Arquitetura de Software

Grupo: Bruno Gomes, João Mairinque, Matheus Andrade, Matheus Vieira e Samara Martins.

Sistema: Twitch.

Motivação: Mudança de arquitetura.

Apresentação do sistema:

Twitch é uma plataforma de transmissão de vídeo ao vivo que se destaca principalmente pelo streaming de jogos. Neste ambiente, os utilizadores têm a oportunidade de interagir uns com os outros, tanto através de um chat disponível como através das transmissões de vídeo e áudio. O sistema foi lançado em junho de 2011 com uma arquitetura monolítica desenvolvida com Ruby on Rails. Entretanto, a base de usuários do Twitch começou a crescer rapidamente, chegando a cerca de 20.000 usuários no mesmo ano do lançamento, o que naquele período era considerado um número elevado. Com esse aumento, problemas como gargalo de desempenho começaram a surgir, onde, partes do sistema começaram a atingir seus limites, prejudicando seu funcionamento como um todo.

Atualmente a twitch alcança média superior a 2 milhões de viewers simultâneos por mês, e teve seu pico máximo de 6.6 milhões em 24 de Junho de 2022. Além disso, conta com a média superior a 80 mil canais ao vivo simultaneamente, com o pico de 233 mil em 30 de novembro de 2020. A Figura 01 abaixo apresenta o gráfico de viewers, demonstrando a média e os picos durante os anos.

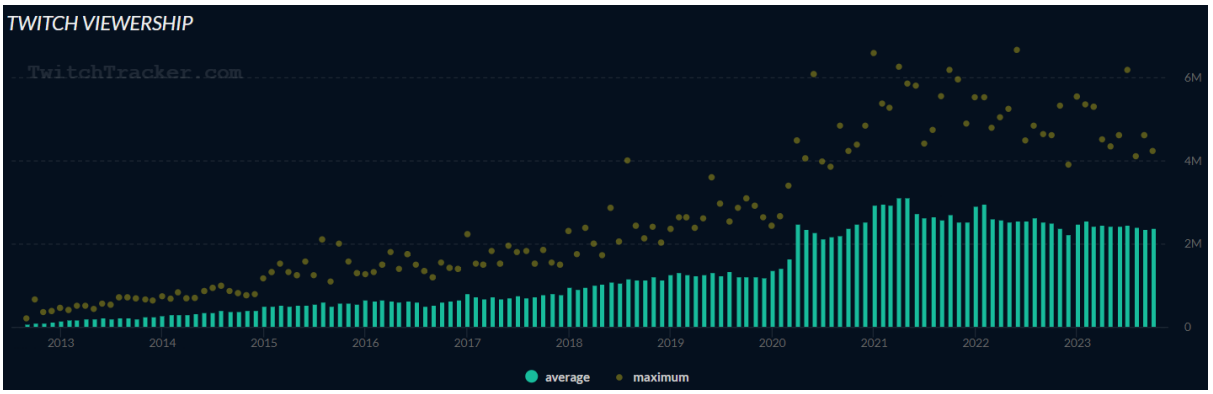


Figura 01: Gráfico com as médias e picos de viewers durante os anos.

A twitch abrange alguns tipos de usuários como os viewers, canais, broadcasters e parceiros. O crescimento da plataforma em relação aos tipos de usuários durante os anos estão apresentados na Figura 02 abaixo.

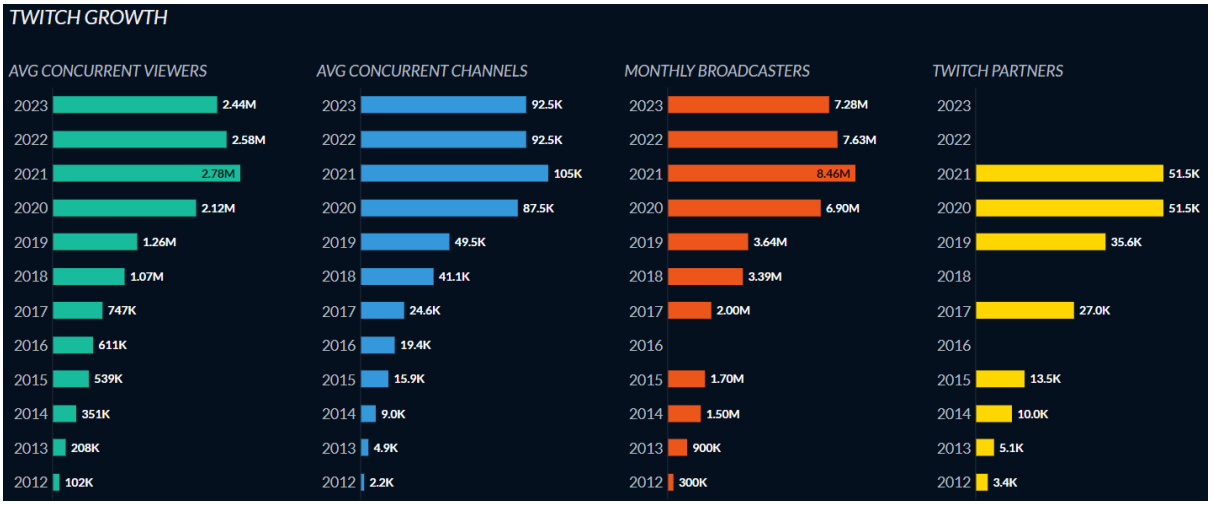
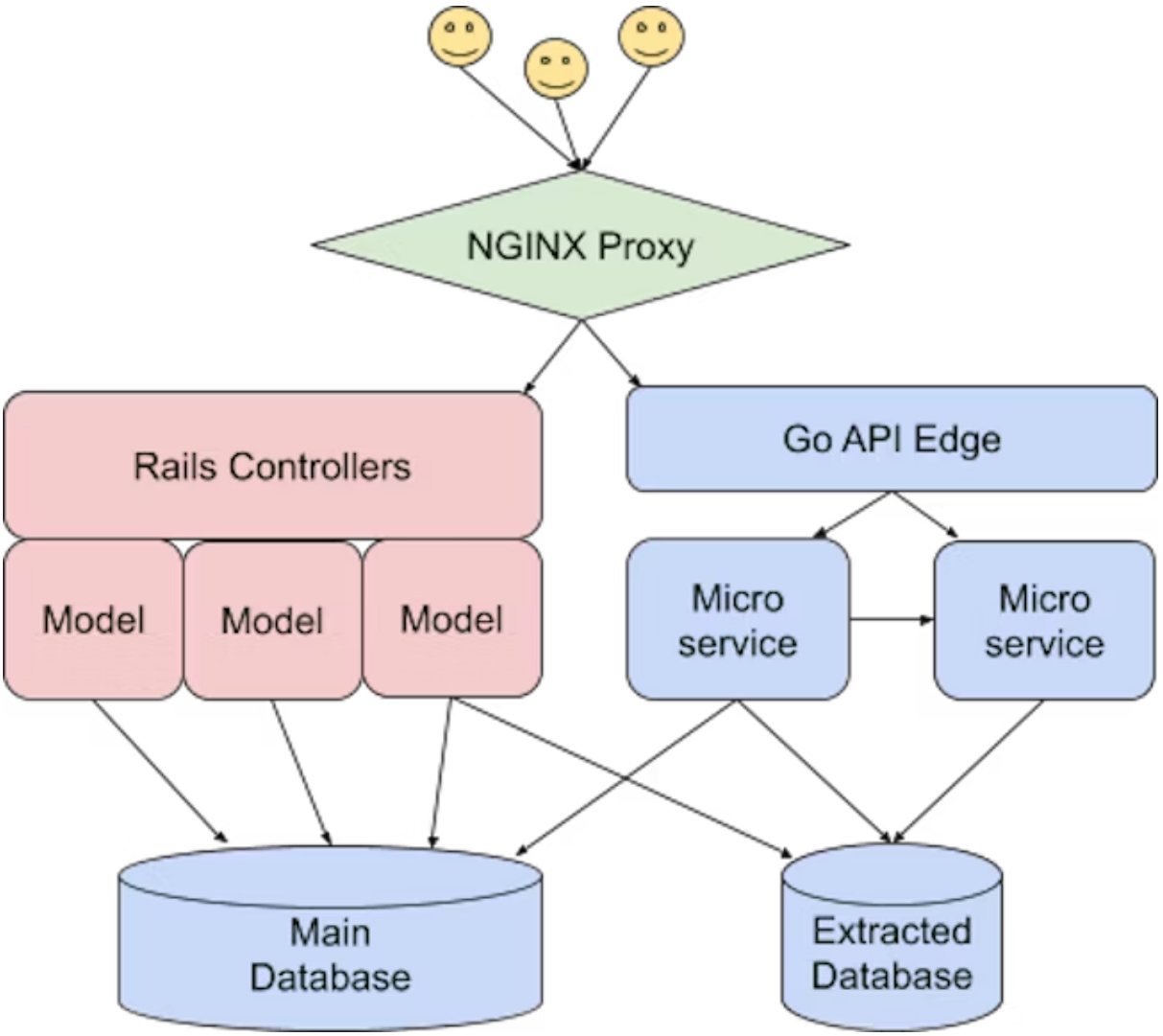


Figura 02: Gráficos em barras demonstrando a relação de crescimento dos usuários na plataforma entre os anos de 2012 a 2023.

Arquitetura de software:

A arquitetura em monolito acarretava em várias dificuldades para os desenvolvedores, como o uso de uma única pipeline deploy para todos, dificuldade de coordenação do ambiente de testes, lentidão na requisição no banco de dados, rastreamento lento devido ao grande volume de *endpoints* da API. O modelo até então desenvolvido em Ruby era caracterizado como uma "*spaghetti architecture*". A transição para a arquitetura de microsserviços implementada principalmente em Go teve seu início e principal esforço a adição de uma camada extra mais externa da API *backend*, um *proxy* reverso chamado NGINX. O NGINX tem como função direcionar parte do tráfego para uma nova parte da borda da API, que utiliza um *middleware* comum para validação, autorização e limitação de taxa; mas delega toda a lógica de negócios aos microsserviços subjacentes.



Com a resolução da implementação do proxy foi definido um processo de migração dos *endpoints*:

1. Desenvolver novo conjunto de microsserviços;
2. Replicar os antigos endpoints na nova borda;
3. Atualizar a configuração do NGINX para direcionar o percentual de tráfego para os novos *endpoints*;
4. Verificar as métricas e incrementar o tráfego de dados até os 100%.

Com os desenvolvedores implementando novas funcionalidades no novo modelo, a arquitetura monolítica passou a ficar muito obsoleta. Foi necessário uma adaptação da equipe para o novo modelo de implementação, que levou de 2016 a 2018, com os desenvolvedores aprendendo como prover os recursos da AWS e adaptando-se ao Go.

O modelo de microsserviços é muito mais complexo que o modelo monolítico. Enquanto no último a chamada de um módulo é o equivalente a chamar uma função

ou método, no primeiro a chamada de um serviço demanda muito mais: tratamento de erro, monitoramento, versão de API, entre outros.

No que diz respeito à carga de trabalho, a migração dos *endpoints* foi o processo mais difícil e demorado apesar de ser a menor parte da transição da arquitetura, seguindo a regra de “pareto”, onde os últimos 20% do trabalho levam 80% do tempo. A dificuldade estava relacionada com segurança e qualidade de código, tendo em vista que atendiam às regras de negócio. Os *endpoints* mantinham tráfego de recursos desconhecidos, talvez *bots* ou *scripts*.

Quando em 2018 quando o tráfego foi desvinculado no modelo monolítico, todas as instâncias EC2 da conta principal da AWS foram interrompidas. Levando mais um ano para a limpeza de dependências da conta e dos pontos de acesso.

Atualmente todo o ambiente de microsserviço é isolado em sua própria conta AWS. Times de produtos distintos podem operar independentemente, gerenciando sua própria infraestrutura e ajustando os serviços conforme necessário.

Impacto Organizacional

Segundo a lei de Conway, as estruturas de um sistema de software tendem a ‘’espelhar’’ a estrutura da organização que o desenvolvem, sendo assim, uma empresa que opta por fazer uma transição para um sistema de microsserviços, naturalmente passa por diversas turbulências até finalmente criar times especializados mas que consigam se comunicar bem, assim como um microsserviço.

Essa tendência também ocorreu na Twitch, onde o crescimento desenfreado da plataforma acabou adicionando uma camada de dificuldade nas migrações que estavam ocorrendo na plataforma. De acordo com o autor do artigo, em 2017 houve um caso onde uma nova versão do Twitch mobile causou um número exagerado de requisições na API, o que aconteceu foi que ambos os times haviam testado suas versões de forma isolada mas não haviam feito um teste de integração entre a versão mobile e a API.

Como uma tentativa de evitar pontos cegos como os citados, as companhias tendem a fazer as chamadas ‘’reorgs’’, que é basicamente uma ação onde os engenheiros de software reavaliam as migrações que estão ocorrendo no momento, reveem quais serviços de fato de fato precisam ser divididos, atualizam permissões de acesso caso necessário e propõem refatorações de código. Esses pontos cegos são praticamente inevitáveis, mas podem ser diminuídos quando se utilizam desse tipo de estratégia.

Apesar das migrações para o microsserviços terem sido árduas e cheias de desafios, o time de desenvolvedores e engenheiros da Twitch conseguiu chegar em um patamar onde a maioria dos times possuem vários serviços onde cada ambiente

é isolado em uma conta separada da AWS. É dito pelo autor do artigo que as migrações na plataforma da Twitch continuam acontecendo conforme novas tecnologias vão surgindo, mas a situação atual da empresa permite que um time consiga implementar essas tecnologias sem que afete os outros times.

Referências

- https://twitchtracker.com/statistics
- https://blog.twitch.tv/en/2023/09/28/twitch-state-of-engineering-2023/
- https://blog.twitch.tv/en/2022/04/12/breaking-the-monolith-at-twitch-part-2/
- https://pt.slideshare.net/InfoQ/twitch-plays-pokmon-twitchs-chat-architecture