CS5033

Practical 1 - Architecture Analysis

190004947

February 21, 2022

University of
St Andrews

Word Count: 2,270

---

1 Abstract

In today's world, video streaming makes up around 80% [1] of global internet traffic and applying advanced software architecture theory is crucial in solving the problem of providing reliable and available solutions at scale. An understanding of important architectural characteristics for video streaming applications is required to identify the most appropriate software architectures. A case study on Netflix, one of the most dominant streaming platforms, can illustrate the requirements of a video streaming architecture and how these requirements can be fulfilled. The goal of this report is to assist in understanding the complexities of a software architecture in the domain of video streaming.

2 Overview of Video Streaming

The domain of video streaming refers to the family of applications that serve users video content through a continuous loop of downloading and playing small chunks of the video file. There are many opportunities for clever software architecture in this domain as video streaming can be optimised in different ways, for example, faster data delivery using a content delivery network (CDN) set up.

Challenges such as diverse environment support and scalability have emerged within the video streaming software domain. Environment support refers to an application's ability to function on different devices with varying levels of computing power and bandwidth capacity. In the context of video streaming, this is realised as being able to serve video at multiple quality levels and in different file types should a client device only accept a particular video format. The challenge of scaling a video streaming platform comes from streaming being a resource intensive activity which requires complex load balancing, efficient data transfer protocols, and resilience against constantly changing environmental conditions in order to run with significant network traffic.

A small video streaming application could use a client/server architecture where the user makes a request to the server for a video and the server responds periodically with small chunks of the video and audio that are temporarily saved and played. This would only be appropriate when anticipated peak traffic is low as the simple architecture would collapse very quickly under stress. For a video streaming platform that is intended to withstand significant amounts of traffic, it is important to be scalable while maintaining high availability and reliability measures. Since microservice architectures are independently scalable, as traffic increases, instead of scaling the entire application, only the most critical microservices need to be scaled. This makes scaling faster and more cost efficient.

Microservice architectures also reduce downtime through fault isolation. If a single service fails, the failure can be isolated to avoid cascading failures that could lead to system outages. Each service is independently deployable which allows dedicated development teams to be assigned to individual services providing the opportunity for autonomy. It is much easier for small development teams to each manage a small codebase representing one service than for a large development team to manage a large, convoluted codebase.

3 Overview of Netflix

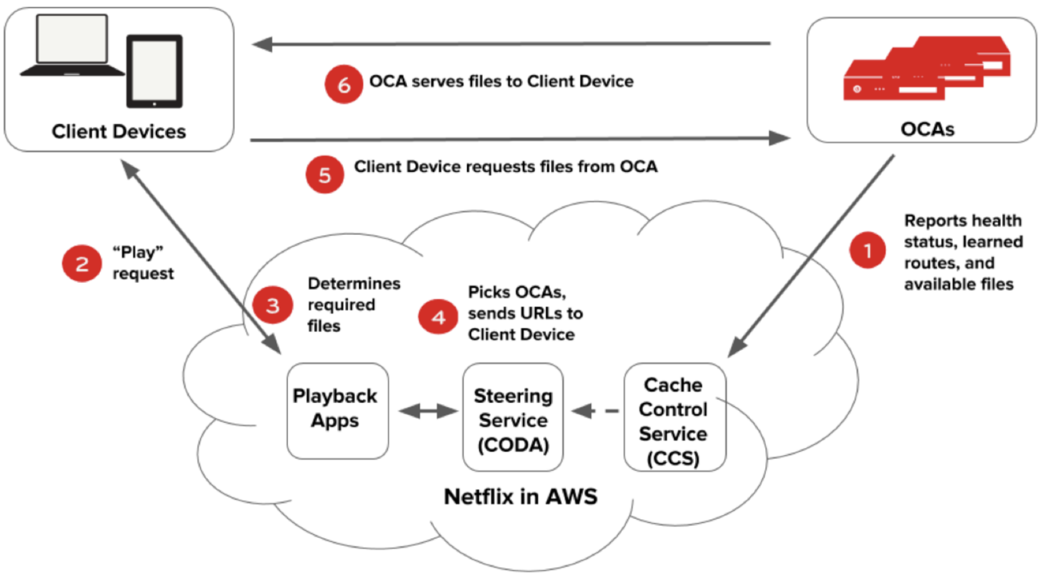
Netflix is the second largest video streaming platform by monthly average users in the world after YouTube [2]. It is a subscription-based service that allows members to watch movies or TV shows without adverts through distribution deals as well as its own productions, called Netflix Originals. The application provides low latency video and audio streams to a global consumer base at very high availability. They also need to support basic login and registration with payment processing for subscriptions. The platform must be able to show available titles in a meaningful order (based on some recommendation algorithm).

4 Interesting Example

As one of the most prominent video streaming platforms, the development team at Netflix have had to pioneer new solutions to scaling and resource management in order to handle the massive amount of traffic that their platform attracts. Netflix has proven how powerful a properly managed microservices architecture can be and what distributed systems can achieve. The media powerhouse makes for an interesting example of software architecture as they, through the use of specialised techniques, maintain a robust and reliable platform [3] while sustaining 34% of streaming time in the US.

Netflix uses a distributed cloud-based architecture organised with astounding precision and automation to achieve high levels of abstraction allowing them to focus resources on business development rather than large technical undertaking. Netflix was one of the leading institutions in utilising cloud technology's ability to provide agility and scale for unpredictable business growth. In 2011, platform growth was accelerating and unpredictable and Netflix needed to be ready for the traffic [4]. They decided to leverage the AWS cloud for their rich feature set, unique scaling opportunity, and high availability.

The main reason I have chosen to highlight Netflix as an interesting example of video streaming architecture is because of their Open Connect Content Delivery Network (CDN). It consists of an array of servers called Open Connect Appliances (OCAs) that are purpose-built to store encoded video/image files and serve these files via HTTP/HTTPS to client devices. The OCAs are installed within internet exchange points (IXPs) or deployed directly inside internet service provider (ISP) networks. OCAs do not store client data; their only purpose is to report their status to the Open Connect control plane services in AWS and to serve content when it is requested by a client. The control plane collects the reported data from OCAs and calculates the most optimal OCAs given their file availability, health, and network proximity to the client. This data is then used to connect the client to the most appropriate OCA which is optimised dynamically to support the specific needs of the client device under current network conditions.


Client Devices
6 OCA serves files to Client Device
OCAs
5 Client Device requests files from OCA
1 Reports health status, learned routes, and available files
2 "Play" request
3 Determines required files
4 Picks OCAs, sends URLs to Client Device
Playback Apps
Steering Service (CODA)
Cache Control Service (CCS)
Netflix in AWS

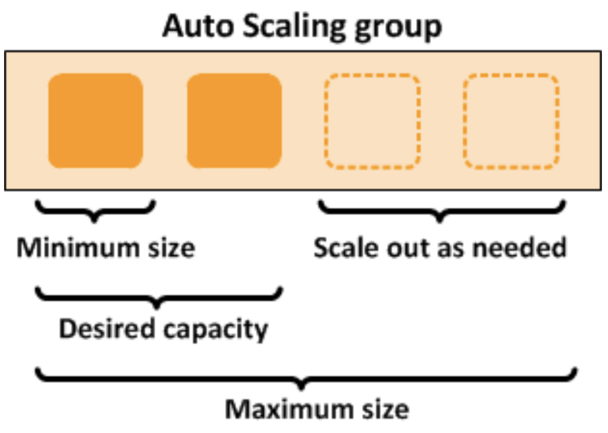
5 Key Architectural Characteristics

A key architectural characteristic that is important to Netflix and to video streaming is scalability. Scalability refers to a system's ability to adapt under load to continue providing a service. Demand is inevitably growing for streaming services, including Netflix, and these systems must be able to grow with the market. Events such as traffic spikes threaten an application's ability to operate at all, let alone with acceptable performance. For streaming, acceptable performance refers to low latency, high video quality, and little to no playback interruption. Latency is an especially important property of these systems as video streaming is very bandwidth intensive and small network delays can cause buffering and poor video quality. For these reasons, it is absolutely critical for software architectures in the video streaming domain to support scaling.

Another key architectural characteristic for video streaming platforms is reliability and availability. In such a competitive market, Netflix has to maintain extremely low measures of mean time to failure and probability of failure on demand in order to preserve user retention and system confidence. Implementing fault tolerance mechanisms is essential for preserving reliability and availability as cascading failures can lead to system failures and downtime. Particularly in platforms employing a paid subscription model, the system must remain functional as consistently as possible to allow customers access to the content they are paying for.

6 Netflix Architecture for Scalability and Reliability

By maintaining a distributed system using AWS cloud, Netflix is able to use AWS Auto Scaling in which AWS EC2 will scale the number of active elastic instances based on a specified criteria to account for traffic fluctuations. This increases compute efficiency, allows for failed nodes to be replaced easily, and creates a buffer for controlling traffic spikes or DDOS attacks. This is an example of dynamic horizontal scaling; Netflix is able to adapt to meet changing demands.



The Open Connect CDN is Netflix's global content delivery network that collaborates with internet service providers (ISPs) and internet exchange points (IXPs) to bring video content physically closer to the client for reduced stream latency. Netflix localise their network traffic, limiting the geographical distances that the video bits must travel during playback. On top of this, OCAs actively manage the portion of the Netflix catalogue that they store by executing 'fill' each night. During the nightly updates, the Open Connect Appliances each download routine software enhancements, if available, and new encoded video files in accordance to the selection criteria determined by the fill pattern. When an OCA downloads a movie or TV show via a regular cache fill, all other OCAs within the same cluster or on the same subnet will attempt to download the video content from the initial OCA through a 'peer fill' as long as routes to the OCAs are advertised over Open Connect peering.

Netflix uses Apache Cassandra which is an open source NoSQL distributed database which is highly scalable and available. Cassandra has no single point of failure and operates an 'eventually consistent' protocol which means that if a service tries to write to multiple databases and one fails, instead of them all failing, it will write to the available databases try to correct the failed one at a later time. Here we see the result of the CAP theorem; in the presence of a network partition, you must choose between consistency and availability. Netflix values availability over having data consistency and so allows Cassandra to hold the stale values.

Fundamental to Netflix's microservices architecture, each microservice utilises EVCache. The EVCache will store recent values returned from the microservice so that successive identical API calls do not need to be recomputed. EVCache is different to regular cache in that write operations write data into local zone cache as well as foreign zone cache for redundancy. Cross-region replication is also used to maintain duplicate caches and invalidate stale cache entries in other region's cache. When set or delete operations are performed on the EVCache in one region, the metadata of the operations are sent to one or more other regions where a 'replication writer' will perform the same operations on their local cache. Local read operations can fallback and read a value from another availability zone rather than returning an error, protecting the system from failure.

Netflix has to anticipate and be prepared for inevitable infrastructure failures. On Christmas Eve of 2012 when the AWS elastic load balancing failed. Netflix had all their traffic running through AWS' US-East-1 region which, upon failure, took the platform offline [5]. Afterwards, Netflix developed a multi-region strategy multiple AWS regions such that if any failed completely, the traffic could be pushed over to the surviving regions which will we scaled by the AWS Auto Scaler to account for the additional load. This mechanism would allow (and has allowed) Netflix to survive total failure of one availability zone with zero downtime [6].

The risk of network latency, congestion, failure, and logical or scaling failure which is introduced when calling a service from another service threatens the availability of applications with microservices architectures. Malformed or unmanaged service dependencies can lead to cascading failure in which a service failure propagates throughout the system potentially leading to a terminal failure. However, Netflix's architecture includes Hystrix, a latency and fault tolerance library written by the Netflix team, which prevents cascading failure by isolating services from their caller in order to catch failures and execute its fallback logic. The fallback protocol can return some static response to allow a customer to continue using Netflix during a service failure. Hystrix is used to decide how long it would like to wait for service responses before it gets the stale data from the cache.

7 Architectural Style Choices

The Open Connect CDN is an administrated peer-to-peer network. Each node connects to all other nodes in the same cluster or on the same subnet that have their routes advertised over Open Connect peering. Each peer only knows about a few others and it is extremely easy to accommodate more peers. Critically, from this style emerges the property of no single point of failure; a failed Open Connect Appliance will not impact any other component of the system and performance change will be negligible.

When a client clicks play on Netflix, a play request is sent to the backend running on AWS which is handled by the AWS Elastic Load Balancer (ELB). The load balancer will then forward the request to an API Gateway Service running on AWS EC2 instances. That API Gateway is called Zuul and was purpose-built by Netflix to manage traffic routing for various purposes such as load testing and routing to different service endpoints under huge workloads. Zuul exhibits a pipe-and-filter style; tasks are split into different filters, and each filter is responsible for carrying out some sort of processing. More specifically, the inbound filters run before proxying a 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, and the outbound filters run after a response has been returned so can be used for operations such as adding or removing custom headers.

8 Critical Analysis

As the many microservices that Netflix employs evolve, their interfaces will evolve too. Communicating updates throughout the lifecycles of these services is critical in avoiding interface clashes. If a particular service pushes an update to production which changes the interface endpoints in a non-backwards-compatible manner, the microservice will be lost. However, due to Netflix's implementation of Hystrix, this service failure will remain isolated and will not propagate throughout the system.

References

[1] *Global 2020 Forecast Highlights.* URL: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2020_Forecast_Highlights.pdf.

[2] Julia Stoll. *U.S. leading streaming video platforms by Monthly average users 2019.* Jan. 2022. URL: https://www.statista.com/statistics/910875/us-most-popular-video-streaming-services-by-monthly-average-users/.

[3] *Netflix revenue and Usage Statistics (2022).* Jan. 2022. URL: https://www.businessofapps.com/data/netflix-statistics/.

[4] Julia Stoll. *Netflix subscribers from 2001 to 2011.* Feb. 2012. URL: https://www.statista.com/statistics/272551/subscribers-of-netflix-since-2001/.

[5] *Auto Scaling Documentation.* URL: https://docs.aws.amazon.com/autoscaling/index.html.

[6] Netflix Technology Blog. *A closer look at the Christmas Eve outage.* Apr. 2017. URL: https://netflixtechblog.com/a-closer-look-at-the-christmas-eve-outage-d7b409a529ee.

[7] Netflix Technology Blog. *Keystone real-time stream processing platform.* Sept. 2018. URL: https://netflixtechblog.com/keystone-real-time-stream-processing-platform-a3ee651812a.

[8] Nick Heath et al. *AWS outage: How netflix weathered the storm by preparing for the worst.* Sept. 2015. URL: https://www.techrepublic.com/article/aws-outage-how-netflix-weathered-the-storm-by-preparing-for-the-worst/.

[9] *Unreeling Netflix: Understanding and improving multi-CDN movie delivery.* URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&amp;arnumber=6195531.

[10] *Mastering chaos - a Netflix guide to microservices.* URL: https://www.youtube.com/watch?v=CZ3wIuvmHeM.

[11] Cao Duc Nguyen. *A design analysis of cloud-based microservices architecture at Netflix.* May 2020. URL: https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f.

[12] *Open Connect Deployment Guide - Netflix.* URL: https://openconnect.netflix.com/en/deployment-guide/.

[13] *Open Connect Overview - Netflix.* URL: `https://openconnect.netflix.com/Open-Connect-Overview.pdf`.

[14] *Netflix Velocity Conference 2011.* URL: `https://www.slideshare.net/adrianco/netflix-velocity-conference-2011`.

[15] Dr. Werner Vogels. *Seamlessly extending the data center - Introducing Amazon Virtual Private Cloud.* Aug. 2009. URL: `https://www.allthingsdistributed.com/2009/08/amazon_virtual_private_cloud.html`.