 is processed before saving there

- Netflix uses AWS for nearly all its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more—hundreds of functions that in total use more than 100,000 server instances on AWS.



Now whenever a video is uploaded it has to be encoded and converted to different formats and resolutions for various devices and compatible services. Doing this increases storage costs, but doing this at the run time according to the user demand means that it will increase the latency and we already know that each video on everage may be viewed approximately a million times and doing this would give us more computational costs than storage costs

- Compressing a 25Gb movie will take a lot of time to solve this problem. Netflix breaks the original video into different smaller chunks and using parallel workers in AWS EC2, it performs encoding and transcoding on these chunks converting them into different formats (MP4, MOV, etc) across different resolutions(4k, 1080p, and more).
    - There are encoding formats like H.264, H.265, AV!, etc that determine how these are compressed. Some of which are proprietary and some opensource.
    - All of them work in entirely different ways but the goal is same to compress the data, convert to the desired format

Video Resolution	Video Bitrate
240p	300 kbps
360p	500 kbps
HD 480p	1000 kbps
HD 720p	1500 kbps
HD 720p	2250 kbps
Full HD 1080p	3000 kbps
Full HD 1080p	4500 kbps
Quad HD 1440p	6000 kbps
Quad HD 1440p	9000 kbps
4K UHD 2160p	13,000 kbps
4K UHD 2160p	20,000 kbps

1. **Uploading content.**

   This is how it all is managed: the team uploads the video content that is processed by the above pipeline and stored in the **s3 bucket**. Other metadata and information are stored in the tables along with it at the time of uploading.

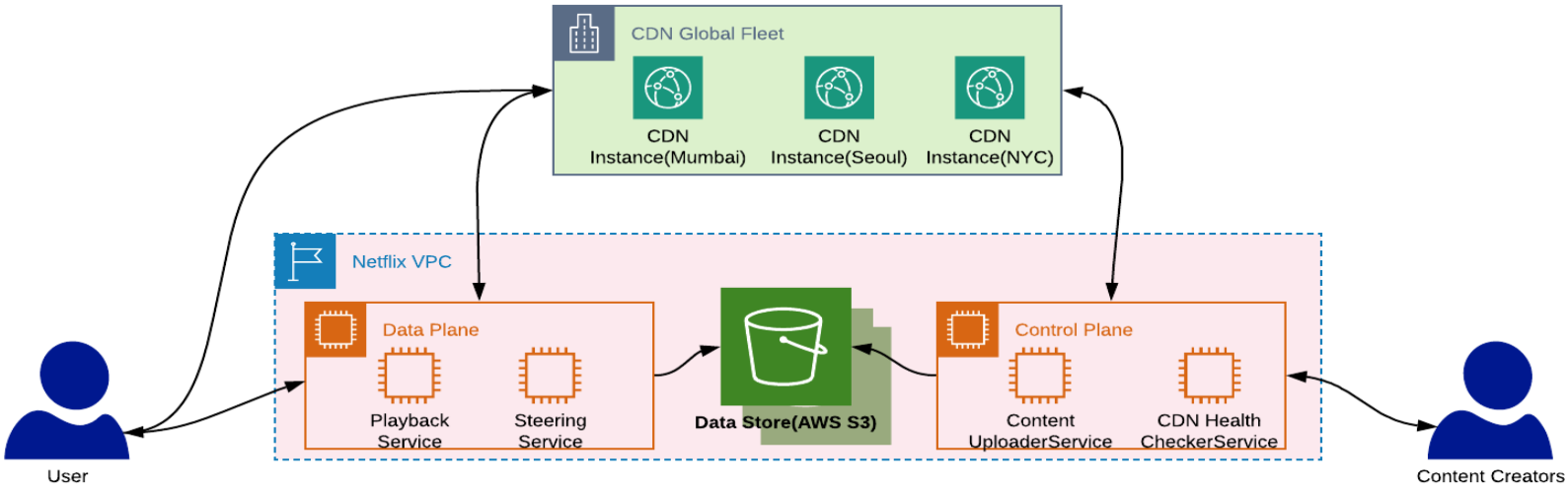
Q- How to handle video streaming for minimum latency

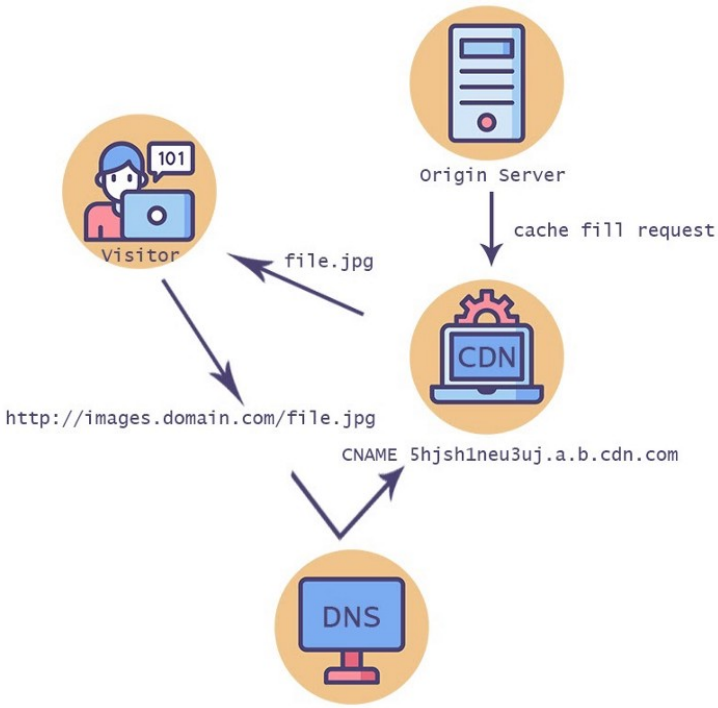
2. **Video streaming**

   Video streaming is a challenging task from both the client and server perspectives. Moreover, internet connection speeds vary quite a lot between different users. To make sure users don't re-fetch the same content, we can use a Content Delivery Network (CDN).

   - **A content delivery network (CDN) refers to a geographically distributed group of servers which work together to provide fast delivery of Internet content. The content is stored in servers closer to the users and fetched directly from there**

   Netflix takes this a step further with its Open Connect program. In this approach, they partner with thousands of Internet Service Providers (ISPs) to localize their traffic and deliver their content more efficiently.





Currently, they have Open Connect Appliances (OCAs) in over **1000 separate locations around the world**. In case of issues, Open Connect Appliances (OCAs) can failover, and the traffic can be re-routed to Netflix servers.

- So based on the traffic and demand netflix replicated the data from their s3 bucket to these cdn's for faster delivery.
- Also this way the latency is low and because it has so **many copies and locations present with the content**, it is a very rare possibility that all of it goes down ensuring the best uptime possible.

- An added advantage of this is that it also helps in **geoblocking of the content**, which is a little more robust and not easy to bypass

Q- How will you handle user interactions

3. user interaction

Now the users can easily get the required content in the desired format with a very low latency and the lowest chances of complete outage. The users can can now watch like dislike and have other interactions with the video

- The user interactions are stored in the **nosql database**. Because it is a no sql system adding and managing data that is not fixed is better in that regard and hence is used for all data like user watched videos likes, dislikes and other data that we mentioned earlier that does not have a fixed schema
- We know that some users will watch and interact more. So the horizontal scalability of this system is better for it.

	MOVIE-1	MOVIE-2	MOVIE-3	MOVIE-4
UID-1	LIKE - Yes TIMESTAMP 2:00:00 END No RECOMENDED Yes	LIKE - No TIMESTAMP 2:03:00 END Yes RECOMENDED No		
UID-2	LIKE Yes TIMESTAMP 1:58:21 END No RECOMENDED No	LIKE Yes TIMESTAMP 2:03:00 END Yes RECOMENDED Yes	LIKE No TIMESTAMP 1:20:00 END No RECOMENDED No	LIKE No TIMESTAMP 1:40:00 END Yes RECOMENDED Yes
UID-3	LIKE Yes TIMESTAMP 3:00:32 END Yes RECOMENDED Yes			

Q- How to handle user recommendations and features needed for it

4. Recommendation system

Usually Netflix will show you only 40 to 50 video options, yet they have many thousands of videos available.

How does Netflix decide? Using machine learning.

That's part of the big data processing and analytics we just talked about. Netflix looks at its data and predicts what you'll like. In fact, everything you see on a Netflix screen was chosen specifically for you using machine learning.
Whenever you access the Netflix service, the recommendations system strives to help you find a show or movie to enjoy with minimal effort. The estimated likelihood that you will watch a particular title in the catalog is based on a number of factors including:

- your interactions with our service (such as your viewing history and how you rated other titles),
- other members with similar tastes and preferences on our service, and
- information about the titles, such as their genre, categories, actors, release year, etc.

In addition to knowing what you have watched on Netflix, to best personalize the recommendations it also looks at things like:

- the time of day you watch,
- the devices you are watching Netflix on, and
- how long you watch.

When you create your Netflix account, or add a new profile in your account, we ask you to choose a few titles that you like. We use these titles to "jump start" your recommendations.

We already know how the different recommendation systems like collaborative filtering, content based filtering and matrix based methods work, the main fascinating thing is how netflix uses all of these to get the best results

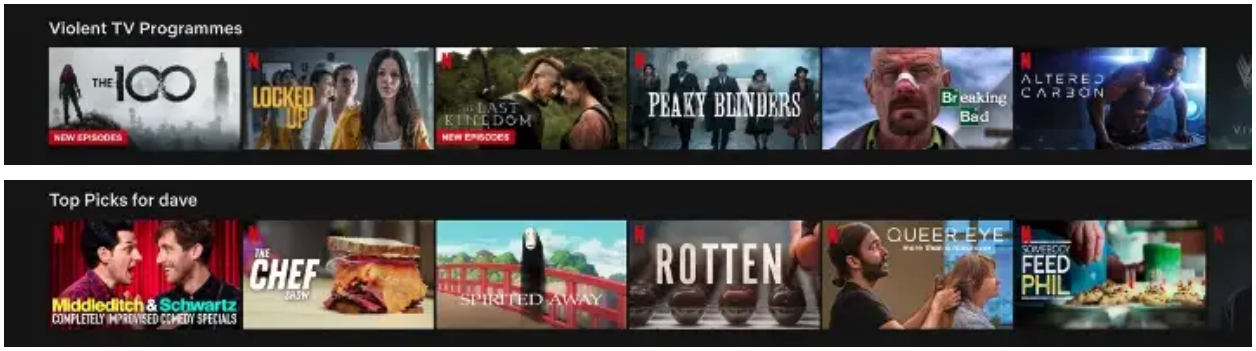
The recommendation system of netflix is a lot more complicated, and it is not just a single algorithm that handles it all. The results are from some different set of algorithms

Q- What kinds of recommendations do you see on netflix and how to recreate them

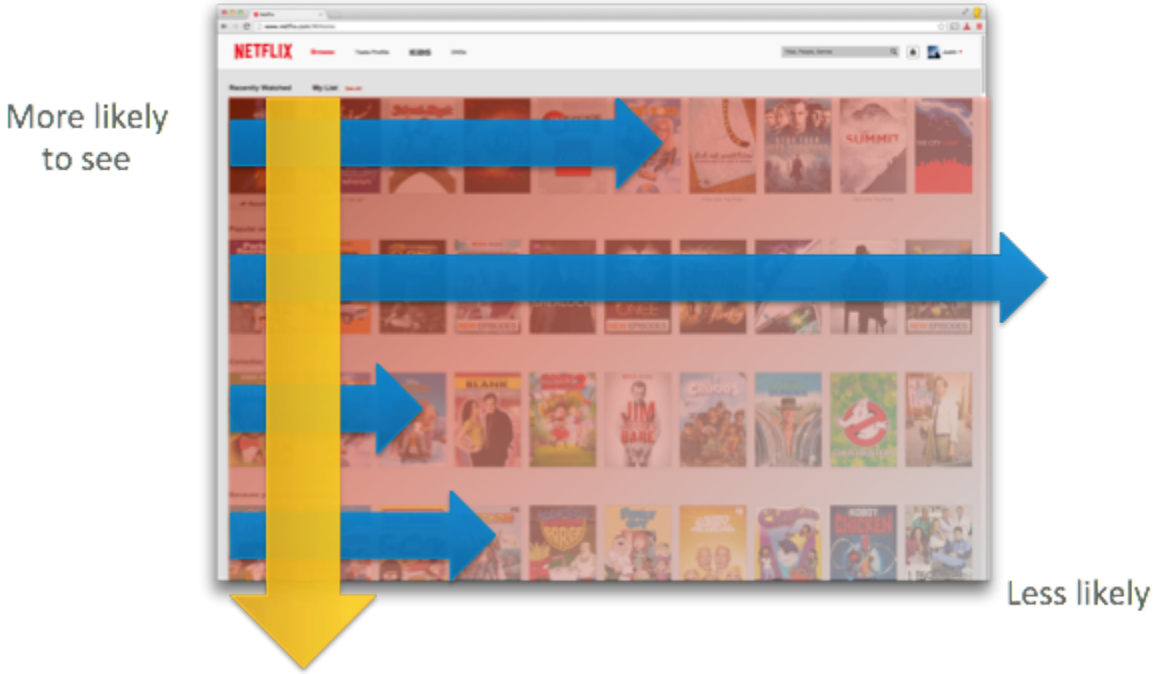
Rows, rankings and title representation
In addition to choosing which titles to include in the rows on your Netflix homepage, the system also ranks each title within the row, and then ranks the rows themselves, using algorithms and complex systems to provide a personalized experience. To put this another way, when you look at your Netflix homepage, the systems have ranked titles in a way that is designed to present the best possible ordering of titles that you may enjoy.

In each row there are three layers of personalization:

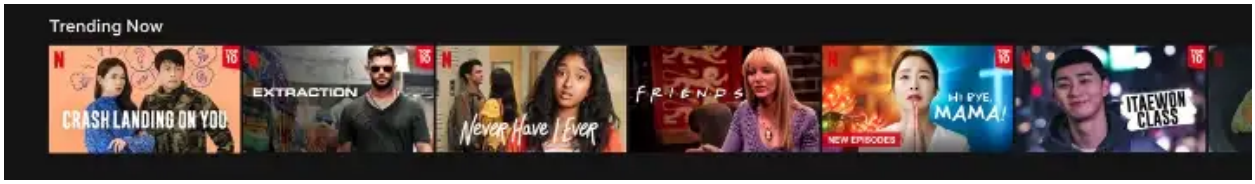
- the choice of row (e.g. Continue Watching, Trending Now, Award-Winning Comedies, etc.)
- which titles appear in the row, and
- the ranking of those titles.



The most strongly recommended rows go to the top. The most strongly recommended titles start on the left of each row and go right -- unless you have selected Arabic or Hebrew as your language in our systems, in which case these will go right to left.



Trending functionality will be based on top of the search functionality. We can cache the most frequently searched queries in the last N seconds and update them every M seconds using some sort of batch job mechanism. Based on the location or language or similar user this can also create a great method to recommend items.
Events that have a seasonal trend and repeat themselves (e.g. Valentines day leads to an uptick in Romance videos being consumed) are also considerd



Video-Video Similarity Ranker —a.k.a. Because you watched (BYW)
This algorithm basically resembles that of a content-based filtering algorithm. Based on an item consumed by the member, the algorithm computes other similar items (using an item-item similarity matrix) and returns the most similar items. Amongst the other algorithms, this one is unpersonalised as no other side features are utilized.

To make our system more resilient we can do the following:

- Running multiple instances of each of our services.
- Introducing load balancers between clients, servers, databases, and cache servers.
- Using multiple read replicas for our databases.
- Multiple instances and replicas for our distributed cache.