



Posted on Jun 24, 2021 • Updated on Jul 27, 2021



372



49



2



[#openconnect](#) [#systemdesign](#) [#netflixbackend](#) [#aws](#)

.....

1 Netflix System Design- How Netflix Onboards New Content

2



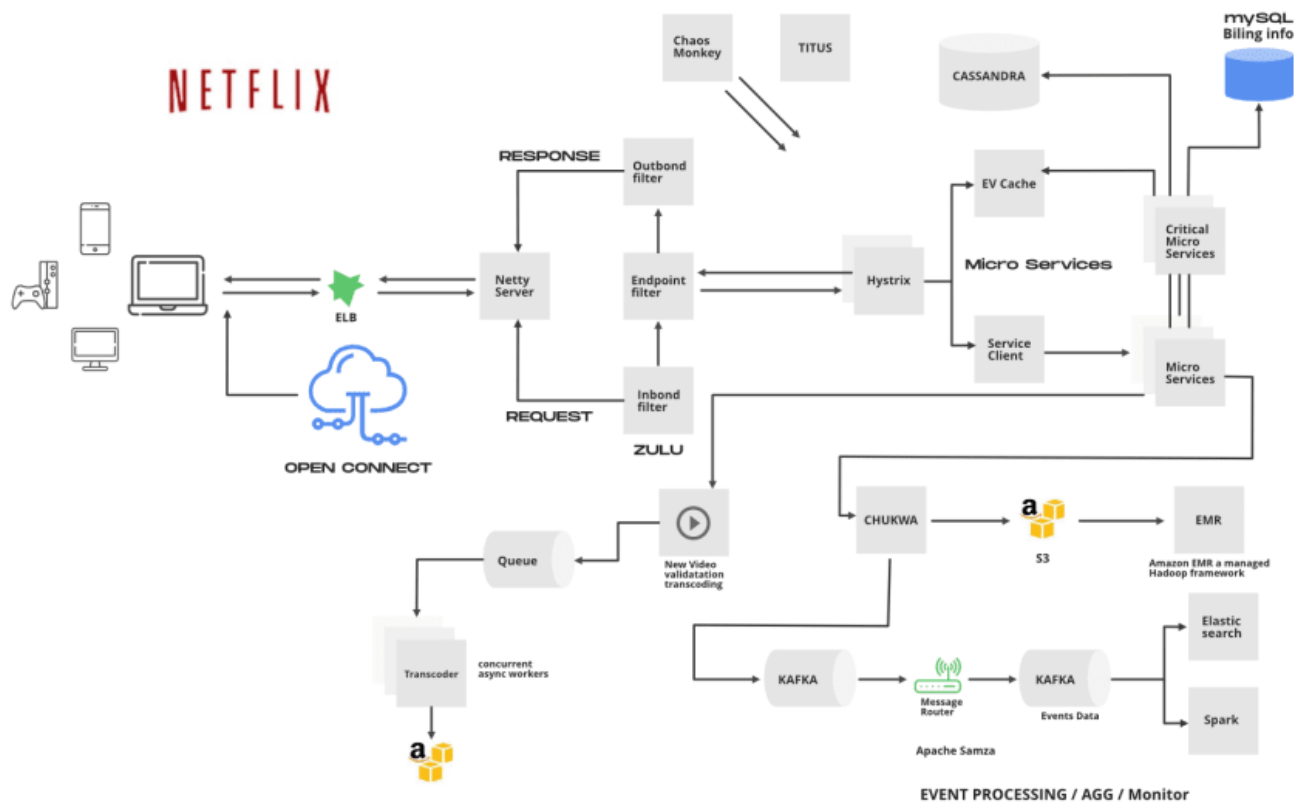
Cover Photo by [Alexander Shatov](#) on [Unsplash](#)

Netflix accounts for about 15% of the world's internet bandwidth traffic. Serving over 6 billion hours of content per month, globally, to nearly every country in the world. Building a robust, highly scalable, reliable, and efficient backend system is no small engineering feat but the ambitious team at Netflix has proven that problems exist to be solved.

This article provides an analysis of the Netflix system architecture as researched from online sources. Section 1 will provide a simplified overview of the Netflix system. Section 2 is an overview of the backend architecture and section 3 provides a detailed look at the individual system components.

.....

Netflix operates in two clouds Amazon Web Services and Open Connect(Netflix content delivery network).



The overall Netflix system consists of three main parts.

- **Open Connect** is Netflix's custom global content delivery network(CDN). These OCAs servers are placed inside internet service providers (ISPs) and internet exchange locations (IXPs) networks around the world to deliver Netflix content to users.
- **Client** - A client is any device from which you play the video from Netflix. This consists of all the applications that interface with the Netflix servers.

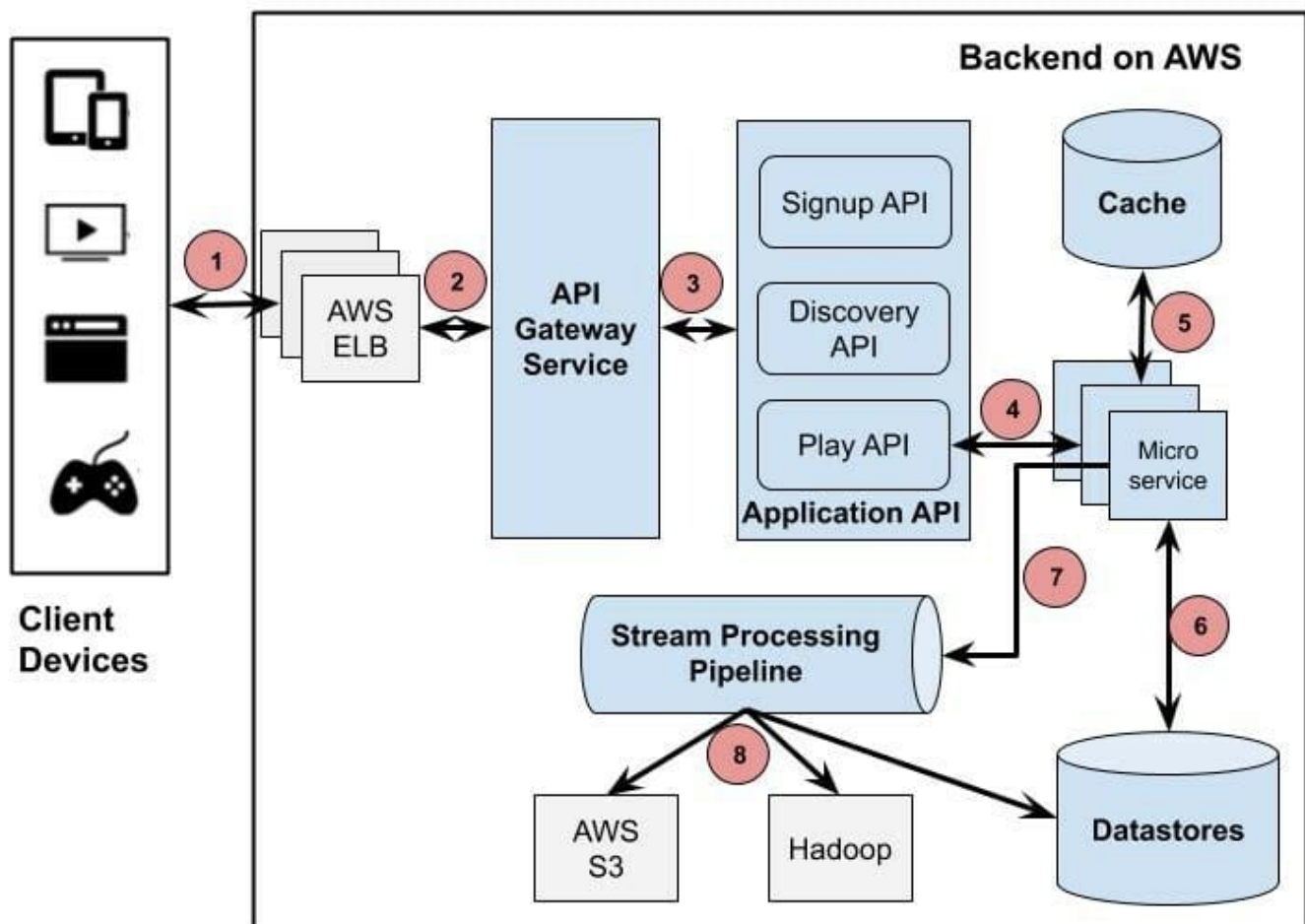
Netflix supports a lot of different devices including smart TV, Android, iOS

platforms, gaming consoles, etc. All these apps are written using platform-specific code. Netflix web app is written using reactJS which was influenced by several factors some of which include startup speed, runtime performance, and modularity.

- ..... - This includes databases, servers, logging frameworks, application monitoring, recommendation engine, background services, etc... When the user loads the Netflix app all requests are handled by the backend server in AWS Eg: Login, recommendations, the home page, users history, billing, customer support. Some of these backend services include (AWS EC2 instances, AWS S3, AWS DynamoDB, Cassandra, Hadoop, Kafka, etc).



Netflix is one of the major drivers of microservices architecture. Every component of their system is a collection of loosely coupled services collaborating. The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. The figure below is an overview of the backend architecture.



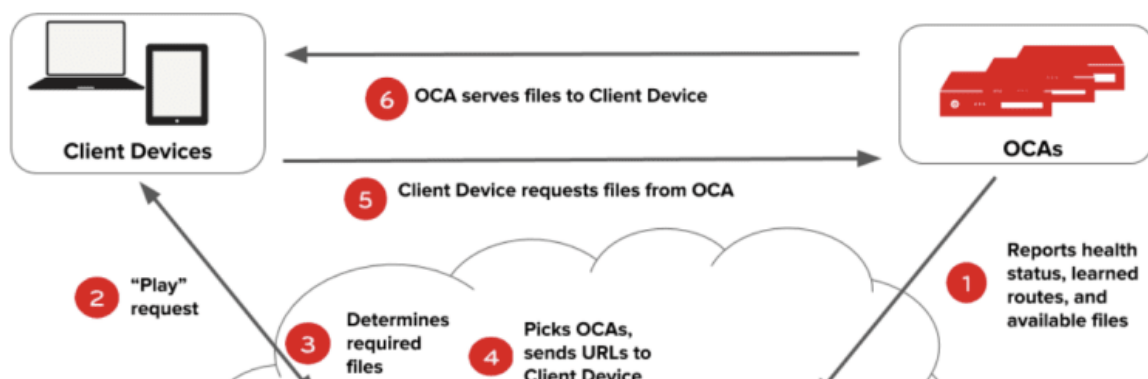
### Backend Architecture

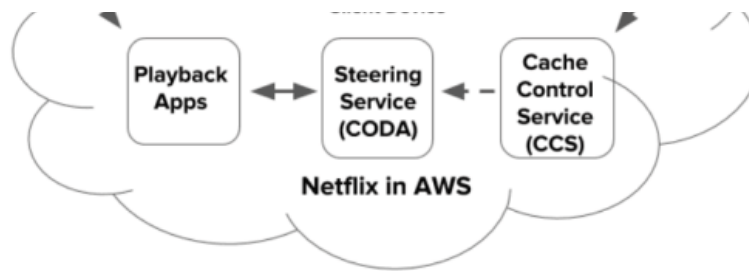
1. The Client sends a Play request to a Backend running on AWS. Netflix uses Amazon's Elastic Load Balancer (ELB) service to route traffic to its services.
2. AWS ELB will forward that request to the API Gateway Service. Netflix uses Zuul as its API gateway, which is built to allow dynamic routing, traffic monitoring, and security, resilience to failures at the edge of the cloud deployment.
3. Application API component is the core business logic behind Netflix operations. There are several types of API corresponding to different user activities such as Signup API, Discovery/Recommendation API for retrieving video recommendation. In this scenario, the forwarded request from API Gateway Service is handled by Play API.
4. Play API will call a microservice or a sequence of microservices to fulfill the request.
5. Microservices are mostly stateless small programs, there can be thousands of these services communicating with each other.
6. Microservices can save or get data from a data store during this process.
7. Microservices can send events for tracking user activities or other data to the Stream Processing Pipeline for either real-time processing of personalized recommendations or batch processing of business intelligence tasks.
8. The data coming out of the Stream Processing Pipeline can be persistent to other data stores such as AWS S3, Hadoop HDFS, Cassandra, etc.

.....

.....

Everything that happens after you hit play on a video is handled by Open Connect. This system is responsible for streaming video to your device. The following diagram illustrates how the playback process works.



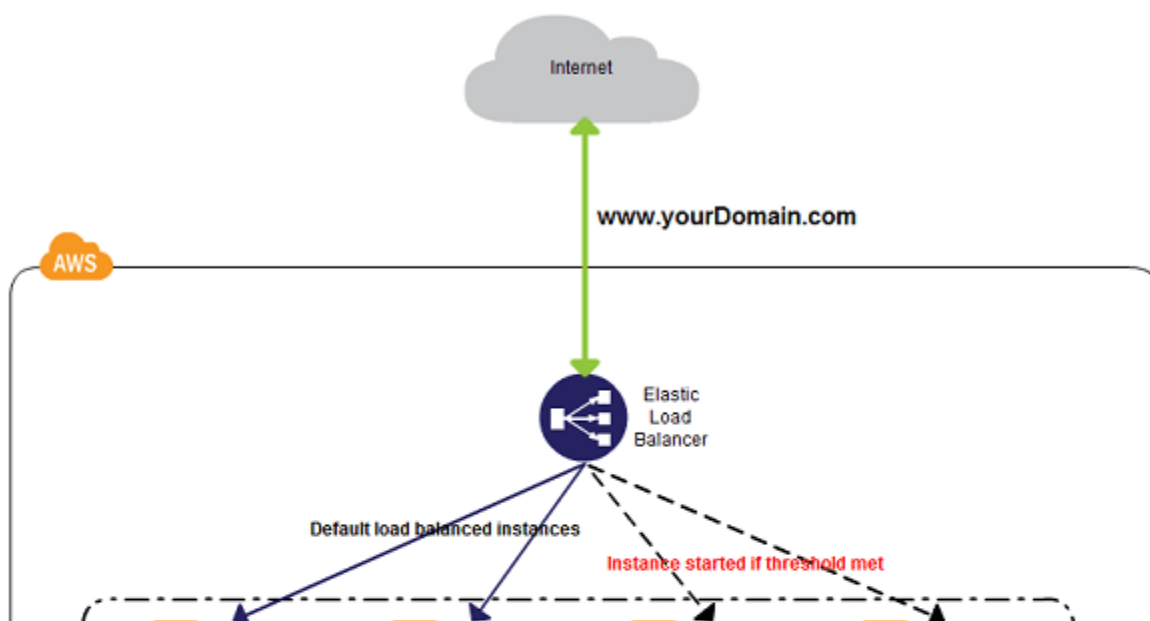


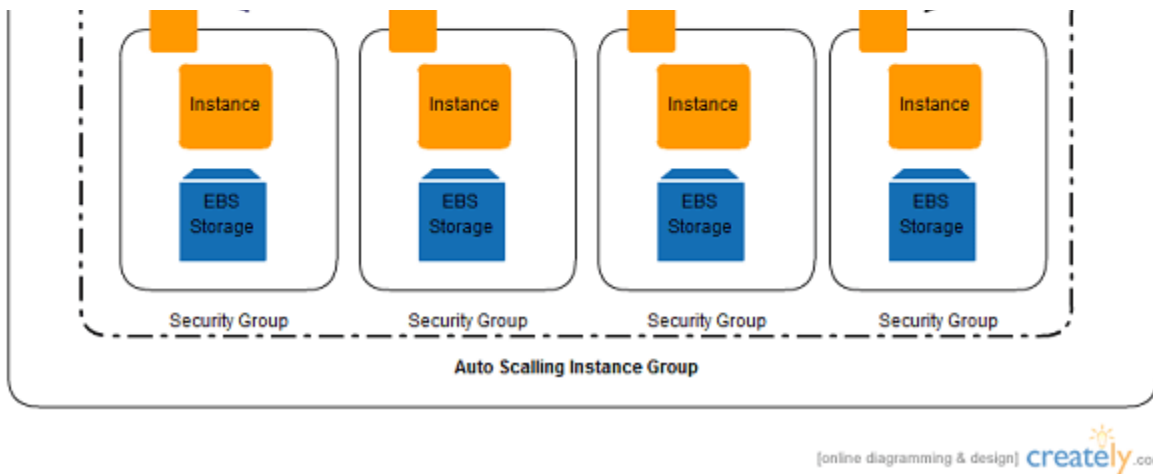
Open Connect Design Image

1. OCAs ping AWS instances to report their health, the routes they have learned, and the files they have on them.
2. A user on a client device requests playback of a title (TV show or movie) from the Netflix application in AWS.
3. The Netflix playback service checks for the user's authorization, permission, and licensing, then chooses which files to serve the client taking into account the current network speed and client resolution.
4. The steering service picks the OCA that the files should be served from, generates URLs for these OCAs, and hands it back to the playback service.
5. The playback service hands over the URLs of the OCA to the client, The client requests for the video files from that OCA.



Netflix uses Amazon's Elastic Load Balancer (ELB) service to route traffic to services. ELB's are set up such that load is balanced across zones first, then instances.





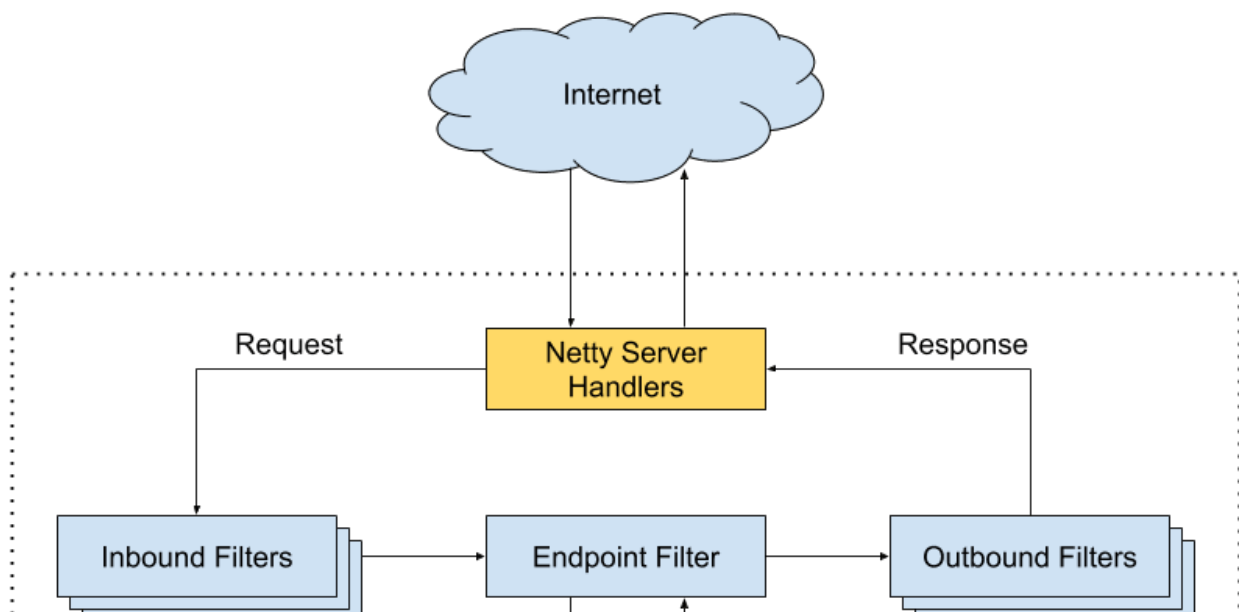
### Amazon Elastic Load Balancer

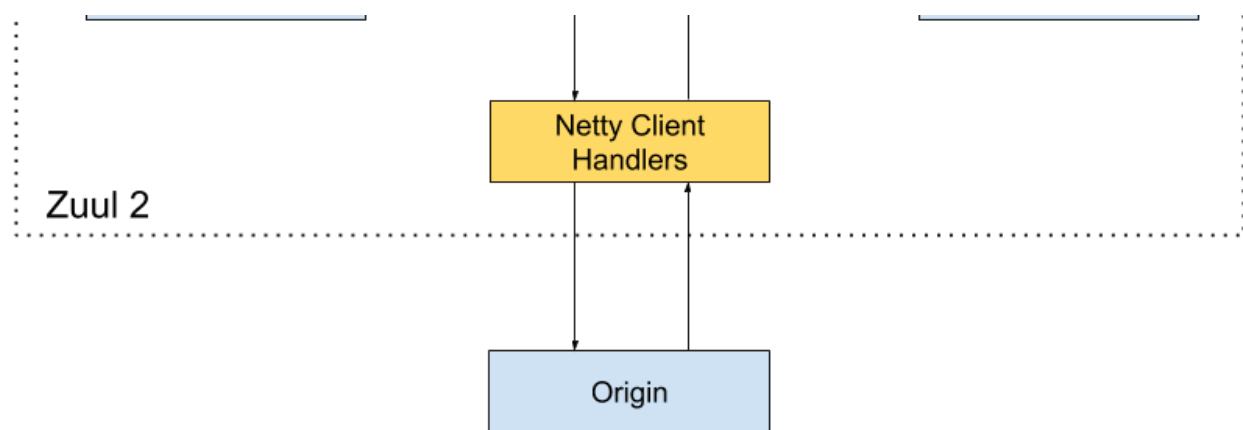
This load balancer routes request to the API gateway service; Netflix uses Zuul as its API gateway, It handles all the requests and performs the dynamic routing of microservice applications. It works as a front door for all the requests.

For Example, `/api/products` is mapped to the product service, and `/api/user` is mapped to the user service. The Zuul Server dynamically routes the requests to the respective backend applications. Zuul provides a range of different types of filters that allows them to quickly and nimbly apply functionality to the edge service.

The Cloud Gateway team at Netflix runs and operates more than 80 clusters of Zuul 2, sending traffic to about 100 (and growing) backend service clusters which amount to more than 1 million requests per second.

### [open-sourcing-zuul-2](#)








### Zuul Architecture

The Netty handlers on the front and back of the filters are mainly responsible for handling the network protocol, web server, connection management, and proxying work. With those inner workings abstracted away, the filters do all of the heavy liftings.

- The inbound filters run before proxying the request and can be used for authentication, routing, or decorating the request.
- The endpoint filters can either be used to return a static response or proxy the request to the backend service.
- The outbound filters run after a response has been returned and can be used for things like metrics, or adding/removing custom headers.

The Zuul 2 Api gateway forwards the request to the appropriate Application API.

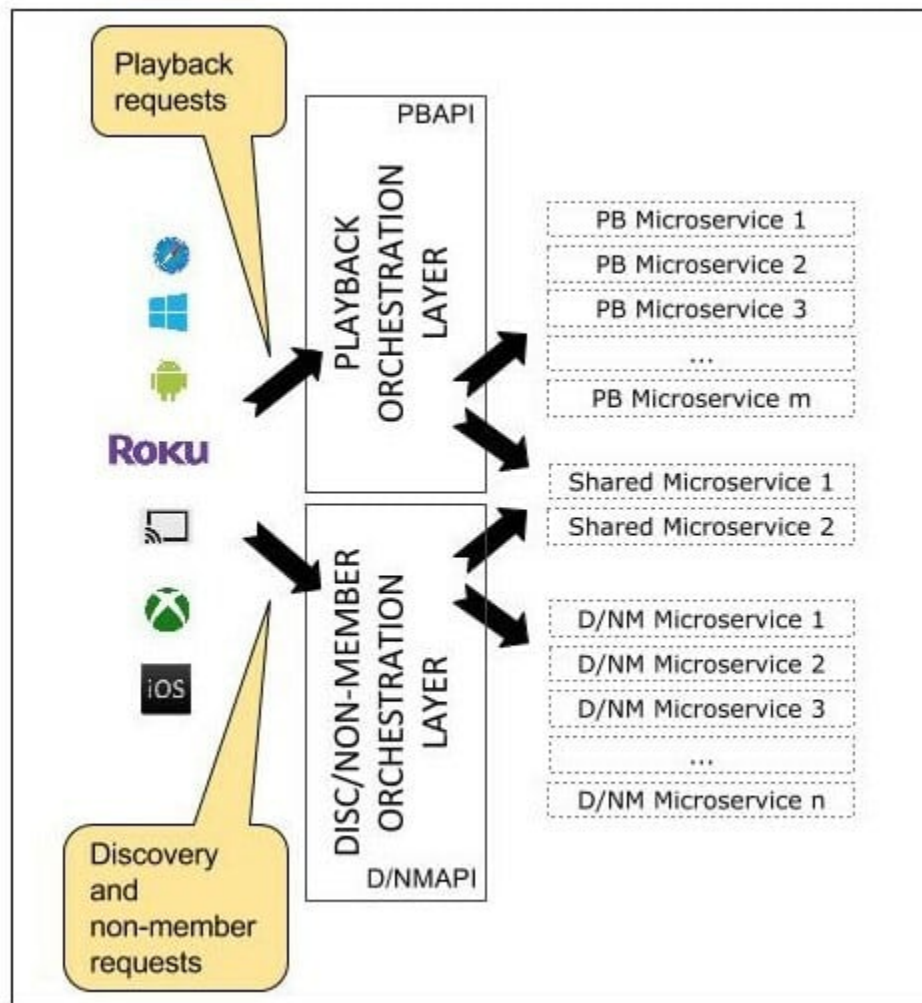


Currently, the Application APIs are defined under three categories:  -for non-member requests such as sign-up, billing, free trial, etc.,  -for search, recommendation requests, and  - for streaming, view licensing requests, etc. When a user clicks signup, for example, Zuul will route the request to the Signup API.

If you consider an example of an already subscribed user. Supposing the user clicks on play for the latest episode of peaky blinders, the request will be routed to the playback API. The API in turn calls several microservices under the hood. Some of these calls can be made in parallel because they don't depend on each other. Others have to be sequenced in a specific order. The API contains all the logic to sequence and parallelize the calls as necessary. The device, in turn, doesn't need



sequence and parallelize the calls as necessary. The device, in turn, doesn't need to know anything about the orchestration that goes on under the hood when the customer clicks "play".

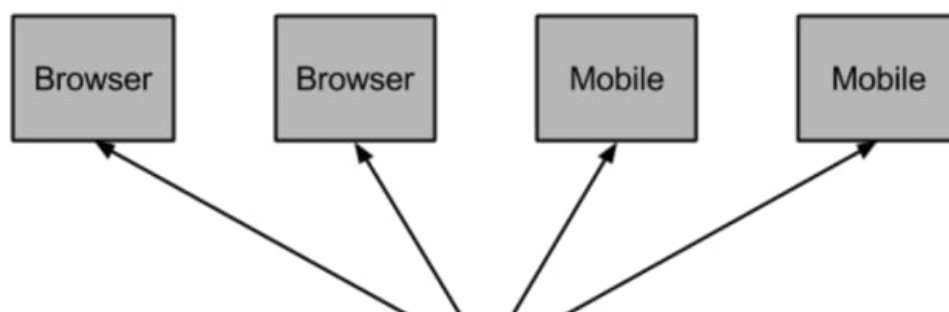


Netflix API Architecture

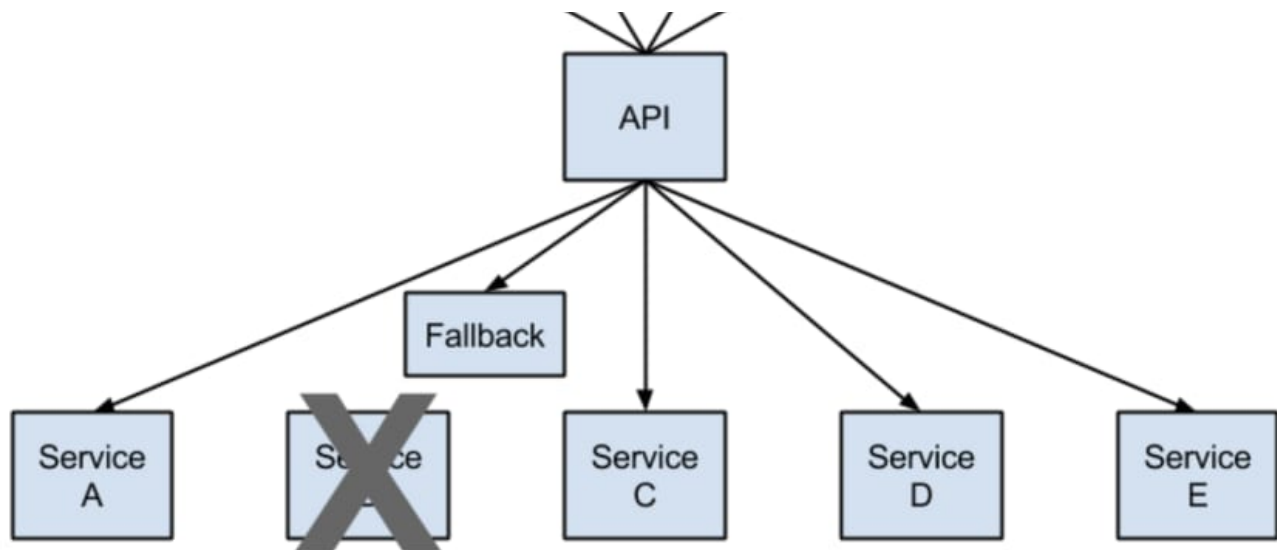
Signup requests map to signup backend services, Playback requests, with some exceptions, map only to playback backend services, and similarly discovery APIs map to discovery services.



## [Hystrix](#)







Hystrix Architecture

In any distributed environment (with a lot of dependencies), inevitably some of the many service dependencies fail. It can be unmanageable to monitor the health and state of all the services as more and more services will be stood up and some services may be taken down or simply break down. Hystrix comes with help by providing a user-friendly dashboard. .... is used to control the interaction between these distributed services by adding some latency tolerance and fault tolerance logic.

Consider this example from Netflix, they have a microservice that provides a tailored list of movies back to the user. If the service fails, they reroute the traffic to circumvent the failure to another vanilla microservice that simply returns the top 10 movies that are family-friendly. So they have this safe failover that they can go to and that is the classic example of first circuit breaking.

..... :

*Netflix Hystrix is no longer in active development and is currently in maintenance mode. Some internal projects are currently being built with resilience4j*

<https://github.com/resilience4j/resilience4j>

..... :

## Titus

Titus is a container management platform that provides scalable and reliable container execution and cloud-native integration with Amazon AWS.