REPORTE A NETFLIX GUIDE TO MICROSERVICES

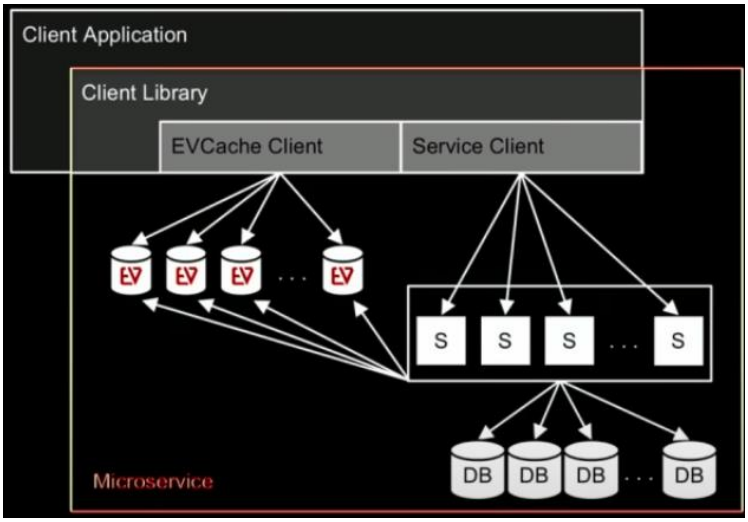
Emanuel Alejandro Gutierrez Romero

NOVEMBER 28, 2023

COMPUTACION TOLERANTE A FALLAS

Prof. Michel Emanuel Lopez Franco

Comenzamos por definir lo que es un microservicio según Martin Fowler “Son un enfoque para desarrollar una única aplicación como un conjunto de pequeños servicios, cada uno ejecutandose en su propio proceso y comunicandose con mecanismos ligeros, comunmente una API de recursos HTTP”. Esto es una definición bastante técnica y abstracta sobre lo que son los microservicios. Después comentemos sobre que los microservicios realmente son una abstracción, comienza a hablar sobre los problemas que existen dentro de los programas que es la escalabilidad pone un ejemplo sobre que utilizar una caché y en caso de que esta falle entonces llamará al servicio que al mismo tiempo llamará a la base de datos.



Esta es una de las formas de ver los microservicios, que son un conjunto de tecnologías , configuraciones “complejas”, así se definiría una estructura del microservicio.

Ahora hablemos sobre la falla dentro del microservicio conocido como “Cascading Failure”, podríamos decir que es un fenómeno en que el fallo de un servicio provocará una reacción en cadena o como su nombre lo indica en cascada de fallos en otros servicios. Esto ocurre usualmente cuando los servicios están interconectados entre sí y dependen de los otros. Los errores más comunes son:

- Sobrecarga: El sobrecargar un servicio puede ocasionar su fallo, es decir si hay un aumento repentino en la demanda de este.

- Errores de configuraciones: Configuraciones erroneas de los microservicios, como por ejemplo una mala configuración de un firewall que impediría que los microservicios trabajen entre ellos.

Ahora analicemos los factores para evitar este tipo de errores:

- Pruebas: Los sistemas de microservicio deben de ser probados y autorizados.
- Monitoreo: Deben de ser monitoreados constantemente para evitar los problemas en cascada.

Ahora las técnicas:

- Circuit breakers: Son dispositivos que pueden interrumpir el flujo de tráficos a un servicio si detecta un fallo.
- Resilience patterns: Los patrones de resiliencia son patrones de diseño que ayudan a los sistemas a ser tolerantes a fallas.
- Health checks: Son comprobaciones que se realizan para determinar el estado de un servicio.

Ahora hablemos sobre los microservicios críticos, son aquellos que son esenciales para el buen funcionamiento de una aplicación o sistema que en caso de fallar puede ocasionar la pérdida de datos o incluso daños financieros. Están interconectados con otros microservicios y en caso de fallar ocasionaría el problema antes mencionado de cascada. Después se menciona sobre la infestación de parásitos (Parasitic Infestation) que ocasionan bastantes problemas desde los lógicos hasta errores físicos.

Continuando tenemos que hablar sobre el teorema de CAP básicamente menciona que es imposible para un sistema de cómputo garantizar simultáneamente:

- Consistenecia
- Disponibilidad
- Tolerancia a particiones.

Puede escoger entre dos, pero nunca puede tomar las tres, el teorema CAP es importante para poder comprender de manera más detallada los sistemas distribuidos para que así los desarrolladores conozcan como construir sus sistemas.

Entonces hablemos sobre “Eventual consistency”, es un modelo de coherencia de datos en el cual las actualizaciones de datos se propagan de nodo a nodo dentro de un sistema distribuido de forma asíncrona. Garantiza que todos los nodos convergerán eventualmente, lo que significa que en cierto punto tendrán la misma versión de datos en algún momento en el futuro. Es modelo de coherencia útil para los microservicios que permite a los sistemas ser escalables y disponibles.

Es turno de hablar sobre “Stateful service” que mantiene el estado de una sesión o transacción entre el cliente y el servidor. Esto significa que el servidor guarda información sobre el estado actual de la sesión o transacción para así poder responder a las solicitudes del cliente de una manera coherente. Estos servicios suelen ser utilizados para aplicaciones que requieren mantener un estado persistente, como el carrito de compras o el historial de pedidos de un cliente. Son simples de implementar y eficientes. Unas de las desventajas sería que son menos escalables ya que el servidor necesita mantener información de estado para cada sesión de cliente y pueden llegar a ser menos tolerante ante fallos ya que la información se pierde si el servidor llega a fallar.

Ahora es importante mencionar sobre la carga excesiva o como se le conoce “Excessive load”, básicamente es una condición en la que un sistema recibe más entradas de las que puede manejar. Esto puede hacer que el sistema se ralentice, se llegue a volver insensible o incluso se bloquee. Existen distintos factores que la ocasionan entre ellas:

- Un aumento en el número de usuarios.
- Un aumento en la complejidad del sistema.
- Aumento en la frecuencia de solicitudes.

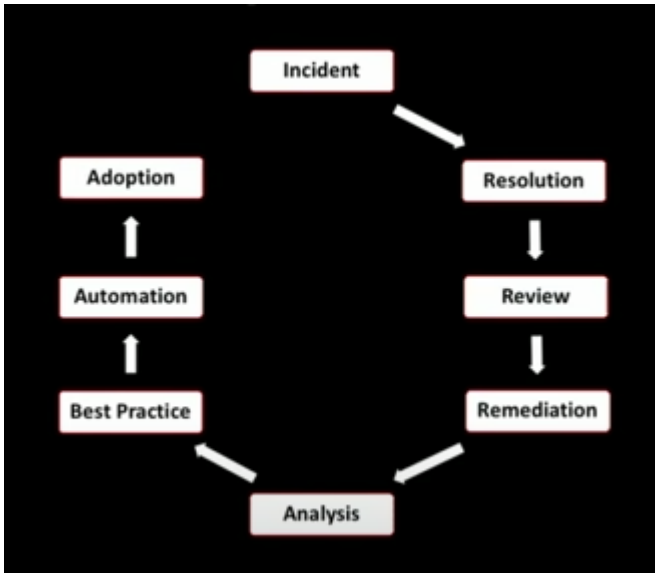
Las consecuencias que se pueden tener debido a las sobrecargas serían:

- Rendimiento reducido.

- Tasas de error más altas.
- Tiempo de inactividad aumentado.

Para evitar las sobrecargas es importante contar con un sistema que sea escalable y que además puedan manejar una carga aumentada. Haciendo uso de técnicas como el equilibro de cargas, caché y otras técnicas para poder distribuir la carga en varios servidores.

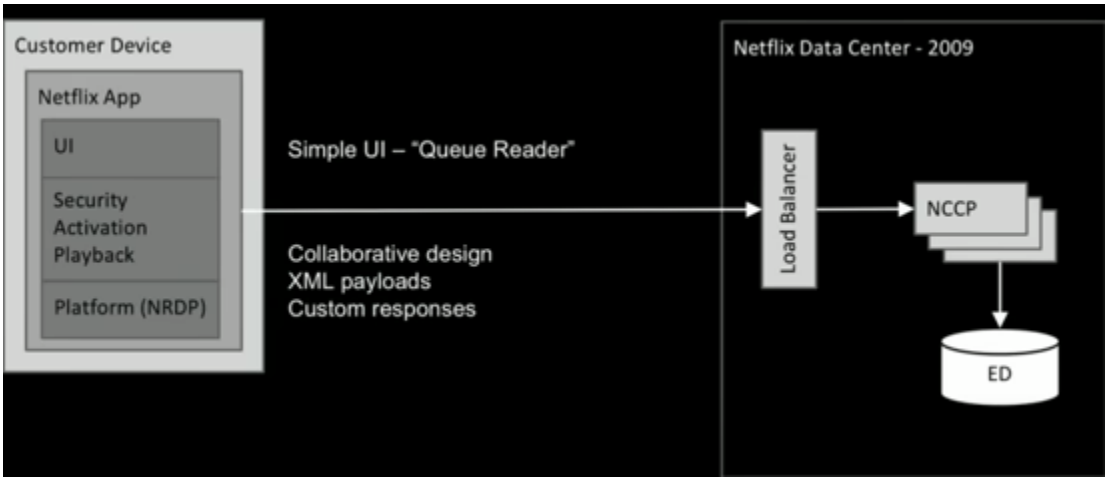
Después seguimos hablando y hay un esquema que me gustó que representaran dentro del video sobre el aprendizaje continuo y automatización.



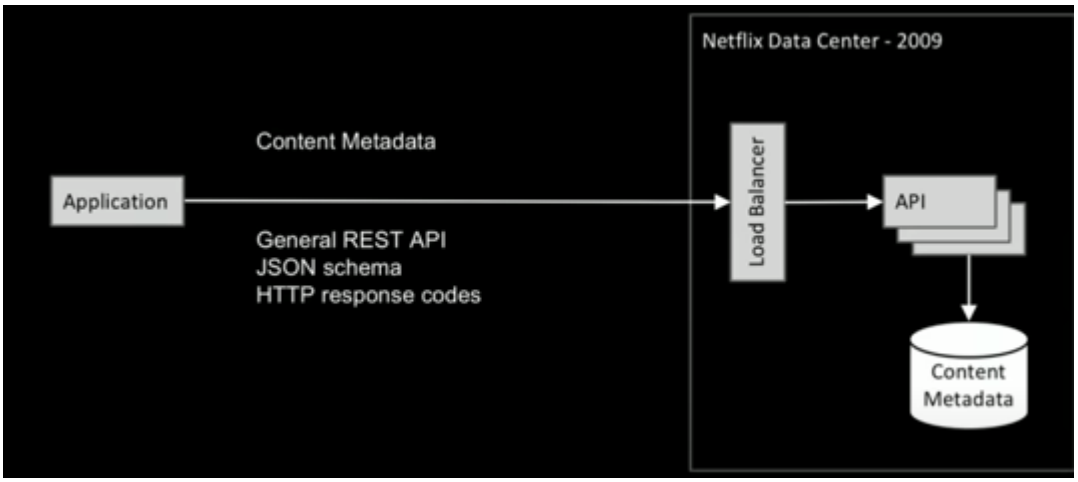
A continuación, en el video se comienza a hablar acerca de cuando la producción ya está lista para comenzar, utilizando diferentes técnicas como auto escalado, consistencia, configuraciones ELB, entre otras. Después habla sobre los distintos caminos que existían para llegar a Netflix, entre ellos estaba el camino pavimentado o como se menciona en el vídeo (The paved road) que habla sobre las diferentes herramientas que utilizaron entre ellas el uso de Java y ec2 para los contenedores. Menciona también del camino que se va construyendo por si solo, haciendo uso de python y de ruby y además reescribiendo el código en node js y tiempo después implementando docker que es cuando las cosas se comenzaron a complicar un poco.

Después se menciona sobre la ruta crítica (Critical path) para los clientes, en este apartado se mencionan como las API tienen la capacidad de poder integrar scripts los cuales pueden llegar a actuar como puntos finales que básicamente desarrollan distintas acciones dentro del sistema que básicamente menciona que dentro de estas rutas pueden existir errores los cuales puden consumir toda la memoria disponible y crear una cadena de errores y por ello una solución para poder evitar esto es tomar esos puntos finales y sacarlos del servicio y moverlos a pequeños nodos de js que trabajan en archivos docker. Para después hablar sobre el Cost of variance, donde se implican la productividad, la fragmentación de imagen, el manejo de nodos, duplicación de bibliotecas/plataformas, curva de aprendizaje. Para después hablar sobre las estancias de estrategia que son estrategias que mencionan en tener una mayor conciencia sobre los costos, priorizar y buscar soluciones.

Ahora se hablará sobre la organización y la arquitectura. Donde en la primera parte únicamente se habla sobre cómo funcionaba una de las primeras versiones de Netflix y muestra un esquema sobre las funcionalidades que básicamente eran haciendo uso de cargas de XML.



Ahora de como la API de Netflix pasó de pública a privada, basado en un esquema JSON, utilizando códigos de respuesta HTTP.



Menciona también sobre una arquitectura híbrida que ahora contaba con una fragmentación entre niveles que ahora funcionan de maneras muy distintas uno basado en JSON,REST, OAuth y el otro usando RPC, XML, NTBA y un mecanismo de seguridad para trabajar con tokens y con un firewall.

Mencionando sobre como trabaja como ejemplo el uso de las licencias de las películas y como cuando el usuario daba click para verla se le prestaba una “licencia” y que cuando terminaba de verla devolvía esa licencia. Se hace mención sobre la ley de Conway (Conway´s Law) que básicamente se basa en la idea de que los sistemas son creados por personas, y las personas están limitadas por sus propias experiencias y perspectivas, esta ley tiene implicaciones importantes para el diseño y arquitectura de los sistemas.