UNIVERSIDADE FEDERAL DO PAMPA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

MICHEL ALMEIDA DA SILVA

ESTUDO DE CASO DE UMA ARQUITETURA DE STREAMING DE VÍDEO PARAMETRIZÁVEL: ANÁLISE DE TRADE-OFFS ENTRE DESEMPENHO E QUALIDADE DE EXPERIÊNCIA

Bagé
2026

MICHEL ALMEIDA DA SILVA

ESTUDO DE CASO DE UMA ARQUITETURA DE STREAMING DE VÍDEO PARAMETRIZÁVEL: ANÁLISE DE TRADE-OFFS ENTRE DESEMPENHO E QUALIDADE DE EXPERIÊNCIA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Leonardo Bidese de Pinho

Bagé
2026

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

S586e Almeida da Silva, Michel

ESTUDO DE CASO DE UMA ARQUITETURA DE STREAMING DE VÍDEO PARAMETRIZÁVEL: ANÁLISE DE TRADE-OFFS ENTRE DESEMPENHO E QUALIDADE DE EXPERIÊNCIA / Michel Almeida da Silva.

- 2026.

136 f.: il.

Orientador: Leonardo Bidese de Pinho

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Campus Bagé, Bacharelado em Engenharia de Computação, 2026.

I. Leonardo Bidese de Pinho. II. Título.

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
Universidade Federal do Pampa

MICHEL ALMEIDA DA SILVA

ESTUDO DE CASO DE UMA ARQUITETURA DE STREAMING DE VÍDEO PARAMETRIZÁVEL: ANÁLISE DE TRADE-OFFS ENTRE DESEMPENHO E QUALIDADE DE EXPERIÊNCIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 06 de Julho de 2026.

Banca examinadora:

Prof. Dr. Leonardo Bidese de Pinho
Orientador
(UNIPAMPA)

Prof. Dr. Bruno Silveira Neves
(UNIPAMPA)

Prof. Dr. Diego Leonel Cadette Dutra
(UFRJ)

Assinado eletronicamente por **LEONARDO BIDESE DE PINHO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 13/07/2026, às 09:15, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Diego Leonel Cadette Dutra**, **Usuário Externo**, em 13/07/2026, às 17:18, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **BRUNO SILVEIRA NEVES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 13/07/2026, às 22:17, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2085548** e o código CRC **6FB6579F**.

Referência: Processo nº 23100.010883/2026-81 SEI nº 2085548

Dedico este trabalho primeiramente à memória de minha mãe, minha eterna inspiração e apoio. Ao meu pai, pelo exemplo de vida. Aos meus amigos, familiares e à minha namorada, que caminharam ao meu lado durante esta jornada acadêmica.

AGRADECIMENTOS

A realização deste trabalho não seria possível sem a presença e o incentivo de pessoas especiais.

Agradeço de coração à minha família, em especial à minha irmã Angelica, ao Ressier, à Diene e à minha tia Terezinha e a Nilza. O apoio de vocês foi essencial para que eu chegasse até aqui.

Um agradecimento especial à minha namorada, Amália. Obrigado por ser meu porto seguro, pela motivação diária e por todo o apoio inestimável durante a elaboração deste TCC.

Ao meu orientador, Prof. Dr. Leonardo Bidese de Pinho, minha sincera gratidão. Obrigado por todo o suporte ao longo da minha trajetória acadêmica e pela orientação precisa que possibilitou o desenvolvimento deste estudo.

Por fim, estendo meus agradecimentos aos meus colegas de faculdade, que tornaram essa caminhada mais leve e enriquecedora através da amizade e do aprendizado compartilhado.

RESUMO

O consumo de vídeo domina o tráfego global da Internet, impondo desafios significativos à eficiência das arquiteturas de *streaming*. Diante da complexidade e da carência de ferramentas abertas para análise, este trabalho desenvolveu e avaliou experimentalmente uma arquitetura de *streaming* de vídeo parametrizável, baseada em tecnologias de código aberto, concebida primeiramente como plataforma industrial de VoD e, por sua instrumentação, empregada na investigação dos *trade-offs* entre desempenho, custo de operação e qualidade, tanto de serviço (QoS) quanto de experiência (QoE), do usuário. A metodologia adotada é mista, dividida em duas fases complementares de um mesmo trabalho: uma exploratória, que conduziu uma Revisão Sistemática da Literatura sobre 19 artigos de alta relevância e fundamentou a proposição do modelo arquitetural; e uma descritiva-explicativa (a fase de execução), na qual a arquitetura foi implementada como uma plataforma de microsserviços e submetida a um estudo de caso instrumental. Para viabilizar a investigação no tempo disponível, adotou-se a estratégia de medição em escala reduzida e projeção: o custo unitário de operação (*footprint* por *viewer*, por requisição e por codificação) foi medido sobre a implementação real e instrumentada, resultando em constantes empíricas que foram usadas como base para projetar o comportamento em larga escala, com custo e pontos de saturação. O estudo de caso decompôs-se em dois experimentos complementares: a avaliação de capacidade e QoE da distribuição por meio de *players* reais, e a análise de eficiência de *codecs* por *rate-distortion* (BD-rate). Os resultados confirmam que cada estágio do *pipeline* é regido por um gargalo próprio: a concorrência e o tráfego de saída (*egress*) na distribuição, e o equilíbrio entre eficiência de compressão e velocidade na codificação. Esses pontos de saturação podem ser antecipados a partir das constantes medidas em escala reduzida. Como principal contribuição, o trabalho disponibiliza uma plataforma aberta, parametrizável e instrumentada como alternativa às soluções comerciais de caixa-preta, acompanhada de um método reprodutível de avaliação ancorado em medição e de evidência empírica dos *trade-offs* do *pipeline* de *streaming*.

Palavras-chave: *Streaming* de Vídeo; Qualidade de Experiência; Análise de Desempenho; Modelo Arquitetural; *Trade-offs* de *Streaming*; *Codecs* de Vídeo; Protocolos de *Streaming*.

ABSTRACT

Video consumption dominates global Internet traffic, imposing significant challenges to the efficiency of streaming architectures. Given the complexity and scarcity of open tools for analysis, this work developed and experimentally evaluated a parameterizable video streaming architecture, based on open-source technologies, conceived primarily as an industrial VoD platform and, through its instrumentation, employed to investigate the trade-offs between performance, operating cost, and quality, both of service (QoS) and of experience (QoE), for the user. The methodology adopted is mixed-method, divided into two complementary phases of a single work: an exploratory phase, which conducted a Systematic Literature Review over 19 highly relevant articles and grounded the proposal of the architectural model; and a descriptive-explanatory execution phase, in which the architecture was implemented as a microservices platform and subjected to an instrumental case study. To make the investigation feasible within the available time, a reduced-scale measurement and projection strategy was adopted: the unit cost of operation (footprint per viewer, per request, and per encode) was measured over the real, instrumented implementation, resulting in empirical constants that were used as a basis to project the large-scale behavior, including cost and saturation points. The case study was decomposed into two complementary experiments: an evaluation of distribution capacity and QoE using real players, and an analysis of codec efficiency through rate-distortion (BD-rate). The results confirm that each stage of the pipeline is governed by its own bottleneck: concurrency and egress in distribution, and the balance between compression efficiency and encoding speed. These saturation points can be anticipated from the constants measured at reduced scale. As its main contribution, this work provides an open, parameterizable, and instrumented platform as an alternative to commercial black-box solutions, together with a reproducible measurement-grounded evaluation method and empirical evidence of the streaming pipeline trade-offs.

Keywords: Video Streaming; Quality of Experience; Performance Analysis; Architectural Model; Streaming Trade-offs; Video Codecs; Streaming Protocols.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABR	*Adaptive Bitrate* (Algoritmo de Adaptação de Taxa de Bits)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
AV1	AOMedia Video 1
AVC	*Advanced Video Coding* (Codificação de Vídeo Avançada)
AWS	*Amazon Web Services*
CBR	*Constant Bitrate* (Taxa de Bits Constante)
CDN	*Content Delivery Network* (Rede de Distribuição de Conteúdo)
CMAF	*Common Media Application Format* (Formato Comum de Aplicação de Mídia)
CORS	*Cross-Origin Resource Sharing* (Compartilhamento de Recursos entre Origens)
CQ	Critério de Qualidade
CTE	*Chunked Transfer Encoding* (Codificação de Transferência em Fragmentos)
DASH	*Dynamic Adaptive Streaming over HTTP* (Streaming Adaptativo Dinâmico sobre HTTP)
DASH-IF	DASH Industry Forum
DNE	*DASH Network Elements* (Elementos de Rede DASH)
DRM	*Digital Rights Management* (Gerenciamento de Direitos Digitais)
E2E	*End-to-End* (Ponta a Ponta)
GGC	*Google Global Cache*
GOP	*Group of Pictures* (Grupo de Imagens)
GSLB	*Global Server Load Balancing* (Balanceamento de Carga Global de Servidores)
HAS	*HTTP Adaptive Streaming* (Streaming Adaptativo HTTP)

HDR	*High Dynamic Range* (Alta Faixa Dinâmica)
HESP	*High Efficiency Streaming Protocol* (Protocolo de Streaming de Alta Eficiência)
HEVC	*High Efficiency Video Coding* (Codificação de Vídeo de Alta Eficiência)
HLS	*HTTP Live Streaming* (Transmissão ao Vivo HTTP)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HTTPS	*Hypertext Transfer Protocol Secure* (HTTP Seguro)
MOQ	*Media over QUIC* (Mídia sobre QUIC)
MOS	*Mean Opinion Score* (Pontuação Média de Opinião)
MPEG	*Moving Picture Experts Group*
MPEG-DASH	*Moving Picture Experts Group - Dynamic Adaptive Streaming over HTTP*
MPEG-SAND	*Server and Network-Assisted DASH*
MPD	*Media Presentation Description* (Descrição de Apresentação de Mídia)
MSE	*Media Source Extensions* (Extensões de Fonte de Mídia)
NAT	*Network Address Translation* (Tradução de Endereços de Rede)
NFV	*Network Function Virtualization* (Virtualização de Funções de Rede)
PoP	*Point of Presence* (Ponto de Presença)
QoE	*Quality of Experience* (Qualidade de Experiência)
QoS	*Quality of Service* (Qualidade de Serviço)
QP	Questão de Pesquisa
QUIC	*Quick UDP Internet Connections* (Conexões Rápidas de Internet UDP)
RFC	*Request for Comments* (Pedido de Comentários)
RL	*Reinforcement Learning* (Aprendizado por Reforço)
RSL	Revisão Sistemática da Literatura
RTMP	*Real-Time Messaging Protocol* (Protocolo de Mensagens em Tempo Real)

RTT	*Round-Trip Time* (Tempo de Ida e Volta)
RUM	*Real User Monitoring* (Monitoramento de Usuário Real)
SAND	*Server and Network-Assisted DASH*
SDN	*Software-Defined Networking* (Redes Definidas por Software)
SRT	*Secure Reliable Transport* (Transporte Confiável Seguro)
SSL	*Secure Sockets Layer* (Camada de Soquetes Seguros)
TCC	Trabalho de Conclusão de Curso
TCP	*Transmission Control Protocol* (Protocolo de Controle de Transmissão)
TLS	*Transport Layer Security* (Segurança da Camada de Transporte)
UDP	*User Datagram Protocol* (Protocolo de Datagrama de Usuário)
VBR	*Variable Bitrate* (Taxa de Bits Variável)
VoD	*Video on Demand* (Vídeo sob Demanda)
VP9	*codec* de vídeo aberto desenvolvido pelo Google
VVC	*Versatile Video Coding* (Codificação de Vídeo Versátil)
WebRTC	*Web Real-Time Communication* (Comunicação em Tempo Real na Web)
WHIP	*WebRTC-HTTP Ingestion Protocol* (Protocolo de Ingestão WebRTC-HTTP)
Wi-Fi	*Wireless Fidelity* (Fidelidade sem Fio)

SUMÁRIO

1 INTRODUÇÃO

O consumo de vídeo constitui, atualmente, a parcela majoritária do tráfego global de Internet, com projeções que indicam um crescimento contínuo e acentuado para os próximos anos (ERICSSON, 2024). Esse crescimento pressiona a eficiência das arquiteturas de *streaming*, especialmente nos *trade-offs* entre qualidade de transmissão, consumo de recursos e latência.

Conforme destacado por Laghari *et al.* (2023) em revisão sistemática sobre o estado da arte em *streaming* de vídeo, a área enfrenta obstáculos complexos relacionados à largura de banda, custos de infraestrutura e heterogeneidade de dispositivos, especialmente na reprodução de conteúdos de alta resolução. Embora se observem avanços notáveis em *codecs* e protocolos, persiste uma lacuna técnica na integração desses componentes. Nota-se, sobretudo, a ausência de uma plataforma aberta, instrumentada e parametrizável, que cubra todo o *pipeline* de vídeo e permita medir, de forma reprodutível, os *trade-offs* entre desempenho, custo e qualidade sob carga. Medir esses *trade-offs* importa porque são eles que dominam o custo e a experiência do serviço: a escolha do *codec* determina a banda e o armazenamento consumidos por título; o dimensionamento da camada de distribuição determina quantos espectadores simultâneos podem ser atendidos e a que custo; e variações de latência e de *rebuffering* degradam diretamente a qualidade percebida pelo espectador. Sem medições reprodutíveis, essas decisões de engenharia são tomadas às cegas, superdimensionando recursos ou comprometendo a experiência, em um cenário cuja escala, quantificada no Capítulo 4, não perdoa erros de dimensionamento. Para operar um serviço de vídeo nessa escala, a alternativa usual a construir uma arquitetura própria é contratar integralmente um serviço gerenciado de terceiros. Essa contratação resolve a distribuição, mas não a investigação: tais soluções proprietárias, do tipo "caixa-preta", cujos mecanismos internos não são observáveis nem ajustáveis pelo pesquisador, não permitem medir os *trade-offs* recém-descritos. Uma plataforma aberta, em contraste, torna a operação em alta escala simultaneamente viável e investigável.

Este trabalho ocupa esse espaço: desenvolve, instrumenta e avalia uma plataforma aberta que cobre o *pipeline* completo de vídeo sob condições controladas. A justificativa para esta pesquisa fundamenta-se na necessidade de superar a barreira de implementação prática das arquiteturas de alta escala. Propõe-se, portanto, o desenvolvimento

de uma plataforma parametrizável que sistematize esses conceitos teóricos em uma implementação funcional e instrumentada.

O trabalho busca conceber tal plataforma primeiramente como uma plataforma industrial: um ambiente de VoD funcional, parametrizável e instrumentado, capaz de operar o caso de uso concreto e de calibrar e dimensionar uma instalação antes de sua entrada em produção (*setup* do sistema), com o modelo de custo e capacidade (Capítulo 6) traduzindo as constantes medidas em decisões de dimensionamento. A mesma instrumentação permite que a plataforma evolua para o uso de pesquisa, em que as medições respondem a questões de conhecimento geral (qual *codec*, qual arquitetura, a que custo); é esse uso que este trabalho exercita nos experimentos, ao observar e quantificar, sob condições controladas, os *trade-offs* entre desempenho, custo e qualidade. O foco está, portanto, na medição reprodutível desses *trade-offs*, e não na otimização de uma métrica específica, que se coloca como desdobramento futuro. A principal contribuição pretendida é dupla: a proposta de uma metodologia de desenvolvimento e avaliação, e a geração de conhecimento explicativo sobre o comportamento do *pipeline* de *streaming* sob condições controladas.

1.1 Problema de pesquisa

O problema central desta investigação reside na complexidade de orquestração de uma arquitetura de *streaming* de Vídeo sob Demanda (VoD) que atenda a requisitos de disponibilidade e escalabilidade, mas em um ambiente controlado. Especificamente, questiona-se: como medir, em escala reduzida e instrumentada, os *trade-offs* entre desempenho, custo e qualidade nas etapas críticas do *pipeline* de VoD (a distribuição dos segmentos e a codificação do vídeo), de modo a projetar o comportamento da arquitetura em escala de produção?

Para tornar essa questão respondível dentro do escopo de um trabalho de conclusão de curso, ela é decomposta em quatro questões operacionais, investigadas por dois estudos de caso específicos (Seção 6.8). Na distribuição (Estudo 1): (i) qual é a capacidade de entrega da arquitetura, em espectadores concorrentes, e com que QoE percebida pelo *player*? (ii) quanto custa operá-la em escala de nuvem? Na codificação (Estudo 2): (iii) qual *codec* preserva a mesma qualidade perceptual com a menor taxa de bits? (iv) a que custo, em velocidade de codificação, essa economia de bits é obtida?

1.2 Objetivos

O objetivo geral do trabalho é conceber, fundamentar e propor uma arquitetura de *streaming* de vídeo, utilizando tecnologias de código aberto, para constituir primeiramente uma plataforma industrial de VoD que, por ser instrumentada e parametrizável, sustenta a investigação experimental dos *trade-offs* entre desempenho, consumo de recursos computacionais, custo de operação em nuvem e Qualidade de Serviço (QoS); a Qualidade de Experiência (QoE) é abordada do lado do *player* como *proxy*, e a percepção em campo é reconhecida como limitação. Dois recortes tornam esse objetivo exequível no escopo de um trabalho de conclusão de curso: embora a arquitetura seja projetada de ponta a ponta, da origem (*storage/origin server*) à borda (*edge*/CDN), a investigação experimental concentra-se nas duas etapas críticas do *pipeline*, a distribuição dos segmentos e a codificação do vídeo, guiada pelas quatro questões operacionais da seção anterior; e as medições são realizadas em escala reduzida e instrumentada, projetando-se o comportamento em escala de produção a partir das constantes empíricas medidas. A partir deste, destacam-se os objetivos específicos da fase exploratória e da fase de execução.

Fase 1 — Abordagem Exploratória:

- Realizar uma Revisão Sistemática da Literatura (RSL) sobre os fundamentos de arquiteturas de *streaming*, abrangendo o estado da arte em protocolos de transporte adaptativo, *codecs* de vídeo modernos e infraestruturas de *Content Delivery Network* (CDN).
- Analisar comparativamente as tecnologias e ferramentas *open-source* disponíveis para cada etapa do *pipeline* de vídeo (ingestão, transcodificação, empacotamento e reprodução), selecionando a *stack* tecnológica mais adequada à solução proposta;
- Definir os requisitos funcionais e não funcionais da arquitetura, estabelecendo métricas-alvo para suportar escalabilidade e disponibilidade;
- Projetar o modelo arquitetural da solução de *streaming* VOD, detalhando o fluxo de dados desde a origem (*storage/origin server*) até a borda (*edge*/CDN), incluindo estratégias de *cache* e balanceamento de carga.

Fase 2 — Abordagem Descritiva e Explicativa (Estudo de Caso):

- Desenvolver e integrar os módulos de ingestão, transcodificação, empacotamento, distribuição, monitoramento, reprodução e containerização/orquestração;
- Elaborar a modelagem de custos e capacidade, estimando os recursos computacionais (CPU, memória e largura de banda) necessários para processar vídeos de características definidas sob a carga estipulada;
- Conceber os cenários de teste e o plano de validação experimental, definindo as variáveis independentes (*codecs*, *bitrates*, latência de rede) e as métricas de QoS e de QoE a serem coletadas;
- Instrumentar um cliente de reprodução web para a coleta e reporte de métricas de QoE, tais como tempo para início da reprodução (*start-up time*), taxa de *buffering* e oscilações na qualidade do vídeo;
- Executar o estudo de caso submetendo o protótipo a diferentes condições operacionais, variando parâmetros como: *codec*, *bitrate*, protocolo de *streaming*, plataforma de distribuição e resolução de vídeo;
- Coletar e tabular as métricas de QoS (desempenho do servidor) e de QoE (cliente) para cada configuração testada;
- Analisar os resultados para descrever o comportamento do protótipo e explicar as relações de causa e efeito entre os parâmetros configurados e as métricas observadas, identificando os *trade-offs* mais relevantes para a engenharia da solução.

1.3 Metodologia

Os fundamentos metodológicos propostos por Prodanov e Freitas (2013) embasaram o percurso adotado para o desenvolvimento da pesquisa. A estrutura metodológica foi concebida para garantir o rigor científico, fundamentando a classificação do estudo e detalhando as etapas procedimentais que conduziram à consecução dos objetivos estabelecidos.

Quanto à sua natureza, a pesquisa classifica-se como aplicada: objetiva gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos. O estudo foca na investigação experimental de *trade-offs* em arquiteturas de *streaming* de

vídeo, visando preencher a lacuna de ferramentas abertas para análise de desempenho e QoE.

Em relação aos objetivos, o trabalho adota uma abordagem mista, combinando características exploratórias, explicativas e descritivas, organizada em duas fases complementares:

- Na fase exploratória, a pesquisa teve natureza predominantemente investigativa. O foco recaiu sobre a investigação e o entendimento do domínio do problema, a seleção das tecnologias *open-source* mais adequadas e a proposição de um modelo arquitetural viável, além de proporcionar maior familiaridade com o problema. Esta etapa foi fundamental para a delimitação precisa do escopo, para a fundamentação das decisões arquiteturais e para a definição dos parâmetros da investigação posterior.
- Na fase de execução, a pesquisa evoluiu para um caráter descritivo e explicativo, materializado por um estudo de caso instrumental. A arquitetura definida na primeira fase foi implementada e seu comportamento foi observado, medido e descrito sob diferentes condições operacionais. Buscou-se relatar os resultados (descrição) e explicar as relações de causa e efeito entre as configurações do sistema e as métricas obtidas (explicação). Para tornar essa investigação viável no tempo disponível, adotou-se a estratégia de medição em escala reduzida e projeção: o custo unitário de operação foi medido sobre uma implementação real e instrumentada e, a partir dessas constantes empíricas, projetou-se o comportamento em larga escala.

Quanto aos procedimentos técnicos, a estratégia de pesquisa selecionada é o estudo de caso. Nessa abordagem, o protótipo de arquitetura desenvolvido funciona como o "caso" ou instrumento por meio do qual o fenômeno (o comportamento e os *trade-offs* do *pipeline* de *streaming*) é investigado em profundidade.

1.4 Organização do texto

Além desta introdução, este trabalho está organizado em mais sete capítulos e dois apêndices. O Capítulo 2 apresenta a sistematização e a avaliação qualitativa da literatura sobre arquiteturas de *streaming*, por meio de uma Revisão Sistemática da Literatura que fundamenta as decisões de projeto. O Capítulo 3 reúne o referencial teórico, detalhando

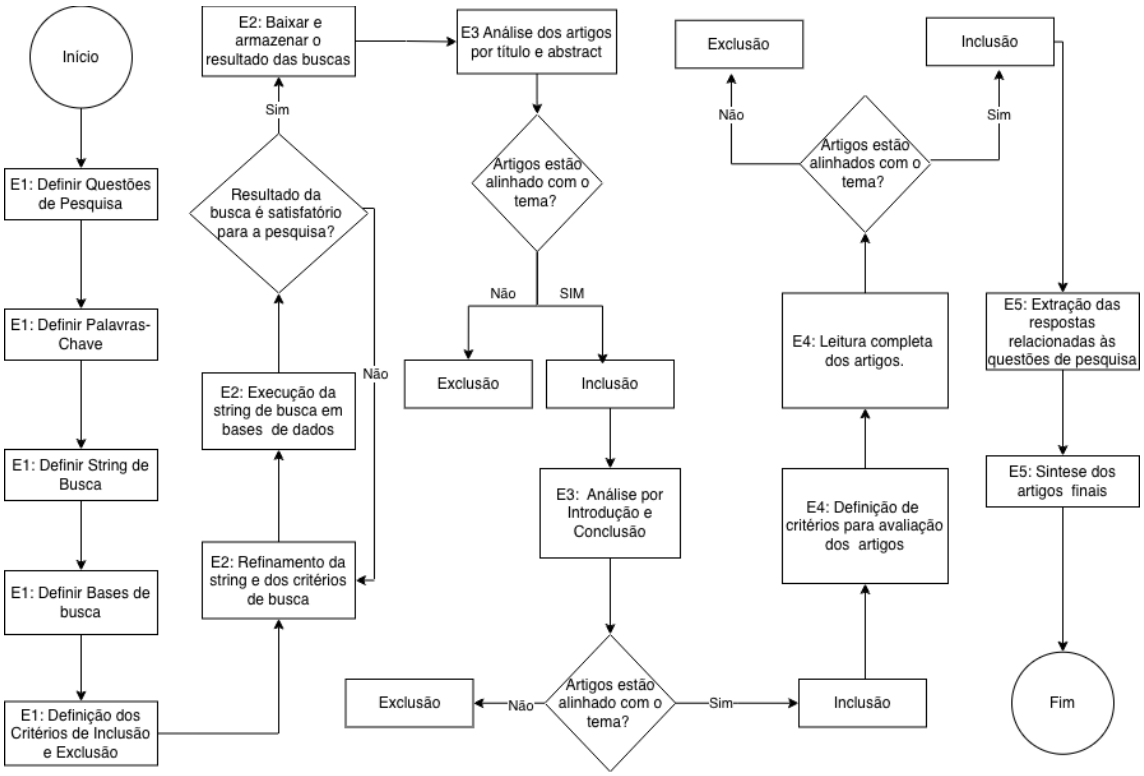
os conceitos de *codecs*, protocolos de transporte adaptativo, métricas de qualidade e infraestrutura de distribuição que embasam a proposta. O Capítulo 4 descreve o estudo de caso que delimita o cenário investigado. O Capítulo 5 apresenta o protótipo arquitetural desenvolvido, com seus módulos e as tecnologias de código aberto empregadas. O Capítulo 6 detalha a metodologia experimental, incluindo o desenho dos experimentos, as variáveis e as métricas coletadas. O Capítulo 7 apresenta e discute os resultados obtidos. Por fim, o Capítulo 8 traz as considerações finais e os trabalhos futuros. Os apêndices complementam o texto: o Apêndice A detalha o teste de carga da distribuição, e o Apêndice B reúne as telas das aplicações desenvolvidas.

2 SISTEMATIZAÇÃO E AVALIAÇÃO QUALITATIVA DA LITERATURA SOBRE ARQUITETURAS DE *STREAMING*

Para o mapeamento do estado da arte, adotou-se o método de RSL. A Revisão Sistemática da Literatura (RSL) é definida como um meio de identificar, avaliar e interpretar toda a pesquisa disponível relevante para uma questão de pesquisa particular, área temática ou fenômeno de interesse (KITCHENHAM, 2012). Diferente de revisões tradicionais, a RSL segue uma metodologia rigorosa e um protocolo predefinido para garantir uma síntese replicável e não enviesada das evidências: os estudos são selecionados e avaliados por critérios explícitos, definidos antes da busca, e não pela preferência do revisor. Este processo é fundamental para estabelecer o estado da arte e identificar lacunas no conhecimento atual sobre arquiteturas de *streaming*, protocolos e métricas de avaliação.

Para o gerenciamento das referências bibliográficas, utilizou-se a ferramenta JabRef. Este *software open-source*, baseado no formato BibTeX, permitiu a organização sistemática dos artigos, incluindo metadados essenciais como título, autores, ano de publicação e resumo. A ferramenta facilitou a classificação dos estudos, a detecção de duplicatas e a exportação das referências no formato BibTeX, garantindo a integridade e a padronização da base de dados da revisão.

Figura 1 – Fluxograma de RSL



Fonte: Elaborado pelo autor.

O fluxograma apresenta uma única realimentação explícita, na calibração da busca (etapa E2): enquanto o resultado não é satisfatório, a *string* e os critérios de busca são refinados. A concentração da iteração nessa fase inicial é intencional: é nela que o protocolo é calibrado; das etapas de seleção em diante, os critérios de inclusão e exclusão são aplicados sem reajuste, preservando a síntese não enviesada. Houve, contudo, um segundo momento de refinamento, posterior e não representado no fluxo: os critérios específicos do *checklist* de avaliação de qualidade foram lapidados após a leitura preliminar de títulos e resumos, antes de sua aplicação aos textos completos, como descrito na Etapa 3.

O processo foi fundamentado no guia prático para a área de Ciência da Computação desenvolvido por Neiva e Silva (2016). A execução de uma RSL bem-sucedida depende da criação de um protocolo de revisão, um documento que detalha previamente todos os passos a serem seguidos, garantindo a transparência e a mitigação de vieses. O fluxo de trabalho seguido foi dividido nas seguintes etapas:

Etapa 1: Planejamento e Estratégia de Busca

Esta etapa definiu o protocolo de pesquisa que guiou todo o processo: o escopo da revisão, a estratégia de busca e os critérios de seleção que asseguram a reprodutibilidade dos resultados.

Definição das Questões de Pesquisa (QP): A formulação das Questões de Pesquisa é o ponto de partida da RSL. Elas delimitam o escopo da investigação e orientam a extração das palavras-chave e a definição dos critérios de seleção. As perguntas foram elaboradas para cobrir os aspectos técnicos e metodológicos das arquiteturas de *streaming*.

As seguintes QP foram definidas para nortear o estudo:

- **QP1:** Quais são as arquiteturas, protocolos e *codecs* predominantes no estado da arte de sistemas de *streaming* de vídeo?
- **QP2:** Quais métricas e metodologias são utilizadas para avaliar o desempenho e a QoE nessas arquiteturas?
- **QP3:** Quais ferramentas e tecnologias *open-source* estão disponíveis para a implementação e experimentação de arquiteturas de *streaming*?

Ressalta-se que, embora os critérios gerais tenham sido definidos a priori, critérios específicos de avaliação da qualidade foram refinados após a leitura preliminar dos títulos e resumos, permitindo uma filtragem mais alinhada com a realidade da literatura encontrada.

Definição das Palavras-Chave: Com base nas questões, realizou-se a extração de termos e sinônimos. Este processo envolveu identificar os conceitos centrais em cada questão e buscar variações, acrônimos e termos alternativos na literatura para garantir que a busca fosse abrangente e não omitisse estudos relevantes.

Definição da String de Busca: As palavras-chave foram combinadas de forma estruturada utilizando operadores booleanos. Tipicamente, o operador ‘OR‘ foi usado para agrupar sinônimos e termos alternativos, enquanto o operador ‘AND‘ foi usado para conectar os diferentes blocos de conceitos, garantindo que os resultados contivessem todos os aspectos de interesse da pesquisa.

Definição das Bases de Busca: A seleção de bases de dados acadêmicas foi criteriosa e justificada, priorizando aquelas de maior relevância para a área de Ciência da Computação, como Google Scholar, IEEE Xplore, ACM Digital Library, SpringerLink e

ScienceDirect (Elsevier). A inclusão de múltiplas bases aumentou a cobertura da busca e a robustez da revisão.

Definição dos Critérios de Inclusão e Exclusão: Critérios objetivos, derivados diretamente das questões de pesquisa, foram definidos para decidir quais artigos seriam selecionados. Estes critérios incluíram restrições como período de publicação, idioma, ou a exclusão de tipos de estudo específicos (e.g., revisões secundárias) para evitar a sobreposição de dados. Os critérios de inclusão definidos foram: (i) artigos publicados nos últimos 3 anos (2022-2025); (ii) escritos em língua inglesa ou portuguesa; (iii) disponíveis na íntegra; e (iv) que abordassem diretamente as questões de pesquisa.

Etapa 2: Execução e Refinamento da Busca

Esta etapa consistiu na aplicação prática da estratégia de busca definida no protocolo. O processo foi iterativo e incremental, permitindo o refinamento da string de busca e dos critérios de seleção à medida que os resultados preliminares eram analisados. O objetivo foi calibrar a sensibilidade e a precisão da busca para maximizar a recuperação de estudos relevantes. O processo resultante consistiu em seis iterações, com as seguintes características e resultados.

Iteração 1: Busca Ampla.

Na primeira iteração, construiu-se uma string de busca abrangente, incorporando termos relacionados a todos os eixos centrais da pesquisa: *streaming*, *codecs*, protocolos, QoE, arquitetura e CDN. A intenção inicial foi capturar o maior espectro possível de trabalhos.

```
String 1:
("Video on Demand" OR "Streaming media" OR
"Delivers video content"  OR "Streaming")
AND ("H.264" OR "H.265" OR "VVC" OR "VP9" OR "AV1")
AND ("RTMP" OR "HLS" OR "DASH" OR "WebRTC" OR "HAS")
AND ("Quality of Experience" OR "Quality of service")
AND ("Architecture" OR "System Architecture" OR "System design")
AND ("Content Delivery Network" OR "CDN" OR "CDNS")
```

Resultado: Aproximadamente 654 artigos. A quantidade excessiva de resultados indicou que a string era muito genérica e precisava de maior especificidade.

Iteração 2: Refinamento de Termos.

Visando aumentar a precisão dos resultados, a segunda iteração refinou os termos de busca. Foram incluídos descritores mais específicos, como *Live Streaming* e QoE, reconhecendo sua ampla utilização na literatura técnica.

```
String 2:
("Video on Demand "OR "Streaming" OR "Live Streaming")
AND ("H.264" OR "H.265" OR "VVC" OR "VP9" OR "AV1" OR "RV40")
AND ("RTMP" OR "HLS" OR "DASH" OR "WEBRTC")
AND ("Architecture" OR "System Architecture" OR "System design")
AND ("Content Delivery Network" OR "CDN" OR "CDNS")
AND ("QUALITY OF EXPERIENCE" OR "QOE")
```

Resultado: Foram escolhidos 222 artigos. A redução foi significativa, mas alguns dos artigos retornados pela busca ainda foram considerados demasiadamente fora do tema.

Iteração 3: Foco em Protocolos Modernos.

Nesta etapa, optou-se pela remoção do protocolo Real-Time Messaging Protocol (RTMP) da string de busca. Esta decisão fundamentou-se no foco do trabalho em arquiteturas de *streaming* modernas baseadas em Hypertext Transfer Protocol (HTTP), dado que o RTMP tem entrado em desuso para entrega final ao usuário.

```
String 3:
("Video on Demand "OR "Streaming" OR "Live Streaming")
AND ("H.264" OR "H.265" OR "VVC" OR "VP9" OR "AV1" OR "RV40")
AND ("HLS" OR "DASH" OR "WEBRTC")
AND ("Architecture" OR "System Architecture" OR "System design")
AND ("Content Delivery Network" OR "CDN" OR "CDNS")
AND ("QUALITY OF EXPERIENCE" OR "QOE")
```

Resultado: Aproximadamente 217 artigos. A alteração resultou em um conjunto de documentos mais alinhado com as tecnologias contemporâneas de distribuição de vídeo, eliminando ruído provocado por estudos focados em tecnologias legadas.

Iteração 4 e 5: Introdução de Novas Tecnologias.

A busca foi reorientada para protocolos emergentes como Warp e o protocolo de transporte Quick UDP Internet Connections (QUIC), o qual é fundamental para o *streaming* de baixa latência, bem como para o Media over QUIC (MOQ). O bloco de *codecs* foi temporariamente removido e depois reavaliado.

```
String 5:
("Video on Demand "OR "Streaming" OR "Live Streaming")
AND ("Warp" OR "MOQ" OR "HLS" OR "DASH")
AND ("Architecture" OR "System Architecture" OR "System design")
AND ("QUIC")
AND ("Content Delivery Network" OR "CDN" OR "CDNS")
AND ("QUALITY OF EXPERIENCE" OR "QOE")
```

Resultado: Aproximadamente 131 artigos. A inclusão explícita do protocolo `QUIC` direcionou a busca para investigações de ponta sobre baixa latência e transporte moderno, elevando a relevância média dos artigos recuperados.

Iteração 6: String Final e Validação.

A string de busca final consolidou os termos que demonstraram maior efetividade na recuperação de estudos relevantes nas etapas anteriores. Esta versão otimizada foi então adaptada para a sintaxe específica de cada base de dados selecionada, garantindo a consistência da busca através das diferentes plataformas.

A execução da busca foi realizada individualmente em cada base de dados, respeitando suas particularidades de sintaxe e limitações de campos de pesquisa. Abaixo, apresentam-se as adaptações da string para o Google Scholar e IEEE Xplore, exemplificando o processo. Observa-se que a sintaxe de busca varia significativamente entre as bases. O Google Scholar exige strings mais simplificadas e possui limitações no uso de operadores booleanos complexos, enquanto a IEEE Xplore permite construções mais robustas com aninhamento de termos. A adaptação foi necessária para manter a equivalência semântica da busca em todas as plataformas.

```
String 6 (Scholar):
("Video on Demand "OR "Streaming")
AND ("MOQ" OR "HLS" OR "WARP" OR "DASH")
AND ("Architecture")
AND ("QUIC")
AND ("CDN")
AND ("QOE")
```

```
String 6 (IEEE):
 AND (HLS OR DASH OR MOQ OR Warp)
 AND (CDN OR "Content Delivery Network")
 AND ("Quality of Experience" OR QoE))
```

Resultado: Após a execução da string final adaptada para cada base de dados (IEEE, ACM, SpringerLink, ScienceDirect) e a remoção de 16 duplicatas, chegou-se a

um total de 150 referências para a fase de seleção (Tabela 1). Este número foi considerado adequado para uma análise aprofundada.

Tabela 1 – Resultados da busca de artigos por base de dados

Base de Dados	**Artigos Encontrados**
Google Scholar	104
IEEE Xplore	5
ACM Digital Library	33
SpringerLink	3
ScienceDirect (Elsevier)	5
Total Bruto	**150**

Fonte: Elaborado pelo autor.

Os resultados de cada busca foram exportados, em formato ‘.bib‘ ou ‘.csv‘, e consolidados em uma ferramenta de gerenciamento de referências chamada JabRef, com o intuito de facilitar a remoção automatizada e manual de duplicatas e leitura dos artigos.

Etapa 3: Seleção e Triagem dos Artigos

Os artigos coletados passaram por um processo de filtragem sistemático e em múltiplas etapas para garantir que apenas os estudos mais relevantes fossem incluídos na análise final.

Triagem Inicial (Título e Resumo): Nesta primeira fase de seleção, os títulos e resumos de todos os artigos recuperados foram examinados. O objetivo foi eliminar estudos claramente irrelevantes ou fora do escopo, classificando-os preliminarmente como ’potencialmente relevante’ ou ’excluído’.

Triagem Secundária (Introdução e Conclusão): Para os artigos aprovados na triagem inicial, realizou-se uma leitura focalizada da Introdução e da Conclusão. Esta etapa permitiu uma avaliação mais precisa da aderência do estudo às Questões de Pesquisa, sem a necessidade de leitura integral imediata, otimizando o processo de seleção.

Etapa 4: Leitura Completa e Avaliação da Qualidade

Os artigos selecionados nas etapas de triagem foram submetidos à leitura integral. Nesta fase, aplicou-se um rigoroso checklist de qualidade para determinar a inclusão final do estudo na síntese. Para quantificar a aderência de cada trabalho aos critérios

de qualidade estabelecidos, utilizou-se uma Escala Likert de 5 pontos (LIKERT, 1932). Esta abordagem metodológica permite uma avaliação mais nuançada do que uma simples classificação binária, atribuindo pontuações baseadas no grau de concordância com as afirmativas de qualidade.

Definição dos Critérios de inclusão e exclusão.

Para formalizar a avaliação da qualidade na Etapa 4, foi desenvolvido um instrumento (Tabela 2) baseado nos critérios extraídos das QP.

Cada artigo foi avaliado em relação a um conjunto de 8 afirmativas, utilizando uma Escala Likert de 5 pontos. Esta abordagem permite uma pontuação mais granular da adesão de cada estudo, em vez de uma avaliação binária (sim/não).

A escala utilizada para pontuar cada Critério de Qualidade (CQ) é definida como:

(1) **Discordo Totalmente** (O artigo não aborda o critério)

(2) **Discordo** (O artigo apenas cita o critério, sem relevância)

(3) **Neutro** (O artigo aborda o critério de forma secundária)

(4) **Concordo** (O artigo aborda o critério de forma clara e relevante)

(5) **Concordo Totalmente** (O critério é um dos focos centrais do artigo)

Tabela 2 – Checklist de Qualidade (CQ) para Avaliação dos Artigos

ID	Critério	Afirmativa de Avaliação (Escala Likert 1-5)
CQ1	Relevância da Arquitetura	O artigo aborda a concepção, componentes (*pipeline*) ou avaliação de arquiteturas de *streaming* de vídeo.
CQ2	Protocolos e *Codecs* (QP1, QP2)	O artigo discute ou compara protocolos de *streaming* e/ou *codecs* de vídeo.
CQ3	Métricas (Desempenho/QoE) (QP2)	O artigo define, coleta ou analisa métricas de Desempenho (e.g., latência) ou de QoE (e.g., *start-up time*, *buffering*).
CQ4	Discussão de *Trade-offs*	O artigo aborda a análise de *trade-offs* entre compressão/qualidade, consumo de recursos e latência (foco central do TCC).
CQ5	Tecnologias *Open-Source* (QP3)	O artigo menciona, utiliza ou compara ferramentas/bibliotecas *open-source* (e.g., FFmpeg) relevantes para a implementação.
CQ6	Parametrização	O artigo trata de arquiteturas que permitem a variação controlada de parâmetros (e.g., *bitrate*, resolução) para investigação sistemática.
CQ7	Distribuição/CDN	O artigo discute aspectos de distribuição de conteúdo, como o uso de CDN.
CQ8	Algoritmos ABR	O artigo discute ou aborda a concepção de algoritmos de taxa de bits adaptável (ABR) que determinam a qualidade da requisição no próximo segmento.

Fonte: Elaborado pelo autor.

A pontuação total de cada artigo foi calculada somando-se os valores atribuídos a cada CQ, resultando em uma pontuação variando de 8 a 40 pontos. Para a seleção final, estabeleceu-se um ponto de corte de 32 pontos (média 4,0 por critério), excluindo-se da síntese os trabalhos que não atingiram este limiar de relevância. Reconhece-se que um limiar fixo pode deixar de fora trabalhos com contribuições pontuais valiosas; duas salvaguardas mitigam esse risco. Primeiro, a soma é compensatória: uma nota baixa em um critério pode ser compensada por notas altas nos demais, pois o corte exige

desempenho médio equivalente a “concordo” (4,0), e não perfeição em todos os quesitos. Segundo, a exclusão restringe-se à síntese sistemática: artigos abaixo do limiar não foram descartados do trabalho e permaneceram elegíveis como fontes complementares do referencial teórico (Capítulo 3), cuja função é conceitual, e não de evidência sistemática.

Etapa 5: Seleção Final, Extração e Síntese dos Dados

Na etapa final, as informações dos artigos selecionados foram extraídas de forma estruturada e consolidadas para responder às questões de pesquisa.

Extração e Síntese de Dados: Na etapa final, as informações dos artigos selecionados foram extraídas de forma estruturada utilizando um formulário padronizado (planilha eletrônica). Este instrumento foi projetado para capturar dados essenciais que respondessem diretamente às Questões de Pesquisa (QP), garantindo a rastreabilidade e a consistência do processo.

As leituras foram organizadas em duas etapas finais de refinamento. Inicialmente, 36 artigos foram selecionados para leitura completa. Deste conjunto, 19 artigos foram mantidos para a extração de dados, sendo que 6 foram excluídos por indisponibilidade de acesso ao texto completo e 11 por não atenderem aos critérios de qualidade mínima após análise detalhada.

A Tabela 3 apresenta os artigos que obtiveram pontuação superior no checklist de qualidade (Total ≥ 32), destacando-se pela relevância e rigor metodológico.

Tabela 3 – Artigos com Pontuação de Qualidade Superior (Total ≥ 32)

Título do Artigo	**Pontuação Total**
A dataset for analyzing *streaming* media performance over HTTP/3 browsers	35
An End-to-End *Pipeline* Perspective on Video *Streaming* in Best-Effort Networks: A Survey and Tutorial	36
Cloud media video encoding: review and challenges	39
Content steering: Leveraging the computing continuum to support adaptive video *streaming*	40
HAS: A Review on Current Advances and Future Challenges	37
Improved Video Broadcasting for multi-homed users in broadband radio networks	32
Is QUIC Quicker with HTTP/3? An Empirical Analysis of Quality of Experience with DASH Video *Streaming*	36
Multimedia *Streaming* in Software-Defined Networking(SDN)/Network Function Virtualization(NFV) and 5G Networks: Machine Learning for Managing Big Data *Streaming*	39
Netflix, Amazon Prime, and YouTube: comparative study of *streaming* infrastructure and strategy	37
Optimizing Video Delivery with Deep Reinforcement Learning (RL) for High Quality of Experience	36
PRIOR: deep reinforced adaptive video *streaming* with attention-based throughput prediction	38
QoE-Driven Adaptive Video *Streaming*: Architectures, Techniques, and Future Research Challenges Toward 6G Networks	36
QoE-driven joint decision-making for multipath adaptive video *streaming*	36
Quality of Experience Assessment for Enhanced Video *Streaming* Services	38
The state of art and review on video *streaming*	35
To DASH, or Not to DASH? Optimal Video *Bitrate* Selection and Edge Network Caching in MEC-Empowered Slice-Enabled Networks	40
Towards enabling cross-layer information sharing to improve today's content delivery systems	32
Video *Streaming* Over QUIC: A Comprehensive Study	40
Toward one-second latency: Evolution of live media *streaming*	40

Fonte: Elaborado pelo autor.

3 REFERENCIAL TEÓRICO

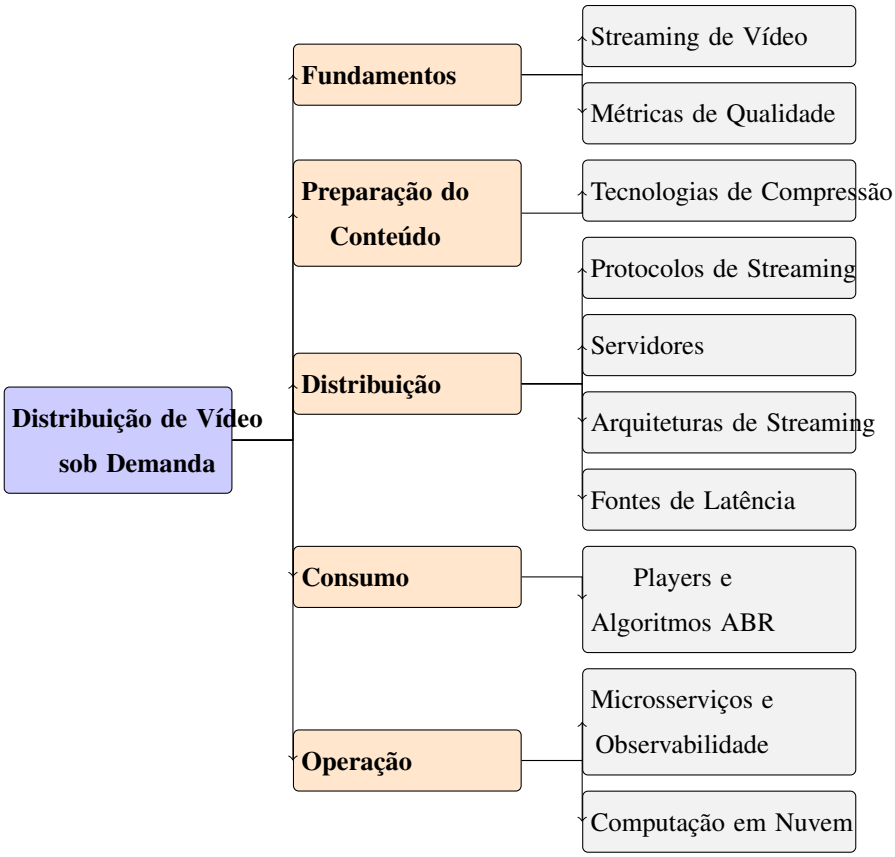
A discussão abrange desde os conceitos fundamentais de *streaming* de vídeo e suas arquiteturas subjacentes até os protocolos de comunicação, tecnologias de compressão (*codecs*) e métricas de avaliação de qualidade. A fundamentação teórica aqui apresentada é alicerçada na RSL conduzida previamente (Capítulo 2), integrando as contribuições mais relevantes e atuais do estado da arte identificadas na metodologia de pesquisa. A RSL, contudo, não foi a única fonte de pesquisa: de forma complementar e não sistemática, recorreu-se a pesquisa bibliográfica dirigida, que abrange documentação técnica e normativa (as RFCs dos protocolos e as especificações de empacotamento e de *streaming* adaptativo), livros e obras de referência metodológica, e relatórios setoriais que fundamentam o estudo de caso (Capítulo 4). A síntese sistemática do Capítulo 2 restringe-se aos estudos selecionados pelo protocolo; este capítulo combina aqueles estudos com essas fontes complementares.

Os tópicos foram organizados hierarquicamente sob o macroassunto da distribuição de vídeo sob demanda, agrupando conceitos do mesmo escopo, da preparação do conteúdo ao seu consumo e à operação da plataforma:

- **Fundamentos:** o conceito de *streaming* de vídeo (VoD e *Live*) e métricas de qualidade (QoE e QoS);
- **Preparação do conteúdo:** tecnologias de compressão (*codecs*) e eficiência energética da codificação;
- **Distribuição:** protocolos de *streaming* (contribuição, ingestão, distribuição e evolução do HTTP), servidores (origem, borda, empacotamento e tecnologias HTTP), CDN e arquiteturas de *streaming*, e fontes de latência End-to-End;
- **Consumo:** *players* e algoritmos de adaptação de taxa de bits (*Adaptive Bitrate*, ABR);
- **Operação:** arquiteturas de microsserviços e observabilidade, somadas à computação em nuvem (escalabilidade, rede e armazenamento), que sustentam a implementação avaliada neste trabalho.

A Figura 2 sintetiza visualmente essa organização, em um organograma que situa cada grupo de tópicos sob o macrotema da distribuição de vídeo sob demanda. As seções seguintes percorrem essa hierarquia, encerrando com a análise dos trabalhos correlatos.

Figura 2 – Organograma hierárquico dos tópicos do referencial teórico

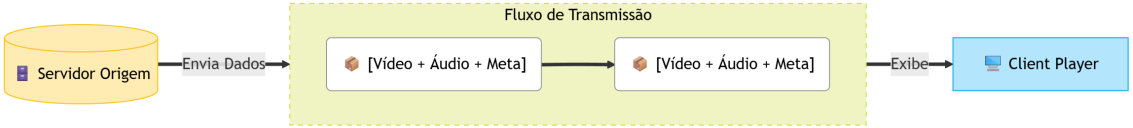


Fonte: Elaborado pelo autor.

3.1 *Streaming* de Vídeo

O *streaming* de vídeo tornou-se o principal meio de consumo de conteúdo multimídia na Internet. Ao contrário do modelo de *download* (que exige a transferência completa do arquivo antes de qualquer reprodução), o *streaming* permite ao usuário assistir enquanto o conteúdo é transmitido. Laghari *et al.* (2023) documentam que a tecnologia se expandiu de plataformas de entretenimento para setores como educação a distância, telemedicina e vigilância, impondo às redes requisitos mais exigentes de capacidade e controle de latência.

Figura 3 – Transmissão de dados entre cliente e servidor



Fonte: Elaborado pelo autor.

Formalmente, o *streaming* de vídeo pode ser definido como a transmissão contínua e sequencial de dados de mídia (incluindo áudio, vídeo e metadados associados) de um servidor de origem para um cliente. A Figura 3 ilustra de forma resumida esse processo de transmissão sequencial de pacotes contendo os diferentes tipos de dados, viabilizando o consumo simultâneo pelo usuário final. Em contraste com o modelo "*download-and-play*", o *streaming* impõe uma restrição temporal: em média, os dados devem chegar ao cliente pelo menos na velocidade em que são consumidos, sob pena de pausas indesejadas para o carregamento de dados (*rebuffering*) (BENTALEB *et al.*, 2026). Essa restrição não implica taxa de transmissão constante: na prática, a taxa instantânea varia bastante, tanto pela codificação em taxa variável (VBR, *Variable Bitrate*), predominante em VoD, quanto pelo transporte em rajadas segmento a segmento, e é o *buffer* do cliente que absorve essas variações. A codificação em taxa constante (CBR, *Constant Bitrate*) permanece relevante sobretudo em transmissões ao vivo, nas quais a folga de *buffer* é pequena e a previsibilidade da taxa importa mais que a eficiência de compressão.

Genericamente o conceito de *Streaming* pode ser categorizado em duas modalidades principais, cada uma regida por requisitos técnicos e operacionais distintos:

- **VoD:** Esta modalidade caracteriza-se pelo acesso sob demanda a conteúdo multimídia previamente gravado e armazenado em servidores ou bibliotecas de conteúdo. O usuário detém controle total sobre a sessão de reprodução, podendo iniciar, pausar, avançar ou retroceder o vídeo a qualquer momento. No VoD, a latência de início não é um requisito estritamente crítico, o que permite aos *players* utilizarem *buffers* de reprodução mais extensos. Essa estratégia de *buffering* ajuda a mitigar variações na largura de banda da rede, garantindo a estabilidade e a alta qualidade da imagem durante a exibição (SUMAN; MOON; CHOI, 2022).
- ***Live Streaming*:** Refere-se à transmissão de eventos capturados e processados em tempo real. A latência (o atraso entre a captura da imagem e sua exibição na tela do espectador) torna-se um requisito crítico e determinante para a QoE. Conforme a classificação estabelecida por Bentaleb *et al.* (2026), o *Live Streaming* depende intrinsecamente da latência para determinar a Qualidade da Experiência (QoE). Os autores categorizam os cenários nas seguintes faixas de operação, considerando suas respectivas aplicações e requisitos técnicos:

- **Alta Latência (>45s):** Caracteriza-se por fluxos unidirecionais voltados para grandes audiências, onde o imediatismo é menos crítico que a escalabilidade e a robustez do serviço oferecido. Prioriza-se a qualidade de imagem e a estabilidade da conexão.
- **Latência Típica (10-45s):** Representa o padrão de mercado para a maioria dos serviços *Over-The-Top* (OTT) convencionais baseados em protocolos de *streaming* adaptativo HTTP (HAS), como o *Dynamic Adaptive Streaming over HTTP* (DASH) e o *HTTP Live Streaming* (HLS). Frequentemente, essa faixa é utilizada como um estado de contingência (*fallback*) quando componentes da entrega de mídia não suportam baixa latência.
- **Baixa Latência (1-10s):** Essencial para proporcionar uma experiência de usuário equiparável à da televisão tradicional (*broadcast*), buscando mitigar o "efeito *spoiler*". Os principais exemplos de aplicação incluem transmissões de esportes *premium*, notícias financeiras, *eSports* e experiências de segunda tela (*second-screen*), onde o evento é consumido simultaneamente em dispositivos diferentes.
- **Ultra-Baixa Latência (<1s):** Fundamental para garantir a integridade de interações em tempo real. Esta categoria é impulsionada por experiências interativas como comentários ao vivo, apostas esportivas (*in-game wagering*), jogos de azar *online* e leilões virtuais. Nesta escala, a latência do vídeo compete com as variações naturais da rede (como *bufferbloat*), exigindo controle rigoroso.
- **Tempo Quase Real (milissegundos):** Faixa de latência na ordem de dezenas até 100 milissegundos, sendo um requisito obrigatório para comunicação síncrona e sistemas que exigem *feedback* de controle imediato. As aplicações englobam videoconferências, cirurgias robóticas, *cloud gaming*, metaverso e o controle remoto de plataformas de *hardware*, como *drones* e veículos autônomos.

3.2 Métricas de Qualidade

No contexto de *Streaming* de Vídeo, Peroni e Gorinsky (2025) definem as diferenças fundamentais entre as métricas de qualidade:

> Qualidade de Experiência (QoE): Uma medida subjetiva que captura o quão satisfeito um usuário está com um serviço, baseada em percepções e experiências individuais. Ela avalia fatores que contribuem para o julgamento pessoal do usuário. Ao contrário de métricas objetivas que focam em parâmetros técnicos, a QoE enfatiza a perspectiva humana.
>
> Qualidade de Serviço (QoS): Uma medida objetiva do desempenho de uma rede ou serviço, que abrange métricas de nível de rede como perda de pacotes, latência e vazão (*throughput*). Ao contrário da QoE, que é subjetiva, a QoS quantifica o desempenho técnico do sistema (PERONI; GORINSKY, 2025, tradução nossa).

É importante distinguir o plano de observabilidade em que cada métrica opera. As métricas de QoS são coletadas no lado da oferta (servidor e rede), enquanto as de QoE podem ser medidas em diferentes camadas: no plano de infraestrutura, pelo *player* em laboratório, ou pelo usuário real em campo. Esta última abordagem, conhecida como *Real User Monitoring* (RUM), consiste em coletar métricas de experiência diretamente dos dispositivos dos usuários durante o uso efetivo da plataforma em produção, capturando as condições reais de rede e *hardware*. Neste trabalho, as métricas de QoE são objetivas e coletadas no plano de infraestrutura e de *player* em ambiente controlado, constituindo um *piso* da experiência real: a percepção em campo agrega ainda o *Round-Trip Time* (RTT), o tempo de ida e volta de um pacote entre cliente e servidor, da última milha e o dispositivo do usuário, acessíveis apenas via RUM.

Métricas de QoS

As métricas de QoS quantificam o desempenho técnico da infraestrutura de entrega:

- **Vazão** (*throughput*): número de requisições atendidas por segundo (req/s), indicador da capacidade de serviço sob carga crescente.
- **Latência**: tempo de resposta medido nos percentis p50, p95 e p99 (ms). O p50 representa o comportamento típico; o p99 revela a cauda extrema que degrada a experiência de uma fração dos usuários.
- ***Egress***: volume de dados transferidos para fora da infraestrutura (GB), principal variável de custo em plataformas de VoD baseadas em CDN.
- **Taxa de *cache-hit***: proporção de requisições atendidas diretamente pela camada de *cache*, sem necessitar do servidor de origem; quanto maior, menor a latência e o custo de *egress*.

- **Taxa de erro**: proporção de requisições que resultam em falha (4xx/5xx), indicador de disponibilidade e estabilidade do serviço.
- **Utilização de recursos**: consumo de CPU, memória e rede nas instâncias de serviço, relevante para o dimensionamento de capacidade e para a detecção de gargalos.

Métricas de QoE

As métricas de QoE capturam a experiência de reprodução percebida pelo cliente (ALSADER; BARAKABITZE; MKWAWA, 2025):

- ***Startup time* / *Time-to-first-frame* (TTFF)**: tempo decorrido entre o início da solicitação de reprodução e o primeiro quadro exibido ao usuário. Trata-se de uma métrica em camadas: no plano de infraestrutura mede-se a entrega do primeiro segmento pelo *backend* (32 ms em condições locais); no plano de *player* soma-se a inicialização e a decodificação (85 ms a 2 s em laboratório); no plano do usuário real agrega-se ainda o RTT da última milha e o dispositivo.
- ***Rebuffering ratio* e *stalls***: proporção do tempo de sessão em que a reprodução foi interrompida por insuficiência de dados no *buffer* (*rebuffering*), acompanhada do número e da duração dos eventos individuais de parada (*stalls*). É um dos fatores de maior impacto negativo na QoE (TIMMERER *et al.*, 2025).
- ***Bitrate* médio e oscilações**: taxa de bits média da sessão (indicador da qualidade visual entregue) e frequência das trocas de representação pelo algoritmo ABR (oscilações abruptas de qualidade degradam a experiência mesmo sem *stalls*).
- **Nível do *buffer***: ocupação do *buffer* de reprodução ao longo do tempo; valores baixos indicam risco iminente de *rebuffering*, enquanto valores estáveis sinalizam entrega confortável.
- **Quadros descartados (*dropped frames*)**: indicador de sobrecarga de decodificação no dispositivo cliente; valores elevados produzem reprodução entrecortada mesmo sem eventos de *rebuffering*.

Destas, este trabalho instrumenta diretamente o *startup time*, o *rebuffering ratio* e o *bitrate* médio (Capítulo 7); as demais são apresentadas para completude do panorama de QoE.

Métricas de Eficiência de Codec

Para a comparação entre codificadores de vídeo, utilizam-se métricas que capturam a relação entre qualidade perceptual e taxa de bits (MOINA-RIVERA *et al.*, 2024):

- *Video Multimethod Assessment Fusion* (VMAF): métrica de qualidade perceptual com referência completa (*full-reference*) desenvolvida pela Netflix: cada quadro do vídeo comprimido é comparado ao quadro correspondente do vídeo original, e as pontuações por quadro são agregadas para o clipe. O nome descreve o mecanismo: em vez de medir um único aspecto do sinal, o VMAF funde múltiplas métricas elementares (fidelidade visual, preservação de detalhe e intensidade de movimento) por meio de um modelo de aprendizado de máquina treinado com avaliações humanas de qualidade, o que explica sua correlação com a percepção subjetiva superior à de métricas puramente matemáticas. A escala vai de 0 a 100; valores acima de 93 indicam qualidade visualmente indistinguível do original para a maioria dos observadores, e por isso esse patamar é adotado como referência de alta qualidade.
- *Peak Signal-to-Noise Ratio* (PSNR): relação sinal-ruído de pico entre o vídeo original e o comprimido, expressa em decibéis (dB). Apesar de amplamente utilizado, correlaciona-se menos com a percepção humana do que o VMAF em cenas com movimento complexo.
- *Bjøntegaard Delta Rate* (BD-rate): métrica que resume, em um único valor percentual, a diferença média de taxa de bits entre dois codificadores mantida a mesma qualidade objetiva. O cálculo ajusta as curvas de *rate-distortion* (qualidade em função da taxa de bits) de cada codificador e integra a diferença entre elas ao longo da faixa de qualidade comum (BJØNTEGAARD, 2001). Um BD-rate negativo indica que o codificador avaliado entrega a mesma qualidade com menos bits do que o codificador de referência; é a única métrica adequada para comparar codificadores cujos parâmetros de qualidade não são numericamente equivalentes entre si.

3.3 Tecnologias de Compressão de Vídeo

A compressão de vídeo é um processo fundamental que visa reduzir a quantidade de dados necessários para representar digitalmente um sinal de vídeo, otimizando tanto o armazenamento quanto a eficiência da transmissão. Este processo baseia-se na exploração e eliminação de redundâncias inerentes ao sinal de vídeo: redundâncias espaciais (dentro do mesmo quadro ou *intra-frame*) e temporais (entre quadros sucessivos ou *inter-frame*). Além disso, técnicas de compressão com perdas removem informações visuais que são irrelevantes ou imperceptíveis ao sistema visual humano, permitindo taxas de compressão significativamente maiores.

Em sua revisão sobre os desafios da codificação de vídeo em nuvem, Moina-Rivera *et al.* (2024) destacam a necessidade premente de equilibrar a eficiência de compressão com a complexidade computacional. O cenário atual de *codecs* é dominado pelas seguintes tecnologias:

- **H.264:** O h.264, ou Advanced Video Coding (AVC), é um padrão maduro e amplamente compatível com a vasta maioria dos dispositivos legados e modernos. É eficiente para resoluções até Full HD e continua sendo a base para muitos serviços de *streaming* devido à sua baixa complexidade de decodificação e amplo suporte de *hardware*.
- **H.265:** Sucessor direto do H.264, o H.265, ou High Efficiency Video Coding (HEVC), oferece uma redução de aproximadamente 50% na taxa de bits para a mesma qualidade perceptual. Essa eficiência é crucial para viabilizar transmissões em resoluções 4K e com High Dynamic Range (HDR). Entretanto, o HEVC impõe uma complexidade computacional significativamente maior e é um padrão sujeito a royalties, com licença paga por fabricantes e desenvolvedores (MOINA-RIVERA *et al.*, 2024).
- **H.266:** Representando a fronteira tecnológica, o H.266, ou também chamado por Versatile Video Coding(VVC), é projetado para suportar as demandas futuras de resoluções 8K, 16K e vídeos imersivos de 360 graus, sendo uma peça-chave para futuras aplicações em redes 6G e em mídia imersiva (XR, telepresença e vídeo 360 graus) (ALSADER; BARAKABITZE; MKWAWA, 2025).
- **AV1:** Desenvolvido pela Alliance for Open Media, o AV1 é um *codec* de código aberto e livre de royalties. Ele supera seus antecessores em eficiência

de compressão (mais de 30% sobre o VP9 e o H.265), mas exige um poder de processamento consideravelmente maior para a codificação (MOINA-RIVERA *et al.*, 2024). Esse custo de codificação ainda limita sua adoção massiva em cenários de transmissão ao vivo de baixa latência, onde o tempo de codificação é crítico.

- **VP9:** O VP9 é a alternativa de código aberto e livre de royalties ao H.265, desenvolvido pelo Google como sucessor do VP8. Lançado em 2013, ganhou ampla adoção devido à sua integração nativa com o YouTube e o sistema operacional Android. Segundo Moina-Rivera *et al.* (2024), foi projetado para reduzir a taxa de bits em 50% em comparação ao seu antecessor, mantendo a mesma qualidade visual, e competindo diretamente com a eficiência de compressão do padrão HEVC. Embora apresente maior complexidade computacional que o H.264, sua natureza aberta o torna uma escolha estratégica para reduzir custos de licenciamento em grandes escalas.

A escolha de um *codec* envolve um compromisso estratégico (*trade-off*) entre três pilares: eficiência de compressão (*bitrate*), qualidade visual e complexidade computacional. *Codecs* avançados como AV1 e VVC são vitais para economizar largura de banda em redes congestionadas e reduzir custos de CDN. No entanto, o aumento exponencial na complexidade algorítmica exige *hardware* de codificação mais robusto e resulta em maior consumo de energia nos dispositivos de reprodução, fatores que impactam diretamente a escalabilidade do serviço.

Eficiência Energética na Codificação

A sustentabilidade tornou-se uma preocupação central na evolução dos *codecs*. Embora o VVC ofereça ganhos impressionantes de compressão, sua complexidade computacional pode ser até oito vezes superior à do HEVC, acarretando um aumento substancial no consumo energético tanto nos *data centers* de codificação quanto nos dispositivos finais de decodificação. Em contrapartida, AV1 oferece um melhor *trade-off* entre a eficiência na codificação e o gasto energético comparado ao AVC, HEVC, VP9 e VVC (TIMMERER *et al.*, 2025).

3.4 Protocolos de *Streaming* de Vídeo

Conforme detalhado na análise de Bentaleb *et al.* (2026), a arquitetura de *streaming* moderna é segmentada em três estágios funcionais, cada um empregando

protocolos específicos otimizados para seus requisitos particulares: Contribuição, Ingestão e Distribuição.

A etapa de contribuição transporta o vídeo de alta fidelidade do local de captura ao centro de processamento, com protocolos tipicamente baseados em *push* que priorizam confiabilidade e baixa latência sobre redes não gerenciadas (BENTALEB *et al.*, 2026). O *Real-Time Messaging Protocol* (RTMP), protocolo legado baseado em TCP, permanece amplamente usado para envio de vídeo a plataformas como YouTube e Twitch por sua onipresença em *software* de transmissão, embora tenha sido descontinuado para entrega final ao usuário. Em seu lugar, o *Secure Reliable Transport* (SRT) emerge como o sucessor moderno: baseado em UDP, oferece baixa latência, criptografia robusta e mecanismos de recuperação de perda de pacotes (ARQ), sendo ideal para contribuição via Internet pública imprevisível.

A ingestão conecta a saída do processo de codificação ao servidor de origem (*origin server*), ponto a partir do qual a infraestrutura de distribuição (CDN) escala a entrega (BENTALEB *et al.*, 2026). A especificação DASH Industry Forum (*DASH-IF*) *Live Media Ingest* padroniza essa interface com verbos HTTP (POST ou PUT) sobre arquivos *Common Media Application Format* (CMAF), facilitando a interoperabilidade entre codificadores de diferentes fabricantes e servidores de origem. Para transmissões de ultra-baixa latência a partir de navegadores, o *WebRTC-HTTP Ingestion Protocol* (WHIP) padroniza a ingestão de fluxos WebRTC via HTTP, simplificando a arquitetura de ponta.

A distribuição é a etapa mais desafiadora em termos de escala, responsável por entregar o conteúdo a milhões de espectadores simultâneos (BENTALEB *et al.*, 2026). Os protocolos MPEG-DASH e HLS, baseados em HTTP (*pull-based*), dominam o mercado: o vídeo é segmentado em pequenos arquivos baixados progressivamente pelo cliente, o que facilita a passagem por *firewalls* e o uso de CDN HTTP padrão para escalabilidade massiva. Para reduzir a latência inerente ao modelo de segmentos, as extensões *Low-Latency* (LL-HLS e LL-DASH) combinam *Chunked Encoding* (subdivisão dos segmentos em unidades menores decodificáveis independentemente) com o *Chunked Transfer Encoding* (CTE) do HTTP/1.1, que permite ao servidor iniciar o envio da resposta antes de completá-la, atingindo a faixa de 2 a 5 segundos de latência. Em direção à sub-segundo, o *High Efficiency Streaming Protocol* (HESP) utiliza *frames* de inicialização rápida para troca instantânea de canais e *buffers* extremamente reduzidos. Complementarmente, o padrão MPEG-SAND define uma comunicação bidirecional entre clientes DASH e elementos de rede (DNE), permitindo que CDN e servidores de borda

auxiliem a tomada de decisão do cliente com informações de desempenho de rede e de cliente e de requisitos de banda (TIMMERER *et al.*, 2025). O processamento na borda (*Edge Computing*) vai além do *cache* convencional, executando tarefas como transcodificação *just-in-time* e inserção de anúncios personalizada; Timmerer *et al.* (2025) apontam que esse processamento beneficiará significativamente aplicações de próxima geração, como vídeo volumétrico e imersivo.

O substrato técnico dos protocolos de distribuição adaptativa (HAS), como o MPEG-DASH e o HLS, é o HTTP, cujas evoluções mais recentes foram impulsionadas pela necessidade de entregar conteúdo multimídia com eficiência e latência reduzida (BENTALEB *et al.*, 2026).

HTTP/1.1: Padronizado originalmente na RFC 2616 (FIELDING *et al.*, 1999) e refinado em diversas atualizações subsequentes (FIELDING; RESCHKE, 2014c; FIELDING; RESCHKE, 2014d; FIELDING; RESCHKE, 2014b; FIELDING; LAFON; RESCHKE, 2014; FIELDING; NOTTINGHAM; RESCHKE, 2014; FIELDING; RESCHKE, 2014a), esta versão introduziu o conceito de conexões persistentes *Keep-Alive*. Embora tenha reduzido a sobrecarga do *handshake* TCP, o protocolo sofre, no nível da aplicação, com o bloqueio de cabeça de fila (*Head-of-Line Blocking*): o atraso no processamento de uma requisição bloqueia as subsequentes na mesma conexão, o que pode ocasionar o esvaziamento do *buffer* do reprodutor (*player*).

HTTP/2: Definido na RFC 7540 e atualizações (BELSHE; PEON; THOMSON, 2015; THOMSON; BENFIELD, 2022), representou uma mudança paradigmática ao introduzir a multiplexação binária de fluxos, que permite ao cliente solicitar áudio, vídeo e metadados simultaneamente na mesma conexão TCP, otimizando o uso da largura de banda. O recurso de *Server Push* viabiliza o envio antecipado de segmentos, embora sua implementação eficaz em *streaming* se revele complexa pela dificuldade de prever o estado do *buffer* e a adaptação de taxa de bits. Apesar das inovações, por operar sobre o TCP, o HTTP/2 permanece suscetível ao bloqueio de cabeça de fila no nível de transporte em redes com perda de pacotes.

HTTP/3: A iteração mais recente (RFC 9114 (BISHOP, 2022)) opera sobre o protocolo de transporte QUIC (IYENGAR; THOMSON, 2021), que utiliza UDP em substituição ao TCP. Devido à independência dos fluxos, a perda de pacotes de um segmento não impacta a entrega de segmentos paralelos, eliminando o bloqueio de cabeça de fila. O QUIC integra em uma única rodada de negociação a conexão

e a criptografia TLS 1.3, reduzindo drasticamente o tempo de início (*startup time*). O uso de *Connection IDs* assegura a continuidade do *streaming* durante a transição entre redes.

Segundo Bentaleb *et al.* (2026), o HTTP/3 é fundamental para viabilizar aplicações de *streaming* de ultra-baixa latência, oferecendo transporte confiável sem o bloqueio de cabeça de fila (*head-of-line blocking*) no nível de transporte típico do TCP.

3.5 Servidores

Os servidores constituem a espinha dorsal da infraestrutura de *streaming*, sendo responsáveis pelo armazenamento, processamento e entrega do conteúdo de vídeo. A escolha e configuração adequadas dos servidores impactam diretamente a escalabilidade, a latência e o custo operacional do sistema. Esta seção detalha os principais tipos de servidores empregados em arquiteturas de *streaming* modernas e as tecnologias de *software* mais utilizadas.

Servidor de Origem (Origin Server)

O servidor de origem constitui o repositório central do conteúdo de vídeo, a fonte da verdade a partir da qual todas as demais camadas da arquitetura derivam suas cópias. Concentra quatro responsabilidades principais:

- **Armazenamento Persistente:** Manutenção de todos os arquivos de vídeo originais (*mezzanine files*) e das representações transcodificadas. Utiliza sistemas de armazenamento de alta capacidade e redundância, como o Hadoop Distributed File System (MOINA-RIVERA *et al.*, 2024). Também utiliza-se serviços de *object storage* em nuvem (Amazon S3, Google Cloud Storage).
- **Processamento de Conteúdo:** Execução de tarefas computacionalmente intensivas, como transcodificação, empacotamento e geração de miniaturas (*thumbnails*). Requer *hardware* com alta capacidade de processamento, preferencialmente com aceleração por GPU para *codecs* modernos.
- **Ingestão de Novos Conteúdos:** Recepção e validação de *uploads* de vídeo, seja através da interface *web* ou da Application Programming Interface (API).

- **Fonte para CDN:** Atua como a fonte primária de dados para os servidores de *cache* da CDN. Implementa políticas de invalidação de *cache* para garantir que atualizações de conteúdo sejam propagadas corretamente.

Devido à sua criticidade, servidores de origem são tipicamente implantados em *data centers* de alta disponibilidade, com redundância geográfica e *backups* regulares.

Servidores de Borda (Edge Servers)

Os servidores de borda são os pontos de contato direto com os usuários finais, posicionados estrategicamente em localizações geográficas distribuídas para minimizar a latência de rede. Suas funções principais são:

- **Cache de Conteúdo:** Armazenamento temporário dos segmentos de vídeo mais populares ou recentemente solicitados. Implementam algoritmos de substituição de *cache* para otimizar a taxa de acerto (*hit rate*).
- **Entrega de Baixa Latência:** Servir conteúdo com o menor RTT possível, aproveitando a proximidade física com o cliente.
- **Balanceamento de Carga:** Distribuição de requisições entre múltiplos servidores de borda em um mesmo *Point of Presence* (PoP) para evitar sobrecarga.
- **Terminação de Conexões Seguras:** Gerenciamento de certificados TLS/Secure Sockets Layer (SSL) e terminação de conexões Hypertext Transfer Protocol Secure (HTTPS), reduzindo a *overhead* de criptografia no servidor de origem.

Em arquiteturas modernas baseadas em *Edge Computing*, esses servidores também executam lógica de aplicação, como transcodificação *just-in-time* e inserção de anúncios.

Servidores de Empacotamento (Packaging Servers)

Servidores dedicados ao empacotamento dinâmico ou *just-in-time* são uma evolução recente que permite maior flexibilidade e eficiência de armazenamento. Suas responsabilidades incluem:

- **Empacotamento Sob Demanda:** Geração de segmentos DASH ou HLS a partir de arquivos CMAF no momento da requisição, eliminando a necessidade de armazenar múltiplas versões pré-empacotadas.

- **Geração de Manifestos Dinâmicos:** Criação de arquivos Media Presentation Description (MPD) ou M3U8 personalizados para cada sessão de usuário, permitindo inserção de anúncios, restrições geográficas ou seleção de CDN.
- **Suporte Multi-Protocolo:** Servir o mesmo conteúdo em diferentes formatos (DASH, HLS, Smooth *Streaming*) a partir de uma única fonte, reduzindo redundância de armazenamento.

Ferramentas como o AWS Elemental MediaPackage (Amazon Web Services, 2025) e o Unified *Streaming* Platform (Unified Streaming, 2025) são exemplos de soluções comerciais. No ecossistema de código aberto, o FFmpeg (FFmpeg Developers, 2024) é a ferramenta de referência para transcodificação e codificação, e o Shaka Packager (Shaka Project, 2025) pode ser configurado para empacotamento dinâmico.

Tecnologias de Servidor HTTP

A escolha do *software* de servidor HTTP é crítica para o desempenho do sistema de *streaming*. O Nginx (NGINX, 2025) é a referência da categoria: seu modelo de processamento assíncrono e orientado a eventos gerencia dezenas de milhares de conexões simultâneas com baixo consumo de recursos, oferecendo *cache* em disco otimizado, balanceamento de carga entre *backends* (*upstream*) e suporte a HTTP/2 e HTTP/3, tornando-o a escolha convencional para servir segmentos de vídeo.

Na implementação final, contudo, a entrega dos segmentos foi delegada a uma CDN (CloudFront), e o plano de controle (consulta de catálogo e composição de manifesto) migrou para um serviço próprio em Go (Capítulo 5). Com a separação entre plano de controle e plano de dados, o papel clássico do servidor HTTP de propósito geral é absorvido pela borda da CDN, dispensando um servidor HTTP dedicado na fronteira de entrega.

CDN e Arquiteturas de Distribuição

Uma CDN é uma infraestrutura de rede geograficamente distribuída, composta por servidores *proxy* e *data centers*. Seu objetivo primordial é assegurar alta disponibilidade e desempenho, aproximando fisicamente o conteúdo dos usuários finais. No contexto de *streaming*, a CDN atua como um *cache* distribuído, armazenando segmentos de vídeo em servidores de borda (*Edge Servers*). Isso reduz drasticamente a latência de rede (RTT) e alivia a carga no servidor de origem, prevenindo gargalos. Grandes provedores globais como Akamai, Cloudflare, Amazon CloudFront e Fastly dominam este setor, oferecendo otimizações específicas para a entrega de vídeo ao vivo e sob demanda.

Uma evolução recente e significativa é o conceito de *Content Steering*, que permite o gerenciamento dinâmico e granular do tráfego entre múltiplas CDN. Rodrigues-Filho *et al.* (2024) exploram o uso do *computing continuum* para suportar o *content steering*, permitindo que provedores de conteúdo alternem entre diferentes CDN ou servidores de borda em tempo real para otimizar a entrega segundo desempenho, QoE e custo de operação. Farahani *et al.* (2023) propõem o CP-Steering, uma solução de *content steering* ciente de protocolo (incluindo QUIC) e de CDN para refinar a seleção de fonte na entrega adaptativa.

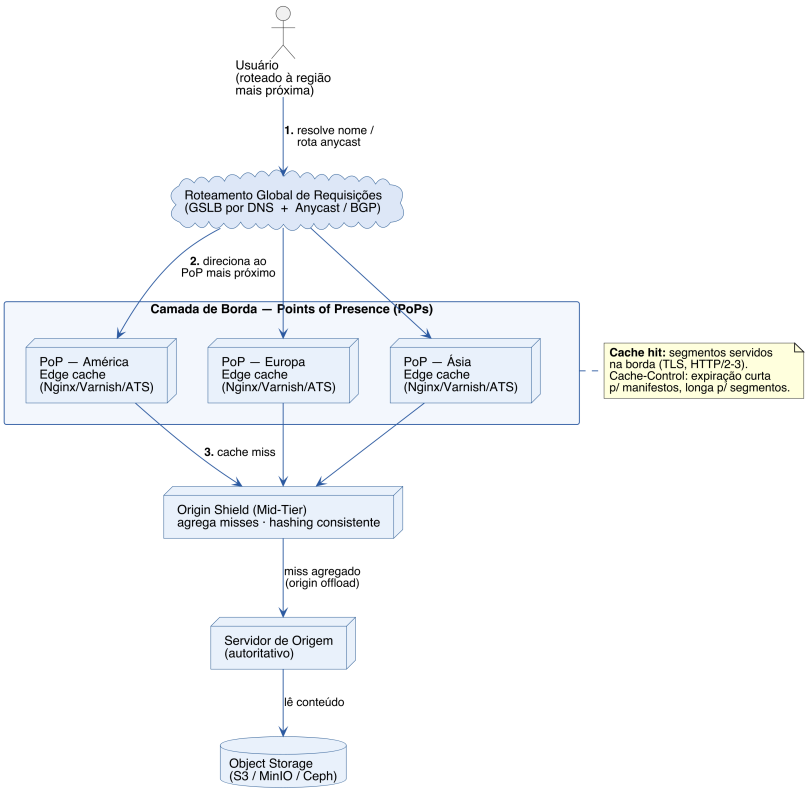
Construção de uma CDN *Self-Hosted* em Escala Global

Embora a maioria dos serviços recorra a CDN comerciais, compreender a construção de uma CDN *self-hosted* (autogerida) é esclarecedor tanto do ponto de vista arquitetural quanto econômico, e delimita até onde uma implementação aberta pode chegar. Uma CDN global é, em essência, uma rede sobreposta (*overlay*) de *caches* HTTP distribuídos geograficamente. Para fins de exposição, este trabalho a organiza em três funções complementares (o roteamento de requisições, a hierarquia de *cache* e a rede física de nós de borda), recorte inspirado na descrição da rede da Akamai, que detalha o sistema de mapeamento (roteamento) e a plataforma de servidores de borda distribuída em milhares de localidades (NYGREN; SITARAMAN; SUN, 2010). A Figura 4 apresenta a topologia conceitual e o caminho de uma requisição através desses subsistemas.

O ciclo de vida de uma requisição, indicado pelos passos numerados na figura, sintetiza o funcionamento do conjunto: (1) o cliente é roteado, por balanceamento global por DNS (*Global Server Load Balancing*, GSLB) ou *anycast*, ao PoP topologicamente mais próximo; (2) o PoP de borda atende a requisição: em caso de *cache hit*, os segmentos são servidos imediatamente na borda, com terminação TLS e suporte a HTTP/2 e HTTP/3; (3) em caso de *miss*, a requisição sobe pela hierarquia de *cache* até o *origin shield* e, somente se necessário, ao servidor de origem, que lê o conteúdo do armazenamento de objetos. Os parágrafos seguintes detalham esses subsistemas, da rede física de PoPs ao roteamento de requisições e à hierarquia de *cache*, encerrando com a operação da frota e seus limites.

A unidade básica dessa rede é o PoP, um conjunto de servidores de *cache* instalados próximos aos usuários, idealmente em *data centers* interconectados a pontos de troca de tráfego (*Internet Exchange Points*, IXP) e diretamente a provedores de acesso (ISP). Cada nó executa um *software* de *cache* HTTP de alto desempenho (*Nginx*, *Varnish* ou *Apache Traffic Server*), responsável por servir o conteúdo quente e por encaminhar

Figura 4 – Topologia conceitual de uma CDN *self-hosted* em escala global



Fonte: Elaborado pelo autor.

os *misses* para a camada superior. A estratégia de *peering* e a colocação dos PoPs determinam diretamente o custo de trânsito e a latência de última milha; a abordagem de "CDN embarcada" da Netflix (*Open Connect*), que instala nós dentro da rede dos ISP, é o exemplo extremo dessa otimização (SUMAN; MOON; CHOI, 2022).

O desafio central de uma CDN global é direcionar cada cliente ao PoP mais próximo e saudável. Duas técnicas dominam: (i) o balanceamento global por DNS (GSLB), em que o servidor DNS autoritativo resolve o nome do serviço para o IP do PoP ótimo segundo a geolocalização e a carga, abordagem detalhada na rede da Akamai por meio de seu sistema de mapeamento (NYGREN; SITARAMAN; SUN, 2010); e (ii) o *anycast* via BGP, em que o mesmo endereço IP é anunciado a partir de múltiplos PoPs e o roteamento da própria Internet (BGP) entrega o pacote à instância topologicamente mais próxima. O *anycast* simplifica o cliente e reage rápido a falhas, mas oferece controle mais grosseiro sobre a distribuição de carga; o GSLB é mais granular, porém limitado pela granularidade e pelo *cache* de resolvedores DNS. Na prática, grandes redes combinam as duas abordagens.

Para evitar que cada PoP sobrecarregue o servidor de origem, intercala-se uma camada intermediária (*mid-tier* ou *origin shield*) que agrega os *misses* de vários PoPs

e oferece um único ponto de *fetch* à origem, maximizando o *origin offload*. O particionamento do conteúdo entre nós emprega tipicamente *hashing* consistente, de modo que a entrada ou saída de um nó redistribua apenas uma fração das chaves. Sobre essa hierarquia operam ainda a terminação TLS na borda, o suporte a HTTP/2 e HTTP/3, e políticas de `Cache-Control` distintas para manifestos (expiração curta) e segmentos (expiração longa).

A orquestração da frota, envolvendo provisionamento, atualização e recuperação de centenas de nós, apoia-se em plataformas de contêineres como o Kubernetes (BURNS *et al.*, 2016), complementadas por observabilidade distribuída para detectar PoPs degradados. O obstáculo decisivo, contudo, não é técnico, mas econômico e logístico: operar uma CDN verdadeiramente global exige investimento de capital em *hardware*, contratos de *peering* e presença física em dezenas de países, barreira que explica por que a esmagadora maioria dos serviços, inclusive este trabalho, delega o plano de dados a uma CDN comercial (CloudFront) e concentra o esforço de engenharia no plano de controle.

3.6 Arquiteturas de *Streaming*

A arquitetura de um sistema de *streaming* de vídeo define a organização estrutural dos componentes, o fluxo de dados entre eles e as estratégias de distribuição empregadas para entregar conteúdo a milhões de usuários simultâneos com alta disponibilidade e qualidade. A evolução das arquiteturas de *streaming* reflete a busca contínua por soluções que equilibrem escalabilidade, custo operacional, latência e qualidade de experiência.

O modelo mais simples, a arquitetura cliente-servidor direta, consiste em um servidor central que armazena e transmite o conteúdo aos clientes solicitantes. Embora conceitualmente simples, esse modelo apresenta limitações críticas de escalabilidade: o servidor central torna-se rapidamente um gargalo quando o número de clientes simultâneos cresce, e a latência de rede aumenta proporcionalmente à distância geográfica. Por isso, é viável apenas para aplicações de pequena escala, como *streaming* corporativo interno ou serviços educacionais com audiência limitada.

Para superar essas limitações, a arquitetura baseada em CDN tornou-se o padrão para serviços de *streaming* em escala global. Nesse modelo, o conteúdo é replicado e distribuído por uma rede hierárquica de servidores geograficamente dispersos, do servidor de origem aos nós de borda próximos ao usuário final (estrutura de PoPs, hierarquia de *cache* e roteamento já detalhada anteriormente nesta seção de distribuição).

A evolução mais recente combina a distribuição geográfica das CDN com capacidades computacionais avançadas nos nós de borda, denominada arquitetura híbrida *Edge/Cloud*. Diferentemente das CDN puramente baseadas em *cache*, os servidores de borda nessa arquitetura executam processamento computacional intensivo: transcodificação *just-in-time* (geração de representações de vídeo sob demanda, adaptadas ao dispositivo e às condições de rede), empacotamento dinâmico entre formatos DASH e HLS, inserção de anúncios personalizada no servidor (*Server-Side Ad Insertion*) e otimização de QoE assistida pela rede por meio do padrão MPEG-SAND.

A comparação entre as maiores plataformas evidencia variações significativas na implementação desse modelo (SUMAN; MOON; CHOI, 2022). A Netflix opera uma CDN proprietária chamada *Open Connect*, com servidores de borda instalados diretamente nas redes dos ISP parceiros, eliminando saltos intermediários e reduzindo custos de trânsito; adota transcodificação por título (*per-title encoding*) e combina H.264, HEVC e AV1 conforme o dispositivo. O Amazon Prime Video integra-se verticalmente à AWS, usando CloudFront como CDN e serviços gerenciados (MediaConvert para transcodificação, MediaPackage para empacotamento *just-in-time* e Lambda@Edge para lógica customizada nos PoPs) que permitem escalabilidade elástica. O YouTube, operado pelo Google, apoia-se na Google Global *Cache* (presente em mais de 100 países) e foi pioneiro na adoção de *codecs* abertos (VP9 e AV1), usando DASH como protocolo primário com *fallback* para *progressive download* em dispositivos legados.

A adoção de estratégias *multi-CDN* com *Content Steering* (RODRIGUES-FILHO *et al.*, 2024) (alternância dinâmica de provedor conforme desempenho) melhora a resiliência, ao permitir o redirecionamento do tráfego para outro provedor em caso de falha ou degradação, e o balanceamento de carga entre CDN. Experimentos com *streaming peer-to-peer* assistido buscam reduzir custos de largura de banda em eventos de alta audiência, enquanto abordagens descentralizadas baseadas em *blockchain* ainda enfrentam desafios de latência e escalabilidade.

3.7 Análise das Fontes de Latência

A latência End-to-End (E2E) em um sistema de *streaming* não é gerada em um único ponto, mas é o somatório cumulativo dos atrasos introduzidos em cada etapa do complexo fluxo de trabalho de mídia. Bentaleb *et al.* (2026) propõem uma categorização

das fontes de atraso em três grupos: latência de preparação, de entrega e de consumo de conteúdo.

A latência de preparação compreende todos os atrasos que ocorrem antes que o conteúdo deixe o servidor de origem. Na captura e processamento, contribuem os atrasos inerentes aos sensores das câmeras, à conversão analógico-digital e ao processamento inicial de sinal. A codificação acrescenta um atraso proporcional ao tamanho do *Group of Pictures* (GOP): GOP menores reduzem a latência, mas comprometem a eficiência de compressão ao reduzir a proporção de quadros preditivos. O empacotamento adiciona o tempo necessário para encapsular o fluxo codificado em segmentos compatíveis com os protocolos de distribuição.

A latência de entrega refere-se ao tempo de propagação do conteúdo pela infraestrutura de rede e divide-se em três saltos: a ingestão (*first mile*), que leva o vídeo do codificador ao servidor de origem ou ao ponto de entrada da CDN; a distribuição, que propaga o conteúdo pela hierarquia de *caches* da CDN até o servidor de borda mais próximo do usuário; e a última milha (*last mile*), trecho final entre a borda e o cliente, frequentemente o mais variável: RTT, *jitter* (a variação do atraso entre pacotes consecutivos), perda de pacotes e congestionamento impactam severamente essa etapa, especialmente em redes móveis e Wi-Fi. O fenômeno de *bufferbloat* (excesso de *buffer* em roteadores intermediários) pode exacerbar significativamente a latência percebida.

A latência de consumo ocorre inteiramente no dispositivo do usuário final. O *buffer* do *player* é frequentemente o componente mais significativo da latência total: o *player* armazena intencionalmente uma quantidade de mídia para absorver variações na chegada dos pacotes (*jitter*) e prevenir interrupções na reprodução. Adicionalmente, o tempo de decodificação e renderização contribui com uma parcela mensurável, ainda que geralmente menor que a do *buffer* de reprodução.

3.8 *Players* e Algoritmos de Adaptação

No lado do cliente, a inteligência do sistema reside no *player* de vídeo. O `dash.js` destaca-se como a implementação de referência de código aberto para o padrão MPEG-DASH, sendo uma ferramenta crucial para a pesquisa e desenvolvimento de novos algoritmos ABR. Além dele, o ecossistema conta com opções como o Shaka Player (reprodução de DASH e HLS em navegadores via MSE/EME, com suporte a DRM), o

hls.js (que viabiliza HLS em navegadores via MSE) e o ExoPlayer (*player* de DASH/HLS para o ecossistema Android) (BENTALEB *et al.*, 2026).

O *player* é responsável por orquestrar toda a sessão de *streaming*: baixar e analisar o manifesto, estimar a largura de banda disponível e solicitar os segmentos de mídia apropriados. A eficácia desses algoritmos determina diretamente a QoE do usuário.

Algoritmos ABR

O núcleo de qualquer *player* HAS é o algoritmo ABR, que toma decisões dinâmicas sobre qual qualidade de vídeo solicitar a cada momento. Os algoritmos ABR são usualmente organizados em quatro famílias, segundo o sinal que orienta a decisão de adaptação (TIMMERER *et al.*, 2025):

- **Baseados em Vazão (*Throughput-based*):** Utilizam médias harmônicas do histórico de *downloads* recentes para prever a largura de banda futura. São simples, mas podem ser instáveis em redes voláteis.
- **Baseados em *Buffer* (*Buffer-based*):** Tomam decisões baseadas principalmente no nível de ocupação do *buffer* de reprodução. Priorizam evitar interrupções (*stalling*), sendo mais robustos a variações bruscas de rede, mas podem demorar a convergir para a melhor qualidade visual.
- **Híbridos:** Combinam estimativas de vazão e o estado do *buffer* para obter o melhor dos dois mundos, equilibrando estabilidade e responsividade.
- **Baseados em Aprendizado de Máquina (ML-based):** Representam a fronteira da pesquisa, utilizando técnicas como *Reinforcement Learning* para aprender políticas de adaptação ótimas através da interação contínua com o ambiente de rede complexo.

Em cenários de baixa latência, a ABR isolada é insuficiente; é necessário gerenciar ativamente o atraso de reprodução em relação ao tempo real. Bentaleb *et al.* (2026) descrevem o *playback speed control* como ferramenta essencial: quando o *player* acumula atraso excessivo em relação ao tempo real (*live edge*), aumenta imperceptivelmente a velocidade de reprodução (por exemplo, para 1,1×), consumindo o *buffer* mais rapidamente sem cortes abruptos no áudio ou vídeo, técnica denominada *catch-up* com correção de *pitch*; inversamente, com o *buffer* perigosamente vazio, o *player* reduz levemente a velocidade (por exemplo, para 0,95×) para acumular dados antes de ser forçado a interromper a reprodução, trocando um breve aumento de latência pela continuidade da exibição.

Independentemente da tecnologia de *player* escolhida, a operação correta do serviço exige configurações críticas no servidor: habilitação de CORS, para que *players* web de domínios distintos acessem os segmentos; desabilitação de compressão gzip/brotli para arquivos de vídeo (já comprimidos), com habilitação para manifestos (MPD, M3U8); suporte a *byte-range requests* (cabeçalho `Range`) para *seeking* eficiente; políticas de `Cache-Control` distintas para segmentos (*cache* longo) e manifestos (*cache* curto ou sem *cache* para conteúdo ao vivo); e reutilização de conexões TCP/TLS (*connection pooling*) para reduzir *overhead* de *handshake*.

3.9 Arquiteturas de Microsserviços e Observabilidade

A evolução de aplicações monolíticas para arquiteturas de microsserviços tornou-se o padrão dominante em sistemas de larga escala que exigem escalabilidade e disponibilidade contínuas. Em uma arquitetura de microsserviços, o sistema é decomposto em serviços independentes, cada um responsável por uma capacidade de negócio, comunicando-se por interfaces bem definidas (tipicamente HTTP ou mensageria assíncrona) (DRAGONI *et al.*, 2017; NEWMAN, 2021). Esse estilo oferece benefícios diretos para uma plataforma de *streaming*: a escalabilidade independente (o serviço de transcodificação, intensivo em CPU, escala de forma distinta do serviço de distribuição, intensivo em rede), o isolamento de falhas (a indisponibilidade de um serviço não derruba o *pipeline* inteiro) e a heterogeneidade tecnológica (cada serviço pode adotar a linguagem e o armazenamento mais adequados à sua função). O caso da Netflix, que migrou de uma aplicação monolítica para uma arquitetura de centenas de microsserviços (MAURO, 2015), é um exemplo emblemático dessa transição em sistemas de *streaming* de larga escala.

Essa decomposição, contudo, introduz custos: a complexidade operacional cresce, a comunicação entre serviços passa a ser uma fonte de latência e falhas, e a consistência de dados torna-se distribuída (NEWMAN, 2021). O uso de mensageria assíncrona (filas e tópicos) é uma técnica recorrente para desacoplar serviços e absorver picos de carga, especialmente em *pipelines* orientados a eventos como o de ingestão e transcodificação de vídeo. Implementações como o RabbitMQ (Broadcom Inc., 2024) adotam o protocolo AMQP para garantia de entrega e persistência de mensagens entre serviços. Essa heterogeneidade estende-se ao armazenamento de cada serviço: bancos de dados de documentos, como o MongoDB, guardam entidades semiestruturadas (caso

dos metadados de um catálogo de vídeos) em documentos flexíveis, sem esquema rígido, enquanto bancos em memória, como o Redis, servem leituras de baixa latência atuando como *cache* dos bancos primários. Dois instrumentos operacionais completam esse estilo arquitetural. A conteinerização empacota cada serviço com suas dependências em uma unidade isolada e portável (o contêiner), executável de forma idêntica em qualquer hospedeiro; composições declarativas, como o Docker Compose, descrevem o conjunto de contêineres, redes e volumes de um ambiente completo em um único arquivo versionável. A infraestrutura como código (IaC) estende o mesmo princípio aos recursos de nuvem: a topologia de serviços gerenciados é descrita em arquivos declarativos, versionados junto ao código, o que torna o ambiente reproduzível e auditável, em vez de configurado manualmente. No plano das linguagens e *frameworks*, serviços de I/O intensivo beneficiam-se de linguagens compiladas com concorrência leve (caso de Go, cujas *goroutines* multiplexam milhares de conexões sobre poucos *threads* do sistema), enquanto as interfaces web modernas apoiam-se em *frameworks* de componentes reativos, como Svelte e NextJS/React.

A natureza distribuída dos microsserviços torna a observabilidade (capacidade de inferir o estado interno do sistema a partir de seus sinais externos) um requisito de primeira ordem, e não um acessório. Ela apoia-se em três pilares: *logs* (registros de eventos discretos), *métricas* (séries temporais agregadas, como vazão, latência e taxa de erro) e *traces* (o caminho de uma requisição através dos serviços) (SRIDHARAN, 2018). No contexto de *streaming*, a observabilidade é o que viabiliza a triangulação entre as métricas do lado do servidor (oferta) e as de QoE do lado do cliente (demanda), permitindo correlacionar configurações de infraestrutura com a experiência percebida. Em ambientes de nuvem, padrões como o *Embedded Metric Format* (EMF) permitem emitir métricas estruturadas a partir dos próprios *logs* da aplicação, reduzindo o acoplamento a agentes de coleta dedicados, abordagem adotada na instrumentação deste trabalho.

A documentação de uma arquitetura de microsserviços, com seus múltiplos serviços e interações, beneficia-se de uma notação que organize a complexidade em níveis de abstração. O modelo C4 (BROWN, 2024) cumpre esse papel ao descrever um sistema de *software* em quatro níveis progressivos: *Context* (o sistema, seus atores e os sistemas externos), *Containers* (as aplicações e os repositórios de dados executáveis independentemente), *Components* (as unidades funcionais internas a cada contêiner) e *Code* (o detalhe de implementação). Cada nível serve a uma audiência e a um propósito

distintos, permitindo ampliar progressivamente o foco, do panorama geral ao detalhe de um serviço. Por sua clareza e independência de ferramenta, o C4 é adequado para documentar plataformas decompostas em serviços, sendo a notação adotada na descrição da arquitetura implementada neste trabalho (Capítulo 5).

3.10 Computação em Nuvem e Escalabilidade

A operação de uma plataforma de *streaming* em escala depende, hoje, da computação em nuvem, que substitui o investimento de capital em *data centers* próprios por um modelo de consumo elástico de recursos sob demanda. Segundo Mell e Grance (2011), a computação em nuvem caracteriza-se por cinco atributos essenciais: autosserviço sob demanda, amplo acesso pela rede, agrupamento de recursos (*resource pooling*), elasticidade rápida e serviço mensurado (*pay-per-use*). É justamente a elasticidade, a capacidade de provisionar e liberar recursos em minutos acompanhando a demanda, que torna a nuvem adequada a um serviço de *streaming*, cuja carga é marcadamente irregular, com picos de concorrência em campanhas e horários de maior audiência.

Modelos de Serviço e Provedores

A literatura distingue três modelos clássicos de serviço, em níveis crescentes de abstração: Infraestrutura como Serviço (IaaS), em que o provedor oferece máquinas virtuais, rede e armazenamento brutos; Plataforma como Serviço (PaaS), que abstrai o sistema operacional e entrega ambientes de execução gerenciados; e *Software* como Serviço (SaaS), em que a aplicação completa é entregue ao usuário pela Internet, sem instalação local (MELL; GRANCE, 2011; ARMBRUST *et al.*, 2010). A esse espectro somou-se mais recentemente o modelo *serverless* ou *Function as a Service* (FaaS), no qual a unidade de implantação é a função, executada sob demanda e cobrada por invocação, sem que o desenvolvedor gerencie servidores, modelo adotado na implantação em nuvem deste trabalho. Armbrust *et al.* (2010) já apontavam a ilusão de recursos infinitos disponíveis sob demanda, a eliminação de compromissos antecipados de capital e o pagamento pelo uso de curto prazo como as três novidades econômicas que definem a nuvem.

O mercado é liderado pela Amazon Web Services (AWS), Microsoft Azure e Google Cloud Platform (GCP), que juntas concentram mais de 60% da infraestrutura em

nuvem global (Synergy Research Group, 2024), com participação relevante de Oracle Cloud e Alibaba Cloud. Embora os serviços fundamentais sejam análogos entre os provedores (computação, armazenamento de objetos, banco de dados gerenciado, rede e CDN), há diferenças de maturidade, presença geográfica e portfólio que orientam a escolha. Todos organizam sua infraestrutura física em regiões (áreas geográficas independentes) subdivididas em zonas de disponibilidade (*Availability Zones*), *data centers* isolados em falha mas interconectados por baixa latência, sobre os quais se constroem a redundância e a alta disponibilidade. Este trabalho adota a AWS por sua maturidade em serviços de mídia e pela ampla documentação de arquiteturas de VoD de referência (Amazon Web Services, 2024b).

Escalabilidade e Rede na AWS

A escalabilidade na nuvem manifesta-se em dois eixos: o vertical (aumento de CPU/memória de uma instância) e o horizontal (adição de instâncias), sendo este último o que sustenta a escala de *streaming*. Na AWS, o escalonamento horizontal de máquinas virtuais (EC2) é automatizado por grupos de *auto scaling*, que ajustam o número de instâncias conforme métricas de utilização; no modelo *serverless* (Lambda), a escala é implícita, com o provedor instanciando cópias da função conforme a concorrência das requisições, sujeita, porém, a uma cota de concorrência que constitui um limite de capacidade concreto. As diretrizes do *AWS Well-Architected Framework* (Amazon Web Services, 2024a) sistematizam essas decisões em pilares de confiabilidade, eficiência de desempenho, otimização de custo, segurança e excelência operacional.

A rede é o substrato dessa escala e organiza-se em torno dos seguintes elementos:

- ***Virtual Private Cloud* (VPC):** rede virtual logicamente isolada, na qual a topologia da aplicação é definida. Subdivide-se em sub-redes públicas (com rota para a Internet) e privadas (acessíveis apenas internamente), distribuídas entre zonas de disponibilidade para tolerância a falhas. *Gateways* de Internet e de tradução de endereços de rede (*Network Address Translation*, NAT) controlam o tráfego de entrada e saída, e *security groups* e *network ACLs* atuam como *firewalls* declarativos.
- **Balanceamento de carga:** o *Elastic Load Balancer* distribui as requisições entre as instâncias saudáveis. O *Application Load Balancer* (ALB) opera na camada de aplicação (HTTP/HTTPS, camada 7), permitindo roteamento por conteúdo e terminação TLS, ao passo que o *Network Load Balancer* (NLB) opera na camada

de transporte (camada 4), priorizando vazão e baixa latência. Para arquiteturas *serverless*, o API Gateway cumpre o papel de *proxy* reverso gerenciado, expondo as funções como API HTTP com autenticação, limitação de taxa (*throttling*) e *cache*.

- ***Proxies* e *cache*:** além dos balanceadores, *proxies* reversos (na borda da aplicação) e camadas de *cache* em memória gerenciado (*ElastiCache* para Redis) reduzem a latência e a carga sobre os bancos primários, estratégia medida empiricamente neste trabalho.
- **Borda global:** o CloudFront (CDN da AWS) e o roteamento por DNS (*Route 53*) levam o plano de dados para perto do usuário, absorvendo a quase totalidade do *egress* de vídeo.

Na arquitetura avaliada neste trabalho, esses elementos materializam-se de forma seletiva: o plano de controle apoia-se no API Gateway (expondo as funções *Lambda* de consulta de catálogo e manifesto) e no ElastiCache/Redis, ao passo que o plano de dados é integralmente delegado ao CloudFront, que absorve o *egress* dos segmentos (Estudo 1). Os demais elementos (VPC com sub-redes, ALB/NLB e *auto scaling* de EC2) permanecem como a variante de implantação baseada em contêineres, não exercida no recorte *serverless* adotado.

A combinação desses elementos permite que o plano de controle escale de forma independente do plano de dados, separação que está no cerne da arquitetura avaliada.

Critérios de Seleção do Modelo de Computação

A escolha entre modelos de computação em nuvem depende do perfil da carga de trabalho. Em plataformas de *streaming*, coexistem tipicamente dois perfis opostos: operações de controle, curtas e de demanda variável (consulta de catálogo, entrega de manifesto), e operações de processamento, longas e intensivas em *hardware* (transcodificação). Os dois modelos disponíveis na AWS, o FaaS (Lambda) e o IaaS (EC2), diferem em dimensões que determinam sua adequação a cada perfil, sintetizadas na Tabela 4.

O FaaS é adequado a operações curtas, *stateless* e de demanda irregular: escala automaticamente com a concorrência, tem custo ocioso nulo e cobra estritamente pelo uso. Sua principal restrição é o limite de tempo de execução (15 minutos na AWS) e a ausência de aceleração por GPU. O IaaS, por sua vez, é adequado a cargas longas, com estado e intensivas em *hardware*: oferece controle total de ambiente e execução ilimitada, ao custo de gestão de capacidade e cobrança contínua enquanto a instância está ligada.

Tabela 4 – Comparação entre AWS Lambda e Amazon EC2

Dimensão	**AWS Lambda (FaaS)**	**Amazon EC2 (IaaS)**
Modelo de execução	Função efêmera, por requisição	Instância (VM) persistente
Escalabilidade	Implícita e automática, por concorrência	Manual ou por *auto scaling* de instâncias
Granularidade de cobrança	Por invocação + tempo (ms) $\times$ memória	Por tempo de instância ligada
Cold start	Sim (mitigável por *provisioned concurrency*)	Não (instância já aquecida)
Tempo máximo de execução	15 minutos	Ilimitado
Estado	*Stateless* (sem disco local persistente)	*Stateful* (disco e memória persistentes)
Aceleração por GPU	Não disponível	Sim (família g, NVENC)
Gargalo típico	Cota de concorrência	CPU, memória ou rede da instância
Custo em regime	Vantajoso em carga esporádica/*bursty*	Vantajoso em uso sustentado/alto *throughput*
Operação	Sem gestão de SO/*patching*	Exige gestão de SO, capacidade e *patching*

Fonte: Elaborado pelo autor.

A aplicação desses critérios à arquitetura implementada neste trabalho é detalhada na Seção 5.4.

Modelo de Custo de Operação em Nuvem

A análise de custo de uma plataforma de *streaming* em nuvem organiza-se em quatro categorias principais de cobrança. O *egress* de rede (o volume de dados transferido para fora da nuvem) é tipicamente o item dominante: cada espectador consome continuamente segmentos de vídeo, e a CDN consolida esse tráfego na borda com preços de volume, reduzindo o custo por GB. O *compute* divide-se entre o modelo por invocação (Lambda: cobrado por número de chamadas e por duração multiplicada pela memória alocada) e o modelo por tempo de instância (EC2: cobrado por hora-instância independentemente do uso efetivo); a escolha entre os dois modelos repercute diretamente no item de custo dominante em cada plano. O armazenamento em S3 é cobrado por GB armazenado acrescido do número de operações (PUT/GET), tornando as políticas de ciclo de vida (*lifecycle*) relevantes em catálogos extensos. A CDN (CloudFront) cobra por GB transferido nos PoPs e por número de requisições HTTP, com variação por região de

borda. Essas categorias estruturam a análise de custo conduzida com preços de tabela da AWS, região `us-east-1`, sem descontos por compromisso (Amazon Web Services, 2024b).

Referência de Preços dos Serviços AWS

Os serviços AWS empregados neste trabalho seguem o modelo *pay-as-you-go*, no qual o custo resulta do consumo efetivo sem compromisso mínimo. A Tabela 5 consolida os preços de tabela vigentes para a região `us-east-1`, modalidade *on-demand*, consultados em junho de 2026. Esses valores são os mesmos utilizados na análise de custo itemizado (Capítulo 7). Os preços de instâncias EC2 são representados pelo tamanho `xlarge` de cada família: `c7i` para cargas CPU-intensivas e `g4dn`/`g6` para aceleração por GPU (NVENC), famílias empregadas nos experimentos de transcodificação. O AWS Batch não incorre em custo adicional além das instâncias EC2 subjacentes.

Armazenamento de Objetos e Alternativas *Self-Hosted*

O armazenamento de vídeo bruto e das *renditions* empacotadas apoia-se no paradigma de armazenamento de objetos, no qual os dados são tratados como objetos imutáveis identificados por chave, acessados por API HTTP e dotados de alta durabilidade por replicação. Na AWS, esse papel cabe ao Amazon S3, que oferece classes de armazenamento por padrão de acesso e políticas de ciclo de vida (*lifecycle*) capazes de mover ou expirar objetos automaticamente, alavanca relevante de custo em catálogos extensos. O S3 atua, ainda, como origem (*origin*) da CDN.

No ambiente local deste trabalho, o papel do S3 é cumprido pelo MinIO, um armazenamento de objetos *self-hosted* compatível com a API do S3; é essa compatibilidade que permite à plataforma alternar entre o ambiente local e a nuvem sem alteração de código. O termo *self-hosted* refere-se à gestão pela própria equipe, não necessariamente à infraestrutura local, o que viabiliza cenários híbridos, multinuvem e de portabilidade entre provedores.

De aplicação mais secundária neste trabalho, registra-se a alternativa *self-hosted* de maior escala: o Ceph, sistema de armazenamento distribuído de código aberto, indicado para evitar a dependência de um provedor específico (*vendor lock-in*) ou para operar em nuvem privada. Seu subsistema RADOS distribui os dados entre nós sem ponto único de falha por meio do algoritmo CRUSH, que calcula a localização de cada objeto deterministicamente, dispensando um diretório central de metadados (WEIL *et al.*, 2006); por meio do *RADOS Gateway* (RGW), o Ceph expõe uma interface compatível

Tabela 5 – Preços de tabela dos serviços AWS (us-east-1, *on-demand*, consultados em jun. 2026)

Serviço	**Instância / classe**	**Preço de referência**
AWS Lambda	—	US$ 0,20/1M req.; US$ 0,0000167/GB-s
API Gateway	HTTP / REST	HTTP: US$ 1,00/1M; REST: US$ 3,50/1M
Amazon CloudFront	—	US$ 0,080/GB *egress*; US$ 0,012/10k req.
Amazon S3	Standard	US$ 0,023/GB-mês (armazen.); PUT: US$ 0,005/mil req.
Amazon S3	Glacier *Flex. Ret.*	US$ 0,004/GB-mês (originais brutos)
ElastiCache Redis	`cache.t3.micro`	US$ 0,017/h
Amazon DocumentDB	`db.r8g.large`	US$ 0,304/h
Amazon MQ	`mq.t3.micro`	US$ 0,044/h
RDS PostgreSQL	`db.t4g.small` (Single-AZ)	US$ 0,032/h
EC2 (*benchmark* CPU)	`c7i.xlarge`	US$ 0,179/h
EC2 (*benchmark* GPU)	`g4dn.xlarge` (T4)	US$ 0,526/h
EC2 (*benchmark* GPU)	`g6.xlarge` (L4)	US$ 0,805/h
Amazon CloudWatch	—	US$ 0,30/métrica-mês; US$ 0,50/GB *log*
AWS Batch	—	Sem custo adicional (usa EC2 subjacente)

Fonte: Amazon Web Services (2026). Acesso em: 19 jun. 2026.

com S3, tanto em *data center* próprio quanto sobre instâncias de IaaS. A avaliação de uma camada dessa natureza está fora do escopo experimental deste trabalho, que adota o S3/MinIO; o Ceph é mencionado como caminho de continuidade para cenários de nuvem privada ou de mitigação de *lock-in*.

3.11 Trabalhos Correlatos

Os trabalhos mais relevantes organizam-se em torno dos eixos que estruturam esta pesquisa (adaptação de *bitrate* e QoE; distribuição e custo; codificação), tendo como pano de fundo as revisões que delineiam o estado da arte. Para cada eixo, indica-se o que esta pesquisa aproveita desses trabalhos e em que ponto deles se separa.

No plano das revisões, Timmerer *et al.* (2025) apresentam um panorama do HAS, da codificação ao consumo pelo usuário final, com foco em algoritmos ABR, QoE e eficiência energética; Alsader, Barakabitze e Mkwawa (2025) enfocam os algoritmos ABR, o *edge computing* e a QoE rumo às redes 6G; Bentaleb *et al.* (2026) sistematizam a evolução do *streaming* de baixa latência; e Moina-Rivera *et al.* (2024) revisam os desafios da codificação de vídeo em nuvem. Essas obras situam o problema e fundamentam o referencial. São, porém, revisões de literatura: cobrem a área em largura, sintetizando resultados de terceiros, sem executar experimentos próprios. Por isso não servem de contraponto experimental, isto é, de base de comparação direta para os resultados medidos nesta pesquisa; esse papel cabe aos trabalhos experimentais dos eixos a seguir.

Um primeiro eixo propõe ou avalia mecanismos de adaptação e a QoE deles decorrente. Yuan *et al.* (2022) introduzem um algoritmo ABR com predição de vazão baseada em atenção e aprendizado por reforço; Zhao e Pan (2023) acoplam, em decisão conjunta, o escalonamento multicaminho e a adaptação de *bitrate*; e Asan (2024) conduz avaliações subjetivas de QoE sobre trocas de resolução, segundo a Recomendação ITU-T P.910. Esses trabalhos otimizam ou medem uma faceta da adaptação, ao passo que esta pesquisa não propõe um novo ABR: mede os *trade-offs* de capacidade, custo e QoE produzidos pelas configurações da plataforma. Deles, este trabalho aproveita o conjunto consolidado de métricas de QoE (tempo de partida, *rebuffering* e oscilações de qualidade), que a plataforma instrumenta no *player*.

Um segundo eixo trata da distribuição na borda e de seus custos. Xenakis (2024) formulam a seleção conjunta de *bitrate* e *caching* de borda em redes MEC com *slicing*, avaliada por emulação; Rodrigues-Filho *et al.* (2024) demonstram o uso

de *Content Steering* com orquestração de contêineres no *continuum* borda-nuvem; e Palmer (2023) propõe o compartilhamento de informação entre as camadas de aplicação e transporte para reduzir interrupções de reprodução. São abordagens de otimização de *caching*, roteamento ou entrega, sem a modelagem empírica de custo (*egress*) e de capacidade (concorrência *serverless*) conduzida neste trabalho. Deste eixo, aproveita-se a caracterização do papel do *cache* e da borda na entrega, que fundamenta o desenho do módulo de distribuição.

No eixo de codificação, Bukhari *et al.* (2022) realizam um estudo experimental de custo e viabilidade da transcodificação, medindo o tempo de codificação, o uso de CPU e a qualidade ao variar *codecs*, configurações de máquina virtual e sistemas operacionais sobre instâncias EC2, o correlato mais próximo do Estudo 2 desta pesquisa. A distinção está no método de comparação: este trabalho adota o BD-rate a qualidade constante e contrasta a codificação por *software* com a aceleração por GPU (NVENC), dimensão ausente naquele estudo. É o correlato com que o Estudo 2 dialoga mais diretamente: o desenho experimental aproveita sua estratégia de variar *codec* e instância, e o contraste com seus resultados é retomado na discussão do Estudo 2.

Por fim, Suman, Moon e Choi (2022) caracterizam a QoS de Netflix, Amazon Prime Video e YouTube a partir da classificação de tráfego criptografado, sem acesso ao código ou à configuração interna dessas plataformas. É o oposto metodológico desta pesquisa: enquanto aquele estudo observa caixas-pretas comerciais a partir de fora, este disponibiliza uma plataforma aberta e instrumentada, cujos parâmetros são diretamente manipuláveis e mensuráveis. Quatro elementos concretos conferem essa abertura: (i) o código-fonte integral dos módulos, publicado em repositórios públicos (Apêndice C); (ii) a composição por tecnologias de código aberto em todas as camadas (FFmpeg na transcodificação, MinIO no armazenamento, Redis, RabbitMQ e MongoDB nos serviços); (iii) os arquivos de implantação (Docker Compose e infraestrutura como código) que permitem recriar o ambiente completo; e (iv) os *scripts* de *benchmark* que reproduzem os experimentos do Capítulo 7.

A Tabela 6 sintetiza esse posicionamento, comparando um representante experimental de cada eixo com a proposta desta pesquisa.

Tabela 6 – Posicionamento da proposta frente a um representante experimental de cada eixo

Dimensão	**Suman, Moon e Choi (2022)**	**Yuan *et al.* (2022)**	**Xenakis (2024)**	**Bukhari *et al.* (2022)**	**Este trab.**
Medição empírica (vs. proposta/otimização)	Sim	Não*	Não*	Sim	Sim
Plataforma aberta e instrumentada	Não	Não	Não	Parc.	**Sim**
Pipeline completo (ingestão→reprodução)	Não	Não	Não	Não	**Sim**
Mede QoE/QoS de reprodução	Sim	Sim	Sim	Não	**Sim**
Mede custo e capacidade em nuvem	Não	Não	Parc.	Sim	**Sim**
Eficiência de *codec* (BD-rate)	Não	Não	Não	Parc.	**Sim**
Medição em escala reduzida + projeção	Não	Não	Não	Não	**Sim**

Fonte: Elaborado pelo autor. Notas: *Proposta de algoritmo ou de otimização avaliada por simulação/emulação. Parc.: parcial.

As dimensões da tabela sintetizam os critérios de comparação. Medição empírica distingue os trabalhos que medem sistemas reais daqueles que propõem algoritmos avaliados por simulação ou emulação. Plataforma aberta e instrumentada indica se o artefato avaliado tem código e configuração acessíveis e expõe métricas (os quatro elementos concretos discutidos acima). *Pipeline* completo verifica se a avaliação cobre todas as etapas, da ingestão à reprodução, ou apenas um estágio isolado. Mede QoE/QoS de reprodução e mede custo e capacidade em nuvem registram quais famílias de métricas cada estudo coleta: as de experiência do espectador e as de operação, respectivamente. Eficiência de *codec* indica o uso de comparação a qualidade constante (BD-rate). Por fim, medição em escala reduzida + projeção denota a estratégia metodológica de medir constantes unitárias em pequena escala e projetar o comportamento em escala de produção (Capítulo 6).

A leitura conjunta evidencia a lacuna que esta pesquisa preenche. Cada correlato aprofunda uma fatia do problema (a adaptação de *bitrate*, o *caching* de borda, a transcodificação ou a caracterização externa de plataformas comerciais), mas nenhum entrega uma plataforma aberta, instrumentada e parametrizável que meça, de ponta a ponta e de forma reprodutível, os *trade-offs* de capacidade, custo e QoE de um *pipeline* completo de *streaming*. É nesse vazio, o de resultados experimentais reprodutíveis sobre uma arquitetura aberta e integral, que se posiciona a contribuição deste trabalho.

4 ESTUDO DE CASO

Este capítulo delimita concretamente o caso particular que o trabalho investiga e que parametriza toda a parte experimental: o contexto de aplicação que motiva a plataforma e a motivação da pesquisa a ele associada. É a partir deste caso, que a arquitetura proposta (Capítulo 5) e a metodologia experimental (Capítulo 6) são instanciadas, e que os dois experimentos (Seção 6.8) são desenhados. O corpus experimental (os 14 clipes sobre os quais as codificações e medições são executadas) é apresentado no Estudo 2 (Seção 6.8), onde seu papel metodológico é detalhado.

Relevância do setor e justificativa do caso. A escolha do setor farmacêutico como caso de aplicação não é arbitrária: trata-se de um dos maiores e mais influentes segmentos da economia global contemporânea, que movimenta cerca de US$ 1,48 trilhão por ano (cifra comparável ao PIB de economias como a da Espanha) e cuja receita praticamente quadruplicou entre 2001 e 2022. No Brasil, o mesmo movimento de expansão se reproduz, com as receitas do setor crescendo cerca de 3,2 vezes na década de 2012 a 2022 (STACCIARINI, 2023). Essa magnitude, aliada a uma audiência ampla e geograficamente distribuída (força de vendas, rede de lojas e profissionais de saúde), caracteriza um cenário de volume e concorrência que justifica o dimensionamento de capacidade e custo conduzido neste trabalho (Tabela 11). A escala potencial desse público é expressiva: somente a RD Saúde, maior rede de farmácias do país, encerrou 2023 com cerca de 47,6 milhões de clientes ativos (RD Saúde, 2024), indicador da ordem de grandeza nacional a que uma plataforma do setor pode ser exposta.

O marketing como vetor do conteúdo audiovisual. Um traço definidor do setor é a centralidade do marketing. Segundo Stacciarini (2023), estima-se que o investimento em divulgação chegue a representar de 25% a 35% das receitas, podendo superar em cerca de duas vezes o gasto em pesquisa e desenvolvimento; no Brasil, esse esforço fez o setor farmacêutico ascender do 12.º ao 5.º lugar entre os maiores anunciantes, um crescimento de aproximadamente 177% no investimento publicitário. Mais relevante para este trabalho é o deslocamento desse investimento da mídia tradicional para canais digitais (internet, plataformas de vídeo e segmentação algorítmica), movimento que sustenta diretamente a demanda por uma plataforma de *Video on Demand* própria, capaz de distribuir, em escala e sob controle, o conteúdo comercial e promocional descrito a seguir.

Contexto de aplicação: o setor farmacêutico. O caso de uso que norteia este trabalho deriva de uma plataforma de *Video on Demand* concebida para atender, principalmente, ao

setor farmacêutico. Nesse setor, o vídeo cumpre dois papéis bem definidos, que delimitam tanto o perfil de consumo quanto as características do conteúdo:

- **Conteúdo comercial e promocional:** peças de curta duração, como comerciais, lançamentos de produtos e campanhas dirigidas à força de vendas e a pontos de atendimento. São vídeos de alta qualidade visual, curtos e com produção elaborada, análogos a *reels* e *stories* quanto à duração e à densidade de informação por segundo.
- **Conteúdo educativo e de treinamento:** material de educação continuada para profissionais de saúde, treinamentos de *compliance* e regulatórios, divulgação científica e capacitação técnica. São vídeos mais longos (palestras, aulas e demonstrações), com movimento moderado e ênfase na inteligibilidade.

Essa dualidade define a magnitude e a natureza do conteúdo efetivamente testado: o corpus experimental foi composto de modo a representar esses dois gêneros (comerciais curtos de alta fidelidade e vídeos educativos mais longos), pois é a combinação deles que caracteriza a carga real da plataforma. O contexto farmacêutico também explica requisitos não funcionais relevantes: alta disponibilidade (campanhas e eventos com picos de audiência concentrada); integridade e rastreabilidade do conteúdo, exigência reconhecida na cadeia farmacêutica (SILVA; MATTOS, 2019); e uma audiência delimitada, porém exigente em qualidade de experiência (*Quality of Experience*, QoE), como a força de vendas e os profissionais de saúde, para quem a educação continuada por meios audiovisuais digitais é cada vez mais central (LANGENDORF; KHALID, 2024). A volumetria quantitativa desse cenário (acessos por dia, concorrência de pico e requisições) é consolidada na Tabela 11 (Capítulo 6), que alimenta o modelo de custo e capacidade.

Dimensionamento do problema de distribuição. Quatro grandezas mostram por que esse cenário configura, de fato, um problema de distribuição de vídeo. Quanto ao público potencial, a audiência direta da plataforma (força de vendas, rede de lojas e profissionais de saúde parceiros) é da ordem de dezenas de milhares de usuários ativos por dia; o teto de exposição, porém, é dado pelo próprio setor: somente a RD Saúde encerrou 2023 com cerca de 47,6 milhões de clientes ativos (RD Saúde, 2024), de modo que campanhas abertas ao consumidor final podem elevar a demanda em ordens de magnitude. Quanto ao catálogo, o ritmo de publicação assumido (cerca de 30 *uploads* administrativos por dia, Tabela 11) produz um acervo da ordem de milhares de títulos já no primeiro ano de operação. Quanto ao tamanho de cada item, os arquivos brutos têm em média 300 MB

(vídeos de aproximadamente cinco minutos), e cada título é transcodificado em múltiplas representações de qualidade (Capítulo 5), o que multiplica o volume armazenado e entregue. Quanto às tendências de carga, projetam-se cerca de 150 mil acessos diários, com concorrência média de cerca de 500 espectadores e picos de aproximadamente 7,5 mil espectadores simultâneos em janelas de campanha de cerca de uma hora, o que representa um salto de cerca de 130 para cerca de 1.875 requisições de segmento por segundo. Em termos de banda, um pico dessa magnitude, servido com representações típicas de alta definição (3 Mbit/s por espectador), exige vazão agregada da ordem de 20 Gbit/s, patamar que um servidor único não alcança sem uma camada de distribuição dimensionada para isso. É esse regime, de catálogo crescente, itens volumosos e carga fortemente concentrada em picos, que caracteriza o problema de distribuição tratado neste trabalho.

5 PROTÓTIPO ARQUITETURAL

Este capítulo detalha o protótipo arquitetural desenvolvido neste trabalho. A solução proposta consolida as melhores práticas identificadas na revisão de literatura e nos trabalhos correlatos, materializando-se em uma plataforma experimental para avaliação de desempenho e QoE em *streaming* de vídeo. A arquitetura foi projetada para ser modular, escalável e baseada inteiramente em tecnologias de código aberto, permitindo a investigação sistemática dos parâmetros de transmissão. As seções iniciais descrevem a concepção modular dos cinco módulos funcionais; a Seção 5.2 reconcilia essa concepção com a implementação efetiva, que evoluiu para uma plataforma de microsserviços poliglota (com serviços escritos em diferentes linguagens) a fim de atender aos requisitos de escalabilidade e isolamento do caso de uso (Capítulo 4). Coerentemente com a metodologia mista do trabalho, o capítulo organiza-se em duas fases: a fase exploratória, que estabelece a concepção modular proposta, e a fase descritiva-explicativa, que descreve a implementação efetiva submetida à medição.

5.1 Requisitos da Arquitetura

Os requisitos foram definidos pelo autor no exercício de levantamento de requisitos do caso de uso (Capítulo 4), a partir de três fontes: as quatro questões operacionais de pesquisa (Seção 1.1), que determinam o que a plataforma precisa medir; o cenário e o dimensionamento do estudo de caso, que determinam a carga e o conteúdo a suportar; e as práticas consolidadas na RSL e no referencial teórico (Capítulos 2 e 3), que determinam os padrões técnicos a adotar. Cada requisito é acompanhado da justificativa que o vincula a essas fontes.

Requisitos Funcionais (RF)

- **RF01 - Ingestão Multi-formato:** O sistema deve aceitar arquivos de vídeo em contêineres padrão (MP4, MKV, MOV) para processamento em lote. Justifica-se pelo perfil de conteúdo do estudo de caso, cujos *uploads* administrativos chegam em formatos brutos heterogêneos.
- **RF02 - Transcodificação Multi-codec:** O sistema deve gerar representações de vídeo utilizando os codecs H.264, H.265 e AV1, permitindo a comparação direta de eficiência. Justifica-se pela questão de pesquisa (iii): comparar a eficiência de

codificadores exige gerá-los na própria plataforma, sob parâmetros idênticos. Os três não se destinam a gerar diferentes qualidades, papel cumprido pela escada de representações dentro de um mesmo *codec*; eles são o objeto de comparação do Estudo 2: três gerações de compressão (H.264 como base de compatibilidade universal, H.265 e AV1 como sucessores de maior eficiência), cujo contraste de eficiência, velocidade e custo responde às questões (iii) e (iv).

- **RF03 - Empacotamento CMAF:** O sistema deve utilizar o formato CMAF para garantir compatibilidade simultânea com DASH e HLS com baixa redundância de armazenamento. Justifica-se pela economia de armazenamento: um único conjunto de segmentos serve aos dois protocolos dominantes, sem duplicar o catálogo.
- **RF04 - Distribuição Otimizada:** O servidor deve suportar conexões HTTP/2 e HTTP/3 e implementar políticas de *cache* configuráveis. Justifica-se pelo dimensionamento do Capítulo 4: os picos de requisições concorrentes fazem de *cache* e multiplexação as principais alavancas de capacidade.
- **RF05 - Reprodução Instrumentada:** O *player* deve adaptar a qualidade do vídeo dinamicamente e registrar métricas de QoE (*startup time*, *rebuffering*, *bitrate* médio) em tempo real. Justifica-se pelas questões (i) e (ii): sem métricas de QoE no *player*, capacidade e custo não podem ser relacionados à experiência do espectador.
- **RF06 - Coleta de Telemetria:** O sistema deve centralizar *logs* e métricas de desempenho dos servidores e dos clientes para análise posterior. Justifica-se pela rastreabilidade metodológica: as análises do Capítulo 7 dependem de séries de métricas centralizadas.

Requisitos Não Funcionais (RNF)

- **RNF01 - Código Aberto:** A *stack* de aplicação deve ser baseada em *software open source*, garantindo a reprodutibilidade e a ausência de custos de licenciamento, com o plano de dados e a infraestrutura de execução delegáveis a serviços gerenciados (por exemplo, CDN e funções *serverless*) sem alterar a lógica dos serviços. Justifica-se pela lacuna que motiva o trabalho: apenas uma *stack* aberta permite observar e ajustar os mecanismos internos medidos.
- **RNF02 - Modularidade e Isolamento:** Os componentes devem ser desacoplados e executados em contêineres isolados, facilitando a manutenção e a substituição de módulos. Justifica-se pelo controle de validade de isolamento de módulos (Capítulo 6): testar cada etapa isoladamente exige componentes desacoplados.

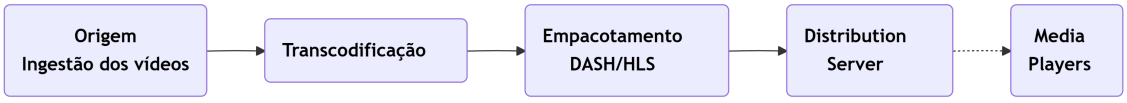
- **RNF03 - Portabilidade:** O sistema deve ser agnóstico à infraestrutura subjacente, podendo ser implantado em ambientes locais ou em nuvem pública via orquestração de contêineres. Justifica-se pela estratégia de medição em escala reduzida e projeção: o mesmo artefato deve executar no ambiente local e na nuvem.
- **RNF04 - Configurabilidade Experimental:** Parâmetros críticos (GOP size, duração de segmento, algoritmo ABR) devem ser expostos via configuração para permitir a variação de cenários de teste. Justifica-se pelo desenho experimental: variar cenários exige expor os parâmetros como configuração, e não como código.

5.2 Concepção, Implementação e Decisões Tecnológicas

A arquitetura proposta fundamenta-se no modelo de referência para sistemas HAS, alinhando-se às especificações MPEG-DASH e HLS. A estrutura do sistema é composta por cinco módulos funcionais interconectados, conforme ilustrado na Figura 5, materializados em microsserviços e justificados pelas decisões tecnológicas descritas nesta seção.

O fluxo de processamento, ilustrado da esquerda para a direita na Figura 5, inicia-se no Módulo de Ingestão (origem), que recebe o conteúdo bruto enviado pelos administradores. Em seguida, o Módulo de Transcodificação processa o vídeo, gerando múltiplas representações de qualidade (*bitrate ladder*). O Módulo de Empacotamento fragmenta essas representações em segmentos e gera os manifestos DASH e HLS. A entrega é realizada pelo Módulo de Distribuição (*Distribution Server*), que atua como servidor de origem e *cache* dos artefatos empacotados. Por fim, o Módulo de Reprodução (*Media Players*) executa a lógica de adaptação no cliente e coleta métricas de QoE. As setas cheias da figura indicam o fluxo de produção do conteúdo, empurrado de um módulo ao seguinte a cada novo vídeo; a seta tracejada marca a fronteira de consumo, na qual a iniciativa se inverte: são os *players* que puxam manifestos e segmentos por HTTP, no ritmo da reprodução.

Figura 5 – Arquitetura Geral do Sistema de Streaming Proposto

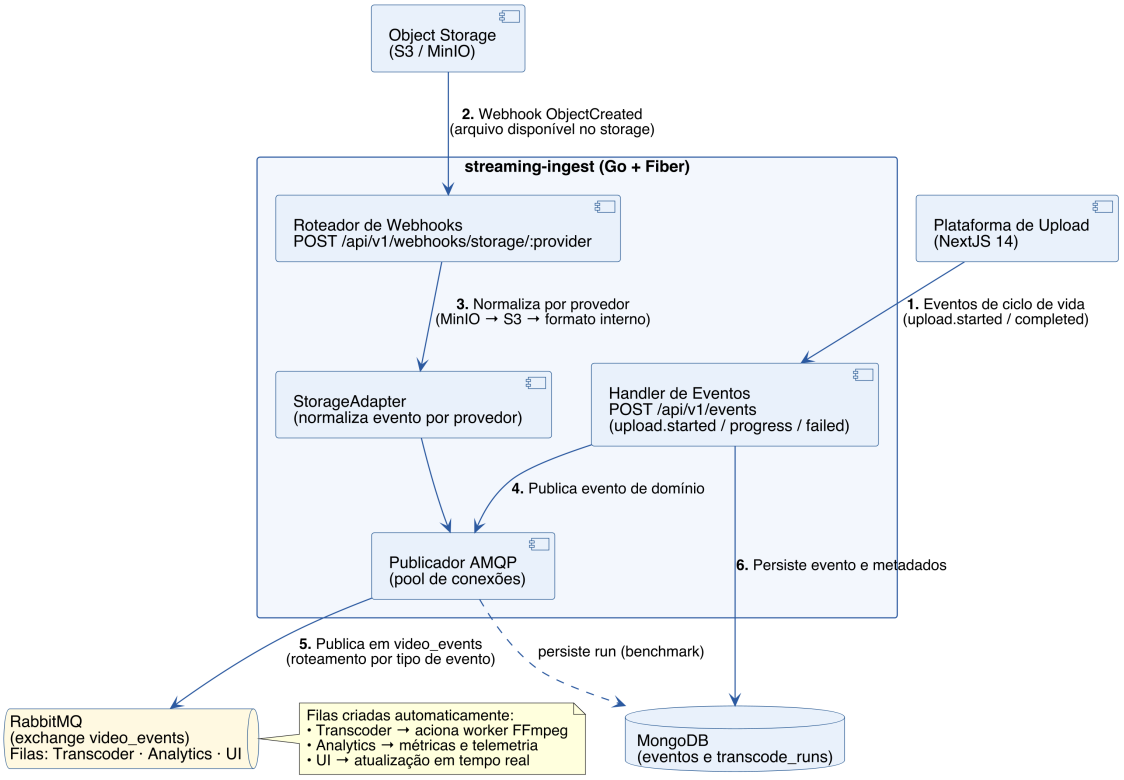


Fonte: Elaborado pelo autor.

Módulo de Ingestão

O módulo de ingestão opera de forma orientada a eventos: ao detectar um novo arquivo no *Object Storage*, um *webhook* `ObjectCreated` é enviado ao serviço *streaming-ingest* (Go + Fiber), que normaliza o evento por meio do `StorageAdapter`: as notificações específicas de cada provedor de armazenamento (os *webhooks* do MinIO e do S3 têm formatos distintos) são convertidas em um evento interno único, independente de provedor, que é então publicado no *exchange* `video_events` do RabbitMQ. Adicionalmente, a plataforma de *upload* notifica diretamente o serviço com eventos de ciclo de vida (`upload.started`, `upload.completed`). O fluxo completo, do gatilho de detecção à publicação nas filas, está esquematizado na Figura 6.

Figura 6 – Módulo de Ingestão

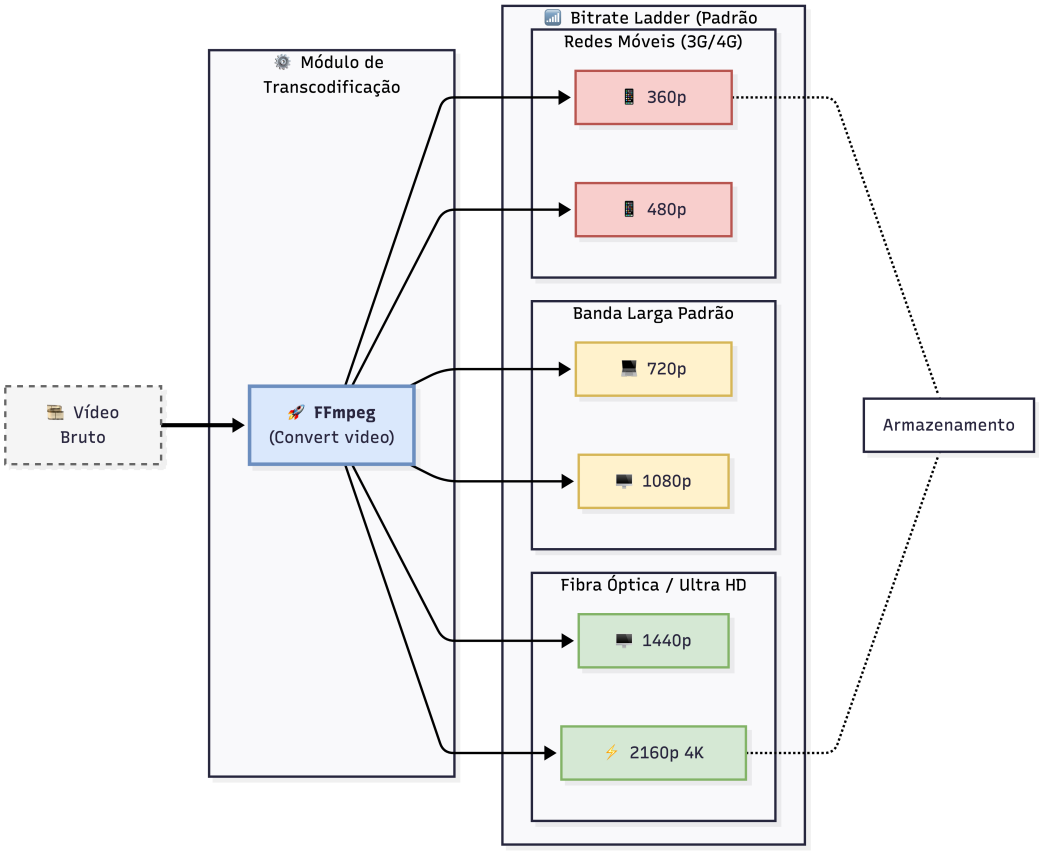


Fonte: Elaborado pelo autor.

Módulo de Transcodificação

Para a etapa de transcodificação, ilustrada na Figura 7, adotou-se o FFmpeg devido à sua flexibilidade e suporte aos codecs alvo deste estudo. Diferente de soluções comerciais "caixa-preta", o FFmpeg permite o controle preciso sobre os parâmetros de codificação, essencial para a análise científica. Na prática, esse controle é exercido

Figura 7 – Módulo de Transcodificação



Fonte: Elaborado pelo autor.

por linha de comando: para cada codificação, o *worker* de transcodificação monta explicitamente o codificador (`libx264`, `libx265`, `libsvtav1` ou as variantes NVENC), o modo de qualidade (CRF na codificação por *software*, CQ no NVENC), o *preset* de velocidade, a resolução e o tamanho do GOP, parâmetros que soluções gerenciadas não expõem ou fixam internamente. Como os comandos são código versionado no serviço `streaming-transcode` e os parâmetros ficam registrados em cada *job*, toda codificação é exatamente reprodutível, condição de que dependem as comparações a qualidade constante do Estudo 2.

Configuração da Bitrate Ladder

A *bitrate ladder* foi definida com base nas recomendações da Apple (*HLS Authoring Specification*). A configuração, apresentada na Tabela 7, busca cobrir um amplo espectro de condições de rede, desde conexões móveis instáveis até banda larga de alta velocidade.

Tabela 7 – Bitrate Ladder Proposta para o Protótipo

Resolução	**Bitrate Vídeo**	**Bitrate Áudio**	**Uso Típico**
360p (640x360)	800 kbps	128 kbps	Redes 3G/4G lentas
480p (854x480)	1400 kbps	128 kbps	Conexões limitadas
720p (1280x720)	2800 kbps	192 kbps	HD padrão
1080p (1920x1080)	5000 kbps	192 kbps	Full HD
1440p (2560x1440)	8000 kbps	256 kbps	2K (opcional)
2160p (3840x2160)	16000 kbps	256 kbps	4K UHD

Fonte: Adaptado de Apple HLS Authoring Specification.

Essa configuração é parametrizável, permitindo ajustes experimentais para avaliar o impacto de diferentes granularidades de adaptação na QoE. Os experimentos do Capítulo 7, registre-se, não exercitam a escada completa: o estudo de distribuição fixa a representação de 1080p (cerca de 5 Mbps de vídeo, $\sim$5,4 Mbps com áudio) para isolar a capacidade de entrega, ao passo que o estudo de *codec* varre pontos de *bitrate* em 480p, 720p e 1080p para levantar as curvas *rate-distortion*. A escada da Tabela 7 é, portanto, a referência de produção, da qual cada experimento adota o recorte adequado ao seu objetivo.

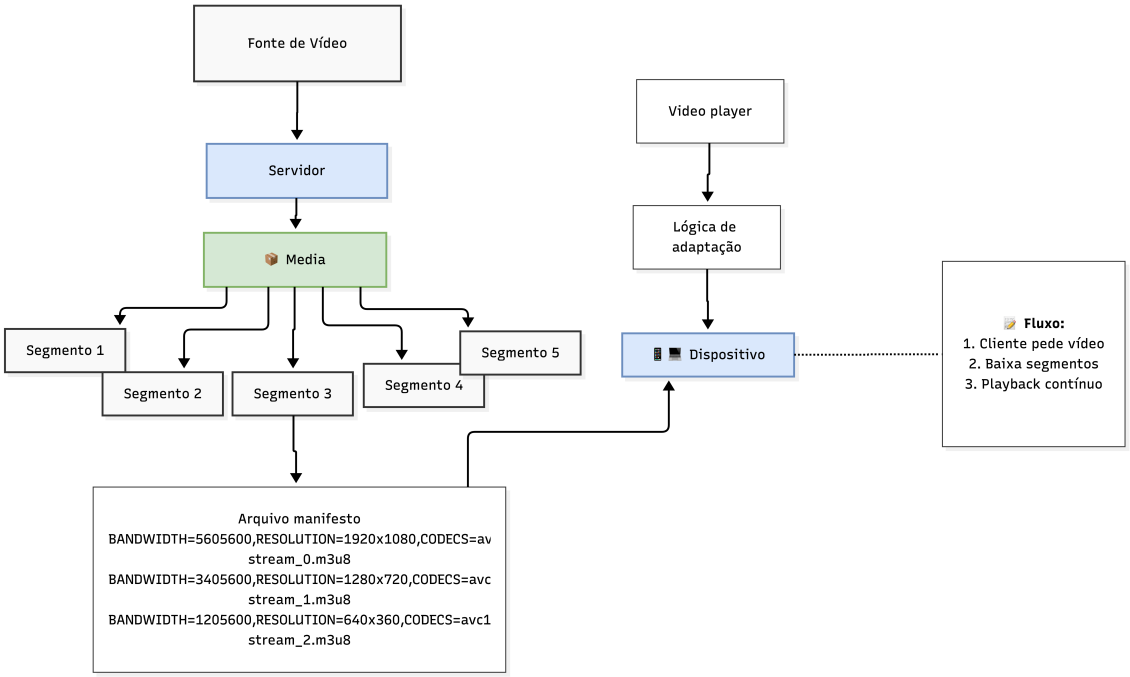
Módulo de Empacotamento

O processo de empacotamento, demonstrado na Figura 8, utiliza o Shaka Packager para gerar segmentos compatíveis com o padrão CMAF. Esta escolha técnica alinha-se à tendência de convergência de formatos discutida no Referencial Teórico, eliminando a necessidade de duplicar o armazenamento para suportar DASH e HLS separadamente.

As configurações aplicadas incluem:

- **Segment Duration:** Fixado em 4 segundos, valor que oferece um compromisso equilibrado entre eficiência de codificação e latência de início. Trata-se de uma decisão de projeto: o *survey* de Bentaleb *et al.* (2026) emprega segmentos de 5 segundos em seus experimentos e aponta durações menores para baixa latência.
- **Manifest Generation:** Geração simultânea de arquivos `.mpd` para o protocolo (DASH) e `.m3u8` para o protocolo HLS, referenciando os mesmos arquivos de mídia `.m4s`.

Figura 8 – Módulo de Empacotamento

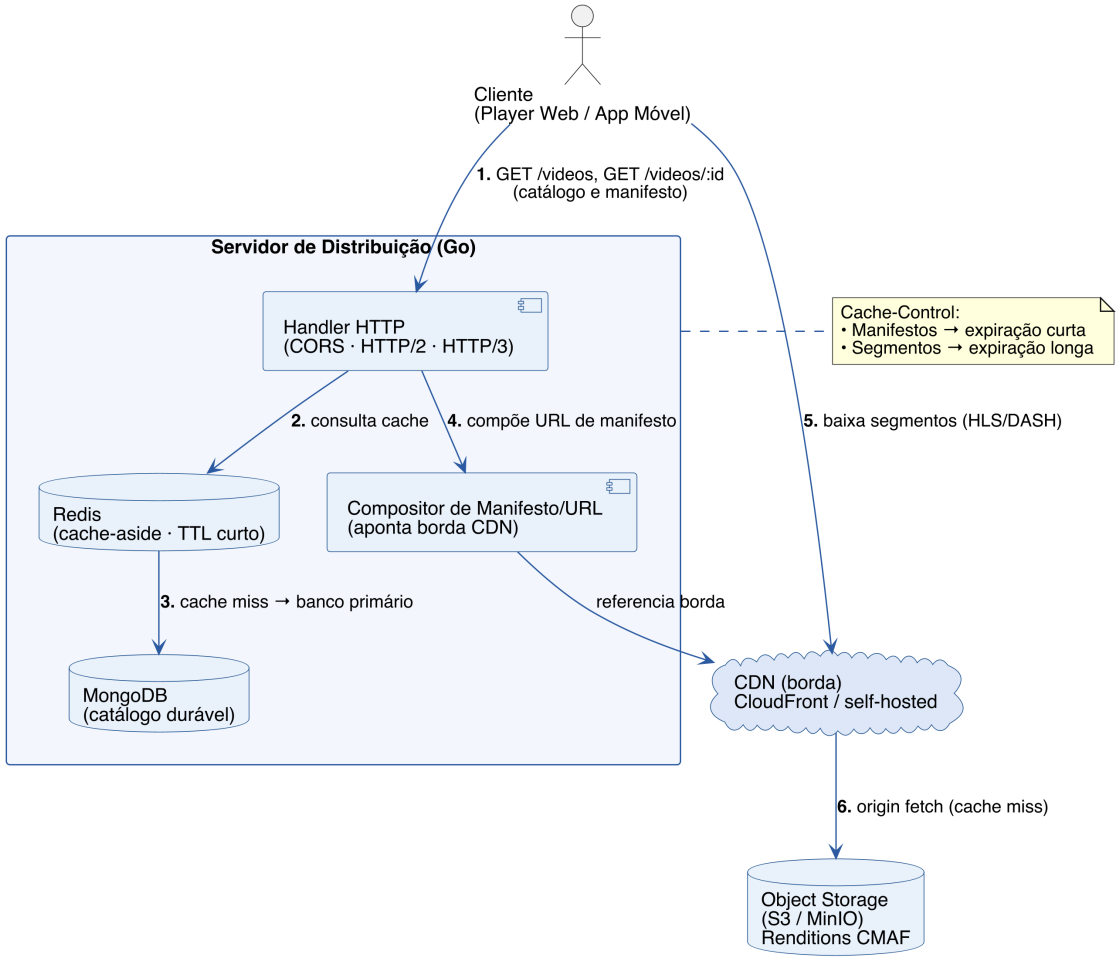


Fonte: Elaborado pelo autor.

Módulo de Distribuição

O componente de distribuição foi implementado como um serviço próprio em Go, escolhido por seu modelo de concorrência leve (*goroutines*) e alto desempenho em I/O. A topologia deste módulo pode ser observada na Figura 9. A configuração do serviço foi otimizada para *streaming* de mídia, incluindo:

Figura 9 – Módulo de Distribuição



Fonte: Elaborado pelo autor.

O fluxo numerado na figura resume a operação do serviço: (1) o cliente requisita o catálogo e o manifesto às rotas HTTP (`GET /videos`, `GET /videos/:id`); (2) o *handler* consulta primeiro o *cache* Redis, no padrão *cache-aside* com TTL curto; (3) em caso de *cache miss*, recorre ao MongoDB, o catálogo durável, e repovoa o *cache*; (4) o compositor de manifesto monta as URLs apontando os segmentos para a borda (CDN); (5) o cliente baixa os segmentos HLS/DASH diretamente da CDN, sem passar pelo servidor de distribuição; e (6) apenas em *cache miss* da borda a requisição desce à origem, o armazenamento de objetos (S3/MinIO) com as *renditions* CMAF. Essa separação faz o servidor Go atender somente o plano de controle (catálogo e manifestos), enquanto o volume de mídia flui pela CDN.

- **CORS:** Habilitado para permitir o acesso aos recursos por *players* hospedados em diferentes domínios.

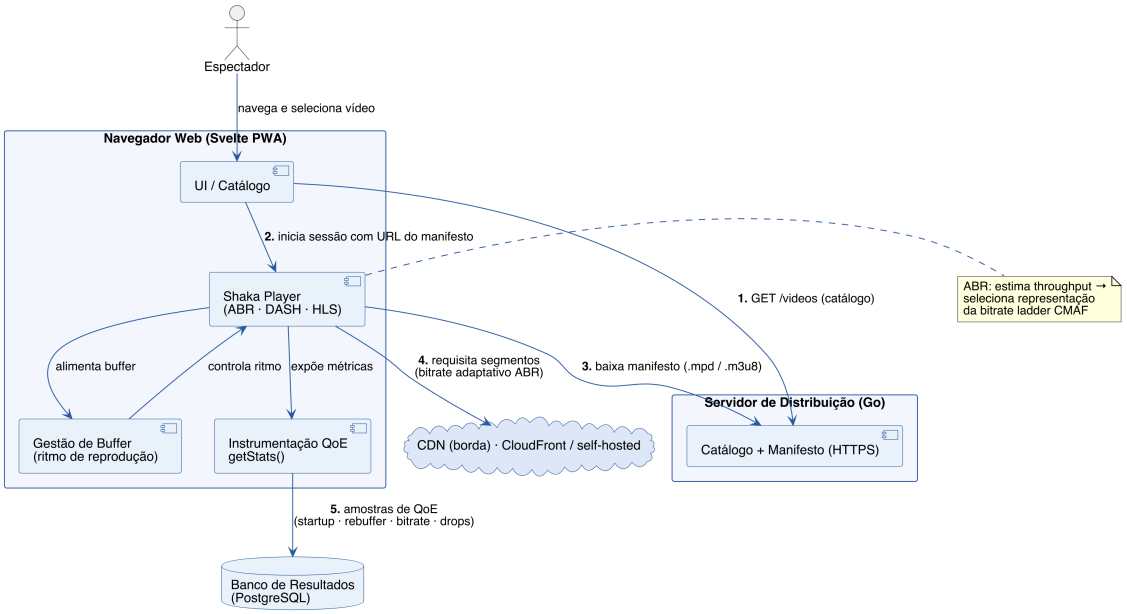
- **Cache-Control:** Definição de cabeçalhos de expiração curta para manifestos (garantindo atualizações frequentes em cenários live) e longa para segmentos de mídia (maximizando o cacheability).
- **Protocolos:** Habilitação de HTTP/2 e suporte experimental a HTTP/3 para avaliação de desempenho em redes com perda de pacotes.

A dualidade entre origem e *cache* é resolvida por composição: a topologia de CDN é emulada no nível de serviço, instanciando múltiplos contêineres do próprio serviço de distribuição configurados como *Edge Servers* (com *cache*) e *Origin Servers*, o que permite exercitar cenários de *cache* hierárquico. A rede entre os componentes, por sua vez, não é emulada: por controle de validade (Seção 6.3), as medições locais ocorrem na rede interna de contêineres, sem injeção artificial de latência ou perda, de modo que as métricas reflitam o sistema sob teste, e não um modelo de rede. As condições de última milha do espectador ficam fora do escopo e são tratadas como limitação: a QoE medida constitui um piso da experiência real. No ambiente de nuvem, a borda é exercida por uma CDN real (CloudFront), dispensando emulação.

Módulo de Reprodução

O módulo de reprodução, detalhado na Figura 10, é executado no navegador web do usuário e é responsável por orquestrar a sessão de *streaming*. O fluxo indicado na figura é o seguinte: (1) o cliente web requisita o manifesto ao módulo de distribuição; (2) o *player* interpreta as representações disponíveis na escada de qualidades; (3) a cada *download*, o algoritmo ABR estima a largura de banda e seleciona a representação do próximo segmento; (4) os segmentos são requisitados no ritmo de reprodução e anexados ao *buffer* do navegador por meio das *Media Source Extensions* (MSE); e (5) em paralelo, o coletor de métricas registra os eventos da sessão e os reporta ao serviço de telemetria.

Figura 10 – Módulo de Reprodução



Fonte: Elaborado pelo autor.

Para a reprodução, adotou-se o Shaka Player (Google), que suporta nativamente DASH e HLS e expõe instrumentação rica de métricas de QoE via `getStats()`, o que o torna a âncora de fidelidade de QoE no estudo de distribuição (Seção 6.8). A implementação própria deste trabalho está no cliente web que o hospeda (`streaming-web-client`, Svelte/PWA): cabe a ele configurar o *player* (parâmetros de *buffer* e de ABR expostos como configuração, RNF04), implementar o coletor que combina os eventos do *player* com amostragens periódicas de `getStats()`, e reportar as métricas ao serviço de telemetria. As bibliotecas de código aberto alternativas (dash.js, hls.js, ExoPlayer) e os algoritmos de adaptação (ABR) são discutidos no referencial teórico (Seção 3.8), não se repetindo aqui.

Instrumentação para Coleta de Métricas

O *player* é instrumentado para coletar as seguintes métricas de QoE, que são enviadas periodicamente a um servidor de telemetria:

- ***Startup Time*:** Tempo decorrido entre o clique no botão de reprodução e o início efetivo da exibição do vídeo.
- ***Rebuffering Events*:** Número, duração e instantes temporais de ocorrência de eventos de parada para carregamento (*stalling*).
- ***Bitrate* Médio e Oscilações:** Taxa de bits média da sessão e frequência de mudanças de qualidade.

- **Nível do *Buffer*:** Ocupação do *buffer* de reprodução ao longo do tempo.
- ***Throughput* Estimado:** Largura de banda estimada pelo algoritmo ABR.
- **Quadros Descartados (*Dropped Frames*):** Indicador de sobrecarga de decodificação.

Containerização e Orquestração

Para garantir a reprodutibilidade dos experimentos e facilitar o *deployment* em diferentes ambientes, toda a arquitetura é containerizada utilizando Docker. Cada módulo é encapsulado em um contêiner independente, permitindo o isolamento de dependências e a escalabilidade horizontal.

A orquestração dos contêineres é realizada através do Docker Compose para ambientes de desenvolvimento e teste local. Em nuvem, a orquestração em maior escala apoia-se no modelo *serverless* da AWS (Lambda e EC2/Batch), conforme detalhado na Seção 5.2.

Telemetria e Monitoramento

O protótipo incorpora um módulo de monitoramento, cujas interações são apresentadas na Figura 11, onde os dados são coletados pelos serviços e persistidos em banco de dados para visualização em dashboards.

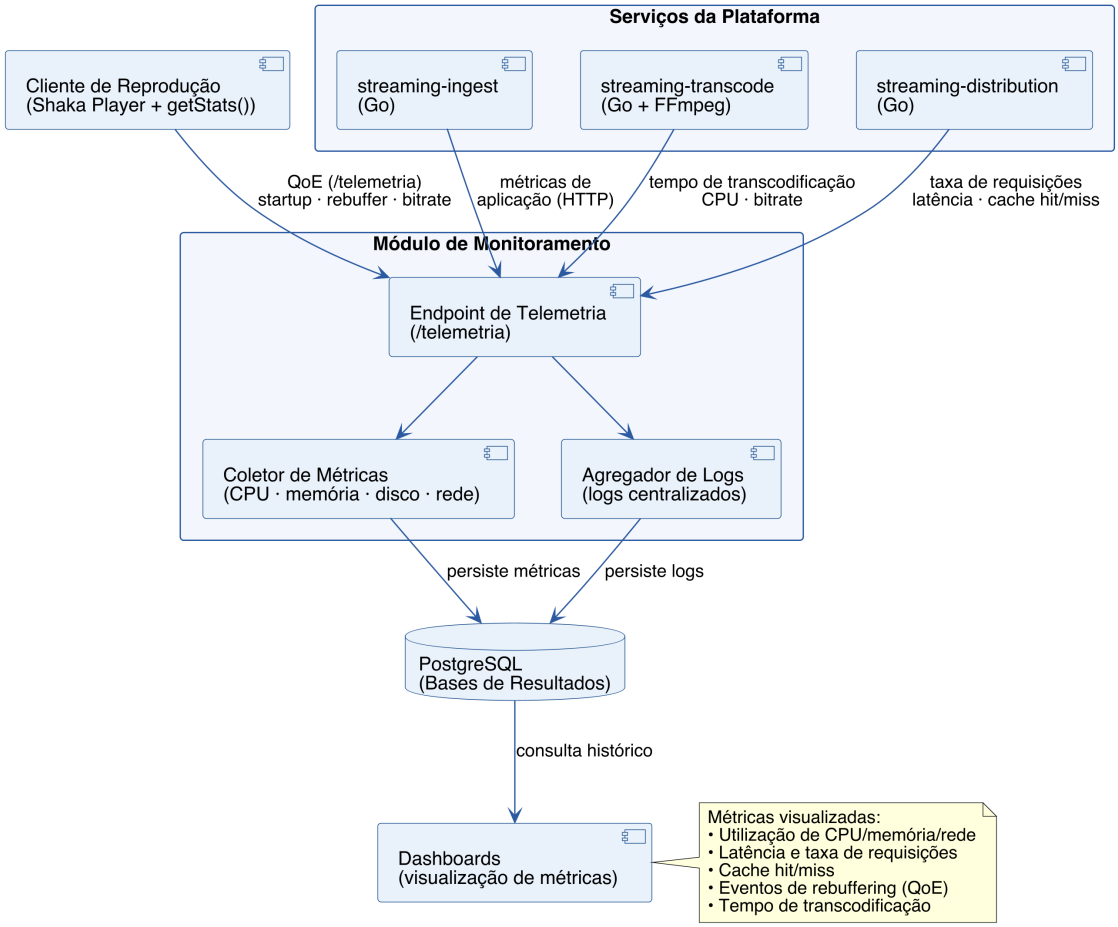
- **Métricas de Servidor:** Utilização de CPU, memória, disco e rede.
- **Métricas de Aplicação:** Tempo de transcodificação, taxa de requisições HTTP, latência de resposta e taxa de *cache* hit/miss.
- ***Logs* Centralizados:** Agregação de *logs* de todos os componentes.
- **Telemetria do Cliente:** Conforme descrito anteriormente, o *player* envia métricas de QoE para um *endpoint* dedicado.

A integração dessas ferramentas permite a triangulação de dados entre as métricas de servidor (lado da oferta) e as métricas de QoE (lado da demanda), viabilizando a análise aprofundada das relações de causa e efeito entre os parâmetros configurados e a experiência percebida pelo usuário.

Da Proposta Conceitual à Implementação

A implementação efetiva materializou os módulos conceituais das subseções anteriores em uma plataforma de microsserviços poliglota, decomposta em serviços independentes por escalabilidade e isolamento de falhas.

Figura 11 – Módulo de Monitoramento



Fonte: Elaborado pelo autor.

A Tabela 8 reconcilia cada módulo conceitual com o serviço que o implementa e a respectiva *stack* efetiva, evidenciando que a implementação não se afastou do modelo proposto, mas o refinou em engenharia, garantindo que os experimentos medem a arquitetura efetivamente concebida.

Tabela 8 – Reconciliação entre a proposta conceitual e a implementação

Módulo (concepção)	**Serviço (implementação)**	***Stack* efetiva**
Ingestão	`streaming-ingest`	Go + Fiber + MongoDB + RabbitMQ (*event gateway*)
Transcodificação	`streaming-transcode`	Worker Go + FFmpeg (HLS/DASH, *retry*/DLQ)
Empacotamento	integrado ao transcode	Shaka Packager (CMAF, DASH + HLS)
Distribuição	`streaming-distribution`	Go + Redis + MongoDB + CDN
Reprodução	`streaming-web-client`	Svelte + PWA (Shaka Player)
Telemetria	instrumentação *embarcada*	EMF + bases de resultados em PostgreSQL
Upload (novo)	`streaming-platform-upload`	NextJS (*multipart* S3/MinIO)
Orquestração	Docker Compose (local) / nuvem	Lambda + API Gateway + S3 + CloudFront

Fonte: Elaborado pelo autor.

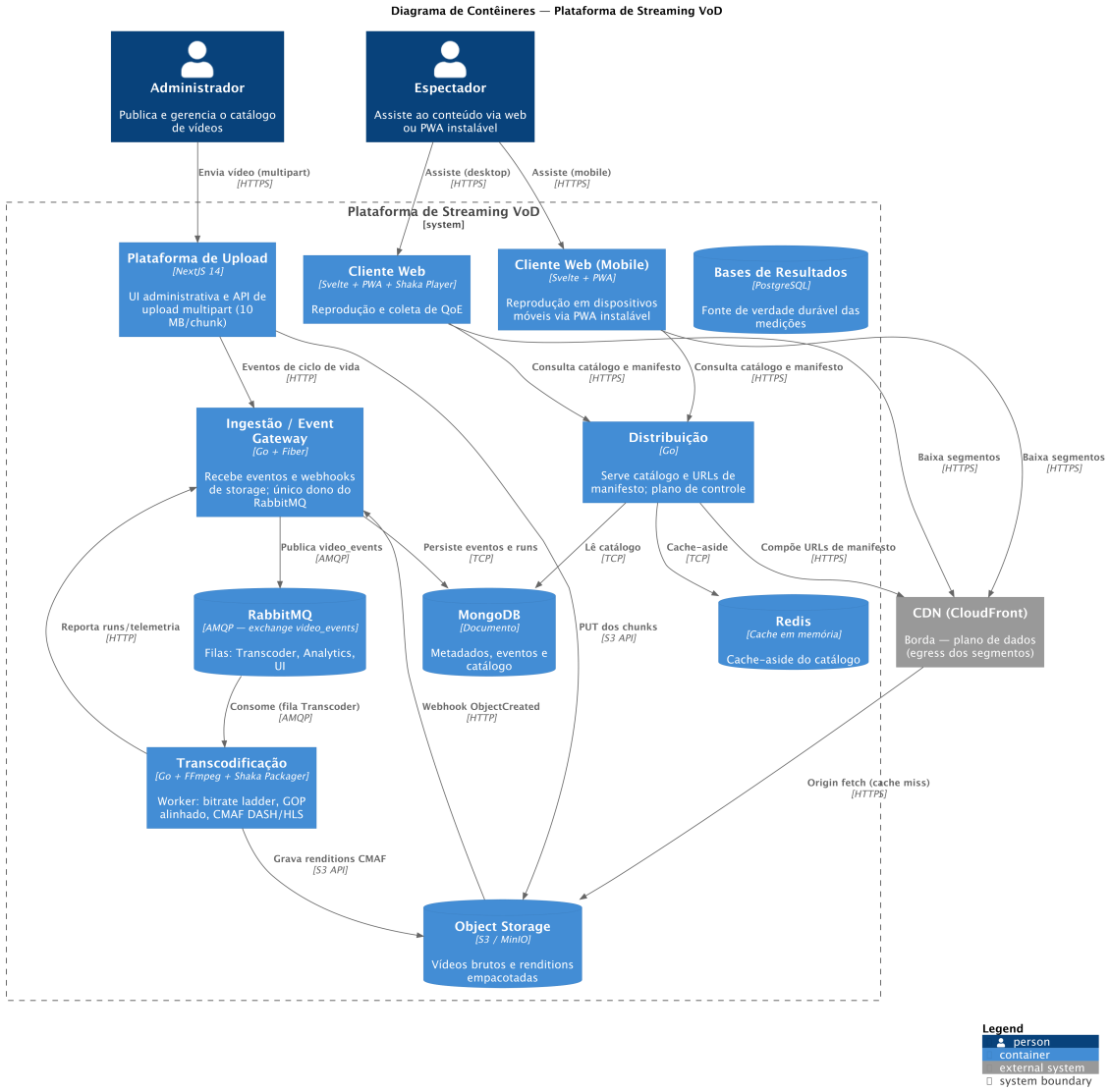
Os termos da coluna *stack* são definidos, de forma sucinta, a seguir; a fundamentação conceitual desses componentes está na Seção 3.9. Go é a linguagem dos serviços de *backend*, e Fiber, o *framework* HTTP minimalista usado sobre ela. MongoDB é o banco de dados de documentos que armazena o catálogo de vídeos, e Redis, o banco em memória usado como *cache* de leitura (Seção 3.9). RabbitMQ é o *broker* de mensagens que desacopla a ingestão do processamento, absorvendo picos de carga (Seção 3.9). FFmpeg é o codificador de linha de comando que executa a transcodificação, e o Shaka Packager, o empacotador que gera os segmentos CMAF, formato comum que serve DASH e HLS com um único conjunto de arquivos (Seção 3.4). Svelte é o *framework* do cliente web de reprodução, empacotado como *Progressive Web App* (PWA), no qual executa o Shaka Player. NextJS é o *framework* da plataforma administrativa de *upload*. EMF (*Embedded Metric Format*) é o formato de métricas embutidas em *logs* adotado pela telemetria, e PostgreSQL, o banco relacional das bases de resultados. Docker Compose orquestra os contêineres no ambiente local; na nuvem, Lambda, API Gateway, S3 e

CloudFront compõem a variante *serverless* (Seção 3.10). As justificativas de cada escolha são consolidadas na Tabela 9.

Visão Geral (Diagrama de Contêineres)

A Figura 12 apresenta o diagrama de contêineres da plataforma. Ele evidencia os dois atores (administrador e espectador), os serviços executáveis (*upload*, ingestão, transcodificação, distribuição e os clientes de reprodução), os repositórios de dados (RabbitMQ, MongoDB, Redis, *object storage* e as bases de resultados em PostgreSQL) e a fronteira com a CDN, que constitui o plano de dados. O diagrama torna explícita a separação entre o plano de controle (consulta de catálogo e manifesto, que transita pelos serviços) e o plano de dados (entrega de segmentos, servida pela borda), distinção central para a escalabilidade discutida ao longo do trabalho.

Figura 12 – Diagrama de Contêineres (C4) da plataforma de *streaming* VoD



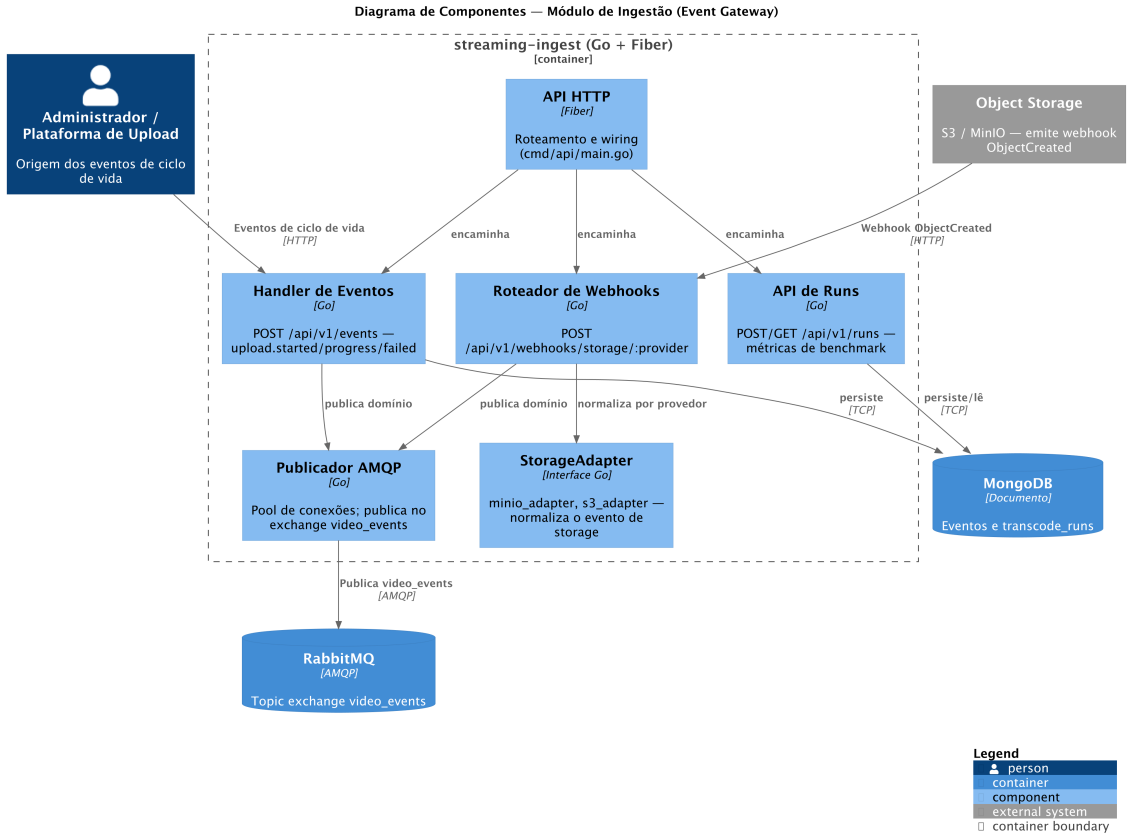
Fonte: Elaborado pelo autor.

Visões Ampliadas por Módulo (Diagramas de Componentes)

Cada serviço é detalhado a seguir em um diagrama de componentes, que abre a "caixa" do contêiner e expõe suas unidades internas e as respectivas tecnologias.

O módulo de ingestão (Figura 13) atua como *event gateway*: a API em Fiber encaminha tanto os eventos de ciclo de vida da plataforma de *upload* quanto os *webhooks* de *storage*, que são normalizados pela interface `StorageAdapter` (com implementações por provedor) e publicados no *exchange* `video_events` pelo publicador AMQP. É o único serviço que detém credenciais do RabbitMQ; regra de ouro do projeto: o *gateway* apenas ingere e publica, nunca consome.

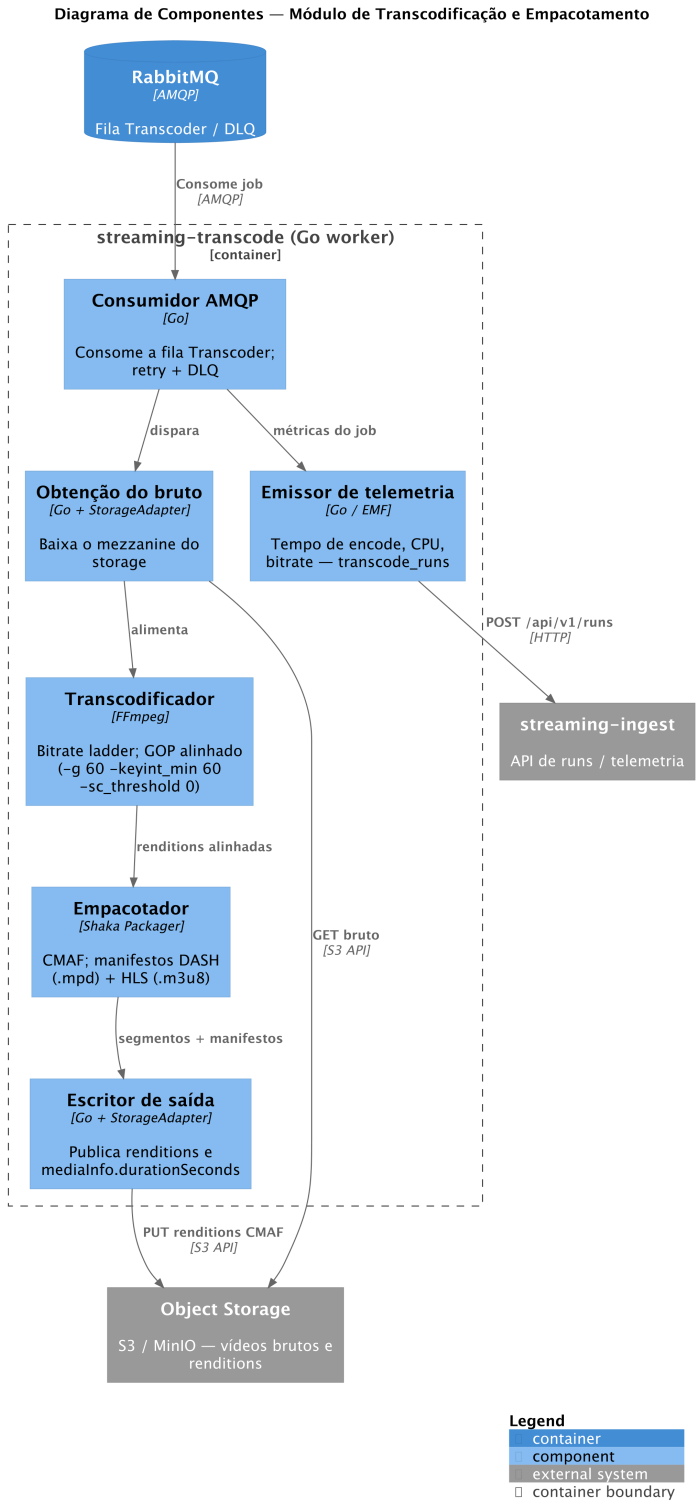
Figura 13 – Diagrama de Componentes (C4) — Módulo de Ingestão



Fonte: Elaborado pelo autor.

O módulo de transcodificação (Figura 14) executa como um *worker*: um processo consumidor que retira da fila `Transcoder` uma tarefa (*job*) por vez; cada *job* descreve a codificação de um vídeo em uma representação de qualidade (*rendition*) da escada da Tabela 7. Para cada *job*, o *worker* baixa o vídeo bruto, executa o FFmpeg (com os GOP alinhados, conforme a restrição de *pipeline*) e o Shaka Packager (gerando CMAF para DASH e HLS), e publica as *renditions* de volta no *object storage*, reportando a telemetria correspondente. Empacotamento e transcodificação, separados na concepção, são integrados neste único *worker*.

Figura 14 – Diagrama de Componentes (C4) — Módulo de Transcodificação e Empacotamento

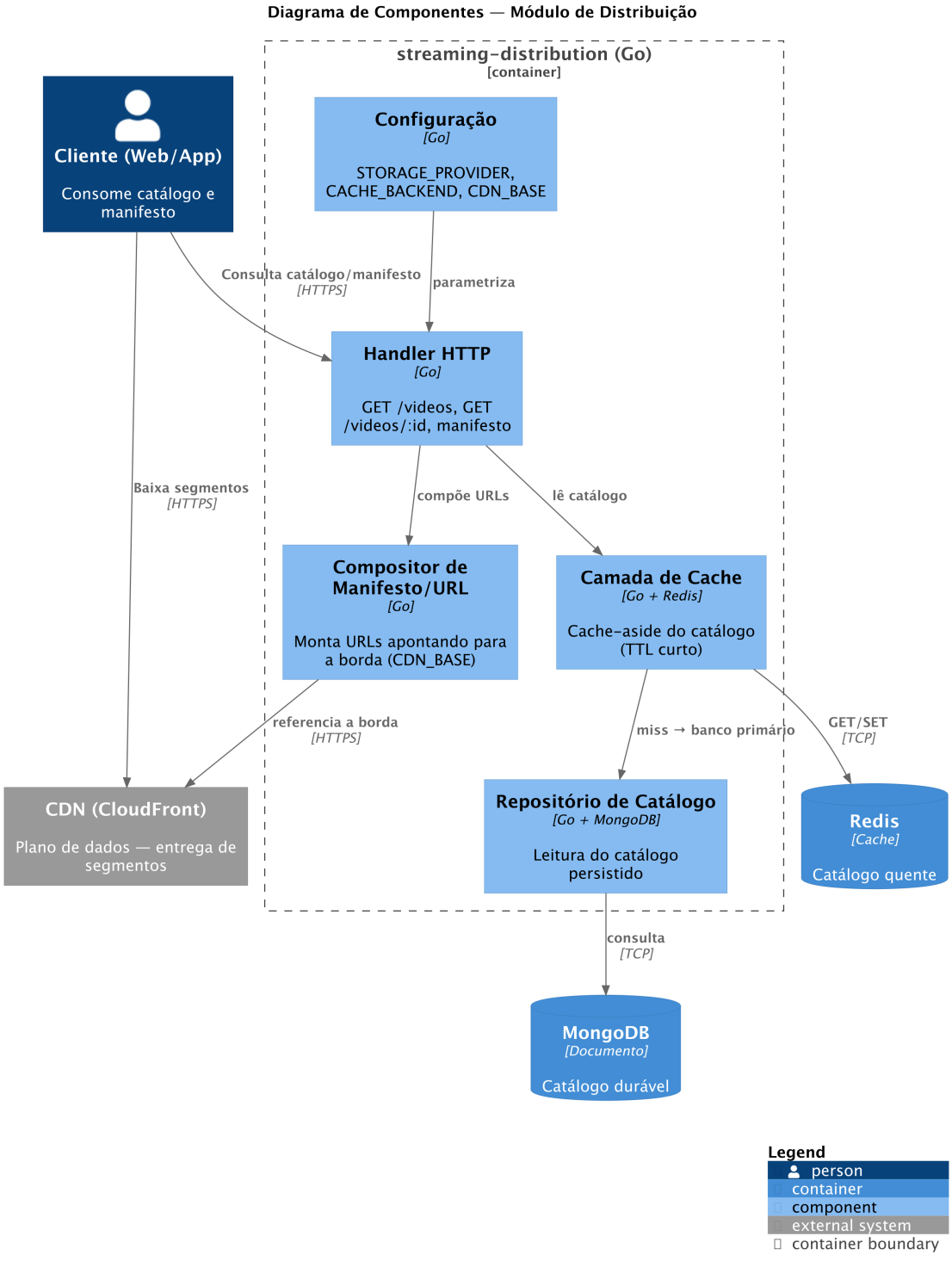


Fonte: Elaborado pelo autor.

O módulo de distribuição (Figura 15) é o plano de controle da entrega: o *handler* HTTP serve o catálogo por meio de uma camada de *cache-aside* em Redis (recaindo no

MongoDB no *miss*) e compõe as URL de manifesto apontando para a borda da CDN, de onde os segmentos são efetivamente servidos.

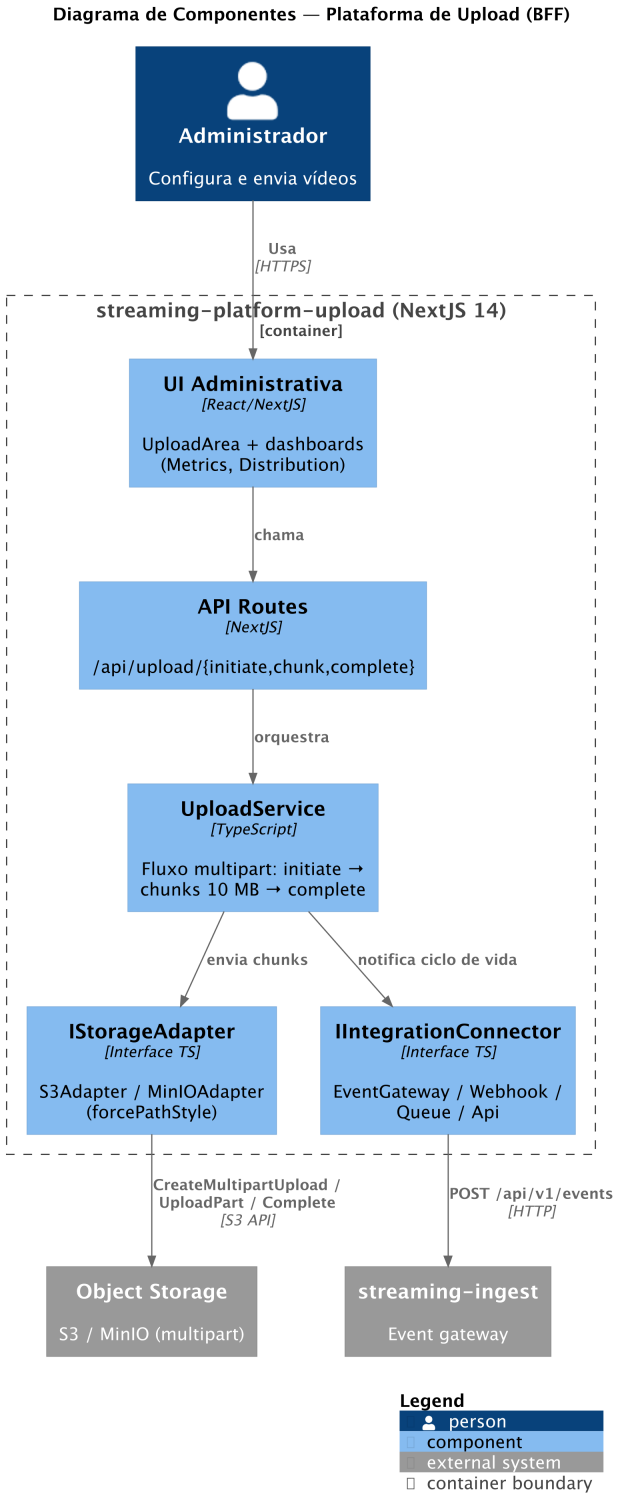
Figura 15 – Diagrama de Componentes (C4) — Módulo de Distribuição



Fonte: Elaborado pelo autor.

A plataforma de *upload* (Figura 16) é um *Backend for Frontend* em NextJS: o `UploadService` orquestra o fluxo multipart (*initiate* → *chunks* de 10 MB → *complete*) sobre a interface `IStorageAdapter`, e notifica o ciclo de vida ao *gateway* de ingestão por meio da interface `IIntegrationConnector`, cujas implementações tornam o *backend* de notificação intercambiável.

Figura 16 – Diagrama de Componentes (C4) — Plataforma de *Upload*

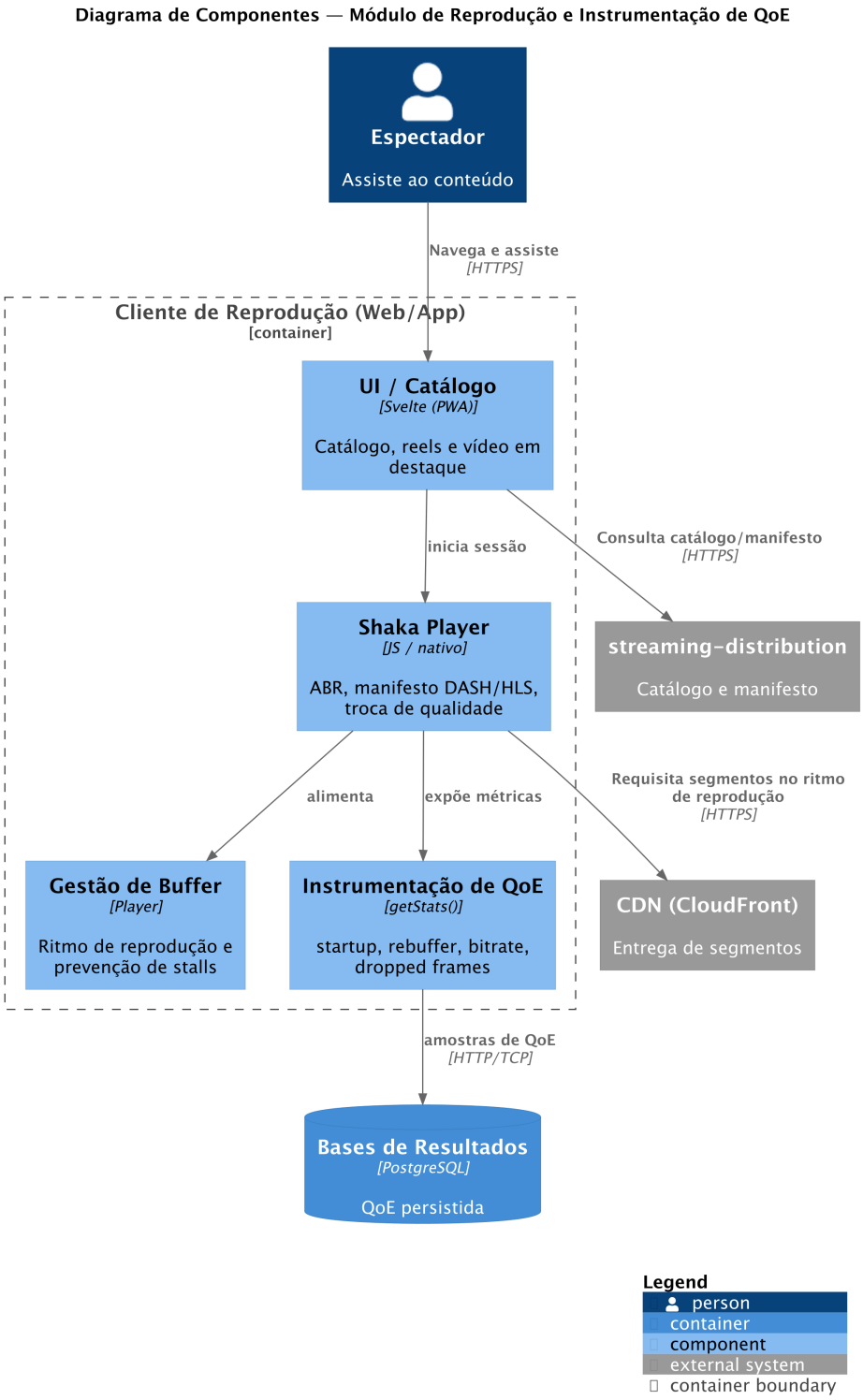


Fonte: Elaborado pelo autor.

Por fim, o módulo de reprodução (Figura 17) materializa o cliente de consumo como uma PWA em Svelte, ancorada no Shaka Player, que executa a adaptação de *bitrate* por ABR e requisita os segmentos no ritmo de reprodução. `getStats()` é a interface

pública do Shaka Player que expõe os contadores internos da sessão, entre eles o tempo de carregamento inicial, o tempo acumulado em *rebuffering*, o *bitrate* corrente e o histórico de trocas de representação. O coletor do cliente amostra essa interface periodicamente e a combina com os eventos do *player*, mapeando os contadores para as métricas de QoE definidas na Seção 3.2 (*startup time*, *rebuffering ratio*, *bitrate* médio e oscilações). A justificativa para adotá-la como âncora de fidelidade é a proveniência: as medidas vêm do próprio reprodutor que decodifica o vídeo, e não de estimativas externas. Em comparação, o gerador leve `vclient` aproxima o consumo para fins de escala sem decodificar mídia, e por isso foi validado contra a âncora Shaka (Apêndice A). As amostras de ambos são persistidas nas bases de resultados, fechando o ciclo de instrumentação entre oferta e demanda.

Figura 17 – Diagrama de Componentes (C4) — Módulo de Reprodução e Instrumentação de QoE



Fonte: Elaborado pelo autor.

Decisões Tecnológicas por Módulo da Plataforma

Cada módulo implementado corresponde a uma decisão tecnológica motivada por uma propriedade específica do caso de uso, articulada com os fundamentos de nuvem (Seção 3.10) e de distribuição (Seção 3.5). A Tabela 9 sintetiza essas escolhas e os estudos que as validam empiricamente.

O perfil de carga do caso de uso tem três propriedades dominantes, que motivam as escolhas: leituras de catálogo muito mais frequentes que escritas, o que motiva o *cache* em Redis à frente do MongoDB; processamento de vídeo esporádico, longo e intensivo em CPU/GPU, o que motiva a fila RabbitMQ com *workers* desacoplados, capaz de absorver picos sem perder *jobs*; e entrega de segmentos dominada por largura de banda, o que motiva delegar o plano de dados à CDN. A escolha de Go com Fiber nos serviços decorre da natureza *I/O-bound* da ingestão e da distribuição, em que a concorrência leve das *goroutines* atende muitas conexões simultâneas com baixo custo de memória; a propriedade é exercitada nas rotas de *webhook* da ingestão e nas rotas de catálogo e manifesto da distribuição, e impacta diretamente a latência do plano de controle e a capacidade de espectadores simultâneos medidas no Estudo 1. A do FFmpeg e do Shaka Packager decorre da exigência de controle e reprodutibilidade científica sobre codificação e empacotamento. A do PostgreSQL para as bases de resultados decorre da necessidade de consultas estruturadas e duráveis sobre as medições.

Três decisões destacam-se por serem transversais. A primeira é a separação entre plano de controle e plano de dados: o catálogo e o manifesto (leves, mas sensíveis à latência) são servidos pelos serviços com apoio de *cache* em Redis, ao passo que os segmentos de vídeo (volumosos, que dominam o *egress*) são delegados à CDN. Essa separação evidencia a modularidade da arquitetura: o plano de dados é um módulo substituível: pode-se trocar a CDN (o CloudFront por outra CDN comercial, ou por uma CDN *self-hosted*) ou reconfigurá-la sem reescrever os demais serviços. É justamente esse desacoplamento que permite contratar apenas a entrega de borda, mantendo o plano de controle aberto e instrumentado. A segunda é o uso de interfaces de adaptação (`StorageAdapter`, `IStorageAdapter`, `IIntegrationConnector`) que isolam o domínio de provedores concretos, permitindo alternar entre MinIO e S3 ou entre diferentes *backends* de notificação sem alterar a lógica de negócio. A terceira é a adoção de bases de resultados dedicadas em PostgreSQL, separadas da telemetria operacional, garantindo a reprodutibilidade e a comparabilidade das medições exigidas pela metodologia experimental.

Tabela 9 – Decisões de tecnologia e respectivas justificativas

Decisão	**Tecnologia**	**Justificativa**
Cache de catálogo	**Redis**	Banco em memória para *cache-aside*; em *cache* quente, decide sozinho a latência de leitura.
Armazenamento de objetos	**S3 / MinIO**	Alta durabilidade, API HTTP e ciclo de vida na nuvem (S3); paridade local com MinIO; origem da CDN (Seção 3.10).
Mensageria	**RabbitMQ**	Desacopla ingestão e processamento por *topic exchange*; absorve picos e habilita *retry*/DLQ no *pipeline* orientado a eventos.
Metadados / catálogo	**MongoDB**	Modelo de documento aderente ao catálogo; banco primário de durabilidade por trás do *cache*.
Bases de resultados	**PostgreSQL**	Fonte de verdade durável e estruturada das medições, desacoplada da telemetria efêmera.
Serviços de *backend*	**Go + Fiber**	Concorrência leve (*goroutines*) e baixa latência para ingestão e distribuição.
Backend for Frontend	**NextJS 14**	UI administrativa e API de *upload* multipart no mesmo *framework*.
Transcodificação	**FFmpeg**	Controle preciso de parâmetros (vs. soluções “caixa-preta”), essencial à análise científica.
Empacotamento	**Shaka Packager**	CMAF único servindo DASH e HLS, eliminando duplicação de armazenamento.
Reprodução	**Shaka Player**	Suporte nativo a DASH/HLS; é o reprodutor do produto e a âncora fiel de QoE (Seção 5.2).
CDN / plano de dados	**CloudFront**	Delega o *egress* global à borda comercial, inviável de autogerir (Seção 3.5).
Orquestração	**Docker / AWS *serverless***	Docker Compose no ambiente local; Lambda + API Gateway + S3 + CloudFront na nuvem (Seção 3.10).

Fonte: Elaborado pelo autor.

Disponibilidade do Código-Fonte

Em coerência com o requisito RNF01 (Código Aberto) e com o compromisso de reprodutibilidade que orienta a metodologia experimental, todo o código-fonte da plataforma está publicamente disponível no GitHub, com cada microsserviço da Tabela 8 em um repositório independente. Além do código dos serviços, os repositórios trazem dois conjuntos de artefatos que sustentam a reprodutibilidade (conceitos apresentados na Seção 3.9). Os arquivos de *deployment* descrevem o ambiente de execução de forma declarativa: o Docker Compose define os contêineres, redes e volumes do ambiente local completo (serviços, MinIO, Redis, RabbitMQ, MongoDB e PostgreSQL), permitindo recriar a plataforma com um único comando, e a infraestrutura como código descreve os recursos da variante em nuvem (Lambda, API Gateway, S3 e CloudFront). Os *scripts* de *benchmark*, por sua vez, automatizam os experimentos do Capítulo 7: preparam o cenário, geram a carga, coletam as métricas nas bases de resultados e produzem as tabelas usadas na análise, de modo que cada experimento pode ser reexecutado de ponta a ponta. A relação completa dos repositórios, com seus respectivos endereços, é apresentada no Apêndice C.

5.3 Detalhamento Operacional

Os experimentos são conduzidos sobre o ambiente de nuvem (AWS Lambda, API Gateway, CloudFront, S3 e ElastiCache), que reproduz a topologia *serverless* de produção. Um ambiente local equivalente em Docker Compose suporta a validação funcional do *pipeline*; a paridade de contratos e esquemas de resultados entre os dois ambientes garante a reprodutibilidade (Capítulo 6).

Sobre esse ambiente, o *cache* opera em duas camadas. O plano de controle emprega *cache-aside* em Redis: leitura atendida do *cache* quente; no *miss*, o MongoDB reabastece-o. O plano de dados opera com `Cache-Control` diferenciado na CDN: `s-maxage` longo nos segmentos (imutáveis por *content-address*) e curto nos manifestos (atualizados a cada segmento em *live*), maximizando o *origin offload*.

O balanceamento de carga, por sua vez, não emprega um balanceador clássico: na variante *serverless* ele resulta da composição de três mecanismos. No plano de dados, o CloudFront distribui as requisições de segmentos entre seus pontos de presença pelo roteamento da própria CDN (DNS/*anycast*, Seção 3.5), aproximando

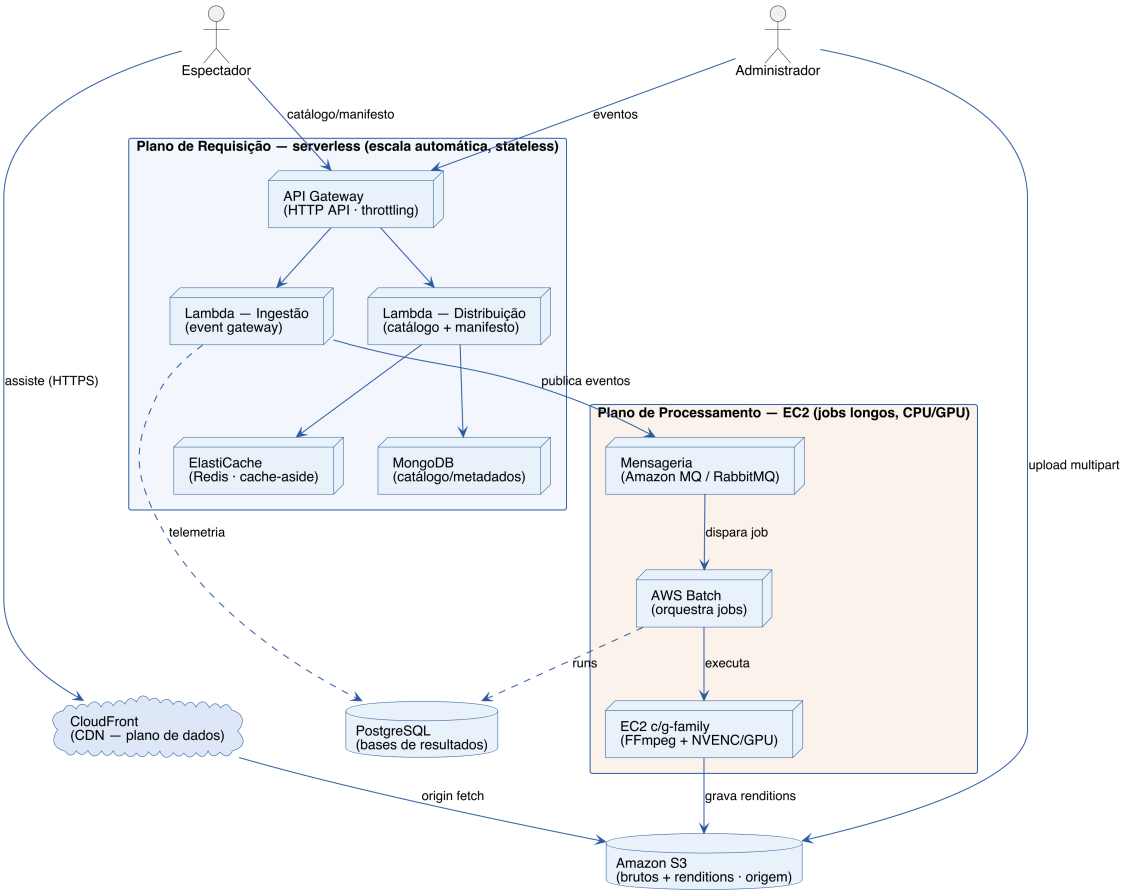
o conteúdo do espectador e diluindo a carga geograficamente; como os segmentos representam a quase totalidade do *egress*, é essa camada que absorve o volume. No plano de controle, o API Gateway repassa cada requisição de catálogo ou manifesto ao Lambda, cuja escala horizontal é implícita: o provedor instancia cópias da função conforme a concorrência, dispensando a distribuição manual entre instâncias. Na variante conteinerizada, não exercida nos experimentos, esse papel caberia ao *Application Load Balancer* (Seção 3.10). Como demonstra o Estudo 1, é justamente a cota de concorrência da função (e não a largura de banda da entrega) o primeiro ponto de saturação da arquitetura, mitigável por concorrência provisionada ou aumento de cota.

Por fim, a instrumentação dos clientes combina o lado do servidor e o lado do cliente. No servidor, a telemetria é emitida no formato EMF e agregada (vazão, latência por percentis, taxa de erro). No cliente, dois instrumentos complementares foram empregados (detalhados no Capítulo 6 e no Apêndice A): o Shaka Player instrumentado via `getStats()` como âncora de fidelidade de QoE, e o gerador leve `vclient` para geração de carga em escala. Todas as amostras são persistidas em bases de resultados dedicadas (PostgreSQL), que constituem a fonte de verdade durável e desacoplada da telemetria efêmera, viabilizando a triangulação entre oferta e demanda.

5.4 Arquitetura de Implantação na AWS

A implantação sobre a AWS materializa a arquitetura em serviços gerenciados com modelo *serverless-first* heterogêneo: o plano de requisição é atendido por funções Lambda e o plano de processamento, por instâncias EC2. A Figura 18 apresenta essa topologia.

Figura 18 – Diagrama de implantação na AWS: plano de requisição (*serverless*) e plano de processamento (EC2/Batch)



Fonte: Elaborado pelo autor.

Mapeamento dos Componentes para Serviços da AWS

A Tabela 10 concretiza a alocação de cada componente a um serviço AWS, registrando o serviço hospedeiro, seu papel e a motivação funcional da escolha: Lambda e API Gateway para operações curtas e elásticas do plano de controle; EC2/Batch para transcodificação longa com GPU; CloudFront como plano de dados da CDN.

Tabela 10 – Mapeamento dos componentes para serviços da AWS

Componente	**Serviço AWS**	**Papel**
Ingestão	AWS Lambda + API Gateway	*Event gateway* sob demanda
Distribuição	AWS Lambda + API Gateway + ElastiCache	Catálogo/manifesto com *cache* em Redis
Transcodificação	AWS Batch sobre EC2 (famílias `c`/`g`)	FFmpeg + Shaka Packager; GPU (NVENC) na família `g`
Mensageria	Amazon MQ (RabbitMQ)	Exchange `video_events`
Object storage	Amazon S3	Brutos e *renditions*; origem da CDN
CDN (plano de dados)	CloudFront	*Egress* dos segmentos na borda
Catálogo/metadados	MongoDB (gerenciado)	Banco de documento
Bases de resultados	PostgreSQL (gerenciado)	Fonte de verdade das medições
Upload	Aplicação NextJS + S3	UI administrativa e *upload* multipart
Borda e segurança	API Gateway + CloudFront	*Throttling*, TLS e *egress*

Fonte: Elaborado pelo autor.

Além da alocação de planos discutida a seguir, três decisões desse mapeamento completam o quadro. Primeiro, os componentes de infraestrutura de dados migram para equivalentes gerenciados preservando o protocolo: o RabbitMQ torna-se Amazon MQ, e o MongoDB e o PostgreSQL passam a instâncias gerenciadas, sem alteração no código dos serviços, em coerência com os requisitos RNF01 e RNF03. Segundo, o S3 acumula dois papéis: repositório dos brutos e das *renditions* e, simultaneamente, origem da CDN, o que elimina a necessidade de um servidor de origem dedicado. Terceiro, as preocupações transversais concentram-se na borda: TLS, limitação de taxa (*throttling*) e *egress* ficam a cargo do par API Gateway + CloudFront, e não dos serviços, mantendo as funções enxutas. Esse uso de serviços gerenciados não conflita com a proposta de arquitetura

aberta: os componentes permanecem de código aberto e portáveis, as interfaces são compatíveis entre ambientes (a API S3, por exemplo, é atendida pelo MinIO no ambiente local), e a AWS entra como alvo de implantação para medição de custo e comportamento, não como dependência da arquitetura.

Alocação de Planos por Modelo de Computação

Os critérios que distinguem o modelo *serverless* (Lambda) do modelo de instância (EC2) foram sistematizados na Seção 3.10 (Tabela 4). A alocação adotada nesta arquitetura aplica esses critérios a dois planos com perfis opostos.

O plano de requisição (a ingestão e a distribuição) é composto de operações curtas, *stateless* e de tráfego irregular (picos de campanha intercalados a períodos ociosos), perfil para o qual o Lambda é ideal: escala automaticamente com a concorrência, escala a zero (não há custo ocioso) e cobra estritamente pelo uso. Como o plano de controle é leve, pois apenas a consulta de catálogo e de manifesto transita pela função, ao passo que os segmentos volumosos são servidos pela CDN; o custo por invocação é baixo e o modelo *serverless* domina em eficiência.

Essa escolha, contudo, traz um limite concreto, explicado a seguir passo a passo. A AWS impõe a cada conta uma cota de concorrência: um número máximo de execuções simultâneas de funções Lambda. Cada espectador ativo gera requisições periódicas de manifesto, e cada requisição ocupa uma execução da função por alguns milissegundos; multiplicando a taxa de requisições pelo tempo médio de execução medido (o *footprint* por espectador do Capítulo 7), obtém-se quantas execuções simultâneas uma dada audiência produz. Projetando esse *footprint*, a cota é atingida com cerca de 25 mil espectadores simultâneos. O gargalo do plano de requisição, portanto, não é a largura de banda, pois os segmentos volumosos fluem pela CDN, e sim essa cota administrativa. As mitigações são o aumento da cota junto ao provedor e, para garantir capacidade dedicada à função crítica, a concorrência provisionada (*provisioned concurrency*), que mantém instâncias pré-aquecidas reservadas. A observação ilustra um traço geral do *serverless*: ele desloca o problema de capacidade da máquina para a cota do provedor. O mesmo mecanismo de concorrência foi, aliás, explorado como freio de custo: zerar a concorrência reservada da função (*reserved concurrency* igual a zero) suspende o plano de requisição sem desfazer a infraestrutura, um *kill-switch* de baixo custo operacional.

O plano de processamento (a transcodificação) tem perfil oposto: *jobs* longos (de minutos a uma hora), intensivos em CPU e GPU, que excedem o limite de 15 minutos do Lambda e exigem aceleração de *hardware* (NVENC) indisponível nele. Esse perfil

mapeia naturalmente para o EC2, orquestrado pelo AWS Batch, que enfileira e despacha os *jobs* sobre uma frota de instâncias da família `c` (CPU) ou `g` (GPU). Não por acaso, a própria campanha de *benchmark* de *codec* (Estudo 2) foi executada sobre instâncias EC2 (`c7i`, `g4dn`, `g6`), pois é o EC2 que expõe o controle de *hardware* exigido por uma medição de eficiência de codificação.

Em síntese, a arquitetura não opta por *serverless* ou por servidores, mas emprega cada modelo onde ele apresenta vantagem: Lambda para o tráfego de requisição, curto e elástico, em que o custo ocioso seria desperdício; EC2/Batch para o processamento, longo e acelerado por GPU, em que o limite de tempo e a ausência de GPU inviabilizariam o *serverless*. Essa correspondência entre o perfil da carga e o modelo de computação é a tradução prática, nesta arquitetura, dos critérios discutidos na Seção 3.10.

A plataforma foi documentada em dois planos. O plano conceitual organiza-se em cinco módulos funcionais: ingestão, transcodificação, empacotamento, distribuição e reprodução, com *bitrate ladder* parametrizável e coleta de métricas de QoE. O plano de implementação efetiva materializa essa concepção em microsserviços políglotas em Go, TypeScript e Python, orquestrados por Lambda e EC2/Batch na nuvem e instrumentados via EMF. A correspondência entre os dois planos, evidenciada pela Tabela 8, é a premissa que garante que os experimentos da Seção 6.8 e do Capítulo 7 medem a arquitetura efetivamente concebida, e não um substituto.

6 METODOLOGIA EXPERIMENTAL

Enquanto a fase exploratória teve caráter de mapeamento, culminando na proposição do modelo arquitetural, esta fase assume natureza descritiva e explicativa, materializada por um estudo de caso instrumental (PRODANOV; FREITAS, 2013). O objetivo é relatar o comportamento da arquitetura sob condições controladas e explicar as relações de causa e efeito entre os parâmetros de configuração e as métricas observadas, de modo a evidenciar os *trade-offs* entre desempenho, custo e QoS e QoE.

O desenho experimental organiza-se em torno de três questões fundamentais: como testar a arquitetura em condições controladas, como metrificá-la de forma objetiva e como projetar os resultados para escalas de produção a partir de medições em escala reduzida. As seções a seguir respondem a cada uma dessas questões e consolidam a metodologia adotada.

Subjacente a essas três questões há uma dimensão de método mais ampla. Propor uma arquitetura de *streaming* é, em si, genérico; o desafio é testá-la de modo que os resultados reflitam um cenário realista, e não um caso abstrato. A resposta deste trabalho é especializar a investigação em um caso particular (a plataforma de VoD do contexto farmacêutico, detalhada no Capítulo 4) e executar a metodologia aqui definida dentro desse estudo de caso: é o estudo de caso que instancia os parâmetros concretos (corpus de vídeos, carga, *codecs* e patamares de escala), exercita a arquitetura e gera os resultados, discutidos no Capítulo 7.

6.1 Estratégia Metodológica: Medição em Escala Reduzida e Projeção

O principal desafio metodológico é quantificar o comportamento da arquitetura em escala de produção sem operar nessa escala, inviável no contexto deste trabalho. Para superá-lo, adotou-se a estratégia de medição em escala reduzida e projeção: mediu-se o custo unitário de operação (o *footprint* por viewer, por requisição e por codificação) sobre uma implementação real e instrumentada, executada em escala reduzida. A partir dessas constantes empíricas, projetou-se o comportamento agregado para os patamares de produção, identificando o custo financeiro, a capacidade necessária e o primeiro ponto de saturação de cada cenário.

A implementação não substitui a literatura; ela a complementa. A literatura fornece as premissas do modelo (faixas de concorrência de pico, regras de compressão

de *codecs*, padrões de consumo), ao passo que a medição fornece as constantes que dependem do código, do ambiente e das escolhas de *stack* específicas, que a literatura genérica não pode oferecer. A combinação dos dois é o que confere validade às projeções.

Quatro cenários de escala norteiam a análise, em ordem crescente de carga:

- **Escala menor (1.000 usuários simultâneos):** linha de base medida em ambiente local controlado;
- **Escala média (10.000 usuários, regime de pico):** cenário de consolidação, medido em infraestrutura de nuvem;
- **Escala nacional (milhões de usuários):** projeção ao patamar de audiência nacional do setor farmacêutico (Capítulo 4), para revelar quais componentes passam a dominar o custo e em que ponto a capacidade satura;
- **Escala internacional (bilhões de requisições):** exercício de projeção de limites extremos, inspirado em plataformas como o YouTube, que reúne bilhões de usuários conectados por mês (YouTube, 2024).

6.2 Caso de Uso e Cenário Experimental

O caso de uso é uma plataforma de VoD no contexto de uma rede de farmácias: administradores (equipe de marketing e treinamento) cadastram vídeos de produtos, campanhas e capacitação, que são consumidos por espectadores (colaboradores das lojas e clientes) em clientes web e móveis. A Tabela 11 consolida as características que parametrizam o cenário experimental e alimentam o modelo de custo e capacidade. O perfil de concorrência concentra cerca de 5% dos acessos diários em uma janela de pico de aproximadamente uma hora (campanhas e eventos), com a concorrência média diária mantendo-se bem abaixo desse patamar.

Tabela 11 – Características do caso de uso

Característica	**Valor**
Atores	Administrador (cadastro) e Espectador (web/móvel)
Acessos (*views*) por dia	$\sim$150 mil
Duração média do acesso	$\sim$5 min
Concorrência de espectadores (média diária)	$\sim$500 espectadores
Concorrência de espectadores no pico ($\sim$1 h/dia)	$\sim$7.500 espectadores (5% dos acessos)
Requisições de segmento por dia	$\sim$11,25 milhões
Vazão de entrega (média / pico)	$\sim$130 / $\sim$1.875 req/s
Tamanho médio de *upload* (bruto)	$\sim$300 MB
Uploads administrativos/dia	$\sim$30
Latência de *streaming* alvo	$<$ 400 ms
Disponibilidade	24/7 (alta disponibilidade)

Fonte: Elaborado pelo autor.

Nota: Requisições de segmento calculadas com segmentos HLS de 4 s (75 req por acesso de 5 min); vazão de pico derivada de 7.500 espectadores simultâneos $\times$ 0,25 req/s por espectador.

6.3 Sistema sob Teste e Controles de Validade

O desenho experimental adota seis controles de validade para garantir que cada medição reflita o sistema sob teste e não fatores externos:

- **Definição precisa do Sistema sob Teste (SUT):** cada estudo declara explicitamente qual serviço e qual rota está sob medição, embrulhando-os em uma aplicação idêntica que isola a variável manipulada;
- **Equivalência verificada:** quando se comparam alternativas que devem produzir a mesma saída, verifica-se a igualdade do resultado por *hash* criptográfico (*sha256*), garantindo que as configurações comparadas estão, de fato, "fazendo a mesma coisa";
- **Cliente único e uniforme:** a carga é gerada por um único cliente de medição, eliminando o fator de confusão de *drivers* distintos;

- **Ambiente controlado:** as medições ocorrem em rede interna (*container* a *container*), eliminando o pedágio de rede da fronteira entre máquina hospedeira e máquina virtual, que penaliza desproporcionalmente componentes leves e sensíveis à rede;
- **Replicação e estatística:** cada ponto de medição é repetido em múltiplas execuções, reportando-se a mediana para descartar *outliers* transitórios;
- **Isolamento de módulos:** cada componente da arquitetura é projetado para teste isolado: testa-se a ingestão independentemente da transcodificação, e a distribuição (API) independentemente do *frontend*.

A geração de carga emprega *players* reais, que processam o manifesto, selecionam uma representação por ABR e requisitam os segmentos no ritmo de reprodução. Essa abordagem simula carga sustentada e realista, distinta tanto de um *download* voraz (pior caso de banda, irreal) quanto de uma simples requisição ao manifesto (que não exercita a entrega de mídia). Em cenários de nuvem, observou-se ainda a necessidade de um gerador de carga dedicado, fisicamente separado do SUT, sob pena de medir a saturação do próprio gerador em vez da capacidade do sistema.

6.4 Métricas Objetivas de QoS e QoE

As métricas de QoE são objetivas e mensuradas no plano de infraestrutura (*startup time*, *rebuffering ratio* e *bitrate* médio) e constituem um *piso* da experiência possível: o que a infraestrutura entrega em condições controladas. A experiência percebida pelo usuário final agrega o *player*, a rede de última milha e o dispositivo, acessíveis apenas por *Real User Monitoring* em campo, o que ficou fora do escopo deste trabalho. A Tabela 12 relaciona cada métrica ao estudo que a coleta.

Tabela 12 – Métricas objetivas coletadas, por dimensão e estudo

Dimensão	**Métrica**	**Estudo**
QoS — vazão	Requisições por segundo (req/s)	Distribuição
QoS — latência	Percentis p50, p95 e p99 (ms)	Distribuição
QoS — entrega	*Egress* (GB), conexões, taxa de *cache-hit*	Distribuição
QoS — recurso	CPU, memória, taxa de erro	Distribuição
QoE — início	*Startup time* / *Time-to-first-frame* (s)	Distribuição
QoE — fluidez	*Rebuffering ratio*, *stalls*	Distribuição
QoE — qualidade	*Bitrate* médio, trocas de ABR	Distribuição
Eficiência	VMAF, PSNR, BD-rate	Codec
Velocidade	Tempo de codificação, *speedup*	Codec
Custo	$/mês (*egress*, computação, armazenamento)	Distribuição

Fonte: Elaborado pelo autor.

O mecanismo de coleta varia por dimensão. As métricas de QoS são medidas do lado da oferta: os serviços emitem telemetria estruturada (formato EMF) a cada requisição, da qual se derivam vazão, percentis de latência, *egress*, taxa de *cache-hit* e consumo de recursos. As métricas de QoE são medidas do lado da demanda: o Shaka Player instrumentado expõe os contadores da sessão via `getStats()`, amostrados periodicamente pelo coletor do cliente, e o gerador leve `vclient` registra as mesmas grandezas nos ensaios de escala (Apêndice A). As métricas de eficiência de codificação são computadas após cada *job*: o VMAF e o PSNR comparam o vídeo codificado ao original, e o BD-rate é integrado sobre as curvas resultantes; o tempo de codificação é cronometrado pelo próprio *worker*. O custo, por fim, não é coletado, e sim calculado: aplica-se a tabela de preços da AWS (Seção 3.10) ao consumo medido.

A persistência das medições emprega bases de resultados dedicadas (em PostgreSQL), funcionando como fonte de verdade durável e estruturada, desacoplada da telemetria operacional (efêmera) do serviço. Essa separação garante a reprodutibilidade da análise e a comparabilidade entre execuções locais e em nuvem.

6.5 Modelo de Projeção

A projeção da escala reduzida para a escala de produção segue um modelo linear ancorado no *footprint* por viewer medido. Para o perfil farmacêutico adotado, a projeção cobre espectadores concorrentes $N \leq 20.000$, com linearidade confirmada experimentalmente até $N = 10.000$ (Capítulo 7) e projeção conservadora até 20 mil, que corresponde ao dobro do ponto medido. Para N espectadores, o agregado de cada métrica, isto é, o valor total demandado pela audiência inteira (a vazão total em requisições por segundo, o *egress* total em gigabytes ou o custo total em dólares, conforme a métrica), é obtido multiplicando-se a constante unitária pelo número de espectadores:

$$\text{Agregado}(N) = \text{footprint\_por\_viewer} \times N \tag{1}$$

ajustado pela inclinação observada entre os pontos medidos, de modo a capturar efeitos não-lineares. Do modelo derivam três entregas: (i) a capacidade (*egress* total em GB/h, requisições por segundo agregadas e conexões concorrentes, estas estimadas pela Lei de Little (LITTLE, 1961) como o produto entre a vazão e a latência); (ii) o custo mensal estimado, decomposto por serviço de nuvem; e (iii) o primeiro ponto de saturação, isto é, o primeiro limite de infraestrutura a ser atingido em cada patamar de escala.

6.6 Modelo de Custo e Capacidade

O modelo de custo parte da volumetria do caso de uso e das regras de compressão por *codec* reportadas na literatura (MOINA-RIVERA *et al.*, 2024), nas quais o H.265 reduz aproximadamente pela metade a taxa de bits do H.264, e o AV1 reduz mais de 30% sobre seus antecessores (VP9 e H.265). Aplicando essas regras a um vídeo de referência, deriva-se o tamanho médio das representações e, a partir do padrão de consumo (cerca de 150 mil acessos diários), o volume diário de *egress*, a principal variável de custo. A concorrência de pico é derivada do próprio perfil de uso (acessos diários e duração média dos vídeos), concentrada em uma janela de cerca de uma hora por dia; adota-se, como premissa de dimensionamento deste trabalho, uma concentração de pico de 5% dos acessos diários, cerca de 7.500 espectadores simultâneos para o perfil de 150 mil acessos/dia. A estrutura de custos e os componentes de infraestrutura considerados

baseiam-se nas diretrizes de referência para arquiteturas de VoD em nuvem (Amazon Web Services, 2024b).

A análise consolidada evidencia que o *egress* de CDN domina a fatura, da ordem de 85% a 95% do custo total no cenário modelo, conforme se incluam ou não as estimativas variáveis de observabilidade e transcodificação (Tabela 19), o que posiciona como principais alavancas de redução: a adoção de *codecs* mais eficientes (reduzindo o volume transmitido), a maximização da taxa de *cache-hit* e a aplicação de políticas de ciclo de vida no armazenamento. Os valores numéricos completos do modelo são apresentados no Capítulo 7, e os preços de referência da AWS que alimentam a estimativa constam na Seção 3.10 (Tabela 5).

6.7 Plano de Análise de Dados

A análise dos dados coletados segue uma abordagem quantitativa e comparativa, estruturada em três dimensões complementares:

- **Estatística descritiva:** caracterização do comportamento do sistema por médias, medianas e desvios (a partir de, no mínimo, três execuções por ponto, reportando-se a dispersão quando material) e, sobretudo, percentis (p50, p95 e p99), uma vez que a latência típica (mediana) e a cauda extrema (p99) revelam comportamentos que a média isolada esconde;
- **Triangulação de dados:** confronto sistemático entre as métricas de desempenho do servidor (oferta) e as métricas de QoE do cliente (demanda), validando as relações de causa e efeito entre configuração e experiência percebida;
- **Comparação com a literatura:** contraste dos resultados com os correlatos que admitem comparação (Seção 3.11). No eixo de codificação, o contraste é quantitativo e emprega as métricas comuns ao correlato experimental mais próximo (BUKHARI *et al.*, 2022): tempo de codificação e eficiência a qualidade constante (BD-rate). No eixo de distribuição, em que os correlatos não publicam medições comparáveis de custo e capacidade, o contraste é qualitativo, posicionando os resultados frente à caracterização externa de plataformas comerciais (SUMAN; MOON; CHOI, 2022) e ao panorama do estado da arte (TIMMERER *et al.*, 2025).

Para cada estudo, declaram-se explicitamente as ameaças à validade: externa (representatividade do ambiente em relação à produção), de instrumento (tetos e vieses

dos geradores de carga e dos modelos de qualidade) e de extrapolação (limites da projeção linear acima do máximo medido), em consonância com o rigor metodológico exigido de um estudo de caso instrumental.

6.8 Desenho dos Experimentos

Esta seção detalha o desenho dos experimentos que exercitam a arquitetura: a entrega de vídeo sob carga (Estudo 1) e a eficiência de codificação (Estudo 2). Para cada experimento, descreve-se a pergunta de pesquisa, o sistema sob teste e o método de medição, segundo os controles de validade definidos na seção anterior (Seção 6.3). Os resultados correspondentes são apresentados no Capítulo 7 (Estudo 1 e Estudo 2, respectivamente), onde também se discutem os *trade-offs*.

Estudo 1: Capacidade e QoE da Distribuição via *Players*

Este estudo mede a entrega de vídeo propriamente dita: a pergunta é quantos espectadores simultâneos o serviço de distribuição suporta, com que QoE e a que custo, e onde está o primeiro ponto de saturação ao projetar a operação para escalas maiores.

A metodologia ancora uma projeção em pontos efetivamente medidos, organizados em três níveis (*tiers*): T1, local (até 1.000 espectadores); T2, em nuvem (até 10.000 espectadores, sobre infraestrutura *serverless* com CDN); e T3, projetado, que estende a análise do ponto medido (10 mil) ao primeiro ponto de saturação da arquitetura ($\sim$25 mil), acima do qual o escalonamento passa a ser uma análise de alavancas (cota de concorrência, *provisioned concurrency* e migração do manifesto à CDN), e não uma medição de custo. A carga reproduz o comportamento real de um *player*: o manifesto é processado, uma representação é escolhida por ABR e os segmentos são requisitados no ritmo de reprodução, simulando o *egress* realista de N espectadores simultâneos.

Empregam-se dois instrumentos com papéis disjuntos. O Shaka Player, executado via navegador automatizado (*o mesmo player utilizado pelo cliente web do produto*), atua como âncora de fidelidade de QoE. Por ser orientado a um único navegador, não escala para milhares de espectadores simultâneos; para isso, desenvolveu-se o `vclient`, um gerador leve de clientes HLS/DASH em *goroutines* que simula o *buffer* e a lógica de ABR sem decodificar vídeo, reproduzindo a QoE do *player* real em alta concorrência. As métricas dos dois são persistidas em uma base de resultados dedicada, fonte de verdade do

estudo. O detalhamento completo dos instrumentos, da largada escalonada e dos excertos de implementação encontra-se no Apêndice A.

Estudo 2: Eficiência de Codec (Rate-Distortion / BD-rate)

Este estudo investiga o eixo de eficiência da codificação, complementando um estudo de velocidade (*throughput*) já realizado, que constatou que a codificação acelerada por GPU (NVENC) é de 2 a 12 vezes mais rápida que a codificação por *software*. A pergunta aqui é distinta: qual *codec* entrega a mesma qualidade perceptual com menor taxa de bits, e a que custo de velocidade?

Os experimentos foram conduzidos sobre um corpus de 14 clipes obtidos na plataforma Vimeo, com licença *Creative Commons* (*by* ou *by-nc-nd*) ou acesso público irrestrito, e selecionados de modo a cobrir o espaço *Spatial Information/Temporal Information* (SI-TI) definido pela Recomendação ITU-T P.910 (ITU-T, 2023), que quantifica, respectivamente, a riqueza de contornos em quadros individuais e a variação de luminância entre quadros consecutivos. A diversidade é deliberada: a eficiência de um *codec* depende fortemente da complexidade da cena, e um corpus restrito a um único gênero produziria conclusões de validade limitada. Comerciais curtos (com cortes rápidos, alta TI e grande SI) e vídeos educativos de plano fixo (com baixa variação temporal) ocupam pólos opostos desse espaço; a combinação dos dois gêneros reflete a carga real da plataforma descrita no Capítulo 4. Todos os clipes foram obtidos na resolução de origem (predominantemente 3840×2160) e recodificados nas três resoluções experimentais (480p, 720p e 1080p). A Tabela 13 lista os 14 clipes, acompanhados de seus metadados técnicos de origem.

Tabela 13 – Clipes de vídeo utilizados nos experimentos de codificação e reprodução

ID Vimeo	Título	Gên.	Licença	Resolução orig.	FPS
Conteúdo comercial/promocional					
1078990193	Lauren — Moisturizer	C	—	3840 × 2160	24
339952895	WASO — Color Smart Moisturizer	C	CC BY-NC-ND	3840 × 2160	30
1185327824	Moisturizer	C	—	2160 × 3810	30
544796409	Versed Skincare Nix-It	C	—	2160 × 3840	24
1075261593	Aotea Skincare	C	—	3840 × 2160	25
765449139	Versine Skincare Commercial	C	—	3840 × 2160	30
Conteúdo educativo/informacional					
1117476549	PGx Testing In a Nutshell	E	—	3840 × 2160	25
1145209824	Dr Guide: Polypharmacy Patients	E	—	3840 × 2160	25
714049017	Alicia John — Pharmacy Apprentice	E	—	3840 × 2160	25
374557639	Total Pharmacy Supply (TPS)	E	—	3840 × 2160	24
1145891430	David Ruane — United Drug	E	—	3840 × 2160	25
833927212	CPS Optimizer Benefits	E	—	3840 × 2160	24
866373019	Game Changers in Health	E	—	3840 × 2160	25
1114576994	RealOptions Obria Medical Clinics	E	—	3840 × 2160	24

Fonte: Elaborado pelo autor.

Notas: Gênero: C = comercial/promocional; E = educativo/informacional. Licença: CC BY-NC-ND = *Creative Commons* Atribuição-NãoComercial-SemDerivações; "—" indica vídeo sem licença CC explícita no metadado Vimeo (acesso público ou irrestrito no período do experimento).

Comparar *codecs* a uma taxa de bits fixa é enganoso, pois cada codificador distribui os bits de modo diferente conforme a complexidade da cena. Por isso, o método controlou a qualidade constante (parâmetros CRF, para *software*, e CQ, para NVENC): cada codificador produziu o arquivo que considerou de qualidade *N*, e mediu-se a taxa de bits resultante e a qualidade verificada independentemente pela métrica VMAF, complementada pelo PSNR. A síntese da comparação é o BD-rate (BJØNTEGAARD, 2001), métrica que integra a diferença de taxa de bits, a igual qualidade, ao longo das curvas *rate-distortion*, e que é a única forma adequada de comparar *codecs* cujos parâmetros de qualidade não são numericamente equivalentes entre si. Adotou-se como âncora o codificador `libx264` executado em instância de referência.

A campanha foi executada sobre três tipos de máquina (`c7i`, *software*, âncora; `g4dn`, NVENC de 7ª geração; `g6`, NVENC de 8ª geração/Ada), três resoluções (480p, 720p, 1080p) e cinco pontos de qualidade por *codec*, sobre esse corpus, totalizando

1.680 codificações com VMAF medido. As demais três máquinas previstas (`c5`, `g5`, `g6e`) foram descartadas por serem cópias na curva de eficiência: a eficiência depende do *encoder*, não do modelo da máquina, e as gerações Turing, Ampere e Ada compartilham o mesmo bloco de codificação para H.264 e H.265 (apenas a Ada agrega o `av1_nvenc`).

7 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados das medições conduzidas no estudo de caso (Seção 6.8) e discute o que cada um revela sobre os *trade-offs* da arquitetura. A organização segue a dos experimentos em execução (distribuição e codificação), encerrando com a síntese que articula os achados em torno dos planos da arquitetura. As constantes empíricas aqui obtidas são as que alimentam o modelo de custo e capacidade (Seção 6.6).

Estudo 1: Capacidade, QoE e Custo da Distribuição

A Tabela 14 sintetiza as medições em nuvem. O serviço de distribuição entregou conteúdo para 10.000 espectadores concorrentes, atendendo integralmente toda a demanda imposta, com *bitrate* cheio (5.400 kbps) e *rebuffer* zero. O *footprint* por espectador permaneceu linear (cerca de 4,9 MB) do menor ponto até 10 mil, confirmando a base da projeção.

Tabela 14 – Medições reais de capacidade e QoE da distribuição em nuvem (T2)

Instrumento	**N**	**Atendidos**	***Startup* p50**	**p95**	**p99**	***Bitrate***
`vclient` (frota 6 nós)	10.000	10000/10000	1,45 s	4,64 s	6,80 s	5129 kbps
`vclient` (1 nó 200 Gbps)	10.000	10000/10000	**32 ms**	58 ms	262 ms	5400 kbps
Shaka (âncora, escalada)	100	100/100	**99 ms**	149 ms	159 ms	6000 kbps

Fonte: Elaborado pelo autor.

A distribuição manteve 100% de entrega, *bitrate* cheio (5.400 kbps) e *rebuffer* zero em todos os cenários medidos. O *startup* de 32 ms (linha `vclient` 200 Gbps da Tabela 14) representa o piso de infraestrutura, ou seja, o tempo que o *backend* demora a entregar o primeiro segmento em condições sem contenção de rede do gerador. Os valores superiores das outras linhas refletem limitações do próprio aparato de medição (placa de rede do `vclient` em frota), não da distribuição; a metodologia de validação do gerador, incluindo a evidência A/B, é detalhada no Apêndice A. A dispersão confirma essa leitura: mesmo no percentil p99, o nó limpo (200 Gbps) permanece em 262 ms, patamar sub-segundo que evidencia a estabilidade do piso de infraestrutura na cauda, ao passo que a cauda larga da frota (p99 de 6,80 s) é artefato da saturação da placa de rede do gerador, e não da entrega.

A partir do *footprint* por espectador, o modelo de projeção (Seção 6.5) estima a capacidade e o custo mensal nos patamares de produção (Tabela 15). O custo é um limite superior, pois assume cada espectador transmitindo continuamente ao longo do mês; o consumo real é intermitente, de modo que o custo efetivo é uma fração desse valor. É essencial distinguir os dois cenários de custo que este trabalho reporta, de unidades distintas: (i) o *teto* da Tabela 15, que assume N espectadores concorrentes transmitindo continuamente, 24 horas por dia, todo o mês (um pior caso teórico de dimensionamento); e (ii) o custo *itemizado realista* da Tabela 19, baseado em usuários por dia com consumo intermitente. A diferença é de ordens de grandeza e de unidade: o teto concorrente-contínuo de 10 mil espectadores chega a ∼US$ 1,27 mi/mês, ao passo que o cenário diário-intermitente de referência (30 mil acessos/dia) custa ∼US$ 7,9 mil/mês. O teto serve para dimensionar capacidade e pior caso; o itemizado, para estimar a fatura efetiva.

Tabela 15 – Capacidade, custo-teto e saturação da distribuição (espectadores concorrentes em transmissão contínua 24/7; medido até 10k, projetado até 20k)

N	**Acessos/dia (24/7)**[†]	**Tipo**	***Egress*/dia**	**Custo/mês (teto, 24/7)**	**Satura**
10.000	∼2,88 mi	medido	486 TB	∼US$ 1,27 mi	—
15.000	∼4,32 mi	projetado	729 TB	∼US$ 1,90 mi	—
20.000	∼5,76 mi	projetado	972 TB	∼US$ 2,54 mi	→ *lambda*[*]

Fonte: Elaborado pelo autor.

Nota: [†] Acessos diários equivalentes com a concorrência sustentada em 100% do dia (cada acesso de 5 min): $N \times (86400/300) = N \times 288$. [*] A concorrência do Lambda satura em $N \approx 25$ mil (Eq. 3); mitigação: *provisioned concurrency* ou aumento de cota.

A análise evidencia que o custo é dominado pelo *egress* da CDN (cerca de US$ 0,080 por GB) e que o primeiro gargalo a saturar não é a largura de banda (os segmentos são servidos pela borda da CDN, que escala), mas a concorrência da função *serverless* responsável pela consulta de manifesto, atingida em torno de 25 mil espectadores neste *footprint*; a mitigação natural é a concorrência provisionada ou o aumento de cota. O 10 mil medido caiu exatamente sobre a reta projetada, validando a linearidade até essa escala; acima dela, os valores são projeção, não medição.

Como o teto da Tabela 15 assume transmissão contínua 24/7, ele não representa a fatura real. A Tabela 16 traduz os mesmos patamares de concorrência para o perfil de consumo *intermitente* do caso de uso, no qual o pico de N espectadores corresponde a

$N \times 20$ acessos diários (concentração de pico de 5%): o custo efetivo é uma fração do teto, pois é dominado pelo *egress* do volume realmente consumido.

Tabela 16 – Projeção realista (perfil intermitente): pico de concorrência, volume diário e custo efetivo

Pico concorrente	**Acessos/dia**	***Egress*/dia**	**Custo realista/mês**
10.000	∼200 mil	∼18,7 TB	∼US$ 50 mil
15.000	∼300 mil	∼28 TB	∼US$ 75 mil
20.000	∼400 mil	∼37 TB	∼US$ 100 mil

Fonte: Elaborado pelo autor.

Nota: acessos/dia equivalentes ao pico de 5% ($N \times 20$). Custo itemizado pela mesma metodologia da Tabela 19, partindo do cenário de referência de 30 mil acessos/dia (∼US$ 7,9 mil/mês): o *egress*, as requisições e os *lookups* de manifesto escalam com os acessos, ao passo que as instâncias gerenciadas permanecem fixas e a observabilidade segue a ∼10% do subtotal. Contrasta com o teto 24/7 da Tabela 15, cerca de 25× maior.

Por que 25 mil: a derivação pela Lei de Little

A natureza desse gargalo é explicada pela Lei de Little, já adotada no modelo de projeção (Seção 6.5). Ela estabelece que o número médio de execuções concorrentes L de um recurso é o produto da taxa de chegada de requisições λ pelo tempo de serviço $\overline{W}$ de cada uma:

$$L = \lambda \cdot \overline{W}. \tag{2}$$

No plano de controle, a função *serverless* não é exercitada *durante* a reprodução (os segmentos são servidos pela CDN), mas apenas pelas consultas de catálogo e de manifesto. A taxa agregada dessas consultas cresce linearmente com o número de espectadores: $\lambda = \overline{r} \cdot N$, em que $\overline{r}$ é a taxa de consultas ao plano de controle por espectador. O ponto de saturação é alcançado quando a concorrência consumida iguala a cota de concorrência C disponível para a função. Substituindo em (2):

$$\overline{r} \cdot N_{\max} \cdot \overline{W} = C \qquad \Longrightarrow \qquad N_{\max} = \frac{C}{\overline{r} \cdot \overline{W}}. \tag{3}$$

Os valores deste *footprint* são consistentes com o teto observado, em ordem de grandeza: o cliente consulta o plano de controle a cada poucos segundos (um *poll* de

catálogo/estado da ordem de $\bar{r} \approx 0{,}3$ req/s por espectador), o tempo de serviço da função é da ordem de dezenas de milissegundos ($\overline{W} \approx 0{,}05$ s) e a cota de concorrência efetiva é da ordem de poucas centenas de execuções. Aplicando a Equação (3), obtém-se $N_{\max}$ na ordem de 10^4 espectadores, coerente com os 25 mil em que a Tabela 15 marca a saturação do *lambda*. O ponto decisivo da Equação (3) não é o valor exato (que depende da cota da conta e do padrão de consulta do cliente), mas sua forma: $N_{\max}$ é linear em C e inversamente proporcional a $\bar{r}$ e $\overline{W}$, o que expõe diretamente as três alavancas de escala.

Projeção: como escalar acima de 25 mil

A Equação (3) indica que escalar além de 25 mil é, antes de tudo, atuar sobre seus três fatores, sem trocar a arquitetura. A Tabela 17 projeta o efeito de cada alavanca, isolada e combinada.

Tabela 17 – Projeção de escala do plano de controle acima de 25 mil espectadores

Alavanca	**Efeito em $N_{\max}$**	**Patamar projetado**
Aumento de cota de concorrência ($C \to kC$)	Linear ($\times k$)	$N_{\max}$ cresce proporcionalmente a k; alavanca mais imediata (solicitação de cota ao provedor)
Provisioned concurrency (elimina *cold start*, reduz $\overline{W}$ em rajada)	$\propto 1/\overline{W}$	Remove a inflação de $\overline{W}$ nos picos de início de sessão
Manifesto/catálogo na CDN (reduz $\bar{r} \to 0$)	Gargalo migra	Tira o *lambda* do caminho crítico; teto passa ao *egress* da CDN

Fonte: Elaborado pelo autor.

As três alavancas expostas pela Equação (3) operam de forma independente. A mais imediata é elevar a cota de concorrência C junto ao provedor, pois $N_{\max}$ cresce linearmente com ela. A segunda é a *provisioned concurrency*, que mantém execuções pré-aquecidas e impede que o *cold start* infle $\overline{W}$ durante rajadas de início de sessão. A terceira é a alavanca estrutural: servindo o manifesto (e o catálogo, com *cache* de expiração curta) pela própria CDN, $\bar{r}$ tende a zero e o plano de controle deixa de ser o

limitante, deslocando o teto para o *egress* da CDN, que escala horizontalmente. Em todos os casos, a entrega de bytes nunca foi o gargalo; a escala do plano de controle é uma questão de cota e de *cache*, não de reescrita da arquitetura.

Quanto à QoE, é fundamental interpretar o *startup* em camadas (Tabela 18): os 32 ms medidos são o piso de infraestrutura (o *backend* entrega o primeiro segmento em dezenas de milissegundos), e não a experiência do usuário. A experiência percebida soma o tempo de inicialização do *player* e de decodificação (estimado em laboratório pela âncora Shaka) e, sobretudo, a rede de última milha e a capacidade do dispositivo do usuário, que só uma instrumentação em campo (*Real User Monitoring*) pode revelar.

Tabela 18 – Interpretação do *startup* em camadas

Camada	***Startup***	**O que mede**
`vclient`	$\sim$32 ms	Infraestrutura: entrega do primeiro segmento pelo *backend*
Shaka (laboratório)	$\sim$85 ms a 2 s	+ inicialização do *player* e decodificação (proxy de QoE)
Usuário real	$\geq$ 2 s	+ RTT, última milha e dispositivo (apenas *RUM* revela)

Fonte: Elaborado pelo autor.

Custo Itemizado da Operação

Enquanto a Tabela 15 projeta o *teto* de custo por número de espectadores *concorrentes* em transmissão contínua, esta seção decompõe o custo mensal *itemizado* no cenário-modelo de referência (30 mil acessos por dia, com consumo intermitente e codificação em H.265), base do modelo analítico de custo e capacidade (Seção 6.6). A Tabela 19 reúne todos os componentes da *stack* implantada, das alavancas dominantes às de valor residual.

Tabela 19 – Custo mensal itemizado da operação (cenário de referência analítico: 30 mil acessos/dia[c], H.265, AWS + CloudFront)

Componente	**Base de cálculo**	**US$/mês**
CDN — *egress* (CloudFront)	84.090 GB × US$ 0,080	6.727,20
CDN — requisições	67,5 M × US$ 0,012/10k	81,00
Lambda (*lookup* de manifesto)[a]	∼0,9 M req + GB-s	0,40
S3 *Standard* (transcodado)	111,9 GB × US$ 0,023	2,57
S3 *Archive* (bruto)	1.200 GB × US$ 0,004	4,80
Cache (ElastiCache Redis)	*cache*.t3.micro × 720 h	12,24
MongoDB (DocumentDB)	db.r8g.large × 720 h	218,88
Mensageria (Amazon MQ)	mq.t3.micro × 720 h	31,45
PostgreSQL (bases de resultados)	db.t4g.small × 720 h	23,00
Compute de transcodificação (GPU)[b]	∼3,3 h/dia × US$ 0,237	24,00
Observabilidade (CloudWatch/EMF)[b]	∼10% do subtotal	710,00
Total (≈)		**7.860**

Fonte: Elaborado pelo autor.

Notas: [a] O custo do *Lambda* é residual, mas seu papel é de capacidade, não de custo: a cota de concorrência da função (da ordem de poucas centenas de execuções) é o que impõe o primeiro ponto de saturação da arquitetura, atingido em torno de 25 mil espectadores concorrentes (Tabela 15, Eq. 3). [b] Valores estimados: o *compute* de transcodificação deriva do *speedup* relativo do Estudo 2 (e não de horas absolutas medidas), e a observabilidade segue a faixa de 5–15% do subtotal reportada na literatura de referência. [c] Cenário analítico de referência em escala reduzida (1/5 do perfil farmacêutico de 150 mil acessos/dia da Tabela 11); para o perfil completo, os volumes de *egress* e requisições escalam linearmente pelo mesmo fator 5.

A decomposição confirma o achado central do modelo: mesmo após a inclusão de todos os componentes residuais, o *egress* da CDN responde por cerca de 85% da fatura total (e por ∼95% do subtotal de infraestrutura, excluídas as estimativas variáveis de observabilidade e transcodificação). Isso mantém as três alavancas de redução de custo (*codec* mais eficiente, alta taxa de *cache-hit* e políticas de ciclo de vida no armazenamento) como as decisões de engenharia financeiramente mais relevantes.

Estudo 2: Eficiência de Codec

A Tabela 20 apresenta o BD-rate de cada *codec* e máquina, relativo à âncora `libx264` (valores negativos indicam maior eficiência, isto é, a mesma qualidade com menos bits).

Tabela 20 – BD-rate por *codec* e resolução (relativo a `libx264`; negativo = mais eficiente)

Encoder	***Backend***	**480p**	**720p**	**1080p**
h264 (libx264) — âncora	*software*	0%	0%	0%
h265 (libx265)	*software*	−13,5%	−23,7%	−33,3%
av1 (libsvtav1)	*software*	−43,4%	−48,4%	**−55,4%**
h264_nvenc (g6)	nvenc	+0,9%	−6,0%	−12,7%
h265_nvenc (g6)	nvenc	−8,4%	−19,9%	−30,6%
av1_nvenc (g6)	nvenc	−34,4%	−42,0%	−49,6%

Fonte: Elaborado pelo autor.

Notas: NVENC de 7ª geração (`g4dn`) deu valores praticamente idênticos aos da `g6` para H.264/H.265 (diferença < 1 ponto). Atenção: os encoders de *software* rodaram no *preset* `veryfast`, que subestima a eficiência de *software*; a magnitude da penalidade do NVENC é um limite superior (ver análise adiante).

Ameaça à Validade: *Preset* de *Software*

Os *encoders* de *software* foram executados no *preset* `veryfast`, calibrado para velocidade e não para qualidade. Isso subestima a eficiência do *software* e torna o NVENC artificialmente mais competitivo; é por esse motivo que o `h264_nvenc` aparece à frente do `libx264` em 480p. A *direção* da conclusão é robusta (o NVENC troca pouca eficiência por muita velocidade), mas a *magnitude* da penalidade deve ser lida como limite superior até a reexecução com *preset* `medium`/`slow`.

Os resultados confirmam a ordem de eficiência esperada da literatura: no *software*, o AV1 ($\sim -50\%$) supera o H.265 ($\sim -25\%$ a -33%), que supera o H.264. O achado central, porém, é que a penalidade do NVENC em relação ao *software* é pequena: em 1080p, o `h265_nvenc` ($-30{,}6\%$) fica apenas $\sim$3 pontos atrás do `libx265` ($-33{,}3\%$), e o `av1_nvenc` ($-49{,}6\%$) $\sim$6 pontos atrás do `libsvtav1` ($-55{,}4\%$). Confirmou-se ainda, empiricamente, que o NVENC é homogêneo entre gerações de GPU para H.264/H.265 (a `g4dn` e a `g6` diferem em menos de um ponto), de modo que, para a análise de eficiência, basta uma GPU por geração de codificação.

Cruzando o BD-rate com o ganho de velocidade do estudo de *throughput* (Tabela 21), obtém-se a afirmação acionável que motivou o estudo: o NVENC troca pouca eficiência por muita velocidade. O `h265_nvenc` entrega 8 a 12 vezes mais velocidade de codificação que o `libx265` pagando apenas $\sim$3% de BD-rate; o `av1_nvenc`, cerca de 6 vezes a velocidade por $\sim$6%.

Tabela 21 – *Trade-off* eficiência $\times$ velocidade em 1080p

Encoder	***Backend***	**BD-rate 1080p**	***Speedup* de *encode***
h264 (libx264)	*software*	0% (âncora)	1$\times$ (âncora)
h265 (libx265)	*software*	$-$33,3%	$\sim$1$\times$
av1 (libsvtav1)	*software*	$-$55,4%	$\sim$1$\times$
h264_nvenc (g6)	nvenc	$-$12,7%	$\sim$4$\times$
h265_nvenc (g6)	nvenc	$-$30,6%	$\sim$8–12$\times$
av1_nvenc (g6)	nvenc	$-$49,6%	$\sim$6$\times$

Fonte: Elaborado pelo autor (BD-rate deste estudo; *speedup* do estudo de *throughput*).

A interpretação para a engenharia da solução é direta: para um *pipeline* com muita recodificação ou requisitos de baixa latência, o NVENC é claramente vantajoso; para um cenário de VOD com catálogo grande e recodificação rara, o *software* compensa pela economia de banda de distribuição. Registra-se, contudo, a principal ameaça à validade deste resultado: o *software* foi executado no *preset* `veryfast`, que é rápido mas de baixa eficiência. Isso subestima a eficiência do *software* e torna o NVENC mais competitivo do que seria contra um *software* bem ajustado (em `medium`/`slow`), motivo pelo qual o `h264_nvenc` chega a aparecer à frente do `libx264` em 1080p. A direção da conclusão é robusta (o NVENC troca pouca eficiência por muita velocidade), mas a *magnitude* da penalidade deve ser relida contra um *software* em *preset* mais lento, passo natural de continuidade do estudo.

Esse *trade-off* tem desdobramento econômico direto. Embora uma instância com GPU seja mais cara por hora (na faixa de US$ 0,526 a US$ 0,805/h para as famílias `g4dn` e `g6`, contra US$ 0,179/h para a `c7i` de CPU), o ganho de 6 a 12 vezes em velocidade mais que compensa a diferença de preço: o custo por codificação cai, tornando a GPU não apenas mais rápida, mas também mais barata por vídeo processado em *pipelines* com recodificação intensiva. É essa economia, somada à baixa penalidade de eficiência (BD-rate), que justifica a alocação do plano de processamento à GPU (EC2/Batch) na

arquitetura proposta (Seção 5.4). A exceção é o catálogo de recodificação rara, no qual o baixo volume de codificação faz a economia de banda de distribuição (favorecida por um *software* bem ajustado) pesar mais que o custo de *compute*.

Síntese dos *Trade-offs*

Os estudos em execução dimensionam dois planos da arquitetura e expõem os *trade-offs* centrais do trabalho, sintetizados na Tabela 22. O Estudo 1 dimensiona o plano de dados (a entrega de bytes), que domina o custo; e o Estudo 2, a preparação (a codificação), cuja eficiência impacta diretamente o volume transmitido. O *footprint* medido no Estudo 1 é a constante que o modelo de custo e capacidade (Seção 6.6) multiplica pelo volume do caso de uso para responder, em termos quantitativos, ao custo de operar a arquitetura em escala.

Tabela 22 – Síntese dos *trade-offs* investigados

Trade-off		**Eixo**	**Achado**
Capacidade custo	×	Distribuição (Estudo 1)	O *egress* domina a fatura; a saturação ($\sim$25 mil) é da concorrência do lookup de manifesto, não da entrega de bytes.
Qualidade velocidade	×	Codec (Estudo 2)	A codificação por GPU (NVENC) troca $\sim$3–6% de eficiência por 6–12$\times$ de velocidade frente ao *software*.

Fonte: Elaborado pelo autor.

8 CONSIDERAÇÕES FINAIS

Este trabalho propôs e avaliou experimentalmente uma plataforma aberta de *streaming* de vídeo, concebida como instrumento de medição dos *trade-offs* entre desempenho, custo e qualidade. Estas considerações refletem sobre o percurso percorrido: as escolhas feitas, as mudanças de rota e os limites do que foi entregue.

A trajetória entre a concepção e a execução impôs revisões deliberadas. A proposta modular inicial evoluiu para uma plataforma de microsserviços poliglota, e a orquestração, antes pensada para contêineres, consolidou-se num recorte *serverless* (Lambda e EC2/Batch) aderente aos dois perfis de carga do problema: o controle, curto e elástico, e o processamento, longo e acelerado por GPU. Decisões de escopo também moldaram o resultado: o estudo de comparação de bancos de dados para o catálogo foi suspenso para concentrar o esforço nos eixos de maior impacto (distribuição e codificação), e o codificador de *software* foi executado no *preset* `veryfast`, escolha consciente de viabilidade de tempo que subestima sua eficiência e que, reconhece-se, seria revista num próximo ciclo por um *preset* mais lento.

Ambas as fases do trabalho foram cumpridas. A exploratória consolidou o referencial a partir de uma RSL rigorosa (19 artigos de alta relevância) e concebeu a arquitetura; a de execução implementou-a, instrumentou-a e a submeteu aos dois estudos de caso, cujas constantes empíricas alimentaram o modelo de custo e capacidade. A principal ressalva é metodológica, e não de completude: a magnitude da penalidade do NVENC deve ser relida contra um *software* mais bem ajustado.

As contribuições organizam-se em três planos. No plano metodológico, propôs-se e validou-se o método de medição em escala reduzida e projeção: a partir de *footprints* unitários medidos sobre uma implementação real e instrumentada, projeta-se o custo e a capacidade em larga escala, ancorando em evidência empírica, e na Lei de Little (LITTLE, 1961), constantes que a literatura genérica não fornece. No plano técnico, entregou-se uma arquitetura de referência integralmente aberta, materializada como microsserviços poliglotas, com artefatos reutilizáveis (bases de resultados duráveis e infraestrutura de *benchmark* provisionada como código). No plano científico, produziu-se evidência empírica dos *trade-offs* centrais: na distribuição, o *egress* domina o custo e o primeiro gargalo é de concorrência, não de banda; na codificação, o NVENC troca apenas cerca de 3 a 6% de eficiência (BD-rate) por um ganho de 6 a 12 vezes em velocidade.

Os resultados situam-se num escopo deliberadamente recortado. A plataforma cobre VoD, não *Live*; a análise empírica restringiu-se a três *codecs* (H.264, HEVC e AV1); e as métricas de QoE constituem um *piso* de infraestrutura, pois a experiência percebida em campo (rede de última milha e dispositivo do usuário) só é acessível por *Real User Monitoring*. As projeções, ancoradas em pontos medidos até 10 mil espectadores, são reportadas como limite superior. Um limite particularmente relevante diz respeito ao plano de dados: por pragmatismo, a entrega dos segmentos foi delegada a uma CDN comercial (CloudFront), de modo que a operação de uma CDN *self-hosted* não foi exercida experimentalmente (Seção 3.5). Ainda assim, o desenho aberto, com armazenamento compatível com a API do S3 via MinIO ou Ceph (Seção 3.10), preserva o caminho para a auto-hospedagem e a independência de fornecedor, mitigando o *vendor lock-in* em cenários de nuvem privada. Segurança, DRM e governança, críticos numa operação farmacêutica regulada, foram tratados apenas em nível de requisito, sem validação experimental. Os demais requisitos da Seção 5.1 foram validados pela própria condução do trabalho: os funcionais, pela implementação e operação de ponta a ponta dos módulos correspondentes nos experimentos; e os não funcionais, pela execução idêntica nos ambientes local e de nuvem (RNF01 e RNF03), pelo teste isolado de cada módulo nos controles de validade (RNF02) e pela variação de parâmetros entre cenários experimentais (RNF04).

Quanto ao processo de desenvolvimento, ferramentas de inteligência artificial e de automação foram empregadas como apoio de engenharia em etapas bem delimitadas: o assistente de codificação Claude Code, operando sobre modelos da família Claude (Anthropic), como ferramenta de produtividade no apoio ao desenvolvimento dos microsserviços e dos *scripts* de *benchmark*; o SonarQube e o CodeQL, na revisão de segurança e na análise estática do código nas *pipelines* de integração contínua dos repositórios; o Dependabot, na varredura de dependências com vulnerabilidades conhecidas; e o Playwright, nos testes automatizados de interface dos clientes web. Na preparação do manuscrito, a revisão textual foi conduzida pelo autor, com o mesmo assistente Claude Code empregado de forma pontual em apoios de escrita, como a sugestão de sinônimos e de alternativas de redação, e na conferência de padronização do texto (siglas, estrangeirismos e formatação de citações e referências segundo as normas da ABNT). Esse ferramental soma-se aos mecanismos de integridade do próprio método: a verificação de equivalência por *hash* criptográfico nos controles de validade (Capítulo 6) e os *scripts* de *benchmark* que reexecutam os experimentos de ponta a ponta, persistindo os

resultados nas bases dedicadas. Todo o código e o texto foram revisados e validados pelo autor; as decisões de arquitetura, o desenho experimental e as análises são integralmente de sua autoria.

Em síntese, demonstrou-se que uma implementação de referência aberta, instrumentada e operada em escala reduzida é capaz de projetar o custo e a capacidade de uma arquitetura de *streaming* em larga escala a partir de constantes empíricas, e não de estimativas genéricas. Os *trade-offs* quantificados (a dominância do *egress* no custo, o gargalo de concorrência do plano de controle e a relação eficiência $\times$ velocidade dos *codecs*) constituem resultados acionáveis para a engenharia de plataformas de VoD.

8.1 Trabalhos Futuros

Os resultados deste trabalho abrem frentes de continuidade diretas:

- **Extensão para *Live Streaming*:** adaptar a arquitetura a transmissões ao vivo de baixa (1–10 s) e ultra-baixa latência (<1 s), integrando protocolos de contribuição, empacotamento de baixa latência (LL-HLS, LL-DASH) e algoritmos ABR otimizados para minimizar a latência fim-a-fim sem comprometer a estabilidade da reprodução.
- **Avaliação de *codecs* emergentes:** estender o estudo de eficiência, hoje restrito a H.264, HEVC e AV1, aos codificadores de próxima geração: o VVC/H.266, considerado apenas no modelo de custo e em adoção crescente (inclusive no padrão brasileiro de TV 3.0), o LCEVC (camada de realce de baixa complexidade), o AV2 (sucessor aberto do AV1) e os codecs baseados em redes neurais (*Neural Network Video Coding*), reancorando as curvas *rate-distortion* e o BD-rate diante do estado da arte da codificação.
- **Transporte de próxima geração:** avaliar empiricamente protocolos de baixa latência identificados na RSL e ainda não exercitados, como o *Media over QUIC* (MoQ) e o Warp, comparando-os à entrega HTTP adaptativa quanto à latência fim-a-fim, à QoE e ao comportamento sob perda de pacotes.
- **Redução do *egress*:** como o *egress* da CDN domina o custo de operação (achado central deste trabalho), investigar mecanismos para reduzi-lo, em especial a entrega *peer-to-peer* assistida em eventos de alta audiência e a otimização das políticas de *cache* e da *bitrate ladder*, quantificando a economia frente ao impacto na QoE.

- **Superação do gargalo de concorrência do plano de controle:** como o primeiro ponto de saturação não é a banda, mas a cota de concorrência da função *serverless* (~25 mil espectadores, derivado pela Lei de Little), avaliar empiricamente as alternativas que movem cada fator da equação $N_{\max} = C/(\bar{r} \cdot \overline{W})$: (i) migrar o plano de requisição para um modelo baseado em instâncias ou contêineres (EC2/ECS/Fargate sobre *auto scaling*), trocando o teto de cota pelo de CPU, memória e conexões, sob o *trade-off* IaaS $\times$ FaaS já discutido; (ii) deslocar a geração e a leitura de manifesto e catálogo para a borda (via *cache* de CDN com TTL curto, CloudFront Functions ou Lambda@Edge), levando $\bar{r} \to 0$ na origem e migrando o teto para o *egress* da CDN, que escala horizontalmente; e (iii) comparar o modelo de concorrência regional do Lambda com plataformas de *edge compute* de concorrência distribuída por *PoP* (por exemplo, Cloudflare Workers ou Fastly Compute@Edge), medindo capacidade, latência e custo em cada recorte.
- **Aprofundamento dos estudos experimentais:** reexecutar o Estudo 2 com o *software* em *preset* mais lento (`medium`/`slow`) e reancorar o BD-rate, para dimensionar a penalidade real do NVENC; instrumentar *Real User Monitoring* (RUM) no cliente, medindo o *startup time* percebido em campo por região e dispositivo; calcular o VMAF de forma assíncrona, reduzindo o custo de computação em GPU; validar cenários de múltiplas CDN com *content steering*; e comparar empiricamente as famílias de algoritmos ABR (baseados em vazão, em *buffer*, híbridos e em aprendizado de máquina) sob condições idênticas, aproveitando o gerador `vclient`.
- **Avaliação de uma camada *self-hosted*:** medir o custo e a viabilidade de operar a CDN e o armazenamento de objetos de forma autogerida (Ceph ou MinIO, compatíveis com a API do S3), quantificando o *trade-off* entre a independência de fornecedor e o esforço operacional em cenários de nuvem privada ou híbrida.
- **Portabilidade entre provedores de nuvem:** reproduzir a arquitetura em outros provedores, como Google Cloud Platform (Cloud Functions/Cloud Run, Compute Engine e Cloud CDN) e Microsoft Azure (Functions, Virtual Machines e Azure CDN/Front Door), comparando empiricamente custo, desempenho, modelo de concorrência *serverless* e preço de *egress* sob carga idêntica. Por se apoiar em ferramentas de código aberto e em armazenamento compatível com a API do S3, a plataforma reduz o atrito dessa migração; medir o esforço de portabilidade e

os diferenciais de custo entre provedores quantificaria o grau de *vendor lock-in* e subsidiaria estratégias *multi-cloud*.

- **Eficiência energética e sustentabilidade:** dimensionar o consumo energético da codificação e o *trade-off* entre compressão e energia (AV1 *versus* VVC), incorporando a sustentabilidade como critério explícito na escolha de *codec*.
- **Análise do consumo de memória sob escala:** perfilar o uso de memória de cada serviço sob carga crescente (em especial o de distribuição e o de empacotamento sob alta concorrência), identificando vazamentos e picos de alocação que complementem o modelo de custo e capacidade com uma dimensão ainda não coberta.

Por fim, a consolidação dos artefatos, do método e dos resultados em um artigo científico facilitaria o acesso, a reprodução e a extensão por outros pesquisadores, dado o uso predominante de ferramentas de código aberto e a disponibilização pública de todo o código-fonte da plataforma (Seção 5.2).

REFERÊNCIAS

ALSADER, M.; BARAKABITZE, A. A.; MKWAWA, I.-H. QoE-driven adaptive video streaming: Architectures, techniques, and future research challenges toward 6G networks. **IEEE Access**, v. 13, p. 157408–157441, 2025.

Amazon Web Services. **AWS Well-Architected Framework**. 2024. Disponível em: <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>. Acesso em: 14 jun. 2026.

Amazon Web Services. **Video on Demand on AWS: Cost**. 2024. Disponível em: <https://docs.aws.amazon.com/solutions/latest/video-on-demand-on-aws/cost.html>. Acesso em: 14 jun. 2026.

Amazon Web Services. **AWS Elemental MediaPackage**. 2025. Disponível em: <https://aws.amazon.com/mediapackage/>. Acesso em: 17 dez. 2025.

Amazon Web Services. **AWS Pricing**. 2026. Disponível em: <https://aws.amazon.com/pricing/>. Acesso em: 19 jun. 2026.

ARMBRUST, M. *et al.* A view of cloud computing. **Communications of the ACM**, ACM, v. 53, n. 4, p. 50–58, 2010.

ASAN, A. Quality of experience assessment for enhanced video streaming services. 2024.

BELSHE, M.; PEON, R.; THOMSON, M. **Hypertext Transfer Protocol Version 2 (HTTP/2)**. [S.l.], 2015. Disponível em: <https://www.rfc-editor.org/info/rfc7540>.

BENTALEB, A. *et al.* Toward one-second latency: Evolution of live media streaming. **IEEE Communications Surveys & Tutorials**, v. 28, p. 2418–2456, 2026.

BISHOP, M. **HTTP/3**. [S.l.], 2022. Disponível em: <https://www.rfc-editor.org/info/rfc9114>.

BJØNTEGAARD, G. **Calculation of Average PSNR Differences between RD-curves**. Austin, Texas, USA, 2001.

Broadcom Inc. **RabbitMQ Documentation**. 2024. Disponível em: <https://www.rabbitmq.com/docs>. Acesso em: 18 jun. 2026.

BROWN, S. **The C4 Model for Visualising Software Architecture**. 2024. Disponível em: <https://c4model.com>. Acesso em: 14 jun. 2026.

BUKHARI, S. M. A. H. *et al.* Video transcoding at the edge: cost and feasibility perspective. **Cluster Computing**, Springer, 2022.

BURNS, B. *et al.* Borg, omega, and kubernetes. **Communications of the ACM**, ACM, v. 59, n. 5, p. 50–57, 2016.

DRAGONI, N. *et al.* Microservices: Yesterday, today, and tomorrow. In: MAZZARA, M.; MEYER, B. (Ed.). **Present and Ulterior Software Engineering**. Cham: Springer, 2017. p. 195–216.

ERICSSON. **Exploring mobile traffic trends 2020-2024**. 2024. In: *Ericsson Mobility Report*. Disponível em: <https://www.ericsson.com/en/reports-and-papers/mobility-report/articles/exploring-mobile-traffic-trends-2020-2024>. Acesso em: 27 ago. 2025.

FARAHANI, R. *et al.* Cp-steering: Cdn- and protocol-aware content steering solution for http adaptive video streaming. In: **Proceedings of the 2nd Mile-High Video Conference**. [S.l.: s.n.], 2023. p. 149–150.

FFmpeg Developers. **FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video**. 2024. Disponível em: <https://ffmpeg.org>. Acesso em: 18 jun. 2026.

FIELDING, R. *et al.* **Hypertext Transfer Protocol – HTTP/1.1**. [S.l.], 1999. Disponível em: <https://www.rfc-editor.org/info/rfc2616>.

FIELDING, R.; LAFON, Y.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Range Requests**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7233>.

FIELDING, R.; NOTTINGHAM, M.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Caching**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7234>.

FIELDING, R.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Authentication**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7235>.

FIELDING, R.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7232>.

FIELDING, R.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7230>.

FIELDING, R.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. [S.l.], 2014. Disponível em: <https://www.rfc-editor.org/info/rfc7231>.

ITU-T. **Recommendation ITU-T P.910: Subjective video quality assessment methods for multimedia applications**. [S.l.], 2023. Disponível em: <https://www.itu.int/rec/T-REC-P.910-202310-I/en>. Acesso em: 19 jun. 2026.

IYENGAR, J.; THOMSON, M. **QUIC: A UDP-Based Multiplexed and Secure Transport**. [S.l.], 2021. Disponível em: <https://www.rfc-editor.org/info/rfc9000>.

KITCHENHAM, B. A. Systematic review in software engineering: where we are and where we should be going. In: **Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies**. New York, NY, USA: Association for Computing Machinery, 2012. (EAST '12), p. 1–2. ISBN 9781450315098. Disponível em: <https://doi.org/10.1145/2372233.2372235>.

LAGHARI, A. A. *et al.* The state of art and review on video streaming. **Journal of High Speed Networks**, v. 29, n. 3, p. 211–236, 2023.

LANGENDORF, L. N.; KHALID, M. S. Discovering improvement opportunities and challenges for pharmaceutical companies adopting digital training technologies: A case study. In: **Learning and Collaboration Technologies. Lecture Notes in Computer Science, vol. 14722. HCI International 2024**. [S.l.]: Springer, Cham, 2024.

LIKERT, R. A technique for the measurement of attitudes. **Archives of psychology**, v. 140, p. 1–55, 1932.

LITTLE, J. D. C. A proof for the queuing formula: $L = \lambda W$. **Operations Research**, v. 9, n. 3, p. 383–387, 1961.

MAURO, T. **Adopting Microservices at Netflix: Lessons for Architectural Design**. 2015. NGINX Blog. Disponível em: <https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/>. Acesso em: 21 jun. 2026.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. Gaithersburg, MD, USA, 2011.

MOINA-RIVERA, W. *et al.* Cloud media video encoding: review and challenges. **Multimedia Tools and Applications**, v. 83, n. 34, p. 81231–81278, 2024.

NEIVA, F. W.; SILVA, R. L. d. S. da. **Revisão Sistemática da Literatura em Ciência da Computação: Um Guia Prático**. Juiz de Fora, 2016.

NEWMAN, S. **Building Microservices: Designing Fine-Grained Systems**. 2. ed. Sebastopol, CA: O'Reilly Media, 2021. ISBN 9781492034025.

NGINX. **Nginx**. 2025. Disponível em: <https://nginx.org/>. Acesso em: 17 dez. 2025.

NYGREN, E.; SITARAMAN, R. K.; SUN, J. The Akamai network: A platform for high-performance internet applications. **ACM SIGOPS Operating Systems Review**, ACM, v. 44, n. 3, p. 2–19, 2010.

PALMER, M. **Towards Enabling Cross-layer Information Sharing to Improve Today's Content Delivery Systems**. Tese (Doutorado) — Universität des Saarlandes, Saarbrücken, 2023.

PERONI, L.; GORINSKY, S. An end-to-end pipeline perspective on video streaming in best-effort networks: A survey and tutorial. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 57, n. 12, jul. 2025. Disponível em: <https://doi.org/10.1145/3742472>.

PRODANOV, C. C.; FREITAS, E. C. d. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. 2. ed. Novo Hamburgo: Feevale, 2013.

RD Saúde. **Relatório Anual e de Sustentabilidade 2023**. 2024. Base de 47,6 milhões de clientes ativos ao final de 2023. Disponível em: <https://rdsaude.com.br/relatorio-2023/>. Acesso em: 21 jun. 2026.

RODRIGUES-FILHO, R. *et al.* Content steering: Leveraging the computing continuum to support adaptive video streaming. In: **Minicursos do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.]: SBC, 2024. p. 1–39.

Shaka Project. **Shaka Packager**. 2025. Disponível em: <https://github.com/shaka-project/shaka-packager>. Acesso em: 17 dez. 2025.

SILVA, R. Brito da; MATTOS, C. A. de. Critical success factors of a drug traceability system for creating value in a pharmaceutical supply chain (PSC). **International Journal of Environmental Research and Public Health**, v. 16, n. 11, p. 1972, 2019.

SRIDHARAN, C. **Distributed Systems Observability: A Guide to Building Robust Systems**. Sebastopol, CA: O'Reilly Media, 2018.

STACCIARINI, J. H. S. **A Consolidação do Setor Farmacêutico na Economia Global: crescimento, influência, desvios e marketing**. 167 p. Tese (Tese (Doutorado em Geografia)) — Universidade Federal de Goiás, Goiânia, 2023.

SUMAN, P.; MOON, Y.-S.; CHOI, M.-J. Netflix, amazon prime, and youtube: comparative study of streaming infrastructure and strategy. **Journal of Information Processing Systems**, v. 18, n. 6, p. 729–740, 2022.

Synergy Research Group. **Worldwide Market Share of Leading Cloud Infrastructure Service Providers**. 2024. Dados compilados pela Statista. Disponível em: <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>. Acesso em: 17 jun. 2026.

THOMSON, M.; BENFIELD, C. **HTTP/2**. [S.l.], 2022. Disponível em: <https://www.rfc-editor.org/info/rfc9113>.

TIMMERER, C. *et al.* Http adaptive streaming: A review on current advances and future challenges. **ACM Trans. Multimedia Comput. Commun. Appl.**, New York, NY, USA, v. 21, n. 7, jul. 2025. ISSN 1551-6857.

Unified Streaming. **Unified Streaming Platform**. 2025. Disponível em: <https://www.unified-streaming.com/>. Acesso em: 17 dez. 2025.

WEIL, S. A. *et al.* Ceph: A scalable, high-performance distributed file system. In: **Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)**. [S.l.: s.n.], 2006. p. 307–320.

XENAKIS, D. To dash, or not to dash? optimal video bitrate selection and edge network caching in mec-empowered slice-enabled networks. **IEEE Transactions on Vehicular Technology**, v. 73, n. 4, p. 5556–5571, April 2024. ISSN 1939-9359.

YouTube. **YouTube for Press**. 2024. Mais de 2 bilhões de usuários conectados mensais. Disponível em: <https://www.youtube.com/about/press/>. Acesso em: 17 jun. 2026.

YUAN, D. *et al.* PRIOR: deep reinforced adaptive video streaming with attention-based throughput prediction. In: **Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video**. [S.l.: s.n.], 2022. p. 36–42.

ZHAO, J.; PAN, J. Qoe-driven joint decision-making for multipath adaptive video streaming. In: IEEE. **GLOBECOM 2023-2023 IEEE Global Communications Conference**. [S.l.], 2023. p. 128–133.

APÊNDICE A – DETALHAMENTO EXPERIMENTAL DO TESTE DE CARGA DA DISTRIBUIÇÃO

Este apêndice reúne o detalhamento experimental do estudo de carga da distribuição (Seção 6.8), incluindo a comparação entre instrumentos de geração de carga, a evidência empírica de que os gargalos observados pertenciam ao aparato de medição, o escalonamento da âncora de fidelidade e excertos da implementação do gerador leve `vclient`. Todos os pontos foram persistidos na base de resultados do estudo; os números de *startup* são reportados em percentis (p50, p95 e p99) por execução, computados nativamente sobre as amostras por *viewer*.

A.1 Comparação de Instrumentos no Tier Local (T1)

Para validar o gerador leve `vclient` contra os instrumentos existentes, executou-se o mesmo ativo local com os três motores (Tabela 23). O `vclient` apresenta baixo *overhead* de inicialização (no mesmo $N = 50$ local, 1,4 s) e casa com a âncora Shaka nos pontos pequenos; a diferença para o Shaka em $N = 10$ (366 ms contra 1.752 ms) corresponde ao *bootstrap* do navegador, que o Shaka inclui e o `vclient` não. A partir de 500 espectadores locais, o *startup* dispara (46 s a 1.000), mas por saturação do *laptop* (que hospeda o armazenamento e os geradores no mesmo *host*), não por limitação do `vclient`.

Tabela 23 – Comparação de instrumentos no T1 local (mesmo ativo)

N	Motor	*Startup* p50	*Bitrate*	Entregues	Obs.
10	shaka	1.752 ms	6000	10/10	inclui *bootstrap* do navegador
10	`vclient`	366 ms	5526	10/10	TTFF puro (*localhost*)
50	`vclient`	1.423 ms	5503	50/50	gerador leve, sem decodificação
100	`vclient`	2.899 ms	4266	100/100	limpo
500	`vclient`	11.806 ms	3000	412/500	*laptop* satura
1.000	`vclient`	46.172 ms	3000	1000/1000	*laptop* satura

Fonte: Elaborado pelo autor.

A.2 Confirmação em Gerador Dedicado (EC2)

Repetindo o mesmo *ladder* a partir de uma instância EC2 dedicada contra a distribuição em nuvem, o *startup* escala suavemente (de 279 ms a 2 s em 1.000), com 100% de entrega, *bitrate* cheio e *rebuffer* zero (Tabela 24). O contraste direto nos pontos que saturavam o *laptop* (Tabela 25) é a evidência empírica de que o gargalo era o gerador co-locado, não a distribuição.

Tabela 24 – `vclient` a partir de EC2 dedicada (distribuição em nuvem)

N	*Startup* p50	p95	p99	*Bitrate*	Entregues	*Rebuffer*
10	279 ms	284 ms	286 ms	5400	10/10	0
50	145 ms	312 ms	339 ms	5400	50/50	0
100	237 ms	455 ms	582 ms	5400	100/100	0
500	1.053 ms	2.116 ms	2.794 ms	5400	500/500	0
1.000	2.001 ms	3.887 ms	5.193 ms	5308	1000/1000	0

Fonte: Elaborado pelo autor.

Tabela 25 – Gerador co-locado (*laptop*) *versus* dedicado (EC2)

N	*Laptop* (co-locado)	EC2 dedicada
500	11.806 ms · 412/500 · 3000 kbps	1.053 ms · 500/500 · 5400 kbps
1.000	46.172 ms · 1000/1000 · 3000 kbps	2.001 ms · 1000/1000 · 5308 kbps

Fonte: Elaborado pelo autor.

A.3 Efeito da Largada Escalonada (*ramp*)

Mesmo na EC2, o *startup* ainda crescia de $\sim$250 ms para $\sim$2 s por causa da largada sincronizada (todos os viewers baixam o primeiro segmento em $t = 0$ e dividem a banda da placa de rede). O parâmetro `-ramp` (viewers por segundo) espaça as partidas e dissolve a rajada, mantendo o *startup* plano (Tabela 26).

Tabela 26 – Efeito do *ramp* de chegada (EC2, *ramp* de 200/s)

N	Sem *ramp*	Com *ramp*	Entregues	*Bitrate*
10	279 ms	37 ms	10/10	5400
100	237 ms	37 ms	100/100	5400
500	1.053 ms	67 ms	500/500	5400
1.000	2.001 ms	336 ms	1000/1000	5400

Fonte: Elaborado pelo autor.

A.4 Âncora de 10 Mil Medida Limpa e a Prova da Placa de Rede

Combinando frota (banda agregada) e *ramp*, mediram-se 10 mil viewers contra a CDN, com *startup* de 1,45 s. A prova definitiva de que o *startup* residual era utilização da placa de rede do gerador veio ao repetir os 10 mil em uma única instância de 200 Gbps: trocando apenas a placa de rede do gerador, o *startup* despencou para 32 ms, o mesmo piso de $N = 10$ (Tabela 27).

Tabela 27 – Os 10 mil espectadores sob diferentes geradores (mesmo sistema sob teste)

Gerador	*Startup* p50	p95	p99	Entregues	*Bitrate*
`vclient` frota (6×, ∼72% da NIC)	1.454 ms	4.642 ms	6.799 ms	9996/10002	5129
`vclient` 1 nó (200 Gbps)	**32 ms**	58 ms	262 ms	10000/10000	5400

Fonte: Elaborado pelo autor.

A.5 Escalonamento da Âncora Shaka (*player* real até 100)

A âncora de fidelidade usa o *player* real (Shaka, em navegador), mantida em amostra pequena por decodificar vídeo. Escalando-a a 100 com *boots* escalonados (mesmo princípio do *ramp*, aqui aplicado à CPU), o *startup* permanece plano (∼83–99 ms p50, p95 $\leq$ 150 ms) com *rebuffer* de ∼0,1–0,2% (Tabela 28). Sem o escalonamento, o pico de CPU da largada simultânea inflava o *startup* a 258 ms (p50), 514 ms (p95) e 614 ms (p99), novamente um artefato do gerador, não da distribuição.

Tabela 28 – Âncora Shaka escalada a 100 *players* reais (*boots* escalonados)

N	***Startup* p50**	**p95**	**p99**	***Rebuffer***	***Bitrate***
1	83 ms	83 ms	83 ms	0,12%	6000
20	84 ms	93 ms	96 ms	0,14%	6000
40	85 ms	99 ms	103 ms	0,15%	6000
60	87 ms	105 ms	107 ms	0,16%	6000
80	94 ms	125 ms	137 ms	0,17%	6000
100	99 ms	149 ms	159 ms	0,19%	6000

Fonte: Elaborado pelo autor.

A.6 Excertos de Implementação do `vclient`

A seguir, os blocos centrais (condensados) que tornam o gerador leve fiel à QoE de um *player* real sem decodificar vídeo. O código completo encontra-se no repositório do serviço de distribuição[1] (Seção 5.2).

A ideia central é simular o *buffer* em vez de decodificar: um segmento baixado entra no *buffer*; a reprodução o drena em tempo real; *buffer* vazio com conteúdo restante caracteriza um *stall* (*rebuffer*).

```
// Play avanca a reproducao em `elapsed` segundos, drenando o buffer.
func (b *Buffer) Play(elapsed float64) {
    consume := elapsed
    if consume > b.level { consume = b.level }
    b.level -= consume
    b.played += consume
    if short := elapsed - consume; short > 1e-9 { // nao coberto pelo buffer
        b.stall += short
        if !b.stalling { b.stalls++; b.stalling = true } // novo rebuffer
    } else {
        b.stalling = false
    }
}
func (b *Buffer) RebufferRatio() float64 { return b.stall / (b.played + b.stall) }
```

Listing A.1: Modelo de buffer (vclient/buffer.go)

A adaptação de *bitrate* combina a estimativa de vazão (média móvel exponencial) com uma guarda de *buffer*:

[1] Disponível em: <https://github.com/Michelalmeidasilva/streaming-distribution>, diretório `bench/scale/`. Acesso em: 22 jun. 2026.

```
func (a *ABR) Observe(bytes int64, dt time.Duration) {
    tp := float64(bytes) / dt.Seconds()
    if a.estBps == 0 { a.estBps = tp } else {
        a.estBps = a.alpha*tp + (1-a.alpha)*a.estBps }
}
func (a *ABR) Decide(rs []Rendition, bufferSec float64) int {
    budget := a.estBps * 8 * a.safety // bits/s sustentavel
    pick := 0
    for i, r := range rs { if float64(r.Bandwidth) <= budget { pick = i } }
    if bufferSec < lowBufferSec && pick > 0 { pick-- } // evita stall
    return pick
}
```

Listing A.2: ABR: vazao (EWMA) + guarda de buffer (vclient/abr.go)

A largada escalonada (*ramp*) espaça as partidas para não dividir a placa de rede em $t = 0$:

```
// Pausa entre largadas para uma taxa de chegada (viewers/s); 0 = todos juntos.
func RampDelay(viewersPerSec float64) time.Duration {
    if viewersPerSec <= 0 { return 0 }
    return time.Duration(float64(time.Second) / viewersPerSec)
}
```

Listing A.3: Largada escalonada (vclient/ramp.go)

Por fim, o motor de projeção extrapola o *footprint* por viewer para N concorrentes, derivando *egress*, conexões (Lei de Little) e o primeiro *tier* a saturar:

```
func Project(fp Footprint, n int64, cm CostModel, basis string) Projection {
    nf := float64(n)
    egressGBH := fp.EgressBytesPerSec * nf * 3600 / 1e9 // GB/h
    aggRPS := fp.RPSPerViewer * nf
    connections := aggRPS * fp.AvgLatencySec            // Lei de Little
    costCDN := egressGBH * 730 * cm.CloudFrontPerGB     // egress da CloudFront domina
    tier := "none"
    if connections > cm.LambdaConcurrencyLimit { tier = "lambda" } // 1o gargalo
    return Projection{NTarget: n, EgressGBH: egressGBH, AggRPS: aggRPS,
        Connections: int64(connections), SaturationTier: tier}
}
```

Listing A.4: Motor de projecao (project/project.go)

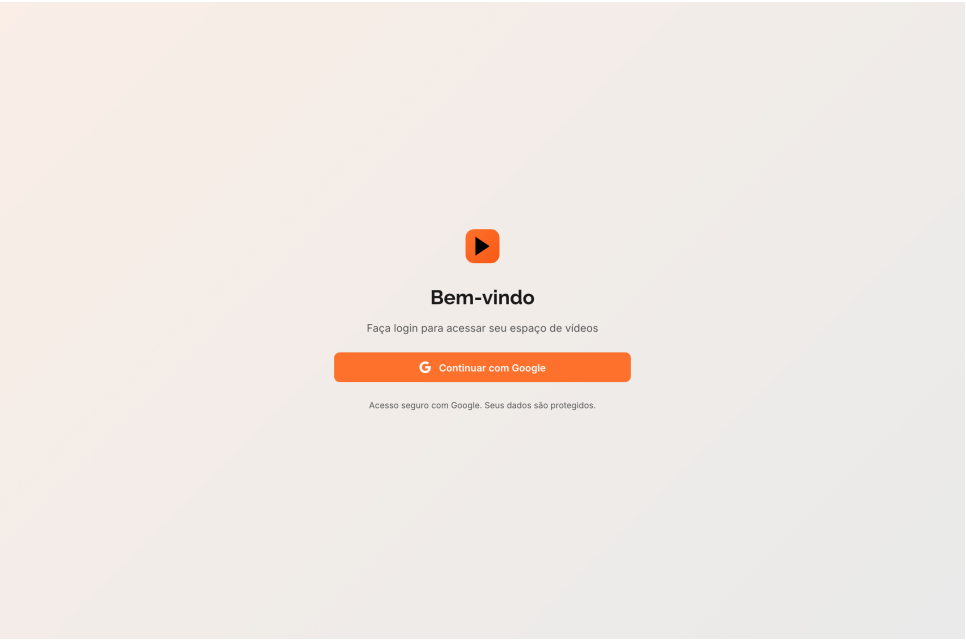
APÊNDICE B – TELAS DAS APLICAÇÕES

Este apêndice apresenta capturas de tela das duas aplicações de *frontend* da plataforma, em execução sobre o ambiente local (*stack* de microsserviços orquestrada por Docker, com catálogo de vídeos real transcodado): o painel administrativo de *upload* (`streaming-platform-upload`) e o cliente *web* de consumo (`streaming-web-client`).

B.1 Painel de *Upload* (`streaming-platform-upload`)

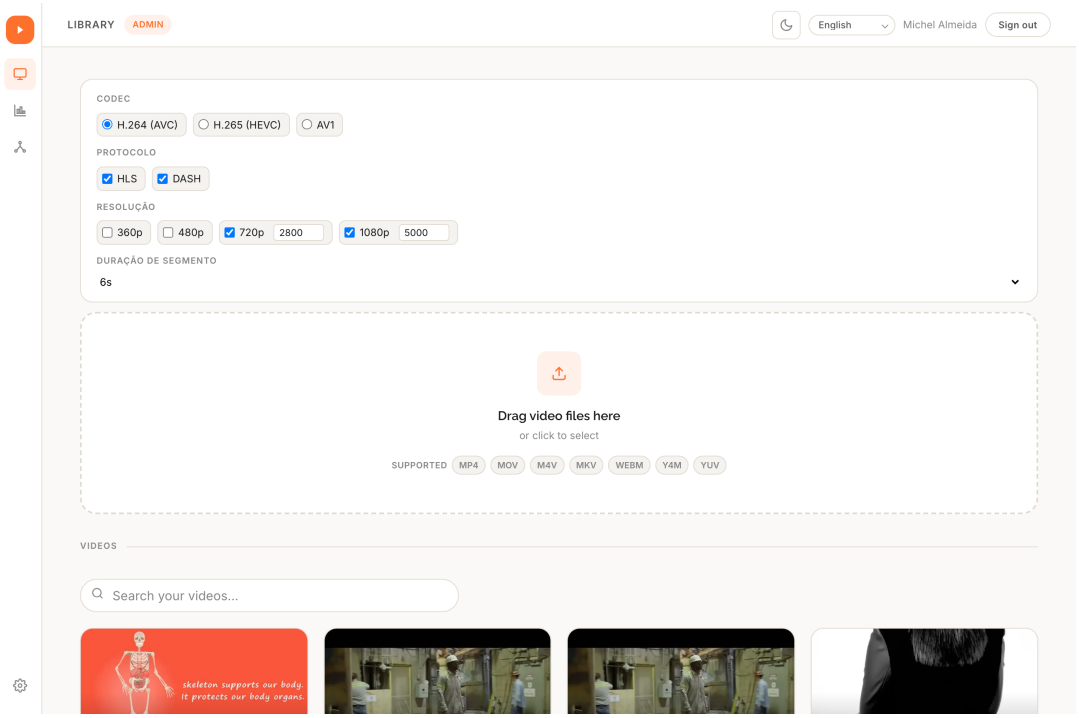
O `streaming-platform-upload` é a interface administrativa para ingestão de vídeos, construída em Next.js. O acesso é protegido por autenticação (Figura 19), com controle de acesso baseado em papéis. Para usuários com perfil de administrador, o painel expõe a configuração parametrizável da ingestão (seleção de *codec*: H.264, H.265 ou AV1; protocolos de empacotamento: HLS e/ou DASH; *bitrate ladder* por resolução; e duração de segmento), acima da área de *upload multipart* (*arrastar-e-soltar*) e do catálogo de vídeos com o respectivo estado de processamento (Figura 20). Essa tela materializa, na prática, a natureza parametrizável da arquitetura discutida ao longo do trabalho.

Figura 19 – Tela de autenticação do painel de *upload*



Fonte: Elaborado pelo autor.

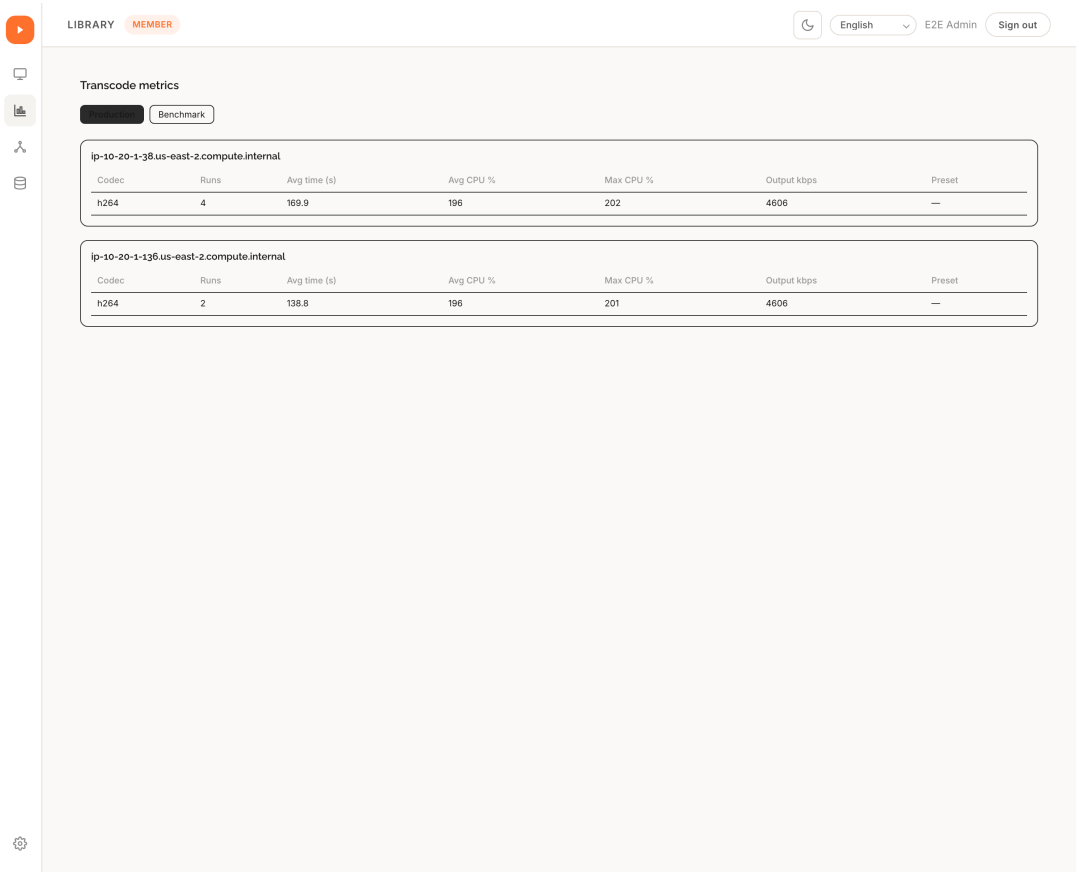
Figura 20 – Painel de *upload* (perfil administrador): configuração parametrizável de *codec*, protocolo, resolução e duração de segmento, área de envio *multipart* e catálogo. O valor de duração de segmento exibido (6 s) é o padrão configurável do painel; os experimentos deste trabalho fixaram segmentos de 4 s.



Fonte: Elaborado pelo autor.

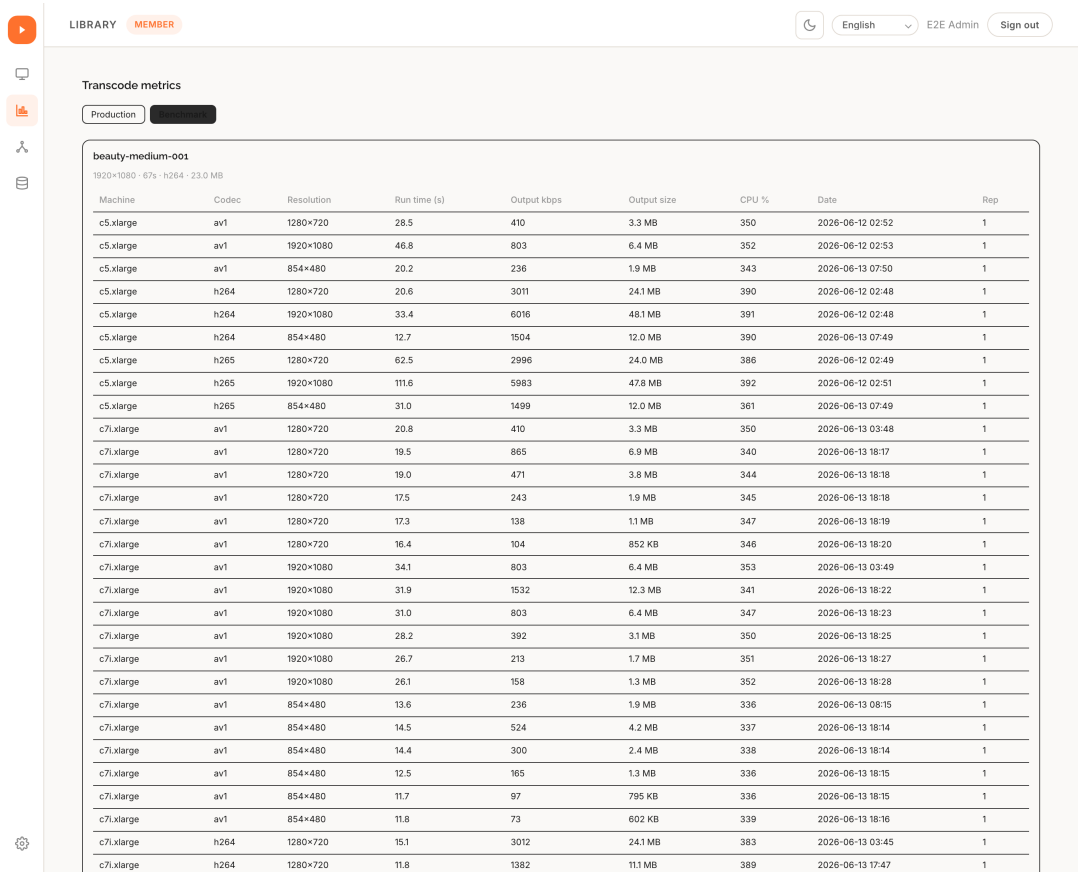
O painel também integra um *dashboard* de métricas de transcodificação, alimentado pela telemetria do serviço de *ingest*. A aba *Production* agrega, por máquina e *codec*, o tempo médio de codificação, o uso de CPU e a taxa de saída dos vídeos efetivamente processados (Figura 21); a aba *Benchmark* expõe os dados brutos da campanha de *benchmark* de *codec* (Estudo 2), com tempo de codificação, *bitrate* e CPU por máquina, *codec* e resolução (Figura 22).

Figura 21 – *Dashboard* de métricas de transcodificação — aba *Production* (estatísticas por máquina e *codec*)



Fonte: Elaborado pelo autor.

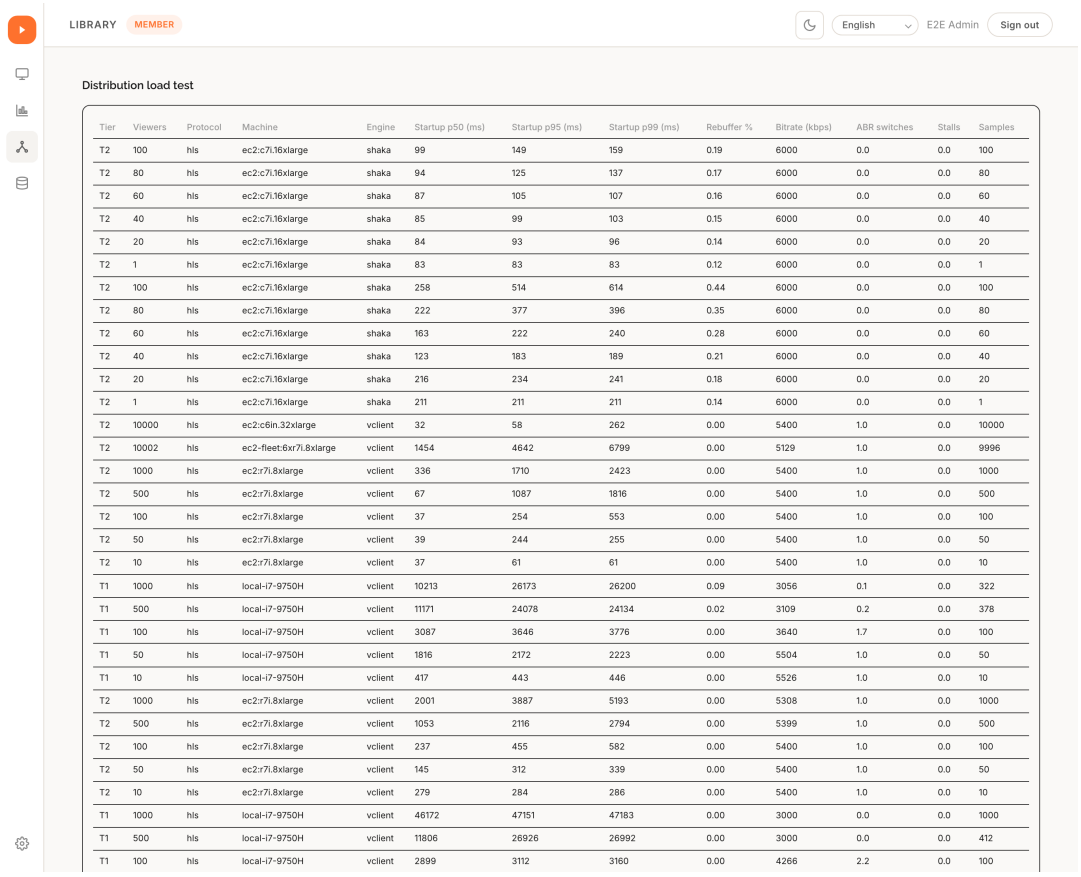
Figura 22 – *Dashboard* de métricas de transcodificação — aba *Benchmark* (dados brutos da campanha de eficiência de *codec*)



Fonte: Elaborado pelo autor.

O painel reúne ainda o *dashboard* do teste de carga da distribuição (Estudo 1), que consolida a QoE medida por *tier*, número de espectadores e instrumento (*startup* em percentis, *rebuffer*, *bitrate*), incluindo o ponto de 10 mil espectadores concorrentes medido limpo (Figura 23).

Figura 23 – *Dashboard* do teste de carga da distribuição (QoE por *tier*, espectadores e instrumento)



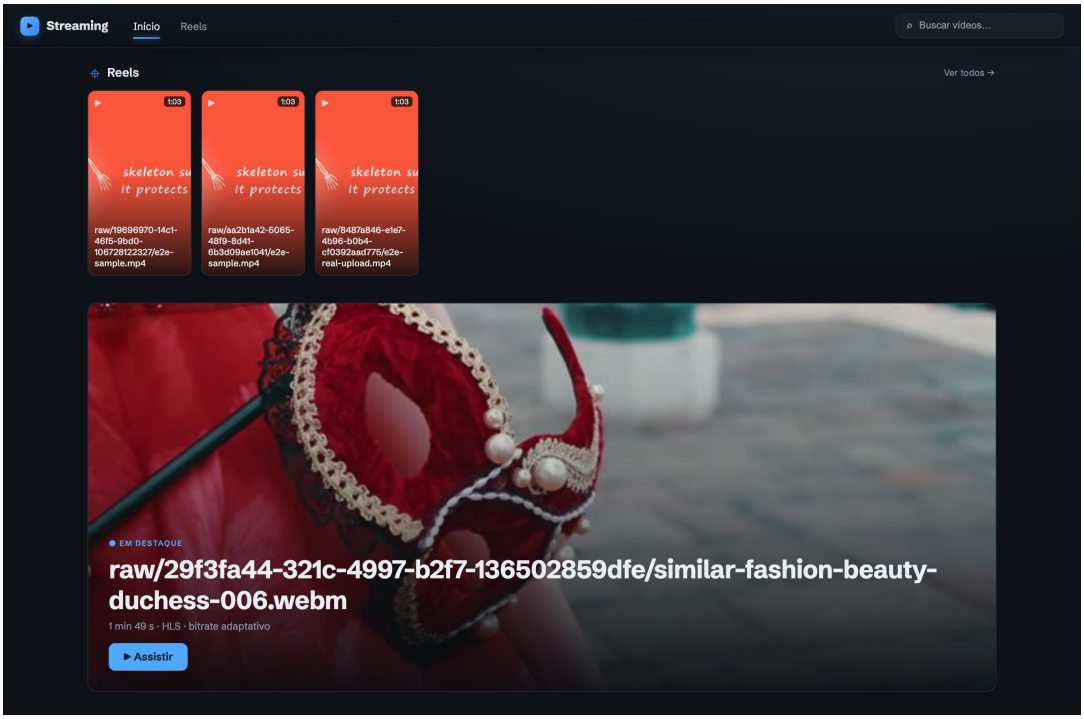
Tier	Viewers	Protocol	Machine	Engine	Startup p50 (ms)	Startup p95 (ms)	Startup p99 (ms)	Rebuffer %	Bitrate (kbps)	ABR switches	Stalls	Samples
T2	100	hls	ec2:c7i.16xlarge	shaka	99	149	159	0.19	6000	0.0	0.0	100
T2	80	hls	ec2:c7i.16xlarge	shaka	94	125	137	0.17	6000	0.0	0.0	80
T2	60	hls	ec2:c7i.16xlarge	shaka	87	105	107	0.16	6000	0.0	0.0	60
T2	40	hls	ec2:c7i.16xlarge	shaka	85	99	103	0.15	6000	0.0	0.0	40
T2	20	hls	ec2:c7i.16xlarge	shaka	84	93	96	0.14	6000	0.0	0.0	20
T2	1	hls	ec2:c7i.16xlarge	shaka	83	83	83	0.12	6000	0.0	0.0	1
T2	100	hls	ec2:c7i.16xlarge	shaka	258	514	614	0.44	6000	0.0	0.0	100
T2	80	hls	ec2:c7i.16xlarge	shaka	222	377	396	0.35	6000	0.0	0.0	80
T2	60	hls	ec2:c7i.16xlarge	shaka	163	222	240	0.28	6000	0.0	0.0	60
T2	40	hls	ec2:c7i.16xlarge	shaka	123	183	189	0.21	6000	0.0	0.0	40
T2	20	hls	ec2:c7i.16xlarge	shaka	216	234	241	0.18	6000	0.0	0.0	20
T2	1	hls	ec2:c7i.16xlarge	shaka	211	211	211	0.14	6000	0.0	0.0	1
T2	10000	hls	ec2:c6in.32xlarge	vclient	32	58	262	0.00	5400	1.0	0.0	10000
T2	10002	hls	ec2-fleet:6xr7i.8xlarge	vclient	1454	4642	6799	0.00	5129	1.0	0.0	9996
T2	1000	hls	ec2:r7i.8xlarge	vclient	336	1710	2423	0.00	5400	1.0	0.0	1000
T2	500	hls	ec2:r7i.8xlarge	vclient	67	1087	1816	0.00	5400	1.0	0.0	500
T2	100	hls	ec2:r7i.8xlarge	vclient	37	254	553	0.00	5400	1.0	0.0	100
T2	50	hls	ec2:r7i.8xlarge	vclient	39	244	255	0.00	5400	1.0	0.0	50
T2	10	hls	ec2:r7i.8xlarge	vclient	37	61	61	0.00	5400	1.0	0.0	10
T1	1000	hls	local-i7-9750H	vclient	10213	26173	26200	0.09	3056	0.1	0.0	322
T1	500	hls	local-i7-9750H	vclient	11171	24078	24134	0.02	3109	0.2	0.0	378
T1	100	hls	local-i7-9750H	vclient	3087	3646	3776	0.00	3640	1.7	0.0	100
T1	50	hls	local-i7-9750H	vclient	1816	2172	2223	0.00	5504	1.0	0.0	50
T1	10	hls	local-i7-9750H	vclient	417	443	446	0.00	5526	1.0	0.0	10
T2	1000	hls	ec2:r7i.8xlarge	vclient	2001	3887	5193	0.00	5308	1.0	0.0	1000
T2	500	hls	ec2:r7i.8xlarge	vclient	1053	2116	2794	0.00	5399	1.0	0.0	500
T2	100	hls	ec2:r7i.8xlarge	vclient	237	455	582	0.00	5400	1.0	0.0	100
T2	50	hls	ec2:r7i.8xlarge	vclient	145	312	339	0.00	5400	1.0	0.0	50
T2	10	hls	ec2:r7i.8xlarge	vclient	279	284	286	0.00	5400	1.0	0.0	10
T1	1000	hls	local-i7-9750H	vclient	46172	47151	47183	0.00	3000	0.0	0.0	1000
T1	500	hls	local-i7-9750H	vclient	11806	26926	26992	0.00	3000	0.0	0.0	412
T1	100	hls	local-i7-9750H	vclient	2899	3112	3160	0.00	4266	2.2	0.0	100

Fonte: Elaborado pelo autor.

B.2 Cliente *Web* de Consumo (`streaming-web-client`)

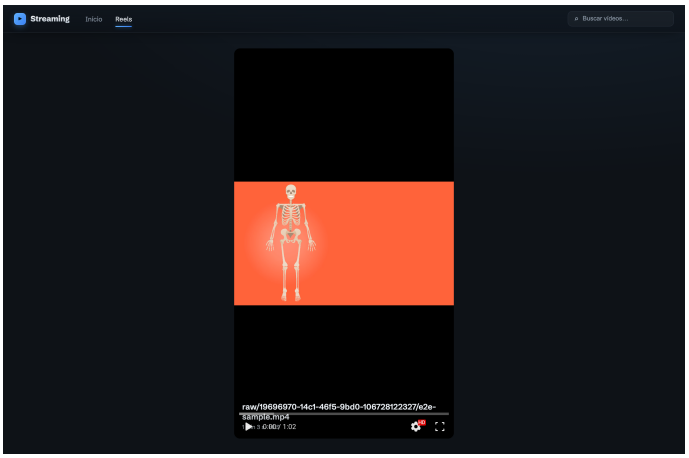
O `streaming-web-client` é a aplicação de consumo (*Progressive Web App* em Svelte), que reproduz o conteúdo distribuído pelo serviço de distribuição via Shaka Player. A tela inicial apresenta o catálogo, com uma faixa de *reels* e um vídeo em destaque (Figura 24); a seção de *reels* oferece a reprodução vertical contínua, com o *player* adaptativo e seus controles (Figura 25).

Figura 24 – Tela inicial (catálogo) do cliente *web*



Fonte: Elaborado pelo autor.

Figura 25 – Reprodução de *reels* no cliente *web* (Shaka Player)



Fonte: Elaborado pelo autor.

APÊNDICE C – DISPONIBILIDADE DO CÓDIGO-FONTE

Em coerência com o requisito RNF01 (Código Aberto) e com o compromisso de reprodutibilidade que orienta a metodologia experimental (Seção 5.2), todo o código-fonte da plataforma, de autoria do próprio autor, está publicamente disponível no GitHub. Cada microsserviço da Tabela 8 corresponde a um repositório independente, relacionado na Tabela 29.

Tabela 29 – Repositórios públicos da plataforma no GitHub

Módulo	**Repositório**
Ingestão	Disponível em: <https://github.com/Michelalmeidasilva/streaming-ingest>
Transcodificação e Empacotamento	Disponível em: <https://github.com/Michelalmeidasilva/streaming-transcode>
Distribuição	Disponível em: <https://github.com/Michelalmeidasilva/streaming-distribution>
Reprodução	Disponível em: <https://github.com/Michelalmeidasilva/streaming-web-client>
Plataforma de *Upload*	Disponível em: <https://github.com/Michelalmeidasilva/streaming-platform-upload>
Infraestrutura	Disponível em: <https://github.com/Michelalmeidasilva/streaming-infra>

Fonte: Elaborado pelo autor.

Os repositórios reúnem, além do código dos serviços, os arquivos de *deployment* (Docker Compose e infraestrutura como código) e os *scripts* de *benchmark* que reproduzem os experimentos do Capítulo 7.