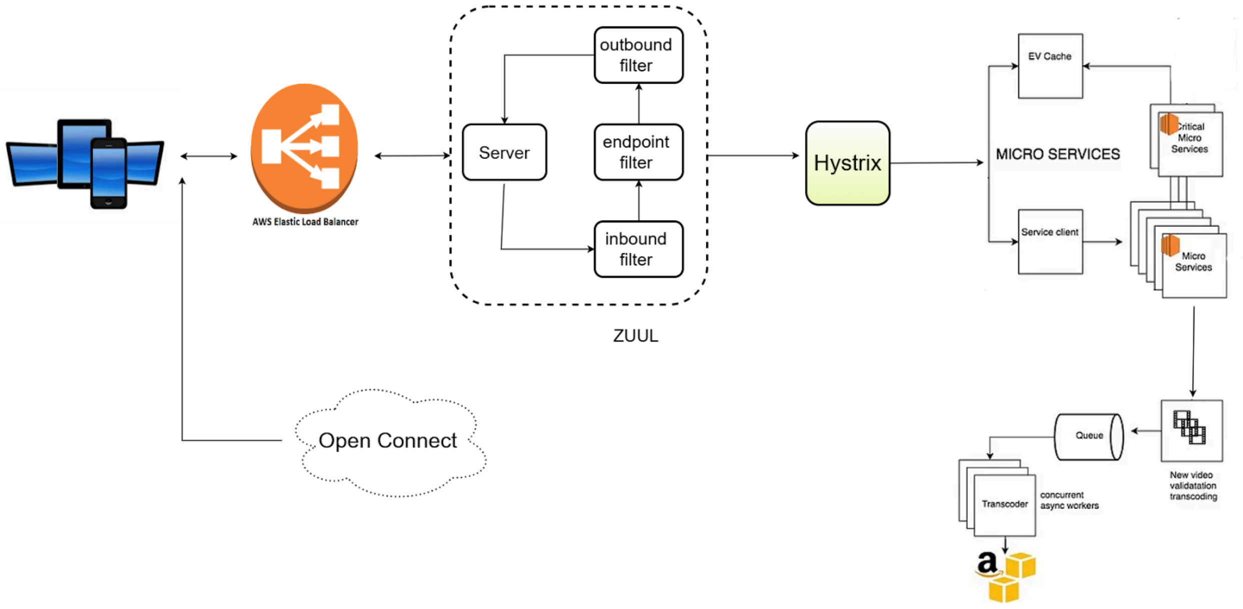
System Design of Netflix

Ref: https://www.geeksforgeeks.org/system-design-netflix-a-complete-architecture/
https://medium.com/@narengowda/netflix-system-design-dbec30fede8d



Netflix operates in two clouds: AWS and Open Connect.

Both clouds must work together seamlessly to deliver endless hours of customer-pleasing video.
High level working of Netflix-

- The client is the user interface on any device used to browse and play Netflix videos. TV, XBOX, laptop or mobile phone etc
- Anything that doesn't involve serving video is handled in AWS.
- Everything that happens after you **hit play i**s handled by O**pen Connect**. Open Connect is Netflix's custom global content delivery network (CDN).
    - Open Connect stores Netflix video in **different locations** throughout the world. When you press play the video streams from Open Connect, into your device, and is displayed by the client.

Netflix has 3 main components -

- OC or netflix CDN
- Backend: AWS cloud
- Client - ReactJs : 1) startup speed, 2) runtime performance, and 3) modularity

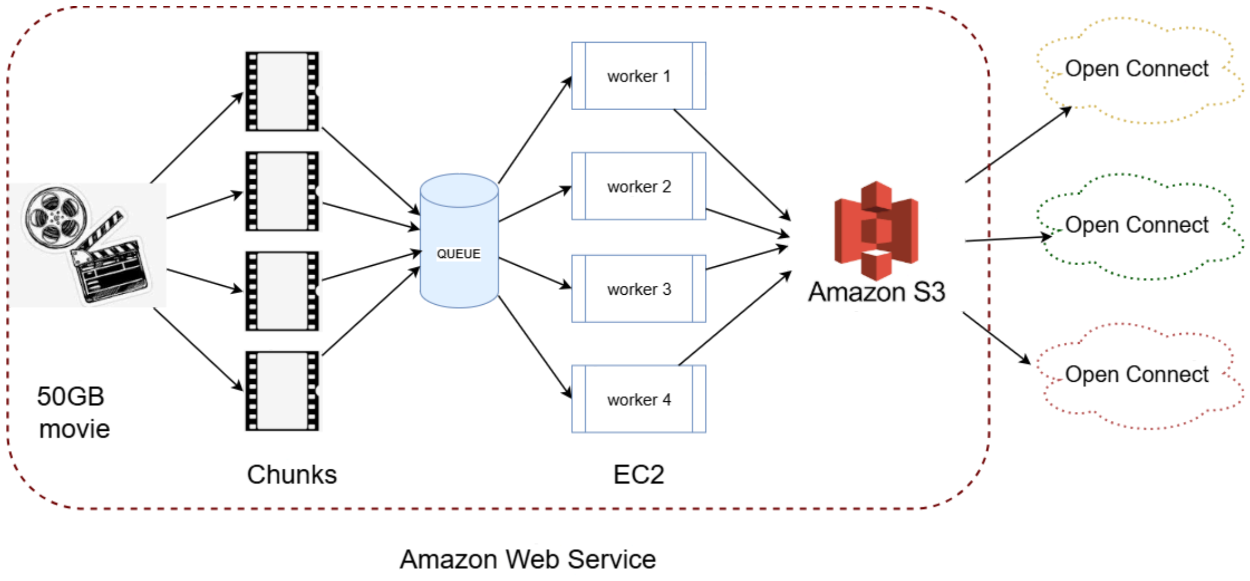
How Netflix onboard a movie/video:

Before this movie is made available to users, Netflix must convert the video into a format that works best for your device. This process is called **transcoding** or encoding.

Whys do we need to do it? why can't we just play the source video?
The original movie/video comes in a high definition format that's many terabytes in size. Also, Netflix supports 2200 different **devices**. Each device has a video format that looks best on that particular device.

Netflix also creates files optimized for different **network** speeds. If you're watching on a fast network, you'll see the higher quality video than you would if you're watching over a slow network.

And also depends on your **Netflix plan**. that said Netflix does create approx 1,200 files for every movie



A **50GB movie** is divided into smaller chunks of a few minutes each. These chunks are then passed into a **queue**. The queued chunks are distributed to **workers**, where each worker is responsible for converting the chunks into a specific **codec and resolution**.

*** Since processing an entire **50GB movie** is a massive task for a single worker, the movie is stored in **chunks in a database**. When delivering the movie to a user, these chunks are **merged together** before being sent.

After conversion, the **workers merge the processed chunks** and store the final version in **Amazon S3**. From there, Amazon S3 sends **1,200 instances of the same movie** to multiple **Open Connect servers**. These Open Connect (OC) servers are responsible for **delivering the content to the end users or clients**.

Advantages of OC

- Less expensive

- Better quality
- More Scalable

How netflix delivers the content

When the user loads Netflix app All requests are handled by the server in AWS Eg: Login, recommendations, home page, users history, billing, customer support etc. Now you want to watch a video when you click the play button of the Video. Your app constantly and automatically figures out the best OC server, best format and best bitrate for you and then the video is streamed from a nearby Open Connect Appliance (OCA) in the Open Connect CDN.

Netflix does not send the entire movie at once; instead, it streams the content in **chunks**. However, these chunks are **not divided based on time**. This is because buffering during a crucial scene would lead to a **bad user experience**.

Instead, **Netflix segments a movie or series based on Scene Recognition**, ensuring that users do not experience buffering in the middle of an **important scene**.

When sending content to users, **Netflix sometimes sends a single chunk and sometimes multiple chunks**. The movie, series, episode, or scene that a user is watching is categorized into two types:

1. Sparse Content

- Users do **not** have a strong preference for watching these movies continuously.
- In this type of content, Netflix **only** sends the **specific chunk** that the user requests.

2. Dense Content

- Users tend to watch multiple chunks continuously in this type of content.
- When a user requests a chunk, Netflix **sends multiple chunks ahead of it**, ensuring smooth playback.

ELB:

Netflix uses Amazons Elastic Load Balancer (ELB) service to route traffic to our front-end services. ELB's are set up such that load is balanced across zones first, then instances. This is because the ELB is a two-tier load balancing scheme.

- The first tier consists of basic DNS based round robin load balancing. This gets a client to an ELB endpoint in the cloud that is in one of the zones that your ELB is configured to use.
- The second tier of the ELB service is an array of load balancer instances (provisioned directly by AWS), which does round-robin load balancing over our own instances that are behind it in the same zone.

Netflix Zuul

Frontdoor/Gateway Service for All Device Requests to the Backend

The **gateway** acts as an **entry point** for all incoming requests and **routes them** to different backend services.

Filter Types in the Gateway Service

1. **Inbound Filters** (Before Proxying)
    - Applied before forwarding the request.
    - Used for **authentication, routing, or modifying requests**.
2. **Endpoint Filters**
    - Can either **return a static response** or **forward the request to the backend service** (also called the origin).
3. **Outbound Filters** (After Response is Returned)
    - Applied **after** receiving a response from the backend.
    - Used for **compression (gzipping), metrics collection, and adding/removing custom headers**.

Services Provided by Zuul

1. Authentication and Security

- Every incoming request is **checked** to ensure it is coming from an **authenticated user**.

2. Insights and Monitoring

- Tracks the **number of requests per second**.
- Monitors the **health of EC2 instances** based on the ratio of **incoming requests vs. successful responses**.

3. Dynamic Routing

- Determines **which microservice** should handle the request.

4. Stress Testing

- **Sends dummy requests** to EC2 instances to check how long they can handle requests before **failing**.

5. Load Shedding

- Allocates **capacity** for different types of requests.
- Drops requests that exceed the allocated **limit**.

6. Static Response Handling

- Some requests **do not need** a microservice for processing.
- In such cases, **Zuul directly responds** instead of routing them to an internal cluster.

7. Multi-Region Resiliency

- When a request arrives, **Zuul determines which AWS region** (e.g., Bangladesh, India, Pakistan) should handle it and routes the request accordingly.

Netflix Hystrix

Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries. which helps in

1. **Stop cascading failures:**

Suppose there is a microservice **X**, which depends on **X1, X2, and X3**.

- If **X3 fails to respond**, then **X2 won't receive a response**.
- If **X2 doesn't get a response**, then **X1 also won't receive a response**.
- As a result, **the user won't receive any response either**.

This **chain reaction of failures** is called **Cascading Failure**.
Detecting such failures requires multiple stages, leading to **wasted time**.

Key Features of Hystrix

1. **Circuit Breaker**
    - Monitors requests to a microservice.
    - If **failure rate exceeds a threshold**, Hystrix **opens** the circuit and **stops further requests**.
    - Once the service is stable again, Hystrix **closes** the circuit and resumes requests.
2. **Timeout Handling**
    - If a service takes too long to respond, **Hystrix automatically fails the request** instead of waiting indefinitely.
3. **Fallback Mechanism**
    - If a service fails, Hystrix **provides an alternative response** (e.g., cached data or a default message).
4. **Bulkheading (Resource Isolation)**
    - Each service gets its **own thread pool**, preventing one slow service from consuming all system resources.
5. **Request Caching**
    - **Reduces duplicate calls** to microservices by caching responses.
6. **Monitoring & Metrics**

    - Provides **real-time insights** into service failures, latencies, and circuit breaker status.

How to make microservice architecture reliable?

1. Hysterix
2. separate critical services: What Netflix does is, they identify few services as critical (so that at last user can see recommended hit and play, in case of cascaded service failure) and these micro-services works without many dependencies to other services.
3. Stateless Services:
   One of the major design goals of the Netflix architecture's is stateless services. These services are designed such that any service instance can serve any request in a timely fashion and so if a server fails it's not a big deal. In the failure, case requests can be routed to another service instance and we can automatically spin up a new node to replace it.

EVCache:

When a node goes down all the cache goes down along with it. so what Netflix did is they came up with EVcache. It is a wrapper around Memcached but it is sharded so multiple copies of cache is stored in sharded nodes. So every time the write happens, all the shards are updated too...

When cache reads happens, read from nearest cache or nodes, but when a node is not available, read from a different available node. It handles 30 million request a day and linear scalability with milliseconds latency.