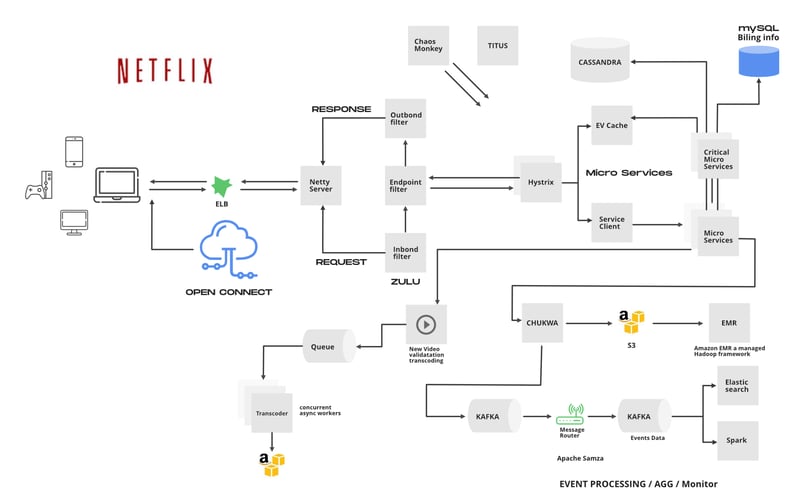
# **Netflix System Design- Backend Architecture**

Below outlines the System design of Netflix:

**Netflix** accounts for about 15% of the world's internet bandwidth traffic, serving over 6 billion hours of content per month 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 analyzes the Netflix system architecture as researched from online sources. Section 1 provides a simplified overview of the Netflix system. Section 2 provides an overview of the backend architecture, and section 3 provides a detailed look at the individual system components.

**1. Overview**

Netflix operates in two clouds Amazon Web Services and Open Connect(Netflix content delivery network).  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F4rq466utjqzmkvt5jjrz.png)

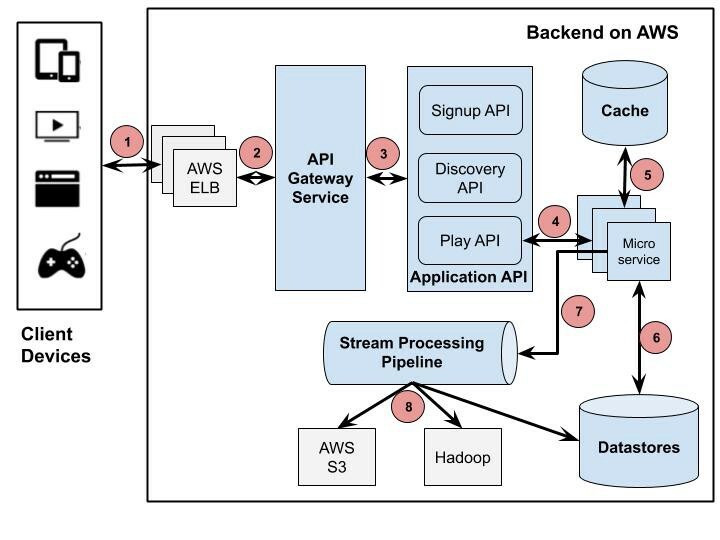
The overall Netflix system consists of three main parts.

* **Open Connect Appliances(OCA)** - Open Connect is Netflix’s custom global content delivery network(CDN). These OCA 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 Netflix videos. This consists of all the applications that interface with the Netflix servers.

Netflix supports many different devices, including smart TVs, Android and iOS platforms, gaming consoles, etc. All these apps are written using platform-specific code. The Netflix web app is written using reactJS, which was influenced by several factors, some of which include startup speed, runtime performance, and modularity.

* **Backend** - 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 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).

**2. Backend Architecture**

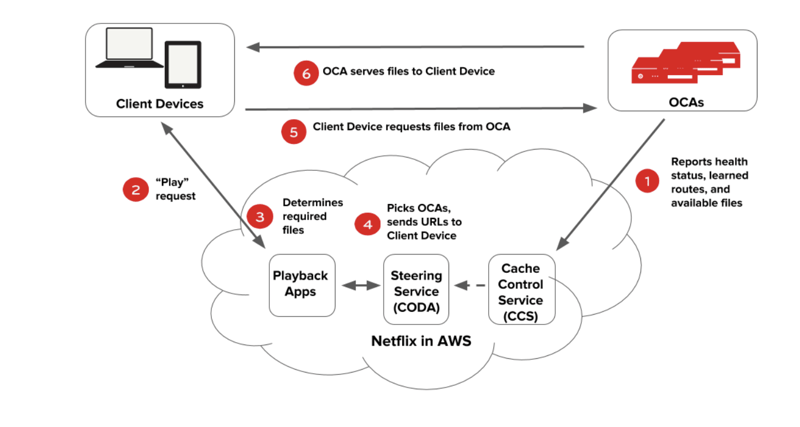
Netflix is one of the major drivers of microservices architecture. Every component of their system is a collection of loosely coupled services that collaborate. The microservice architecture enables the rapid, frequent, and reliable delivery of large, complex applications. The figure below is an overview of the backend architecture.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fuzh1rz679blx2rlhm9da.jpeg)

Backend Architecture

1. The Client sends a Play request to a Backend running on AWS. Netflix routes traffic to its services using Amazon's Elastic Load Balancer (ELB) service.
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, security, and resilience to failures at the edge of cloud deployment.
3. The application API component is the core business logic behind Netflix's operations. Several types of API correspond to different user activities, such as the Signup API and the Discovery/Recommendation API for retrieving video recommendations. In this scenario, the forwarded request from the API Gateway Service is handled by the 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. Thousands of these services can communicate with each other.
6. Microservices can save or get data from a data store during this process.
7. Microservices can send events to track 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 Stream Processing Pipeline data can be persistented to other data stores such as AWS S3, Hadoop HDFS, Cassandra, etc.

**3. Backend Components**

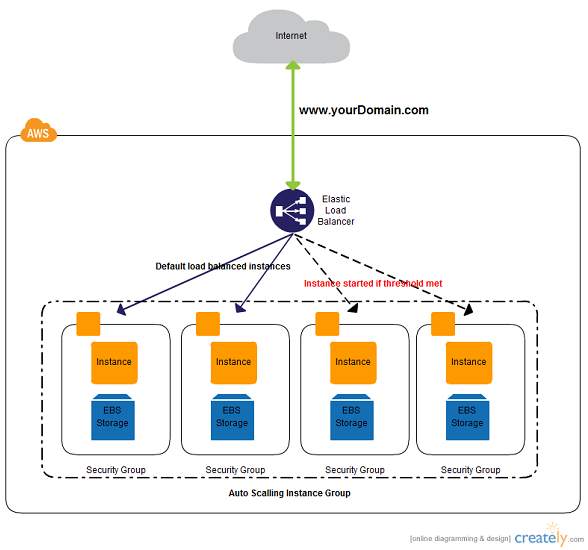
**Open Connect**

Open Connect handles everything that happens after you hit play on a video. This system is responsible for streaming video to your device. The following diagram illustrates how the playback process works.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fhdg8io3upt0ioogjvz6v.png)

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 from which the files should be served, generates URLs for these OCAs, and hands them back to the playback service.
5. The playback service hands over the URLs of the OCA to the client, and the client requests the video files from that OCA.

**Zuul2-API GATEWAY**

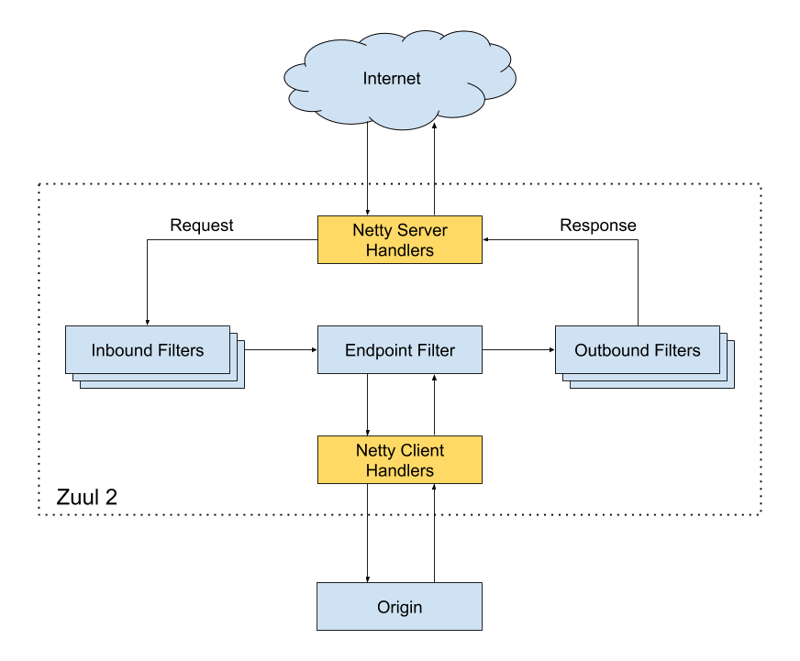
Netflix uses Amazon's Elastic Load Balancer (ELB) service to route traffic to services. ELBs are set up such that the load is balanced across zones first, then instances.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fvb4u8c24qg9ftca8qgp9.png)

Amazon Elastic Load Balancer

This load balancer routes requests 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 allow 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](https://netflixtechblog.com/open-sourcing-zuul-2-82ea476cb2b3)  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fqkjl4fb94l51dx0qsbj3.png)

Zuul Architecture

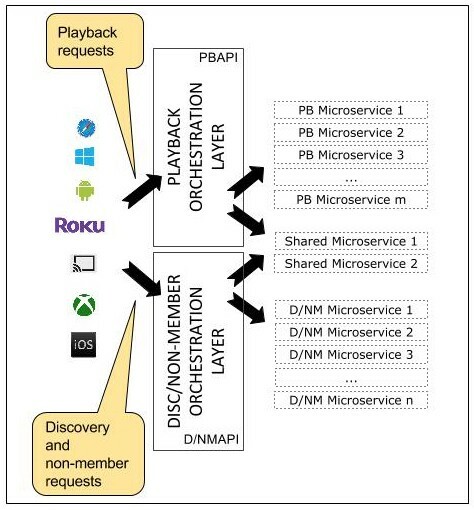
The Netty handlers on the front and back of the filters are mainly responsible for handling the network protocol, web server, connection management, and proxying work. With those inner workings abstracted away, the filters do all the heavy lifting.

* The inbound filters run before proxying the request and can be used for authentication, routing, or decorating the request.
* The endpoint filters can either 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 to add or remove custom headers or metrics.

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

**Application API**

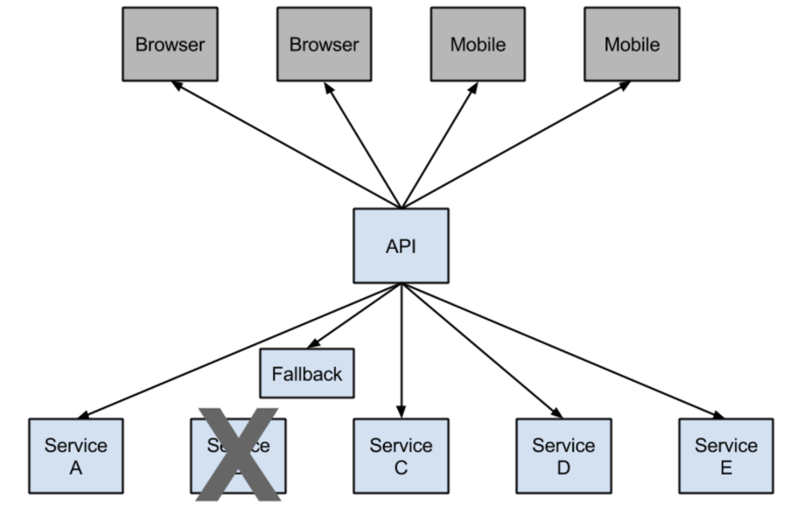
Currently, the Application APIs are defined under three categories: **Signup API** -for non-member requests such as sign-up, billing, free trial, etc., **Discovery API**-for search, recommendation requests, and **Play API**- for streaming, view licensing requests, etc. When a user clicks signup, for 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 to know anything about the orchestration that goes on under the hood when the customer clicks “play.”  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fk298u17sizjb8ib7xg65.jpeg)

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- Distributed API Services Management**

[Hystrix](https://netflix.github.io/titus/)  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F6ry3q3t2vizu7o55ixg3.png)

Hystrix Architecture

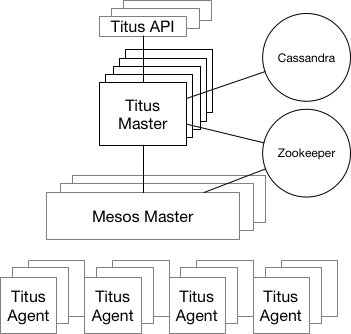
In any distributed environment (with a lot of dependencies), inevitably, some of the many service dependencies will fail. It can be unmanageable to monitor the health and state of all the services as more and more services will be stood up, and some services may be taken down or simply broken down. Hystrix comes with help by providing a user-friendly dashboard. **Hystrix library** 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 family-friendly movies. So they have this safe failover that they can go to, and that is the classic example of the first circuit breaking.

**Note**:

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

**Titus- Container Management**

[Titus](https://netflix.github.io/titus/)  
Titus is a container management platform that provides scalable and reliable container execution and cloud-native integration with Amazon AWS.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F2k22xkjv5ed0vbc69oay.png)

Titus Architecture

It is a framework on top of Apache Mesos, a cluster management system that brokers available resources across a fleet of machines.  
Titus is run in production at Netflix, managing thousands of AWS EC2 instances and launching hundreds of thousands of containers daily for both batch and service workloads. Just think of it as the Netflix version of Kubernetes.

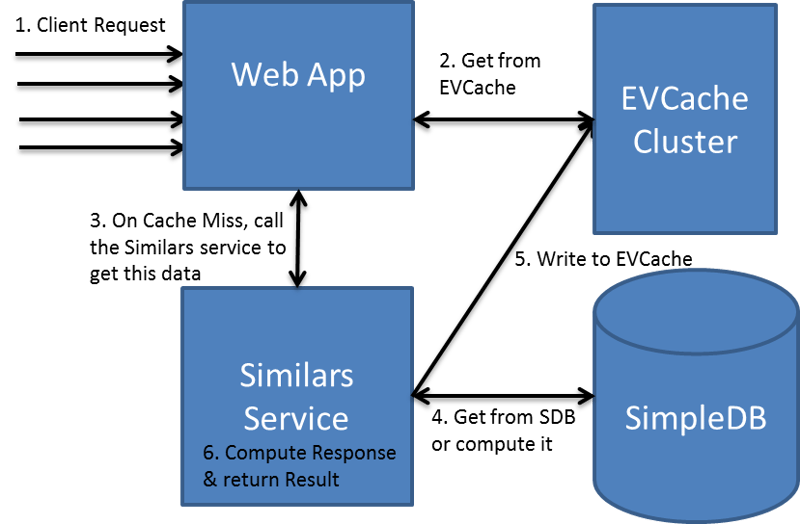
*Titus runs about 3 million containers per week.*

**Datastores**

**EVCache**

A cache's primary purpose is to increase data retrieval performance by reducing the need to access the underlying slower storage layer. Trading off capacity for speed, a cache typically stores a subset of data transiently.  
[EVCache](https://github.com/Netflix/EVCache)

Two use cases for caching is to:

* Provides fast access to frequently stored data.
* Provides fast access to computed(memoized) data. Netflix's microservices rely on caches for fast, reliable access to multiple types of data like a member’s viewing history, ratings, and personalized recommendations.EVCache Diagram

EVCache is a Memcached and spymemcached-based caching solution mainly used for caching frequently used data on AWS EC2 infrastructure.  
EVCache is an abbreviation for:

* Ephemeral - The data stored is for a short duration as specified by its TTL (Time To Live).
* Volatile - The data can disappear at any time (Evicted).
* Cache - An in-memory key-value store.

**SSDs for Caching**

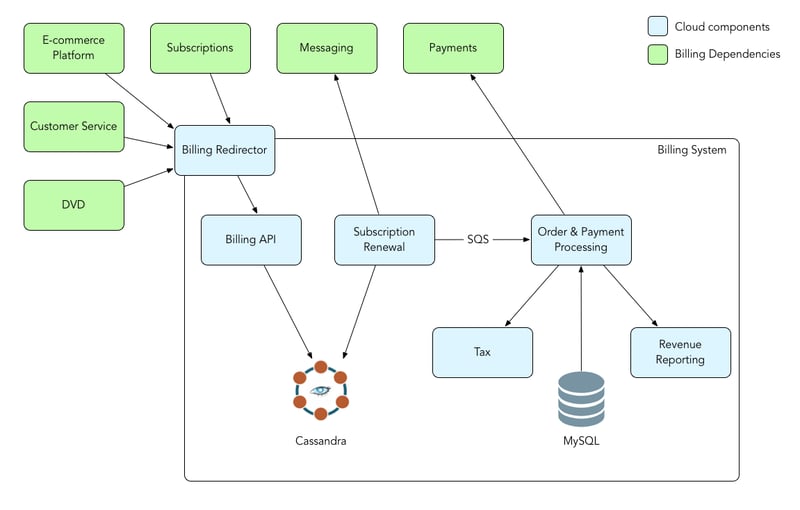
Traditionally, caching is done on RAM. Storing large amounts of data on RAM is expensive, so Netflix decided to move some caching data to SSD.

Modern disk technologies based on SSD are providing fast access to data but at a much lower cost when compared to RAM. The cost to store 1 TB of data on SSD is much lower than storing the same amount using RAM.

[Evolution of application Data Caching](https://netflixtechblog.com/evolution-of-application-data-caching-from-ram-to-ssd-a33d6fa7a690)

**MySQL**

Netflix uses AWS EC2 instances of MYSQL for its Billing infrastructure. Billing infrastructure is responsible for managing Netflix members' billing states. This includes keeping track of open/paid billing periods, the amount of credit on the member’s account, managing the member's payment status, initiating charge requests, and what date the member has paid through.

The payment processor needed the ACID capabilities of an RDBMS to process charge transactions.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F2cdd1qpqbl3ckw6ebyn4.png)

Netflix Datastore

**Apache Cassandra**

Cassandra is a free and open-source distributed wide-column store. The NoSQL database is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

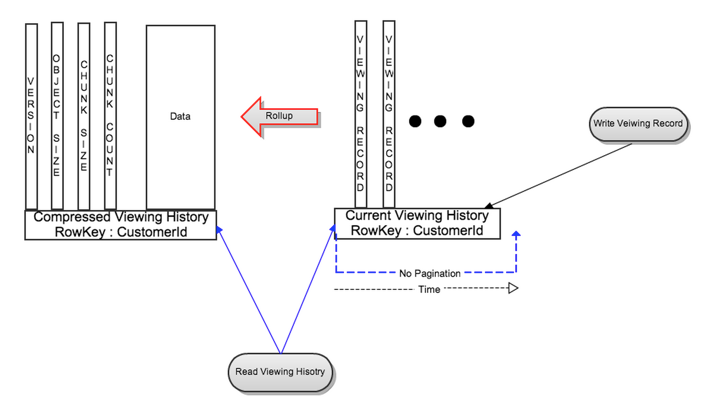
Netflix uses Cassandra for its scalability, lack of single points of failure, and cross-regional deployments. ” In effect, a single global Cassandra cluster can simultaneously service applications and asynchronously replicate data across multiple geographic locations.

Netflix stores all kinds of data across its Cassandra DB instances, including all user-collected event metrics.

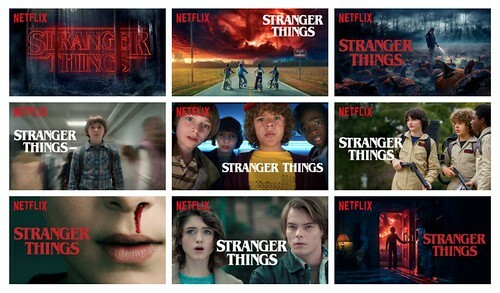
As user data began to increase, there needed to be a more efficient way to manage data storage. Netflix Redesigned data storage architecture with two main goals in mind:

* Smaller Storage Footprint.
* Consistent Read/Write Performance as viewing per member grows.

The solution to the large data problem was to compress the old rows. Data were divided into two types:

* Live Viewing History (LiveVH): Small number of recent viewing records with frequent updates. The data is stored in uncompressed form.
* Compressed Viewing History (CompressedVH): A large number of older viewing records with rare updates. The data is compressed to reduce the storage footprint. Compressed viewing history is stored in a single column per row key.Compressed Viewing History

**Stream Processing Pipeline**

Did you know that Netflix personalizes movie artwork just for you? You might be surprised to learn the image shown for each video is selected specifically for you. Not everyone sees the same image.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F6zbpyp23tozgvhsng18z.jpeg)  
Netflix tries to select artwork that highlights the most relevant aspect of a video based on the data it has learned about you, such as your viewing history and interests.

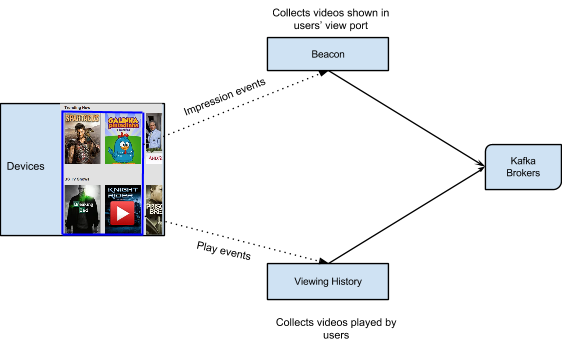
Stream Processing Data Pipeline has become Netflix’s data backbone of business analytics and personalized recommendation tasks. It is responsible for producing, collecting, processing, aggregating, and moving all microservice events to other data processors in near real-time.

Streaming data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, e-commerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers.

[AWS- What is streaming Data?](https://aws.amazon.com/streaming-data/)

This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows and used for a wide variety of analytics, including correlations, aggregations, filtering, and sampling.

Information derived from such analysis gives companies visibility into many aspects of their business and customer activity, such as service usage (for metering/billing), server activity, website clicks, and the geo-location of devices, people, and physical goods, and enables them to respond promptly to emerging situations. For example, businesses can track changes in public sentiment on their brands and products by continuously analyzing social media streams and responding promptly as necessary.

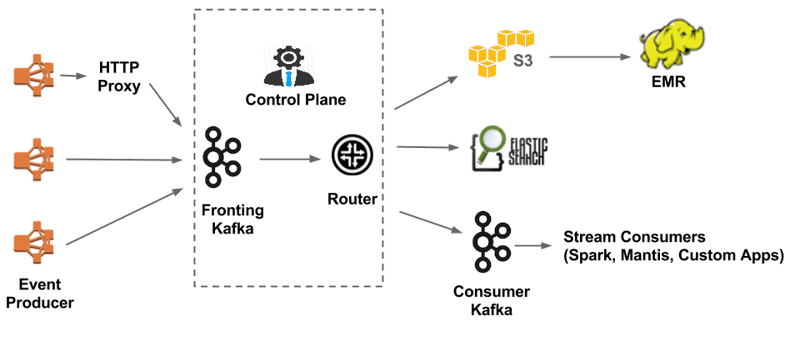
The stream processing platform processes trillions of events and petabytes of data per day.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fyfv8wddthnk9m37uf628.png)

The Viewing History Service captures all the videos played by members. Beacon is another service that captures all impression events and user activities within Netflix. All the data collected by the Viewing History and Beacon services is sent to Kafka.

**Apache Kafka- Analyzing Streaming Data**

Kafka is open-source software that provides a framework for storing, reading, and analyzing streaming data.

Netflix embraces Apache Kafka® as the de-facto standard for its eventing, messaging, and stream processing needs. Kafka acts as a bridge for all point-to-point and Netflix Studio-wide communications.

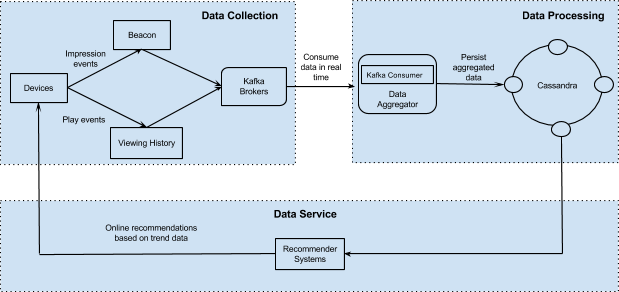
[How Netflix uses Kafka](https://www.confluent.io/blog/how-kafka-is-used-by-netflix/)  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F78gmjgardflv97rrwxyj.png)

**Apache Chukwe- Analyzing Streaming Data**

Apache Chukwe is an open-source data collection system that collects logs or events from a distributed system. It is built on top of HDFS and Map-reduce framework. It comes with Hadoop’s scalability and robustness features. It includes a lot of powerful and flexible toolkits to display, monitor, and analyze data. Chukwe collects the events from different parts of the system; From Chukwe, you can do monitoring and analysis, or you can use the dashboard to view the events. Chukwe writes the event in the Hadoop file sequence format (S3).

**Apache Spark - Analyzing Streaming Data**

Netflix uses Apache Spark and Machine learning to recommend movies. Apache Spark is an open-source unified analytics engine for large-scale data processing.

On a live user request, the aggregated play popularity(how many times a video is played) and take rate(Fraction of play events over impression events for a given video) data, along with other explicit signals such as members’ viewing history and past ratings, are used to compute personalized content for the user. The following figure shows the end-to-end infrastructure for building user movie recommendations.  
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F3335yl4ojj3h2fyn95hx.png)

Data Processing Engine

**Elastic Search - Error Logging and Monitoring**

Netflix uses elastic search for data visualization, customer support, and error detection in the system.

Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

With elastic search, they can easily monitor the state of the system and troubleshoot error logs and failures.