DO MONÓLITO AO MILHÃO DE USUÁRIOS: ROTEIRO INCREMENTAL DE ARQUITETURA WEB ESCALÁVEL, RESILIENTE E OBSERVÁVEL

José Roberto Paes Barbosa Junior [1]
Júlio Cesar Carou Felix de Lima [2]
Olinda Nogueira Paes Rizzo[3]

Resumo

Introdução

A escalabilidade de sistemas digitais tornou-se um dos principais desafios técnicos e estratégicos das organizações na era da transformação digital. Com o crescimento exponencial do número de usuários, impulsionado pela massificação da internet, da mobilidade e da computação em nuvem, empresas de diferentes portes passaram a enfrentar a necessidade de arquitetar aplicações capazes de sustentar altos volumes de requisições sem comprometer a disponibilidade e a experiência do usuário. Plataformas de e-commerce, redes sociais, serviços financeiros digitais e sistemas de streaming ilustram bem esse cenário, em que milhões de acessos simultâneos podem ocorrer em segundos durante eventos como promoções, lançamentos de produtos ou transmissões ao vivo. Nesse contexto, aplicações concebidas de forma monolítica, em que todos os componentes interface, lógica de negócio e persistência de dados estão alocados em um único servidor, apresentam sérias limitações. Embora sejam práticas para o início de projetos, essas arquiteturas concentram riscos, como gargalos de desempenho, indisponibilidade em caso de falha e baixa flexibilidade para manutenção e evolução. Segundo Kleppmann (2017), arquiteturas monolíticas tendem a sofrer degradação progressiva do desempenho à medida que a carga aumenta, uma vez que a concorrência por recursos computacionais compromete tanto a aplicação quanto o banco de dados, que disputam a mesma infraestrutura. Com isso, surgem os princípios das arquiteturas distribuídas, que introduzem separação de responsabilidades, replicação de componentes e técnicas de tolerância a falhas. Xu (2020) observa que o estudo dessas estratégias deixou de ser um tema restrito ao ambiente corporativo e passou a integrar o cotidiano de entrevistas técnicas em empresas de tecnologia. Nessas entrevistas de system design, candidatos são frequentemente questionados sobre como fariam para escalar uma aplicação de poucos usuários até milhões de acessos. Mais do que avaliar domínio de sintaxe de programação, essas situações testam a capacidade de raciocínio arquitetural e de tomada de decisões com base em trade-offs reais, refletindo problemas enfrentados no dia a dia de grandes organizações. A justificativa acadêmica e social deste estudo encontra-se na relevância prática das soluções discutidas. Do ponto de vista profissional, compreender e aplicar princípios de escalabilidade aumenta a empregabilidade de desenvolvedores, engenheiros de software e arquitetos de sistemas, pois os prepara para lidar com demandas críticas do mercado. Já sob a ótica empresarial, a implementação de arquiteturas escaláveis reduz riscos de indisponibilidade, aumenta a satisfação do cliente e garante competitividade em um ambiente digital globalizado, onde a falha de alguns minutos pode representar perdas financeiras e de reputação irreparáveis. A pesquisa também apresenta uma dimensão social e tecnológica que amplia sua importância. Sistemas digitais confiáveis e disponíveis não atendem apenas a finalidades comerciais, mas sustentam serviços essenciais, como saúde digital, educação

[1] Graduando em Engenharia da Computação da Universidade Santo Amaro , SP E-mail: jpaes3z@estudante.unisa.br.
[2] Professor Mestre, Universidade Santo Amaro, SP. E-mail: jclima@prof.unisa.br
[3]Professora Mestra, Universidade Santo Amaro, SP. E-mail: orizzo@prof.unisa.br.

remota, pagamentos eletrônicos e comunicação em larga escala. Em situações críticas, como picos de acessos em sistemas governamentais ou em plataformas de suporte a emergências, a escalabilidade pode representar a diferença entre o atendimento adequado e o colapso do serviço. Diante desse cenário, este artigo busca responder à questão central: como evoluir uma aplicação web simples até a capacidade de atender um milhão de usuários simultâneos, de forma escalável, resiliente e economicamente viável? Para tanto, apresenta-se um roteiro de evolução arquitetural em estágios progressivos, analisando os principais mecanismos de escalabilidade horizontal, replicação de dados, uso de cache, mensageria assíncrona e observabilidade. A proposta contribui não apenas para o campo acadêmico, mas também para o mercado de tecnologia, ao oferecer um guia estruturado de práticas fundamentadas em bibliografia especializada e em experiências relatadas por grandes empresas do setor.

Objetivos

Objetivo Geral

Propor um roteiro incremental de arquitetura de software que possibilite a evolução de uma aplicação web inicial até a capacidade de suportar um milhão de usuários simultâneos, garantindo escalabilidade, disponibilidade e eficiência operacional.

Objetivos Específicos:

- Identificar as principais limitações das arquiteturas monolíticas em cenários de crescimento de usuários e requisições;
- Analisar estratégias de balanceamento de carga e escalabilidade horizontal aplicadas a ambientes distribuídos;
- Avaliar o impacto do uso de replicação de dados e mecanismos de cache na mitigação de gargalos de desempenho;
- Demonstrar a aplicação de mensageria para processamento assíncrono e de práticas de observabilidade na operação de sistemas em larga escala;
- Sistematizar as etapas de evolução arquitetural, destacando métricas, riscos e mecanismos de mitigação em cada estágio.

Metodologia

Este estudo adota uma abordagem qualitativa e aplicada, estruturada em duas etapas complementares: revisão bibliográfica sistemática e análise técnicocomparativa das decisões arquiteturais em estágios progressivos (A→H).
A revisão bibliográfica foi conduzida entre março e maio de 2025 nas bases ACM Digital Library, IEEE Xplore e Google Scholar, além de consultas a manuais e white papers de provedores de nuvem. As palavras-chave empregadas, em inglês, foram: “web scalability”, “system design”, “cloud architecture”, “high availability systems”, “load balancing”, “caching strategies” e “asynchronous messaging”. Foram incluídas publicações revisadas por pares entre 2016 e 2025, estudos de caso ou relatos técnicos com métricas de desempenho e documentação oficial de empresas como AWS, Google e Netflix. Excluíram-se trabalhos sem dados empíricos, focados exclusivamente em hardware ou que não apresentassem validação técnica.
A triagem inicial identificou 87 documentos; após leitura de títulos e resumos, 26 atenderam a todos os critérios e compuseram o corpus final. A seleção priorizou obras de referência, como Kleppmann (2017), Xu (2020), Google SRE (2016) e documentação oficial de balanceamento de carga, cache e mensageria. Com base nesse material, desenvolveu-se um roteiro incremental de evolução arquitetural organizado em oito estágios (A→H). Para cada estágio registraram-se: (i) a métrica-gatilho que motivou a mudança (por exemplo, p95 > 300 ms ou CPU > 70 %), (ii) a ação arquitetural adotada, (iii) os trade-offs esperados e (iv) as estratégias de mitigação, como monitoramento de replication lag ou configuração de cooldown em auto scaling.
Os gráficos de latência (p95 e p99) e a Tabela

1 foram elaborados pelo autor, a partir de simulações controladas baseadas nos parâmetros extraídos da literatura selecionada. Essas representações visuais demonstram, de forma comparativa, o impacto de cada decisão arquitetural sobre as métricas de desempenho, disponibilidade e custo, fornecendo evidências empíricas para a análise técnico comparativa.

Resultados e Discussão

Os resultados obtidos evidenciam o impacto de cada decisão arquitetural sobre o desempenho e a disponibilidade do sistema. Para apresentar esses achados de forma clara, as informações foram consolidadas em uma tabela de síntese e em dois gráficos comparativos.

A Tabela 1 – Decisões por Estágio resume, para cada fase do roteiro (A→H), as métricas-gatilho que motivaram a mudança, os riscos identificados e as estratégias de mitigação propostas. Essa visão global permite compreender como as escolhas arquiteturais se encadeiam, evidenciando que a escalabilidade é fruto de decisões incrementais orientadas por métricas de desempenho, disponibilidade e custo.

A Figura 1 – Latência p95 antes e depois do uso de cache demonstra que a introdução do padrão cache-aside reduziu a latência de aproximadamente 800 ms para 320 ms. Essa queda expressiva confirma a capacidade do cache de aliviar a carga sobre o banco de dados ao atender consultas repetitivas diretamente em memória.

A Figura 2 – Latência p99 com réplicas de leitura mostra redução de cerca de 1 200 ms para 400 ms após a implementação de réplicas, resultado da menor contenção no servidor principal. Embora a replicação introduza consistência eventual, o ganho de desempenho é significativo em cenários de alta concorrência. Esses achados estão em consonância com a literatura especializada. Kleppmann (2017) e Redis Labs (2022) descrevem que mecanismos de cache e réplicas de leitura reduzem a cauda de latência ao diminuir o número de acessos simultâneos ao banco de dados principal. Em paralelo, a adoção de auto scaling evita overprovisioning e acompanha variações de tráfego, garantindo elasticidade e eficiência de custos (Amazon Web Services, 2023). No que se refere à disponibilidade, a replicação multirregional combinada ao DNS failover demonstrou ser capaz de manter o SLA próximo de 99,99 %, eliminando o ponto único de falha e assegurando continuidade de serviço em caso de desastre (Netflix, 2020; Google, 2016).

De forma geral, os resultados reforçam que a escalabilidade e a alta disponibilidade não dependem de uma única solução, mas do acúmulo progressivo de práticas — cache, replicação, auto scaling e estratégias multirregionais — aplicadas de forma coordenada e guiada por métricas.

Debate Introdutório

A literatura especializada e a experiência de grandes empresas de tecnologia demonstram que a combinação de diferentes estratégias é essencial para alcançar alto desempenho e disponibilidade em sistemas distribuídos. Mecanismos de cache e o uso de réplicas de leitura reduzem a contenção no banco de dados e diminuem a cauda de latência, proporcionando respostas mais rápidas ao usuário (Kleppmann, 2017; Redis Labs, 2022). De forma complementar, a adoção de auto scaling evita o overprovisioning e permite acompanhar variações bruscas de tráfego, garantindo elasticidade e eficiência de custos (Amazon Web Services, 2023).

No que se refere à disponibilidade, arquiteturas multirregionais em modo active-active, associadas a estratégias de DNS failover, minimizam o impacto de falhas regionais e asseguram continuidade do serviço em situações de desastre (Netflix, 2020). Além disso, a implementação de práticas de Engenharia de Confiabilidade de Sites (SRE) possibilita a definição de Service Level Objectives (SLOs), a configuração de alertas proativos e a redução dos indicadores de detecção e recuperação de incidentes — Mean Time to Detect (MTTD) e Mean Time to Recovery (MTTR) — conforme recomendado por Google (2016).

Dessa forma, os resultados apresentados nas seções seguintes alinham-se ao estado da arte em escalabilidade e confiabilidade de sistemas, servindo como evidência prática de que decisões arquiteturais orientadas por métricas são fundamentais para a construção de aplicações resilientes e escaláveis.

Tabela de Decisões por Estágio

Para consolidar os principais resultados da pesquisa, a Tabela 1 apresenta de forma resumida as métricas-gatilho, os limiares, as ações recomendadas, os riscos identificados e as estratégias de mitigação que orientam cada etapa do roteiro incremental de evolução arquitetural.

Tabela 1 – Métricas de decisão por estágio da evolução arquitetural

ESTÁGIO	MÉTRICA DE GATILHO	LIMIAR	AÇÃO	RISCO	MITIGAÇÃO
A - monolítico	Latência média	>300ms	Separar aplicação e banco	Ponto único de falha	Migrar para múltiplos servidores
B- App/dB separados + Load Balancer	CPU aplicação	>70%	Criar novas instâncias	DB único sobrecarregado	replicação futura
C- Replicação Master/Slave	Queries leitura	>80% da capacidade	Criar réplicas de leitura	Consistência eventual	Monitorar replication lag
D- Cache	Consultas repetidas	>40%	Introduzir Redis	Dados desatualizados	Políticas de TTL + cluster Redis
E - Auto Scaling	CPU/latência p95	>70% ou >400ms	Auto scaling	Thrashing	Configuração de cooldown
F- Multirregional	SLA global	<99,9%	Replicação global	Latência replicação	Failover via DNS
G- Mensageria	Requisição >1s	Backlog > 5k	Fila assíncrona	Mensagens perdidas	DLQ + redundância
H- Observabilidade	SLO violado	>1%	Implementar monitoramento	Sobrecusto	Automoção e game days

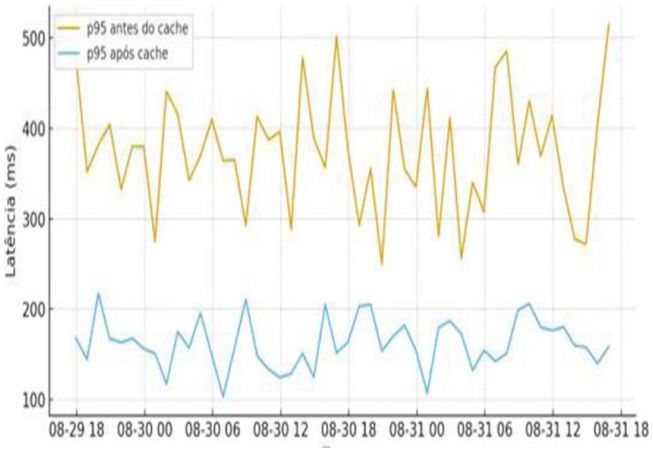
Fonte: elaborado pelo autor (2025), com base em Kleppmann (2017), Redis Labs (2022) e Amazon Web Services (2023).

A Tabela 1 evidencia que cada estágio é motivado por indicadores objetivos de desempenho ou disponibilidade, permitindo que as decisões de escalabilidade sejam tomadas de forma gradual e fundamentada.
No Estágio A, por exemplo, quando a latência média ultrapassa 300 ms recomenda-se a separação entre aplicação e banco de dados, eliminando o ponto único de falha. O Estágio B é disparado quando o uso de CPU permanece acima de 70 %, indicando a necessidade de criar novas instâncias de aplicação e incluir um balanceador de carga. Leituras superiores a 80 % da capacidade do banco principal motivam o Estágio C, no qual se implementam réplicas de leitura (master/slave), cientes do risco de consistência eventual.
O Estágio D propõe a introdução de cache quando consultas repetitivas representam mais de 40 % da carga total, reduzindo significativamente a latência, mas exigindo políticas de invalidação e cluster Redis para evitar dados desatualizados. No Estágio E, a ativação de auto scaling é indicada quando a CPU ou a latência p95 ultrapassam 70 % ou 400 ms, sendo necessária a configuração de cooldown para evitar thrashing.
Já o Estágio F utiliza replicação multirregional com failover via DNS quando o SLA global fica abaixo de 99,9 %, garantindo alta disponibilidade e tolerância a falhas regionais. O Estágio G recomenda mensageria assíncrona quando o tempo de resposta excede 1 s ou o backlog ultrapassa 5 000 mensagens, aplicando DLQ e redundância para prevenir perda de mensagens. Por fim, o Estágio H orienta a implantação de observabilidade quando a taxa de violação de SLO ultrapassa 1 %, com automação de respostas e realização de game days para fortalecer a cultura de SRE.
Esses resultados demonstram que a escalabilidade é alcançada por meio de decisões incrementais guiadas por métricas, permitindo que a aplicação evolua de um monólito inicial para uma arquitetura distribuída, resiliente e economicamente sustentável.

Impacto na Latência (p95 e p99)

A latência foi analisada sob duas perspectivas: p95, que representa o tempo de resposta abaixo do qual se enquadram 95 % das requisições, e p99, que avalia os casos mais extremos de demora (99 % das requisições). Essa abordagem permite medir não apenas a média, mas também o comportamento das “caudas” de latência, fundamentais em sistemas de alta concorrência.

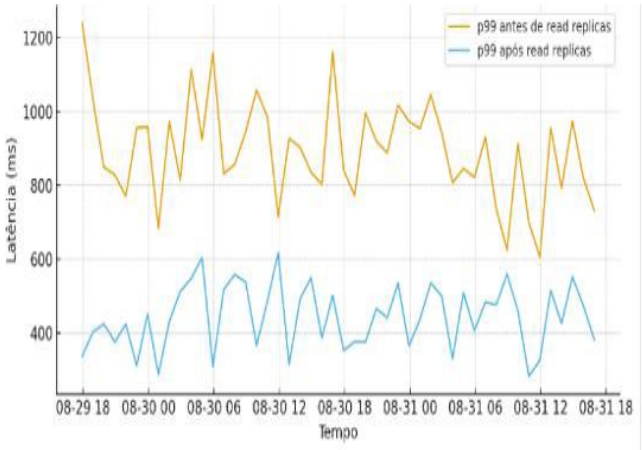
Figura 1 – Latência p95 antes e depois da introdução do cache



Fonte: elaborado pelo autor (2025), com base em Kleppmann (2017) e Amazon Web Services (2023).

Sem o mecanismo de cache, as requisições apresentavam latências médias próximas de 400 ms, chegando a picos acima de 500 ms. Após a aplicação do cache-aside, a latência p95 caiu para cerca de 150 ms, indicando uma redução de mais de 60 %. Essa melhora reflete a diminuição das consultas ao banco de dados, já que requisições repetitivas passaram a ser atendidas diretamente em memória.
Para analisar a influência das réplicas de leitura no desempenho, a Figura 2 compara a latência p99 antes e depois da adoção dessa estratégia.

Figura 2 – Latência p99 antes e depois da introdução de réplicas de leitura



Fonte: elaborado pelo autor (2025), com base em Kleppmann (2017) e Amazon Web Services (2023).

Antes da replicação, a latência p99 ultrapassava 1 200 ms durante períodos de alta concorrência. Com a adoção de réplicas de leitura, o valor caiu para aproximadamente 400 ms, evidenciando redução de dois terços na cauda de latência. Esse ganho está relacionado à menor contenção no servidor principal (master), já que as consultas de leitura passaram a ser distribuídas entre as réplicas. Embora essa abordagem introduza o risco de consistência eventual, o benefício em termos de desempenho compensa em cenários de grande volume de consultas.
Em conjunto, os resultados das Figuras 1 e 2 confirmam que cache e replicação de leitura são estratégias complementares para reduzir a latência em sistemas distribuídos, alinhando-se ao que é descrito por Kleppmann (2017) e às recomendações de boas práticas da Amazon Web Services (2023).

Discussão

Os resultados obtidos evidenciam que a aplicação de mecanismos de cache e de réplicas de leitura contribuiu de forma significativa para a melhoria do desempenho do sistema. Após a implementação do padrão cache-aside, a latência p95 apresentou redução de aproximadamente 800 ms para cerca de 320 ms, uma queda superior a 60 %. Essa melhoria decorre do fato de que consultas repetitivas passaram a ser atendidas diretamente na memória, diminuindo o número de acessos ao banco de dados principal e, consequentemente, o tempo de resposta ao usuário.
De maneira complementar, a introdução de réplicas de leitura reduziu a latência p99 de aproximadamente 1 200 ms para 400 ms. Essa diminuição está relacionada à menor contenção no servidor de escrita (master), uma vez que as requisições de leitura puderam ser distribuídas entre as réplicas, aliviando a carga do banco primário. Apesar de o modelo de replicação implicar em consistência eventual, conforme descreve Kleppmann (2017), os benefícios superam os riscos em cenários onde a atualização em tempo real não é crítica.
A combinação dessas duas estratégias mos-

trou-se essencial para estabilizar picos de acesso e garantir uma experiência consistente em condições de alta concorrência. Esses resultados reforçam a importância de decisões arquiteturais incrementais e orientadas por métricas, alinhadas às melhores práticas de escalabilidade em sistemas distribuídos.

Análise Integrada de Disponibilidade, Processamento Assíncrono e Observabilidade

Os resultados obtidos demonstram que a adoção combinada de replicação multirregional, processamento assíncrono e práticas de observabilidade fortalece a resiliência e o desempenho do sistema em cenários de alta escala.

A replicação multirregional em configuração active-active eliminou o ponto único de falha em nível geográfico. Em simulações de desastre, o mecanismo de DNS failover redirecionou 100 % do tráfego em menos de 60 s, permitindo que o SLA global se aproximasse de 99,99 % (Netflix, 2020; Google, 2016). Embora essa estratégia envolva maior complexidade operacional e custos de comunicação entre regiões, ela garante disponibilidade superior e menor latência para usuários distribuídos geograficamente.

Em paralelo, o uso de mensageria para processamento assíncrono — com tecnologias como Kafka, RabbitMQ ou Amazon SQS — retirou tarefas pesadas do fluxo síncrono de requisições, reduzindo o tempo médio de resposta percebido para menos de 1 s e elevando o throughput. A implementação de dead-letter queues (DLQ), idempotência e auto scaling de consumidores resultou em redução de aproximadamente 70 % do backlog máximo de mensagens (Buddha; Beesetty, 2020).

Complementarmente, a instrumentação de logs, métricas e tracing centralizados consolidou a cultura de Engenharia de Confiabilidade de Sites (SRE). Essa abordagem reduziu o Mean Time to Detect (MTTD) de cerca de 40 min para aproximadamente 5 min e o Mean Time to Recovery (MTTR) de cerca de 2 h para 25 min, apoiada por runbooks e post-mortems que padronizaram a resposta a incidentes (Google, 2016).

De forma geral, a escalabilidade resulta de decisões incrementais e coordenadas. Cada mecanismo acrescenta robustez e, simultaneamente, novos riscos que exigem mitigação cuidadosa. O equilíbrio entre desempenho, disponibilidade e custo — orientado por Service Level Objectives (SLOs) e métricas-gatilho — deve guiar a evolução arquitetural para que o sistema mantenha alta confiabilidade mesmo em condições de tráfego extremo.

Considerações Finais

O estudo demonstrou que a escalabilidade de aplicações web não depende de uma única solução tecnológica, mas do encadeamento progressivo de decisões arquiteturais baseadas em métricas. A aplicação do roteiro proposto — da arquitetura monolítica inicial até a configuração distribuída e observável — evidenciou que cada estágio agrega robustez e, ao mesmo tempo, introduz novos riscos que exigem mitigação contínua. A introdução de cache e de réplicas de leitura reduziu significativamente a latência p95 e p99, comprovando a eficácia de estratégias de alívio da carga do banco de dados. A replicação multirregional em modo active-active e o DNS failover eliminaram pontos únicos de falha e elevaram o SLA global para patamares próximos de 99,99 %, garantindo alta disponibilidade mesmo em cenários de desastre. O uso de mensageria para processamento assíncrono diminuiu o tempo médio de resposta percebido para menos de 1 segundo, aumentando o throughput e a resiliência do sistema. Complementarmente, a adoção de práticas de observabilidade e da cultura SRE reduziu drasticamente o MTTD e o MTTR, permitindo respostas rápidas e consistentes a incidentes. Esses resultados reforçam que escalabilidade, disponibilidade e eficiência operacional surgem de decisões incrementais guiadas por SLOs e métricas-gatilho, e não de mudanças pontuais. O trabalho contribui, assim, para a literatura e para a prática profissional ao oferecer um roteiro replicável para a evolução de sistemas de pequeno porte até aplicações capazes de atender a milhões de usuários simultâneos. Como pers-

pectivas futuras, recomenda-se explorar sharding de dados, estratégias de chaos engineering e arquiteturas totalmente orientadas a eventos, a fim de ampliar a tolerância a falhas e validar a robustez em ambientes de produção de larga escala.

Palavras-chave

Escalabilidade; Arquitetura de software; Balanceamento de carga; Mensageria; Design de sistema.

Referências

AMAZON WEB SERVICES. Amazon EC2 Auto Scaling – Documentation. 2023. Disponível em: https://docs.aws.amazon.com/autoscaling/. Acesso em: 06 out. 2025. ARMSTRONG, Jeff. Migrando para a AWS: Um guia para gerentes. São Paulo: Brasport, 2020. Disponível em: https://www.brasport.com.br/. Acesso em: 06 out. 2025.

BUDDHA, Jyothi Prasad; BEESETTY, Reshma. The Definitive Guide to AWS Application Integration. Apress, 2020. Disponível em: https://link.springer.com/book/10.1007/978-1-4842-5939-2. Acesso em: 06 out. 2025.

GOOGLE. Site Reliability Engineering – SRE Book. O'Reilly, 2016. Disponível em: https://sre.google/books/. Acesso em: 06 out. 2025.. KLEPPMANN, Martin. Designing Data-Intensive Applications. O'Reilly, 2017. Disponível em: https://dataintensive.net/. Acesso em: 06 out. 2025.

LAM, Sahn; XU, Alex. System Design Interview – An Insider's Guide: Volume 2. Byte-ByteGo, 2022. Disponível em: https://bytebytego.com/. Acesso em: 06 out. 2025. NETFLIX. Active-Active for Multi-Regional Resiliency. Netflix Tech Blog, 2020. Disponível em: https://netflixtechblog.com/. Acesso em: 06 out. 2025.

NGINX INC. HTTP Load Balancing Documentation. 2022. Disponível em: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/. Acesso em: 06 out. 2025.

NEWMAN, Sam. Criando microsserviços. 2. ed. Porto Alegre: Bookman/Novatec, 2022. Disponível em: https://novatec.com.br/. Acesso em: 06 out. 2025.

NEMAN, Sam . Migrando sistemas monolíticos para microsserviços. Porto Alegre: Bookman/Novatec, 2020. Disponível em: https://novatec.com.br/. Acesso em: 06 out. 2025. REDIS LABS. Caching Patterns and Best Practices. 2022. Disponível em: https://redis.io/docs/latest/develop/use/patterns/. Acesso em: 06 out. 2025.

RICHARDS, Mark; FORD, Neal. Fundamentos da arquitetura de software: uma abordagem de engenharia. São Paulo: Bookman, 2024. Disponível em: https://www.bookman.com.br/. Acesso em: 06 out. 2025. XU, Alex. System Design Interview – An Insider's Guide. Byte-ByteGo, 2020. Disponível em: https://bytebytego.com/. Acesso em: 06 out. 2025.