# Netflix Onboarding New Content

## Uploading new content

### Challenges we face

The challenges we will face is we need to store in multiple **format** because many people will have different internet speed.

The way we compress the video is called **codec.**

If we reduce the file size the quality of the video will go down as it is **lousy compression.**

**Second thing** what Netflix does is toplay with different resolution. When we play it with mobile and monitor we need different resolutions.

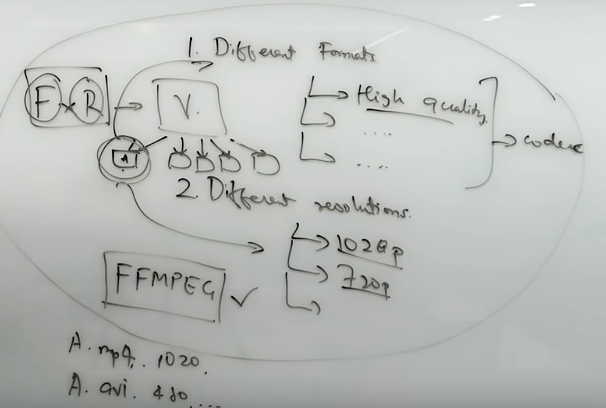
So single video will have multiple formats and multiple resolutions. Each of the format and resolution will form a **tuple.**

Number of Format (F) \* Number of Resolution(R) = Number of videos we will be processing (V).

To make the video processed we need to run it through the process which will may make a 6GB video compress to 1GB. But this process will take time and if we give all the responsibility to single computer it will take time and could be a single point of failure.

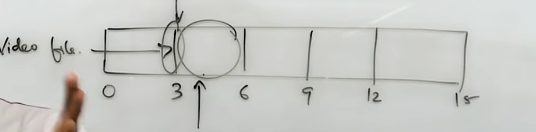
Netflix take each video and break it to **chunks** and each of the chunk will be run through different **resolution** and **format.**

So one resolution, one format and one chunk that’s one **task.**



### Story of processing chunks

Initially we will start with one chunk, where video file is broken into chunks of some minutes (ex. 3min) each. So every processor is doing the equal amount of work.



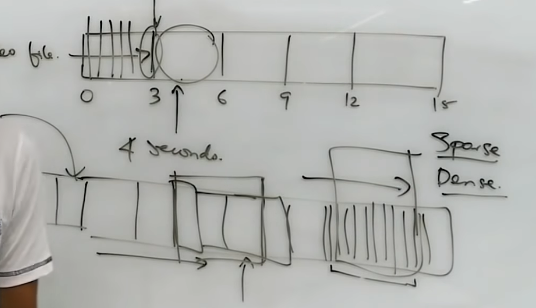
Here the problem is if there is a continuous scene (car chase) and exactly at 3min for next chunk api call happens then the video will start processing and buffering which will impact the user experience.

So instead of time we can think of braking on **scene.** Where each scene will have multiple **shots** of many fine grained chunks (ex. 4 sec each shot). We can collate shots to make a **scene.**

Here if the user has clicked on any point of the video which belongs to a particular scene, so the entire block of the scene will be loaded at one shot and user experience will be better.

Netflix sees the entire movie and breaks it inform of **chunks.** If we arbitrarily moves from point to different point, Netflix consider this movie is a **sparse** movie. Then its recommendation algorithm will being smart will not send lot of data but rather only the data user is asked for. Because user could probably clicking different points in that buffer.

For very engaging or dense movie where peoples are seeing continuously then there is a high probability people will watch it continuously. So Netflix will go ahead and load future parts and gets it on to your computer and shows it.



## Storing the data and performance

Netflix stores all the data in **Amazon S3** bucket which is cheap and used to store static data don’t change so frequently.

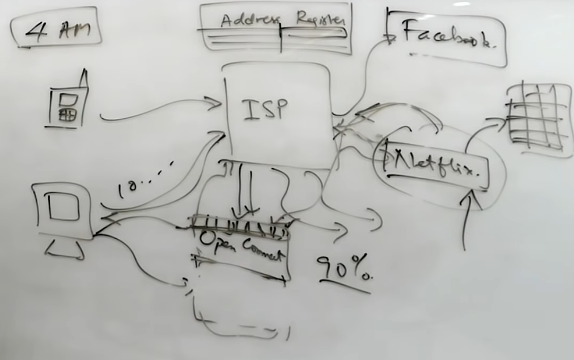
### Performance

Netflix even YouTube leverage the **ISP (**Internet Service Provider) to **cache** the data (pre computed) at ISP end. Where the popular and trending videos will be cached. So on request rather going to Netflix server and load the data which will be slow, data will be loaded from ISP cache.

These boxes are like hard drive placed by Netflix on every ISP end, called **Open Connect.**

**90%** of traffic is getting handled from these boxes.

In the night time when load is slow data from Netflix can be written back to these ISP boxes.



# System Architecture

## Getting started

### Components involved

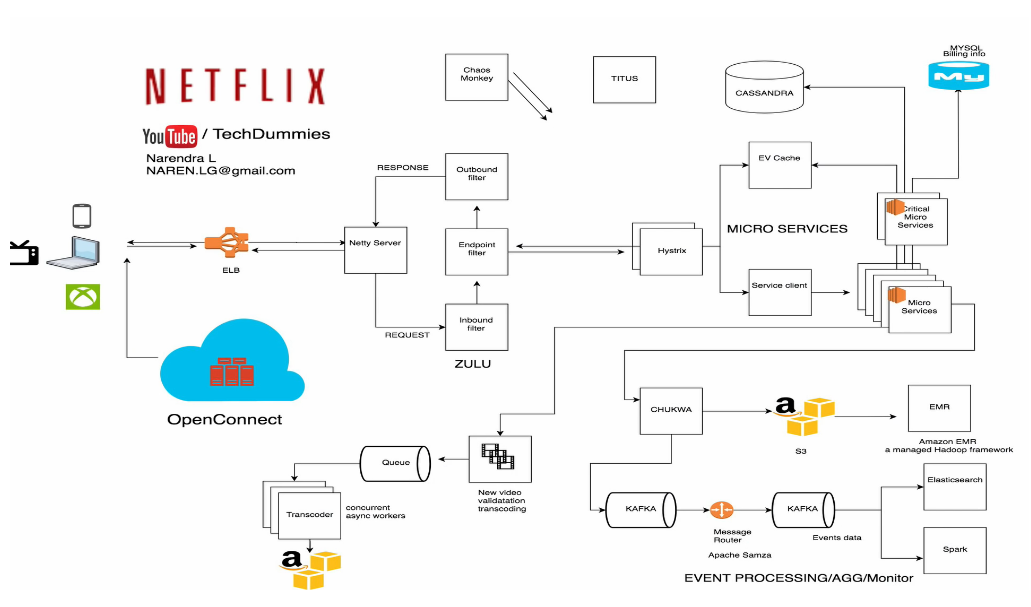
Below are the technical components Netflix used to deliver seamless service to users.

* Open Connect
* AWS
* Backend
* Client

**Open Connect** is Netflix own content delivery network (CDN). Which are also called **Edge locations.** Where we can keep the static and frequently accessed data w.r.t. that geographical location for faster access of data if request comes from that area.

## Netflix system design

### Architecture diagram



## Discussion individual client

Apart from Open Connect (Netflix’s CDN) everything is in AWS cloud.

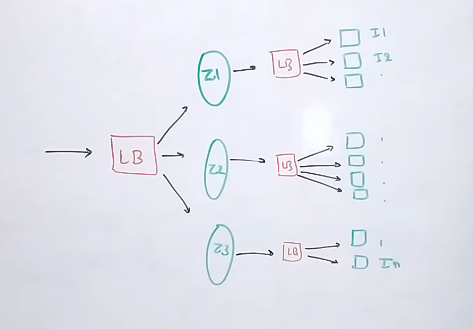
### Client

Netflix supports verities of range of client’s smart TV, android, ios, gaming consoles, web app. The web app is written using React JS.

### Elastic Load Balancer (ELB)

Netflix uses ELBs to route the traffic to multiple front end service. Using **ELB** load is balanced in **two tier.** In first load will be balanced between **Zones** using DNS based round robin and from there request will be forwarded to other load balancer and load will be balanced across instances.

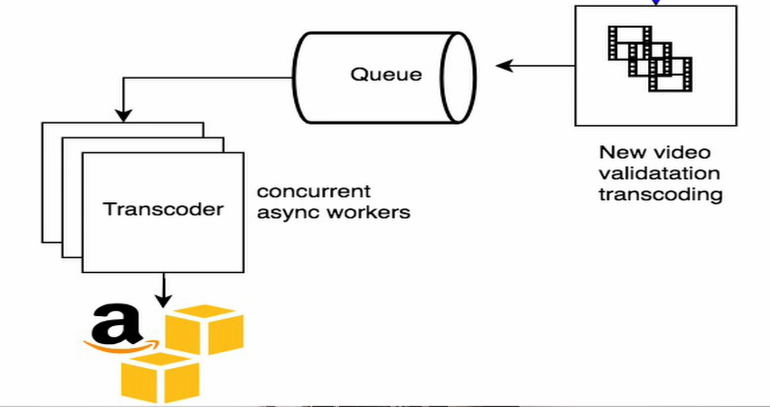
**Zones** are kind of logical grouping of servers. There could be 3 zones in India and 10 zones in United State.

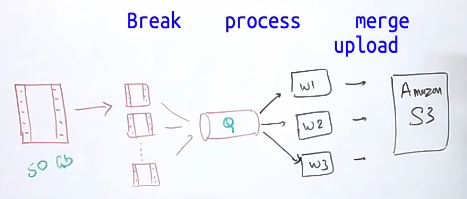


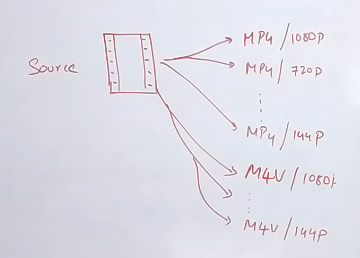
### Video onboarding

Before video is made available to user. Netflix do lots of preprocessing for finding errors, converting different formats and resolutions and these process is called **transcoding**.

To make the video streams in verities of network speed Netflix creates many different processing file for original file in different resolutions and loads it based on network speed.





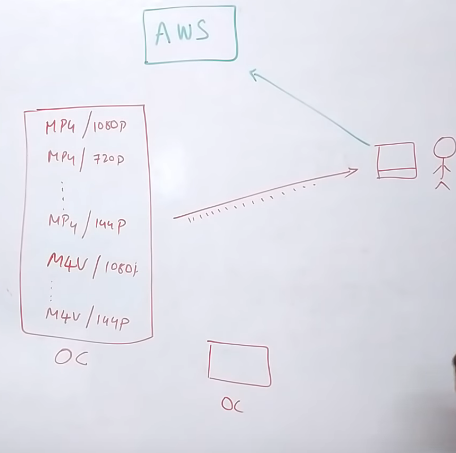


All these copies will be pushed to each and every **open connect** distributed server present across the world.

On user loads the Netflix app on its mobile or device. All the request like request like

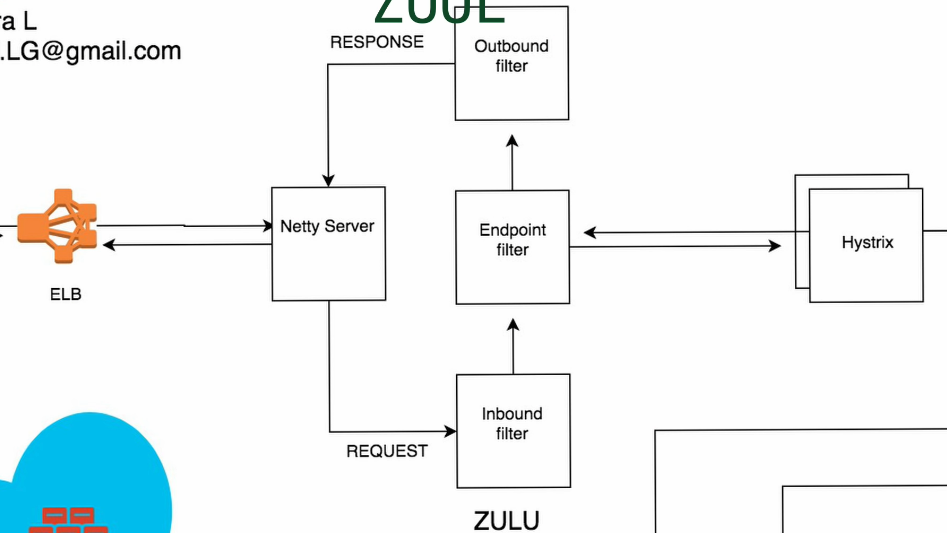
* Login, recommendation, customer support, Billing are handled by **AWS** instances
* Moment we found the video to watch and hit play button. The request will try to find best **Open Connect** server nearer for seamless experience and loads from that and also the clients are so intelligent that while video is streaming it searches for other nearer **open connect** server based on quality and bandwidth to serve with which helps maintaining the **load** between Open connect servers.

Your search, video viewing patterns, likes will be saved in AWS data centers and Netflix does create **Machine Learning** model using that data to build the recommendation engine.



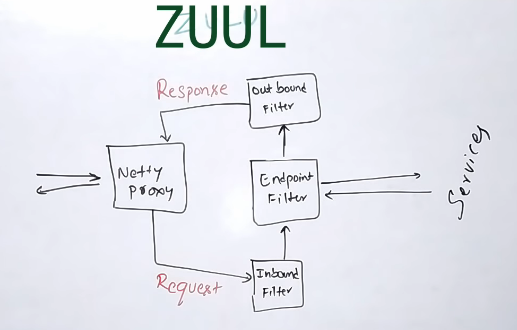
### Gateway service

Netflix has its own gateway service **ZUUL.**



ZUUL is a **Gateway Service** provides

* Dynamic routing
* Monitoring
* Resiliency
* Security
* Connection management
* Proxying the request



This is **Netty Server** based proxy, which will take the request first and will proxying the request to **Inbound filter.**

Inbound filter run before proxying the request and can be used for authentication, routing or decorating the requests.

**Endpoint filter** can be used to return the static response or to forward the request to backend services. Once backend filter responds the endpoint filter will forward the request to **outbound filter.**

**Outbound Filter** can be used for gzing the content, for add or remove header, calculating the metrics etc and will send back the response to client through Netty server.

Advantage of using gateway service

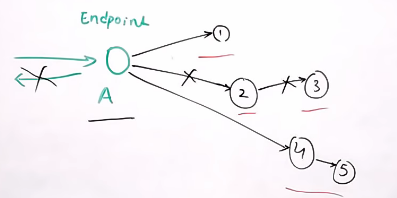
* Shard or distribute the traffic between servers of different version by putting rules in endpoint filter.
* We can do load testing for new on boarded server by routing traffic to that server.
* We can test upgraded services in one or few server in real time with some part of traffic
* We can filter bad request.

### Circuit breaker with Hystrix

Hystrix is a latency and fault tolerant library design to isolate the points of access to remote system, services or third party libraries.

Of one of the call is slow or down, the whole end point might suffer for that cause even other services or calls are well enough.

Hystrix stops cascading failure of the system. We can decorate each and every microservice.



Application of Hystrix

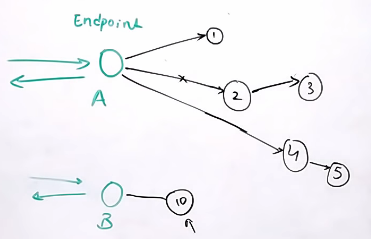
* Timeout calls > time
* Reject request when thread pool is full
* Disconnect the service > errors percentage threshold
* Fail back to default response
* Metrics

### So many microservices

* Critical endpoint
* Stateless (all the endpoint stateless)

With so many microservices in place communicating with each other, how we can have a reliable communication because failing could be cascading.

One of the solution is to separate out the **critical services.** If one endpoint is very critical for our web app to load then we can **separate** that endpoint and have less dependency for that.



Critical end points are the endpoints, say in worst case all of the microservices are down but still user can do some basic stuffs like view home page, search, favorite video. And these end points are critical endpoint. we need to separate those out and make them **highly available.**

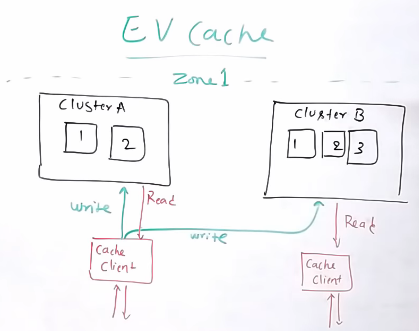
**Stateless**

Next critical thing is to make all the end points stateless. That is when one m/s is taking to other endpoint and because of some reason that endpoint is throwing error. We should always switch back to different server automatically and get the response. That means state should not be preserve in server cache or local memory. So we should build our endpoint keeping **stateless** in mind so always the user should get the response through any different server available at time.

### EV Cache

In any application there are so many endpoints so the data can be cached or response can be cached. These API are very good candidate for relieving the pressure on actual server.

Netflix builds their own Caching layer called EV Cache on top of **Memcache.** They have deployed number of clusters on EC2 instances in which there are so many nodes of Memcache db. And they even have their own **cache client**



Whenever there is a **write** is happening to cache. The **Cache Client** writes it to every cluster i.e. every node available in that particular cluster.

And when **read** happens, read happens in that cluster. That way cache is distributed across the server and read is happening from very nearest server.

Advantage of having caching layer in system

* Throughput
* Latency
* Cost saving for deploying more number of endpoint server.

Netflix own implementation is on **SSD** not on **RAM.** Which saves the cost incur for deploying RAM.

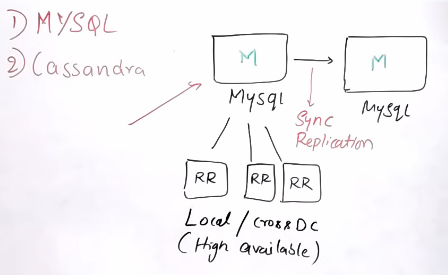
### Databases

Netflix uses two different databases, for different kind of purpose

* MySql – Billing, Transaction, User information’s as these needs ACID compliance.
* Cassandra – User history, search history, viewing patterns kind of big data storage

Netflix has a kind of **Master – Master** setup for Mysql replication (deployed on AWS EC2 large instances).

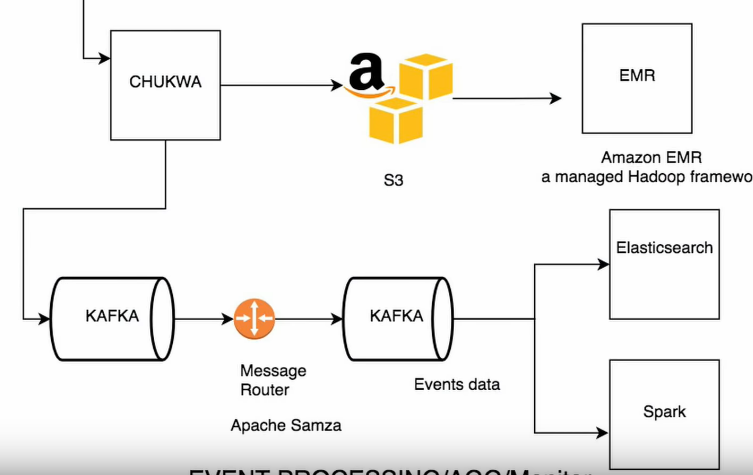
Netflix also has **read replicas** for each and every nodes that way the **scalability, high availability** and **resilience** can be managed for **RDBMS** databases.



All the read queries will be redirected to **RR (Read Replicas)** and only the write queries are redirected to **Master.**

In case of **Master failure** just need to change the DNS configuration in **Route 53** to redirect to backup Master (all the write queries).

### Kafka and Chukwa



Netflix uses **Kafka** for event based communication (event sourcing). Approximately 500B events everyday 1.3 PB of data.

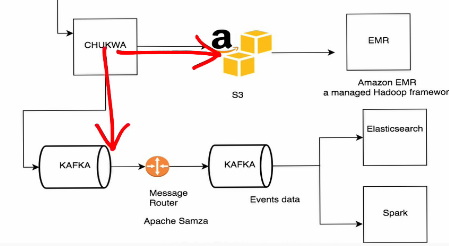
**Apache Chukwa** is open source data collecting system for collecting logs or events in a distributed system. It’s built on top of HDFS and Map Reduce framework.

All the logs and events from different part of distributed system will be sent to Chukwa. In Chukwa we can either do monitoring or analysis or use dashboard to view the events. Once Chukwa get the data it forwards to AWS S3 and Kafka

**Kafka routing service** is responsible for sending data from fronting Kafka to various sinks like AWS S3, Elastic search and other secondary Kafka.

Routing of these messages is done using **Apache Samza** framework.

When Chukwa sends data to Kafka, it can send full or filter stream. Sometimes we need to apply further filter on Kafka streams returned from Chukwa, which is why we have **router** to consume from one Kafka topic and produce to another Kafka topic.



### Elastic search

150 clusters

3500 instances

Netflix has very huge data produced everyday approximately 150 PB of data. Which needs to be saved for visualization using Kibana.

Netflix using Elastic search for customer support for **debug** or trace for customer queries to see what’s happening in the system.

Ex. Customer is complaining about video is not streaming properly.

Here the customer support needs to know what is happening. All they do is go to elastic search, search with user information and get to know all the events happening for that particular user and what error is been thrown.

Can be useful to check

* Errors in signup
* Errors in login
* Keep track of resource uses.

### Spark

Netflix is using Spark and Machine learning to do all recommendation kind of stuff. Spark is used for content recommendation and personalization. Majority of M/C learning models runs on large Spark clusters and then these models are used to do

* Sorting
* Row selection
* Relevance rank

Which decides when user login, and front page loads, what content should be displayed to the user.

Netflix personalizes the row selection for particular user and all decision happens based on user historical data and again based on this selection it will sort the data on the way based on user personalization again.

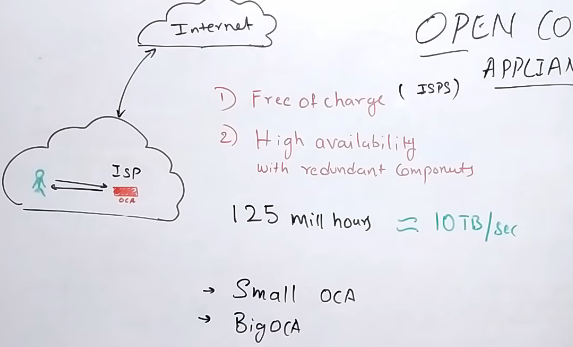
Relevance ranking will be calculated for showing user recommendation.

### Movie recommendation system

Movie recommendation system is very crucial system in Netflix, which shows relevance movie/content list to user to retain the user base. Because Netflix has very huge base and user can’t search everything by their own they mayn’t know all the lists which Netflix has. So in only way it can be reached to the user is through recommendation engine.

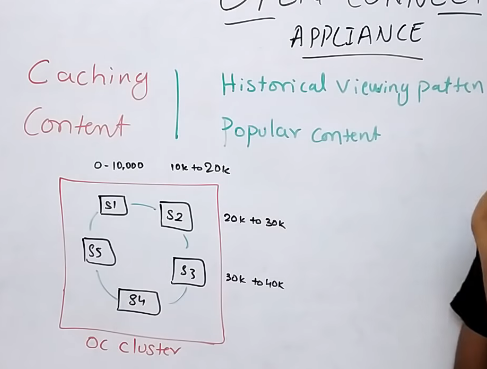
* Collaborating filtering (Movie title, user taste, director, time of the day)
* Content based filtering

### Open connect



**Open Connect** is also a cluster. Netflix distributes the cache among multiple servers in Open connect cluster using **consistent hash ring.**

**Hash** the file name, based on the value allocate it to a pre-configured server.



When the request for same file comes to cluster again same thing happens. Take the file name, do the hash, with the hash value check for the server where the range falls into. If the cache is not present then the request will be forwarded into Netflix main server in AWS cloud get the content and save in OC cluser.