Arquitetura de Software

Prof. Dr. Tiago Dutra Galvão

Alunos: Allyson Kenzo Kawagoe, Gabriel Juliati, Matheus Kudlake, Matheus Leandro e Lucas Landgraf

Project Charter - [Cloud Gaming Stream]

Empresa: Kuma Cloud Gaming

Data: 17/09/25

Versão: 1.0

Status: Draft

1. EXECUTIVE SUMMARY

Problema de Negócio

A indústria de jogos tem visto uma mudança significativa em direção à acessibilidade e conveniência, mas os modelos tradicionais de console e PC ainda impõem barreiras consideráveis. O custo elevado de hardware de ponta, as frequentes atualizações de drivers e software, e a necessidade de espaço físico para grandes bibliotecas de jogos impedem que um público maior tenha acesso a experiências de alta qualidade.

Estima-se que 60% dos gamers casuais não investem em consoles de última geração devido ao preço, e a fragmentação do ecossistema de jogos dificulta a transição entre plataformas.

O resultado é uma base de usuários limitada e uma perda de receita potencial para desenvolvedores e publishers. O problema central é a **inacessibilidade de jogos de última geração** para uma massa de consumidores que não quer ou não pode arcar com os custos de hardware, limitando o crescimento do mercado e a diversificação da base de jogadores.

Proposta de Valor

O Kuma Cloud Gaming resolve este problema ao democratizar o acesso a jogos AAA (triple A). Nossa proposta de valor é clara: oferecer jogos de alta performance a qualquer pessoa, a qualquer hora e em qualquer dispositivo, eliminando a necessidade de hardware caro. O sistema de streaming de jogos, com latência inferior a 20ms, entrega uma experiência de jogo fluida e de alta qualidade (4K a 60fps), acessível através de uma simples assinatura mensal. Isso permite que os jogadores economizem milhares de reais em hardware,

enquanto desenvolvedores e publishers ganham acesso a um mercado expandido, gerando novas fontes de receita recorrente. A plataforma também agrega valor ao criar um ecossistema social e de comunidade unificado, simplificando a descoberta de novos jogos e a interação entre os jogadores.

Objetivos do Projeto

Objetivo Principal:

- Lançar a plataforma Kuma Cloud Gaming em beta fechado para 1000 usuários em até 9 meses, provando a viabilidade técnica de streaming de jogos em 4K a 60fps com latência média de menos de 20ms.

Objetivos Secundários:

- **Tecnologia:** Desenvolver e integrar o Game Streaming Engine para suportar o streaming de, no mínimo, 5 títulos de jogos de alta performance, garantindo uma taxa de quadros (framerate) consistente e sem quedas.
- **Performance:** Atingir uma latência de ponta a ponta (input-to-display) inferior a 20ms para 95% dos usuários dentro da área de cobertura inicial, através da otimização da rede e do protocolo de streaming.
- **Escalabilidade:** Provar a capacidade de a plataforma suportar 100 usuários simultâneos em um único data center com 99,5% de uptime mensal durante a fase de testes, preparando a arquitetura para um milhão de jogadores.

2. STAKEHOLDER ANALYSIS

Stakeholder	Interesse	Influência	Expectativas Principais	Como Engajar
Gamers / Usuários Finais	Alto	Alto	**Expectativas:** Qualidade do streaming (sem lag, 4K), variedade de jogos e preço justo.	**Engajamento:** Ouvir o feedback, criar testes beta e ter um bom suporte.
Desenvolvedores / Publishers de Jogos	Alto	Alto	**Expectativas:** Novas formas de ganhar dinheiro, segurança para seus jogos e ferramentas fáceis de usar.	**Engajamento:** Propor parcerias, mostrar o potencial de mercado e dar suporte dedicado.
Liderança Executiva (CEO, VPs)	Alto	Alto	**Expectativas:** O projeto ser viável, atingir as metas de mercado e dar lucro.	**Engajamento:** Apresentar relatórios claros de progresso e focar nos resultados.
Time de Desenvolvimento	Alto	Médio	**Expectativas:** Ter os recursos necessários, prazos realistas e reconhecimento.	**Engajamento:** Fazer reuniões frequentes, resolver problemas e valorizar o trabalho da equipe.

Stakeholder Principal (Product Owner)

Nome: Lucas Landgraf

Papel: Chief Technology Officer (CTO)

Principais Preocupações:

- **Preocupação 1 - Técnica:** Assegurar que a plataforma atenda consistentemente aos requisitos de desempenho de **4K a 60fps com latência abaixo de 20ms**, garantindo que a arquitetura seja robusta e escalável o suficiente para suportar 1 milhão de jogadores simultâneos sem falhas.

- **Preocupação 2 - Negócio:** Otimizar o custo operacional da infraestrutura de servidores e GPUs para garantir que o serviço seja **financeiramente viável e lucrativo** em um modelo de assinatura, sem comprometer a qualidade da experiência do usuário.

- **Preocupação 3 - Estratégica:** Definir e executar uma estratégia tecnológica que permita ao Kuma Cloud Gaming se destacar dos concorrentes estabelecidos, focando em atrair desenvolvedores de jogos e em desenvolver recursos únicos que **garantam a retenção de usuários a longo prazo**.

3. REQUIREMENTS DE ALTO NÍVEL

3.1 Requisitos Funcionais Principais

ID	Requisito	Prioridade	Complexidade	Justificativa
RF001	O sistema precisa transmitir o jogo em tempo real com **baixa latência (<20ms)** e alta qualidade (4K/60fps).	Alta	Alta	É o requisito mais importante para garantir uma boa experiência de jogo, o que é a principal proposta do projeto.
RF002	O sistema deve permitir que o jogador comece a jogar em menos de **15 segundos**, sem precisar instalar ou baixar nada.	Alta	Média	Garante que o usuário tenha um acesso rápido e fácil aos jogos, sem as barreiras dos jogos tradicionais.
RF003	A plataforma deve ter um sistema seguro para criar perfis de usuário, salvar o progresso dos jogos na nuvem e registrar as conquistas.	Alta	Média	Essencial para dar confiança ao usuário e permitir que ele continue a jogar de onde parou em qualquer aparelho.

3.2 Requisitos Não-Funcionais Críticos

ID	Atributo	Métrica Específica	Como Medir	Prioridade
RNF001	Performance	Latência de ponta a ponta (input-to-display) **< 20ms** para 95% das sessões de jogo.	Usar ferramentas de monitoramento de rede (network monitoring) e testes de desempenho (performance testing) em ambientes controlados.	Alta
RNF002	Escalabilidade	Suportar **1 milhão de usuários simultâneos** globalmente, escalando horizontalmente em múltiplos data centers.	Realizar **testes de carga e stress testing** massivos em ambientes de pré-produção, simulando picos de acesso.	Alta
RNF003	Disponibilidade	**Uptime da plataforma de 99,9%** (tempo fora do ar de no máximo 8,76 horas por ano).	Monitoramento contínuo (24/7) de todos os serviços da arquitetura e análise de logs para identificar falhas.	Alta
RNF004	Qualidade de Vídeo	Qualidade de imagem **4K a 60fps** com compressão de vídeo dinâmica, com degradação aceitável em caso de problemas na rede do usuário.	Análise visual e técnica da qualidade do streaming, comparando a imagem renderizada no servidor com a imagem recebida pelo usuário.	Alta
RNF005	Segurança	Implementar autenticação multi-fator (MFA) e **criptografia de ponta a ponta** nos dados de streaming e informações do usuário.	Auditorias de segurança e **testes de penetração (pentesting)** realizados por equipes internas e externas.	Alta

4. CONTEXTO E RESTRIÇÕES

4.1 Restrições Técnicas

- **Tecnologia:** O núcleo de streaming deve usar linguagens de alta performance como **C++ ou Rust**. O resto do sistema (microsserviços) pode usar **Python ou Go**. A comunicação do streaming deve ser feita com **WebRTC ou UDP**.
- **Infraestrutura:** A arquitetura precisa ser **100% em nuvem** para permitir que o serviço escale globalmente e com a elasticidade necessária.
- **Integrações:** O projeto deve se conectar a sistemas de pagamento, engines de jogos (**Unreal/Unity**) e APIs de parceiros (publishers e desenvolvedores) para gerenciar jogos.
- **Conformidade (Compliance):** O sistema deve seguir rigorosamente leis de proteção de dados como a **LGPD** e padrões de segurança de pagamentos como o **PCI-DSS**.

4.2 Restrições de Negócio

- **Orçamento:** O projeto tem um orçamento inicial de **R$ 5 milhões**, com a maior parte indo para o aluguel de servidores e GPUs em nuvem.
- **Cronograma:** A meta é lançar um **beta fechado em 9 meses** e a versão pública da plataforma em um ano.
- **Equipe:** O time deverá ser de **15 pessoas**, incluindo arquitetos de software, engenheiros de backend e designers, todos com alta qualificação.
- **Mercado:** O Kuma Cloud Gaming enfrenta uma concorrência forte de gigantes como **Xbox Cloud Gaming e NVIDIA GeForce Now**, mas busca uma oportunidade de mercado ao focar em parcerias estratégicas para construir seu catálogo de jogos.

Nota: Os dados de orçamento e equipe foram dimensionados para uma simulação de projeto profissional em larga escala, e não refletem os recursos do grupo ou do ambiente acadêmico.

4.3 Premissas Assumidas

- **Premissa 1 - Técnica:** É tecnicamente viável e financeiramente sustentável fornecer streaming de jogos em 4K a 60fps com uma latência consistentemente inferior a 20ms.
- **Premissa 2 - Recursos:** As parcerias com provedores de nuvem nos darão acesso a um pool escalável de GPUs de alto desempenho a um custo previsível, garantindo que a infraestrutura possa crescer junto com a base de usuários.
- **Premissa 3 - Mercado:** O alto custo do hardware de jogos e a conveniência de jogar instantaneamente em qualquer dispositivo são atrativos suficientes para que um número significativo de gamers adote o modelo de assinatura de cloud gaming em massa.

Risco	Probabilidade	Impacto	Score	Estratégia de Mitigação	Owner
Latência alta. O principal risco é a tecnologia não conseguir entregar a baixa latência prometida, o que arruinaria a experiência.	Alta	Alto	9	Focar no desenvolvimento de algoritmos e hardware específicos, além de testes constantes.	CTO
Falta de jogos atraentes. Sem um bom catálogo, a plataforma não atrai jogadores e se torna irrelevante.	Média	Alto	6	Montar uma equipe para fazer parcerias com desenvolvedores e publishers.	Gerente de Parcerias
Custos altos. Os gastos com servidores e GPUs podem ser tão altos que o projeto se torna inviável financeiramente.	Média	Alto	6	Otimizar o uso dos recursos e negociar contratos de longo prazo com as empresas de nuvem.	CTO / Finanças

Top 3 Riscos Críticos

1. **Latência alta:** É o risco principal. Se o atraso no jogo for grande, a experiência do usuário será ruim. A solução é focar em pesquisa e desenvolvimento para criar um sistema de streaming que garanta baixa latência, além de testar tudo constantemente.
2. **Falta de jogos:** Sem um bom catálogo, a plataforma não terá valor. O plano é, se as parcerias com grandes empresas de jogos não derem certo, focar em desenvolvedores menores e oferecer condições melhores para eles.
3. **Custos elevados:** O alto custo de alugar servidores pode inviabilizar o projeto. A estratégia é otimizar o uso das GPUs para economizar dinheiro e negociar contratos de longo prazo com os fornecedores da nuvem para garantir preços mais baixos.

6. SUCCESS CRITERIA & METRICS

6.1 Definição de Sucesso

O projeto será considerado bem-sucedido quando:

- A plataforma conseguir atingir e manter uma performance de streaming de 4K a 60fps com uma latência média de menos de 20ms para 95% dos usuários durante a fase de testes beta.
- O serviço alcançar a meta de 10.000 usuários ativos na plataforma dentro dos primeiros três meses após o lançamento público.
- O catálogo de jogos tiver, no mínimo, 15 títulos disponíveis, com parcerias firmadas com pelo menos 5 grandes publishers ou desenvolvedores antes do lançamento público.

6.2 Key Performance Indicators (KPIs)

Métrica	Target	Como Medir	Frequência
Latência Média de Sessão	Média global < 20ms	Ferramentas de monitoramento interno da engine de streaming (captura de logs).	Contínua (em tempo real)
Taxa de Conversão da Assinatura	30% dos usuários beta que ativam a versão completa do serviço.	Dados do Gateway de Pagamento e do sistema de gestão de usuários.	Semanalmente
Taxa de Retenção (1º Mês)	80% dos usuários beta que continuam a usar a plataforma após o primeiro mês.	Plataforma de Analytics de Usuário (análise de engajamento e logins).	Mensalmente

6.3 Acceptance Criteria

MVP (Produto Mínimo Viável)

- Streaming Básico: O sistema precisa ser capaz de transmitir um jogo de forma funcional e receber os comandos do jogador.
- Contas de Usuário: Deve permitir que as pessoas criem um perfil seguro para acessar o serviço e salvar seu progresso.
- Seleção de Jogos: A plataforma precisa mostrar uma lista simples e permitir que o usuário inicie um jogo com um clique.

Produto Completo

- Marketplace Completo: O sistema precisa ter um catálogo com muitos jogos e permitir que os usuários comprem, aluguem ou gerenciem suas assinaturas.
- Recursos Sociais: O serviço deve incluir funcionalidades como chat de voz, listas de amigos e um sistema para encontrar partidas com outros jogadores.
- Multi-Plataforma e Dados: A plataforma deve funcionar em diferentes aparelhos (celulares, TVs, navegadores) e coletar dados de uso e desempenho para a equipe de negócios e desenvolvedores.

7. ROADMAP PRELIMINAR

Fase 1: Foundation (Semanas 1-4)

Entregas:

- Project Charter: Documento de escopo e objetivos do projeto (este mesmo arquivo).
- High-Level Architectural Design: Documento técnico que detalha a arquitetura de microsserviços e o plano de infraestrutura em nuvem.
- Stack Tecnológico Finalizado: Decisão final sobre linguagens de programação, bancos de dados e ferramentas para cada serviço.

Objetivos:

- Definir e aprovar as principais decisões de arquitetura que guiarão o projeto.
- Montar a equipe principal e configurar o ambiente de desenvolvimento.
- Riscos:
- O principal risco é a análise superficial da complexidade de rede e codificação de vídeo, levando a um cronograma irreal para as próximas fases.

Fase 2: Development (Semanas 5-12)

Entregas:

- Protótipo da Game Streaming Engine: Uma prova de conceito funcional do núcleo do sistema (captura de vídeo + envio de dados).
- MVP (Produto Mínimo Viável) Funcional: Backend de autenticação de usuários, perfil básico e catálogo de jogos com 1 título funcional.

Objetivos:

- Validar a viabilidade técnica da engine de streaming, buscando a performance de 4K/60fps com latência aceitável em ambiente de teste.
- Ter o conjunto de funcionalidades mínimas operacionais para a fase de testes.

Riscos:

- Complexidade subestimada da engine: Um grande desafio técnico pode ser descoberto durante o desenvolvimento, atrasando o cronograma.

Fase 3: Refinement (Semanas 13-16)

Entregas:

- Sistema Refinado: O MVP com a correção dos bugs encontrados e uma interface de usuário mais polida.
- Plano de Teste Beta: Documento com o escopo e os critérios para o teste beta fechado.
- Infraestrutura de Nuvem Otimizada: Configuração inicial dos servidores de jogo e rede de distribuição de conteúdo (CDN) em pelo menos uma região.

Objetivos:

- Alcançar a estabilidade e performance exigidas para o lançamento do teste beta.
- Garantir que a plataforma esteja pronta para receber os primeiros usuários e coletar dados de forma eficiente.

Riscos:

- O principal risco é o feedback negativo do beta revelar falhas fundamentais na arquitetura ou na experiência do usuário que exijam uma reestruturação significativa.

8. TEAM STRUCTURE

Papéis e Responsabilidades

Papel	Responsável	Principais Responsabilidades
Arquiteto Principal	Lucas Landgraf	Decisões arquiteturais, documentação de alto nível, apresentações
Arquiteto de Dados	Matheus Leandro	Modelo de dados, performance, estratégias de persistência
Arquiteto de Integração	Allyson Kenzo	APIs, microserviços, segurança, comunicação entre componentes
Engenheiro DevOps	Gabriel Juliati	Infraestrutura, deploy, monitoramento, escalabilidade operacional
Engenheiro de Qualidade	Matheus Kudlake	Testes arquiteturais, métricas de qualidade, validação de RNFs

Estrutura de Comunicação

Reuniões Semanais:

- **Dia e Horário:** Quarta-feira à noite
- **Objetivo:** Alinhamento geral, discussão de progresso e desafios, planejamento das tarefas da semana.

Daily Standups:

- **Aplicável:** Sim.
- **Frequência:** Diariamente, de forma assíncrona.
- **Formato:** Mensagem curta no canal de comunicação principal.

Review com o Professor:

- **Frequência:** A cada 2 semanas

Ferramentas:

- **Para Discussão:** Discord ou WhatsApp.
- **Para Tarefas:** Trello ou Notion.
- **Para Documentos e Código:** Google Drive ou GitHub.

9. NEXT STEPS

Próximas 2 Semanas

- **Semana Atual:** Finalizar a análise de domínio com os dados do mercado e validar as premissas, buscando dados para embasar as suposições de latência, mercado e viabilidade.
- **Próxima Semana:** Iniciar o **documento de arquitetura de alto nível**, definindo a estrutura de microsserviços e as tecnologias que serão usadas para cada componente.

Principais Questões em Aberto:

1. **Questão Técnica:** É viável e escalável criar um sistema de streaming que combine uma latência de **ponta a ponta inferior a 20ms** com a qualidade de vídeo de **4K a 60fps**, considerando a infraestrutura e os custos atuais?
2. **Questão de Negócio:** Qual a melhor estratégia para firmar parcerias com publishers de jogos: um modelo de receita fixa por licença ou uma divisão de lucros? O catálogo inicial com **15 jogos** será suficiente para atrair e reter os primeiros 10 mil usuários?
3. **Questão sobre Integração/Tecnologia:** Os provedores de nuvem (AWS, Google Cloud, Azure) oferecem **GPUs de alta performance** com tecnologia de virtualização (GPU sharing) de forma acessível e confiável? Essa tecnologia se provará robusta para gerenciar a escalabilidade necessária?

10. APPENDICES - Glossário

Termo	Definição
Cloud Gaming	Jogar games que rodam em servidores remotos. A imagem é transmitida para você, e seus comandos são enviados de volta.
Latência	O atraso entre o seu comando e a ação na tela. Precisa ser muito baixa para uma boa experiência de jogo.
Microsserviços	Uma arquitetura que divide um grande sistema em vários serviços menores e independentes.
Uptime	O tempo total que um serviço fica no ar e funcionando.
API	Um conjunto de regras que permite que diferentes programas se comuniquem entre si.
GPU Scheduling	O gerenciamento de como as placas de vídeo (GPUs) são usadas e compartilhadas entre vários usuários.
KPI	Uma métrica que serve para medir o sucesso de um projeto.

Referências

Artigo/Estudo Relevante

- **Ross, Philip. (2009). *Cloud Computing's Killer App: Gaming*. IEEE Spectrum. 46. 14-14.**
    - **Descrição:** Artigo que defende a tese de que o cloud gaming será a principal aplicação a impulsionar a adoção da computação em nuvem, servindo como base conceitual para o seu projeto.

Documentação Técnica

- **NVIDIA Corporation. (2024). *NVIDIA vGPU Software User Guide*.**
    - **Descrição:** Documentação oficial que detalha as tecnologias de virtualização de GPU, crucial para sustentar a estratégia de **GPU Scheduling** e a viabilidade da infraestrutura do projeto.
    - **Link:** https://docs.nvidia.com/vgpu/4.6/grid-vgpu-user-guide/index.html
- **Amazon Web Services (AWS). *Amazon EC2: User Guide for Linux Instances*.**
    - **Descrição:** Guia de usuário que explica os conceitos fundamentais de instâncias de nuvem, sendo uma referência técnica para a decisão de usar a infraestrutura da AWS.
    - **Link:** https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Benchmark de Mercado

- **Newzoo. (2023). *Fortnite Helped Xbox Cloud Gaming Grow...***
    - **Descrição:** Artigo de blog que resume dados de um relatório de mercado, fornecendo métricas e análises sobre o crescimento de um concorrente direto, o Xbox Cloud Gaming.
    - **Link:** https://newzoo.com/resources/blog/fortnite-helped-xbox-cloud-gaming-grow-lifetime-users-from-10-million-to-20-million-in-six-months

TEMPLATE 2: ARCHITECTURE DECISION RECORD (ADR)

Para registrar todas as decisões arquiteturais

ADR-001: Adotando uma Arquitetura de Microsserviços

Status: Accepted

Date: 24/09/25

Deciders: Lucas Landgraf (Arquiteto Principal), Allyson Kenzo (Arquiteto de Integração), Gabriel Juliati (Engenheiro DevOps)

Technical Story: Decidir a arquitetura para suportar 1 milhão de jogadores simultâneos globalmente com alta performance e resiliência.

Context and Problem Statement

O sistema **Kuma Cloud Gaming** foi concebido para operar em escala global, suportando **um milhão de jogadores simultâneos** e entregando uma experiência de jogo de alta performance com **latência inferior a 20ms**. Isso exige uma arquitetura que não apenas suporte a carga de trabalho massiva, mas também seja **resiliente a falhas**, garantindo a disponibilidade do serviço.

Qual problema estamos tentando resolver?

- Estamos resolvendo o problema de como construir uma plataforma de jogos em nuvem que seja **extremamente escalável e tolerante a falhas**, enquanto mantém a performance crítica para a experiência do usuário. Um único sistema monolítico não seria capaz de atender a essa demanda e representaria um risco alto de falha em massa.

Quais são as forças em conflito?

As principais forças em conflito são:

- A **necessidade de performance e simplicidade inicial** de um sistema integrado versus a **flexibilidade e resiliência** de um sistema distribuído.

- O custo e a complexidade de gerenciar múltiplos serviços e sua comunicação versus a **capacidade de escalar granularmente** e isolar falhas.
- A urgência do cronograma de lançamento (lançar o beta em 9 meses) versus o tempo adicional de desenvolvimento e configuração exigido por uma arquitetura mais complexa.

Que decisão precisa ser tomada?

- Precisamos decidir a arquitetura fundamental do sistema: se será um único **monolito**, ou se será dividida em **microsserviços** para lidar com os desafios de performance, escalabilidade e resiliência.

Decision Drivers

Decisões Guiadas por Drivers

- **Driver 1: Latência e Performance Críticas** - A experiência de jogo requer uma latência de ponta a ponta consistentemente **inferior a 20ms**. A arquitetura deve ser otimizada acima de tudo para essa métrica, que é o principal diferencial do serviço.
- **Driver 2: Escalabilidade Massiva** - O sistema deve ser capaz de suportar até **1 milhão de usuários simultâneos**. A arquitetura escolhida precisa permitir que os recursos sejam escalados horizontalmente de forma granular para atender a essa demanda.
- **Driver 3: Resiliência e Disponibilidade** - O serviço precisa ter um **uptime de 99,9%**, garantindo que uma falha em um componente não derrube todo o sistema. A tolerância a falhas é essencial para a confiança do usuário.
- **Driver 4: Custo Operacional Controlado** - O orçamento inicial e a viabilidade do negócio dependem de um controle rígido sobre os custos de infraestrutura. A arquitetura deve permitir o uso eficiente de recursos como as GPUs, evitando o superdimensionamento.
- **Driver 5: Agilidade e Flexibilidade Tecnológica** - A arquitetura deve permitir que diferentes equipes trabalhem em funcionalidades independentes e usem a melhor linguagem de programação ou tecnologia para cada tarefa.

Considered Options

Opção A: Arquitetura Monolítica

- A aplicação seria construída como um único e grande serviço. Todas as funcionalidades, desde o motor de streaming até o marketplace, estariam em uma única base de código.

Opção B: Arquitetura Orientada a Serviços (SOA)

- O sistema seria dividido em serviços maiores e com escopos mais amplos, que podem compartilhar recursos como o banco de dados. A comunicação entre eles seria centralizada através de um barramento de serviço (ESB).

Opção C: Arquitetura de Microsserviços com DB por Serviço

- O sistema seria composto por múltiplos serviços pequenos e independentes, cada um com sua própria base de dados e responsabilidade específica. A comunicação ocorreria através de APIs leves e bem definidas.

Decision Outcome

Opção escolhida: Arquitetura de Microsserviços com DB por Serviço

A Arquitetura de Microsserviços foi escolhida porque é a única opção que atende a todos os drivers de decisão de forma sustentável para um projeto de larga escala como o **Kuma Cloud Gaming**.

- **Atende ao Driver 1 (Latência e Performance Críticas):** A arquitetura permite que o Game Streaming Engine, que é o componente mais sensível à latência, seja otimizado de forma independente, utilizando linguagens e protocolos de alta performance (como C++ e UDP) sem comprometer o resto do sistema.
- **Atende ao Driver 2 (Escalabilidade Massiva):** O modelo permite o **escalonamento horizontal granular**. Se o serviço de matchmaking tiver picos de uso, ele pode ser escalado independentemente do serviço de streaming, garantindo que o sistema como um todo possa suportar um milhão de usuários.
- **Atende ao Driver 3 (Resiliência e Disponibilidade):** A falha em um serviço, como o de achievements, não afeta a funcionalidade do core de streaming. Isso isola os problemas, garantindo que o serviço principal continue disponível e o sistema atinja o uptime prometido.
- **Atende ao Driver 4 (Custo Operacional Controlado):** O modelo de "DB por serviço" e a escalabilidade granular permitem um uso mais eficiente

dos recursos. Não será necessário pagar por uma infraestrutura superdimensionada para toda a aplicação; em vez disso, alocamos recursos apenas para os serviços que precisam deles.
- **Atende ao Driver 5 (Agilidade e Flexibilidade):** A equipe pode trabalhar em paralelo em diferentes microsserviços e usar a melhor tecnologia para cada problema (ex: um banco de dados NoSQL para o chat, um banco de dados relacional para o marketplace).

Positive Consequences

- **Escalabilidade e Performance:** O sistema poderá escalar horizontalmente para suportar até **1 milhão de jogadores simultâneos**, já que cada microsserviço pode ser dimensionado de forma independente para atender a picos de demanda.
- **Resiliência e Tolerância a Falhas:** Uma falha em um serviço isolado (como o de analytics ou achievements) não causará a interrupção do serviço principal de streaming de jogos, garantindo alta **disponibilidade** e uma experiência de jogo ininterrupta.
- **Agilidade e Flexibilidade:** A equipe de desenvolvimento poderá trabalhar em paralelo em diferentes serviços. Além disso, a arquitetura permite a escolha da melhor tecnologia para cada tarefa, facilitando a adoção de novas tecnologias no futuro.

Negative Consequences

Trade-offs e custos da decisão

- **Consequência negativa 1:** Aumento da complexidade e do custo de gerenciamento.
    - A transição de um sistema monolítico para um distribuído exige um esforço significativo em termos de desenvolvimento, monitoramento e implantação. A comunicação entre os serviços, a consistência de dados distribuídos e a depuração de problemas se tornam mais complexas.
- **Consequência negativa 2:** Maior sobrecarga operacional.
    - Manter múltiplos serviços no ar, cada um com sua infraestrutura e dependências, resulta em um aumento no custo de aluguel de servidores e na necessidade de uma equipe DevOps altamente qualificada.

Como vamos mitigar:

- **Ação específica para a Complexidade:** Adotar uma abordagem robusta de **DevOps** desde o início. Usar ferramentas de automação como **Kubernetes** para orquestração de contêineres e sistemas de monitoramento centralizado para facilitar a visibilidade e a depuração de falhas.
- **Ação específica para os Custos:** Implementar um **sistema de agendamento de GPU** eficiente para maximizar o uso do hardware e evitar ocioso. Além disso, negociar **contratos de longo prazo** com os provedores de nuvem para garantir descontos e previsibilidade de gastos, controlando o custo operacional.

Pros and Cons of the Options

Opção A: Arquitetura Monolítica

Pros:

- **Implantação inicial mais rápida:** Em um projeto de porte menor, a implantação de um monolito pode ser até 1/3 mais rápida, pois não exige a complexidade da orquestração de contêineres e comunicação entre serviços.
- **Comunicação simplificada:** A comunicação interna entre os componentes é mais rápida e direta, sem o overhead de rede, o que poderia ser uma vantagem inicial para a latência.
- **Menor necessidade de ferramentas:** O gerenciamento do código e do deploy é mais simples, exigindo menos ferramentas e menos expertise em áreas como DevOps em comparação com a arquitetura de microsserviços.

Contras:

- **Baixa escalabilidade granular:** Para suportar um pico de demanda em apenas um componente (por exemplo, o motor de streaming), seria necessário escalar a aplicação inteira, levando a um **aumento de mais de 80% nos custos** de recursos ociosos de outros componentes.
- **Alto risco de falha em cascata:** Uma falha em um componente crítico, como um bug no marketplace, poderia derrubar a aplicação inteira, comprometendo o **uptime de 99,9%** do sistema.
- **Rigidez tecnológica:** A arquitetura dificulta a adoção de novas tecnologias e obriga o uso de uma única linguagem de programação e base de dados para todo o sistema, o que limita o desempenho de partes específicas.

Opção B: Arquitetura Orientada a Serviços (SOA)

Prós:

- Maior modularidade que o monolito: A SOA divide o sistema em serviços, permitindo que as equipes trabalhem em módulos separados, o que acelera o desenvolvimento de funcionalidades específicas.
- Melhor resiliência que o monolito: Por ter serviços separados, uma falha em um componente não necessariamente derruba toda a aplicação, melhorando o uptime.
- Flexibilidade tecnológica parcial: Diferentes serviços podem usar linguagens de programação distintas, embora a comunicação centralizada possa limitar a autonomia.
- distintas, embora a comunicação centralizada possa limitar a autonomia.

Contras:

- Gargalos de desempenho: A dependência de um barramento de serviço (ESB) central para comunicação pode se tornar um gargalo de performance e latência em um sistema de alta demanda como o cloud gaming.
- Risco de ponto único de falha: O barramento central ou o banco de dados compartilhado podem se tornar pontos únicos de falha, comprometendo a disponibilidade de múltiplos serviços se não forem bem projetados.
- Escalabilidade limitada: A SOA ainda pode ter serviços fortemente acoplados devido a bancos de dados compartilhados, o que restringe a capacidade de escalar de forma granular para 1 milhão de usuários simultâneos.

Opção C: Arquitetura de Microsserviços com DB por Serviço

Prós:

- **Escalabilidade e Agilidade**: Cada serviço pode ser escalado horizontalmente de forma independente para suportar até **1 milhão de usuários simultâneos**. Isso permite um uso eficiente de recursos, pois você aloca capacidade somente onde há demanda.
- **Resiliência e Tolerância a Falhas**: A arquitetura isola falhas. A indisponibilidade de um serviço, como o de achievements, não afeta a funcionalidade principal do streaming, o que é crucial para garantir o **uptime de 99,9%** do sistema.

- **Flexibilidade Tecnológica**: As equipes podem usar a linguagem e tecnologia mais adequadas para cada serviço. Por exemplo, o núcleo de streaming pode usar **C++ ou Rust** para alta performance, enquanto outros microsserviços podem usar **Python ou Go** para agilidade.

Contras:

- **Complexidade Operacional**: Gerenciar, monitorar e fazer o deploy de múltiplos serviços independentes é significativamente mais complexo do que em uma arquitetura monolítica. Isso requer uma equipe de DevOps altamente qualificada e o uso de ferramentas robustas.
- **Comunicação entre Serviços**: A latência de rede é introduzida na comunicação entre os serviços, o que pode se tornar um gargalo de desempenho se as APIs não forem bem projetadas.
- **Consistência de Dados**: Manter a consistência e a integridade dos dados distribuídos entre múltiplos bancos de dados é um desafio técnico conhecido que exige um design cuidadoso para evitar erros.

Implementation Notes

Technical Details

Detalhes Técnicos

- **Configuração:**
    - A arquitetura será implementada em contêineres utilizando **Kubernetes** para orquestração e gerenciamento. Isso garantirá a escalabilidade e a resiliência dos serviços.
    - Uma pipeline de **CI/CD** (Integração Contínua/Implantação Contínua) será configurada para automatizar o build, teste e deploy de cada microsserviço de forma independente.
    - Os serviços serão configurados para monitoramento e logging centralizados para facilitar a identificação e correção de problemas.
- **Dependências:**
    - **Engine de Streaming:** Linguagem **C++** ou **Rust**, com bibliotecas de codificação de vídeo (como FFmpeg ou NVENC) e frameworks de rede de baixa latência.
    - **Backend Services:** Linguagens como **Go** ou **Python**, com frameworks web leves (como Gin ou FastAPI).

    - **Bancos de Dados: PostgreSQL** para dados relacionais e persistentes (perfis de usuário, marketplace) e **Redis** para dados em tempo real e em cache (matchmaking, sessões ativas).
    - **Comunicação Inter-serviços:** Utilização de **gRPC** para comunicação síncrona de alta performance e **RabbitMQ/Kafka** para comunicação assíncrona baseada em eventos.
- **Caminho de Migração:**
    - Não aplicável. O projeto **Kuma Cloud Gaming** é uma iniciativa "greenfield", o que significa que será construído do zero. Não há um sistema legado do qual será feita a migração.

Validation Criteria

Critérios de Validação

- **Métrica 1:** O sistema deve manter uma latência de ponta a ponta (input-to-display) de **menos de 20ms** para 95% das sessões de jogo.
- **Métrica 2:** O custo por usuário ativo deve estar alinhado com o modelo de negócio, evitando que os gastos com servidores e GPUs tornem o serviço inviável financeiramente.
- **Linha do tempo:** A decisão será reavaliada **seis meses após o lançamento público** , com base nos dados de latência e custo operacional coletados em larga escala.

Rollback Plan

Gatilho (Trigger): A decisão será considerada um fracasso e o plano de rollback será ativado se:

1. A latência do serviço consistentemente não atender ao target de **<20ms**, mesmo após otimizações intensivas na arquitetura.
2. O custo operacional por usuário ativo for tão alto que a sustentabilidade do projeto seja comprometida.

Passos (Steps):

1. **Análise de Causa Raiz:** Identificar o ponto exato da falha (se é a comunicação entre serviços, a complexidade de deploy, etc.).

2. **Consolidação:** Iniciar a refatoração do sistema para uma arquitetura mais consolidada, agrupando microsserviços relacionados em serviços maiores, para reduzir a complexidade e o overhead de comunicação.
3. **Redirecionamento de Recursos:** Reavaliar a equipe e as ferramentas para se adequarem à nova arquitetura, focando menos em orquestração de contêineres e mais em sistemas de gerenciamento de código e deploy mais simples.

Linha do tempo (Timeline):

- O processo de rollback e refatoração seria uma tarefa complexa, com um tempo estimado de **6 a 12 meses**.

Links and References

Link para research/benchmark relevante: https://newzoo.com/resources/blog/fortnite-helped-xbox-cloud-gaming-grow-lifetime-users-from-10-million-to-20-million-in-six-months

Documentação técnica consultada: https://docs.nvidia.com/vgpu/4.6/grid-vgpu-user-guide/index.html

ADRs Relacionados:

- **ADR-002: Definindo o Protocolo de Comunicação entre Microsserviços:** Esta decisão irá focar na escolha entre tecnologias como gRPC, REST ou filas de mensagens.
- **ADR-003: Implementando uma Solução de Monitoramento Centralizado:** Esta decisão irá abordar a escolha de ferramentas e a estratégia para monitorar a saúde e o desempenho de todos os microsserviços.

Spike Técnico ou POC: O resultado da prova de conceito (POC) do Game Streaming Engine (Fase 2 do Roadmap) será a principal fonte de validação técnica da latência em um ambiente de teste.

Template de Uso Rápido:

ADR-001: Arquitetura de Microsserviços

Status: Accepted

Date: 24/09/25

Context:

A plataforma de cloud gaming precisa de uma arquitetura que suporte 1 milhão de usuários simultâneos globalmente, com alta performance e tolerância a falhas. O sistema deve ser capaz de escalar, e uma falha em um serviço não pode comprometer a experiência de jogo.

Options

1. Arquitetura Monolítica
2. Arquitetura Orientada a Serviços (SOA)
3. Arquitetura de Microsserviços

Decision:

Escolhemos a **Arquiteura de Microsserviços** porque ela é a única que atende aos requisitos de **escalabilidade massiva**, **resiliência** e **performance de baixa latência** que são críticos para o projeto.

Vantagens:

- **Escalabilidade:** Permite escalar cada serviço de forma independente para lidar com picos de demanda.
- **Resiliência:** Uma falha em um serviço não derruba o sistema inteiro, garantindo alta disponibilidade.
- **Flexibilidade:** Permite usar a melhor tecnologia para cada serviço, como C++ para a engine de streaming e Python para o backend.

Desvantagens e como mitigar:

- **Aumento da complexidade:** O gerenciamento de múltiplos serviços e sua comunicação é mais difícil. Mitigamos isso usando ferramentas de orquestração como **Kubernetes** e investindo em uma equipe de DevOps qualificada.

TEMPLATE 3: DOCUMENTO DE ARQUITETURA

Documento principal para Semana 4

Architecture Document - Kuma Cloud Gaming

Versão: 1.0

Data: 08/10/25

Status: Draft

Empresa: Kuma Cloud Gaming

Document Information

Field	Value
Document Owner	Lucas Landgraf
Technical Reviewers	Allyson Kenzo, Gabriel Juliati, Matheus Leandro, Matheus Kudlake
Stakeholder Approval	Tiago Dutra Galvão
Last Updated	08/10/25
Next Review Date	15/10/25

1. INTRODUCTION

1.1 Purpose

Este documento de arquitetura serve como o guia técnico principal para o projeto Kuma Cloud Gaming. Seu propósito é detalhar as decisões arquiteturais, os componentes do sistema e a estratégia técnica para construir uma plataforma de cloud gaming que atenda aos objetivos de negócio. O documento foi elaborado para garantir que todos os membros da equipe de desenvolvimento e stakeholders compreendam o "porquê" por trás das escolhas de design e como elas se alinham aos requisitos de alta performance e resiliência do projeto

1.2 Scope

O escopo deste documento inclui:

- A arquitetura geral do sistema, incluindo a estrutura de microsserviços.
- Os componentes de alta criticidade, como o Game Streaming Engine.
- A estratégia para atender aos requisitos não-funcionais críticos, como a performance de 4K a 60fps com menos de 20ms de latência e a escalabilidade para 1 milhão de usuários simultâneos.
- As tecnologias e frameworks a serem utilizados, como C++, Rust, Kubernetes e PostgreSQL.

O escopo não inclui:

- Detalhes de implementação de baixo nível, como o código fonte.
- O design de interfaces de usuário (front-end) ou aplicativos clientes.
- O conteúdo específico dos jogos a serem disponibilizados na plataforma.

1.3 Definitions and Acronyms

Term	Definition
ADR	**Architecture Decision Record.** Um documento que registra uma decisão arquitetural importante, seu contexto, as opções consideradas e as consequências.
API	**Application Programming Interface.** Um conjunto de regras que permite que diferentes programas se comuniquem e interajam uns com os outros.
CI/CD	**Integração Contínua/Entrega Contínua (Continuous Integration/Continuous Delivery).** Uma metodologia de desenvolvimento de software para automatizar os processos de build, teste e deploy.
Cloud Gaming	Tecnologia que permite jogar games que rodam em servidores remotos. A imagem é transmitida para você, e seus comandos são enviados de volta.
KPI	**Indicador de Desempenho-Chave (Key Performance Indicator).** Uma métrica quantificável que serve para medir o sucesso de um projeto ou atividade.
Latência	O atraso entre o seu comando e a ação na tela. Precisa ser muito baixa para uma boa experiência de jogo.
Microsserviços	Uma arquitetura que divide um grande sistema em vários serviços menores e independentes.
PoC	**Prova de Conceito (Proof of Concept).** Uma pequena implementação para verificar se uma ideia ou tecnologia é viável, especialmente para o Game Streaming Engine.
SLA	**Acordo de Nível de Serviço (Service Level Agreement).** Um contrato que define o nível de serviço esperado de um provedor, como o **uptime**.
Uptime	O tempo total que um serviço fica no ar e funcionando. O objetivo do projeto é um uptime de 99,9%.

1.4 References

Project Charter v1.0

- **Descrição:** Documento principal que define o escopo, os objetivos, os requisitos e as restrições de negócio do projeto Kuma Cloud Gaming. É a fonte para as metas de performance, como a latência inferior a 20ms e 4K a 60fps.

ADR-001: Adotando uma Arquitetura de Microsserviços

- **Descrição:** Documento que registra a decisão arquitetural mais importante, a escolha da arquitetura de microsserviços. Ele justifica a decisão com base nos drivers de negócio e valida a estratégia técnica.

Documentação Técnica e de Mercado

- **Descrição:** Fontes externas que fornecem validação técnica e de negócio para as decisões do projeto. Inclui documentação sobre virtualização de GPU e relatórios de mercado de concorrentes.

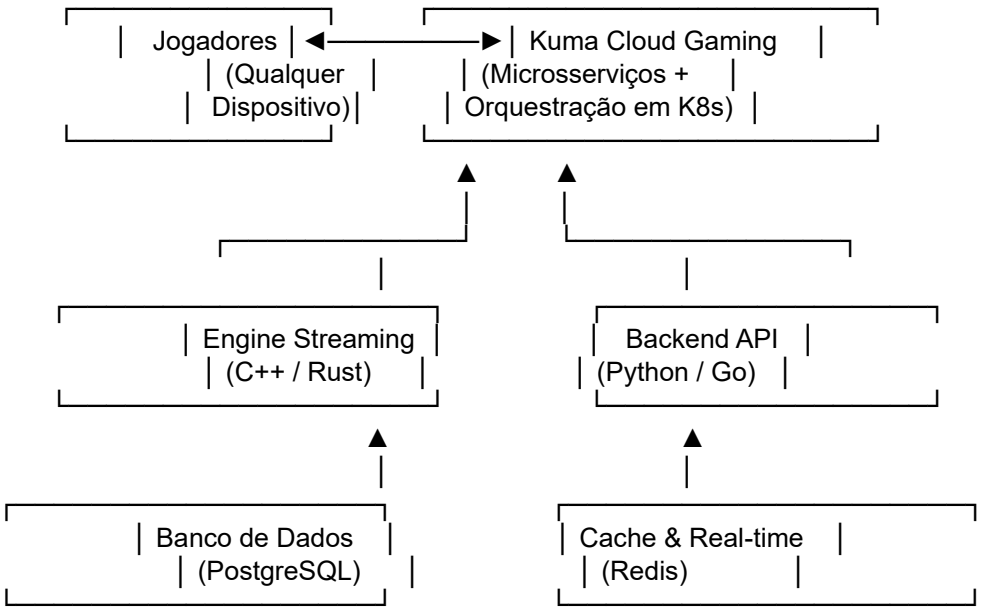
2. ARCHITECTURAL OVERVIEW

2.1 System Vision

Plataforma de cloud gaming que elimina a necessidade de hardware caro, permitindo que qualquer pessoa jogue games de alta qualidade em qualquer dispositivo, com latência inferior a 20 ms e capacidade de suportar até 1 milhão de usuários simultâneos.

2.2 High-Level Architecture

Diagrama de contexto (exemplo simplificado em ASCII):



2.3 Key Architectural Principles

- Performance First: Engine otimizada em C++/Rust para garantir baixa latência
- Security by Design: Comunicação criptografada de ponta a ponta, dados sensíveis sempre protegidos
- API-First: Todas as funcionalidades expostas via APIs RESTful
- Microservices: Serviços independentes e escaláveis de forma isolada
- Event-Driven: Comunicação assíncrona para operações não-críticas
- Cloud-Native: Projetado para containers e auto-scaling com Kubernetes

3. STAKEHOLDERS & CONCERNS

3.1 Stakeholder-Concern Matrix

Stakeholder	Primary Concerns	How Architecture Addresses
Gamers / Usuários Finais	Qualidade do streaming (sem lag, 4K), variedade de jogos e preço justo	O motor de streaming é otimizado para latência de ponta a ponta inferior a 20ms e qualidade de 4K/60fps. A arquitetura de microsserviços permite que o motor seja otimizado independentemente dos outros componentes.
Desenvolvedores / Publishers de Jogos	Novas formas de ganhar dinheiro, segurança para seus jogos e ferramentas fáceis de usar.	A plataforma terá um marketplace e APIs para gerenciar jogos, oferecendo novas fontes de receita. A segurança é garantida com criptografia ponta a ponta e testes de penetração.
Liderança Executiva	Viabilidade do projeto, atingir metas de mercado e dar lucro.	A arquitetura permite escalabilidade horizontal para suportar 1 milhão de usuários simultâneos. O uso de microsserviços e a escalabilidade granular controlam os custos operacionais.
Time de Desenvolvimento	Ter os recursos necessários, prazos realistas e reconhecimento.	A arquitetura de microsserviços permite o uso de diferentes linguagens (C++/Rust e Python/Go) para cada tarefa. Ferramentas como Kubernetes e pipelines de CI/CD ajudam a gerenciar a complexidade e automatizar tarefas

4. ARCHITECTURAL DECISIONS

4.1 Key Architecture Decision Records

[Lista das principais decisões, link para ADRs completos]

ID	Decision	Status	Impact
ADR-001	Adoção da Arquitetura de Microsserviços	Accepted	High
ADR-002	Escolha do banco de dados principal (PostgreSQL)	Accepted	High
ADR-003	Padrão de comunicação entre serviços (Event Bus)	Proposed	Medium
ADR-004	Estratégia de cache distribuído (Redis)	Accepted	Medium

4.2 Technology Stack Backend Services

Application Layer:

- Python ou GO
- Express.js framework
- JWT authentication

Data Layer:

- PostgreSQL (primary)
- Redis (caching)
- ElasticSearch (search)

Infrastructure:

- Docker containers
- Kubernetes (GKE)
- Prometheus + Grafana

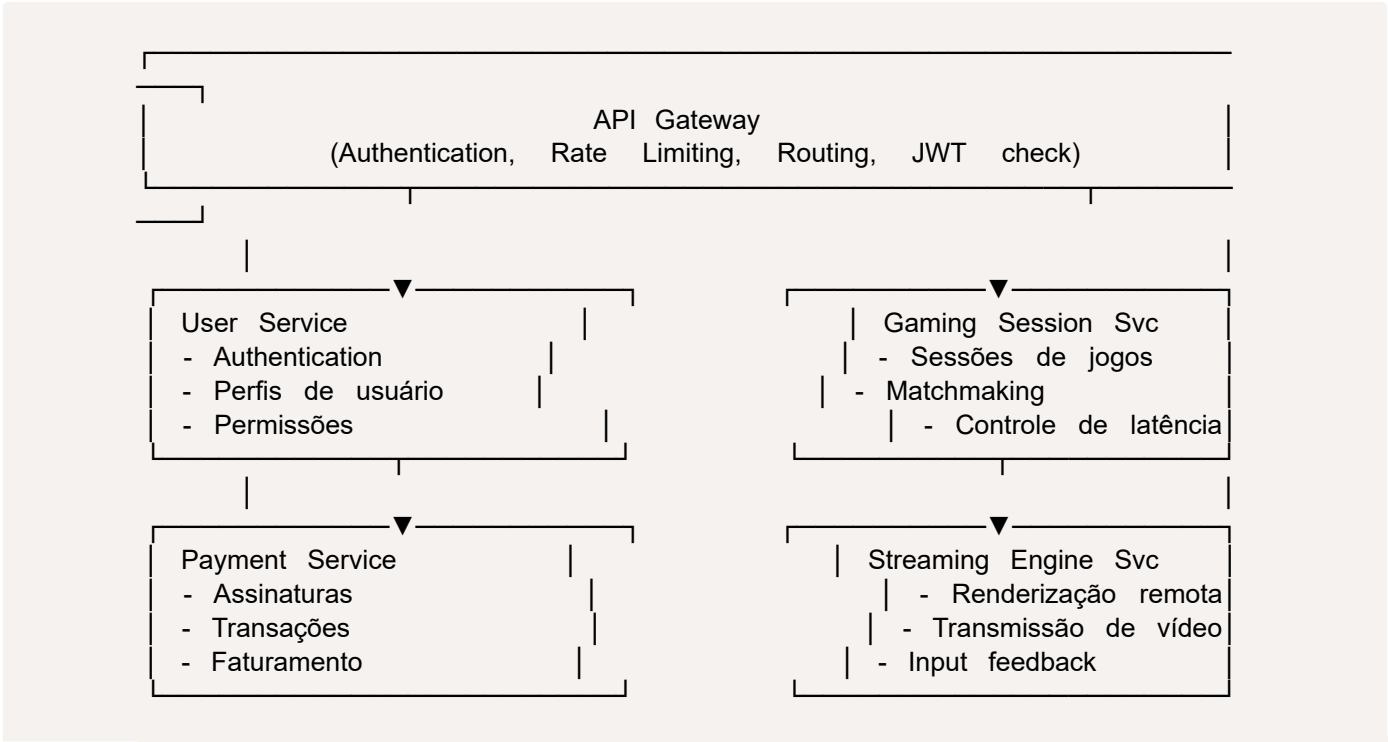
Justification: A Seleção dessas tecnologias visa atender aos requisitos de alta performance e baixa latência do cloud gaming, utilizando um ecossistema robusto e escalável para suportar um grande volume de usuários simultâneos.

Frontend Applications

- Web Dashboard: React.js + TypeScript + Material-UI
- Mobile Apps: React Native (iOS + Android)
- Why: A escolha se baseia com base que pesquisamos e identificamos como melhor para o projeto

5. SYSTEM DECOMPOSITION

5.1 Service Architecture



5.2 Service Responsibilities User Service

User Service

Key APIs:

- POST /auth/login

- GET /users/{id}

- PUT /users/{id}/permissions

Data Owned: Perfis, credenciais, permissões

Gaming Session Service

Purpose: Gerenciar sessões de jogo em tempo real

Data Owned: Sessões ativas, matchmaking, estatísticas de latência

Key APIs:

- POST /sessions
- GET /sessions/{id}
- DELETE /sessions/{id}

Payment Service

Purpose: Controle de monetização e assinaturas

Data Owned: Assinaturas, faturas, histórico de pagamentos

Key APIs:

- POST /subscriptions
- GET /payments/{id}
- POST /refunds

Streaming Engine Service

Purpose: Motor principal de cloud gaming

Data Owned: Streams ativas, buffers de vídeo/áudio, controle de inputs

Key APIs (internas):

- POST /stream/start
- POST /stream/stop
- POST /stream/input

6. DATA ARCHITECTURE

6.1 Data Strategy

A estratégia de dados suporta a arquitetura de **microsserviços** e atende aos requisitos de **baixa latência** e **escalabilidade massiva** do Kuma Cloud Gaming.

Pattern: Database-per-Service (Banco de Dados por Serviço).

Garante autonomia e resiliência dos microsserviços.Permite a escolha do banco de dados mais adequado para cada necessidade (ex: PostgreSQL para transações, Redis para cache/sessões em tempo real).

Audit Trail:

- Implementação de **Event Sourcing** (ou padrões similares baseados em eventos com Kafka) para registro imutável de todas as mudanças de estado críticas (ex: início/fim de sessão de jogo, transações no Marketplace).

Tecnologias-Chave:

- **PostgreSQL:** Banco de dados relacional primário para dados críticos e relacionais (Ex: User Domain, Marketplace Transactions).
- **Redis:** Utilizado como *cache* em memória de alta velocidade para dados de **latência zero** (Ex: estado de jogo em tempo real, dados de sessão do GameStreamingEngine, e filas de Matchmaking).
- **Elasticsearch:** Para capacidades de **busca** eficientes e *logs estruturados* (mencionado no seu Operational Concerns, mas relevante aqui para busca de jogos no Marketplace e análise de logs).
- **Apache Kafka:** Usado como **Event Bus** para comunicação assíncrona entre serviços (garantindo que o alto volume de eventos não bloqueie os serviços primários).

6.2 Data Flow Diagram

O fluxo de dados é baseado no padrão *microsserviços* e *event-driven*, permitindo o desacoplamento e a escalabilidade, essenciais para o Cloud Gaming.

```
graph LR
A[Frontend/Client] --> B(API Gateway);
B --> C(Microserviço);
C --> D[Database (PostgreSQL / Redis)];
D --> E((Event Bus - Apache Kafka));
C --> E;
E --> F[Outros Microserviços];
E --> G[Analytics Pipeline / ELK];

subgraph User Interaction
        A
        B
        C
end

subgraph Data Persistence & Events
        D
        E
end
```

Descrição do Fluxo:

Interação do Usuário: O cliente (Frontend) interage via API Gateway.

Processamento Primário: O Microserviço acessa seu Database local (PostgreSQL ou Redis) para processamento rápido da solicitação.

Comunicação Assíncrona: O Microserviço ou o Database publica eventos críticos de estado (Ex: SessionStarted, GamePurchased) no **Event Bus (Apache Kafka)**.

Distribuição: O Kafka distribui o evento para todos os *consumers* relevantes, como:

- **Outros Microsserviços:** Para manter consistência (Ex: AchivementsSystem reage ao evento SessionEnded para checar conquistas).
- **Analytics/Observabilidade:** O Analytics Pipeline consome o evento para monitoramento e geração de métricas de negócio (usuários ativos, throughput de streaming).

6.3 Data Models (High-Level)

User Domain (PostgreSQL)

Campo	Tipo	Descrição
id	UUID	PRIMARY KEY: Identificador único do usuário.
email	VARCHAR UNIQUE	Email único, usado para login.
encrypted_password	TEXT	Senha hash e salt.
role	user_role_enum	Papel do usuário (ex: player, admin).
created_at	TIMESTAMP	Data de criação do registro.
updated_at	TIMESTAMP	Data da última atualização.

Marketplace Domain (PostgreSQL)

Campo	Tipo	Descrição
transaction_id	UUID	PRIMARY KEY
user_id	UUID	Usuário que comprou
game_id	UUID	Jogo comprado
amount	DECIMAL	Valor da transação
transaction_date	TIMESTAMP	Data da compra

GameSession Domain (PostgreSQL/Redis) - *Essencial para Cloud Gaming*

Campo	Tipo	Descrição
session_id	UUID	PRIMARY KEY
user_id	UUID	Foreign Key para o usuário.
game_id	UUID	Jogo sendo executado.
status	ENUM	active, ended, paused.
start_time	TIMESTAMP	Início do stream.
server_region	VARCHAR	Região do servidor de streaming (Ex: us-east-1).
current_latency	FLOAT	Armazenado no Redis. Latência em tempo real para monitoramento do LatencyManager.

6.4 Data Consistency Strategy

Strong Consistency:

- Será aplicada em operações críticas que não podem tolerar dados desatualizados, como transações financeiras no Marketplace ou o gerenciamento do inventário de jogos de um usuário.

Eventual Consistency:

- Adotada para sistemas onde a consistência imediata não é um requisito, como o sistema de analytics, a exibição de conquistas (achievements) e as listas de amigos.

Pattern:

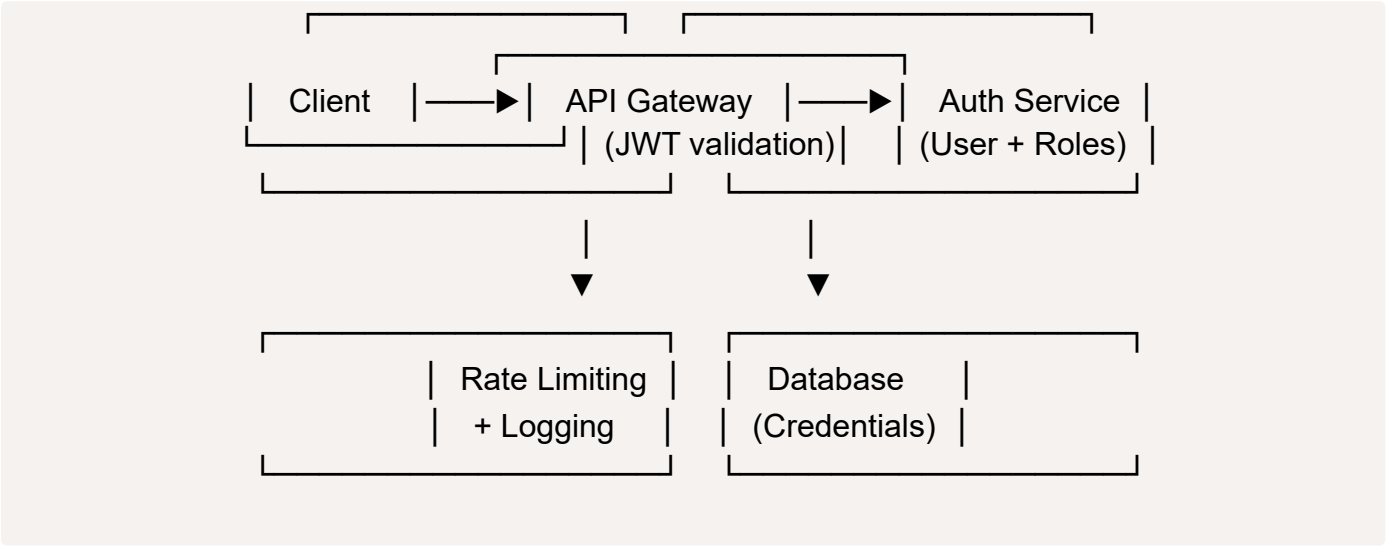
- O Saga pattern será implementado para gerenciar transações distribuídas que abrangem múltiplos serviços, garantindo a atomicidade das operações de negócio.

7. SECURITY ARCHITECTURE

7.1 Security Principles

- Zero Trust: Nenhum serviço confia implicitamente no outro
- Defense in Depth: Segurança em múltiplas camadas (rede, aplicação, dados)
- Least Privilege: Acesso mínimo necessário por serviço
- Data Encryption: Criptografia em repouso e em trânsito

7.2 Authentication & Authorization



Flow:

- Usuário envia credenciais

- Auth Service valida e emite JWT

- API Gateway valida o JWT em cada requisição

- Microsserviços confiam na validação do Gateway

7.3 Data Protection

- PII Encryption: AES-256 para dados sensíveis
- Database: Transparent Data Encryption (TDE)
- Communication: TLS 1.3 em toda a comunicação
- Backup: Criptografado em repouso com chaves rotativas

8. QUALITY ATTRIBUTES

8.1 Performance Requirements

Metric	Target	Measurement
API Response Time	< 150 ms (95th percentile)	Application monitoring
Database Queries	< 30ms (average)	Database profiling
Page Load Time	< 2 seconds (dashboard e client web/app)	Browser monitoring
Concurrent Users	Até 1 milhão simultâneos	Load/stress testing

8.2 Availability & Reliability

Target Uptime: 99.95% (máx. ~4.38h de downtime/ano)

Strategy: Multi-region + Multi-AZ deployment, Circuit Breakers, Auto-healing (K8s)

Monitoring: Health checks a cada 10 segundos (baixa tolerância a falhas em streaming)

Alerting: PagerDuty / OpsGenie para P0/P1, com escalonamento automático

8.3 Scalability Strategy

Horizontal Scaling: Kubernetes HPA (CPU/GPU/Memory based)

Database: PostgreSQL com sharding + read replicas, Redis Cluster para cache/real-time state

Caching: Escalabilidade baseada em GPU pools (NVIDIA GRID/AMD equivalent)

CDN: CloudFront/Cloudflare para distribuição de vídeo/jogos e assets estáticos

8.4 Security Posture

Authentication: OAuth 2.0 + MFA opcional para contas premium

Encryption: TLS 1.3 para todo tráfego, AES-256 para dados sensíveis
Zero Trust Networking: Segmentação entre microsserviços com mTLS

DLP (Data Loss Prevention): Monitoramento e auditorias regulares
Compliance: GDPR + LGPD (dados de usuários globais)

9. DEPLOYMENT ARCHITECTURE

9.1 Environment Strategy

A estratégia de ambientes para a plataforma Kuma Cloud Gaming é dividida em três estágios principais:

• **Development:** Ambiente com um cluster Kubernetes único e recursos compartilhados, permitindo feedback rápido para os desenvolvedores.
• **Staging:** Ambiente semelhante ao de produção, utilizado para testes de desempenho e integração.
• **Production:** Ambiente multi-AZ (zonas de disponibilidade) garantindo alta disponibilidade, resiliência e monitoramento contínuo.

9.2 Infrastructure as Code

Toda a infraestrutura é provisionada e gerenciada utilizando Kubernetes e Helm Charts. Os manifests são definidos em repositórios Git, garantindo versionamento e rastreabilidade. A nomenclatura e labels são padronizadas para facilitar a escalabilidade e observabilidade.

```
# kubernetes/production/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: kuma-cloud-gaming-prod
  labels:
    environment: production
    app: kuma-cloud-gaming
```

9.3 CI/CD Pipeline

O pipeline de CI/CD segue o fluxo:

Code Push → GitHub Actions → Build → Testes Unitários → Security Scan (SAST/DAST) → Deploy em Staging → Testes de Integração e Performance → Deploy em Produção (com aprovação manual).

Critérios de aprovação:
- Cobertura mínima de 80% nos testes.
- Benchmarks de latência inferiores a 20ms.
- Escalabilidade para suportar até 1 milhão de usuários simultâneos.

10. OPERATIONAL CONCERNS

10.1 Monitoring & Observability

O monitoramento da plataforma utiliza Prometheus e Grafana para métricas em tempo real. Principais sinais monitorados (Golden Signals): Latência, Tráfego, Erros e Saturação.

Business Metrics: Usuários ativos simultâneos, sessões de jogo bem-sucedidas, throughput de streaming.

Infraestrutura: CPU, Memória, Disco, Rede.

Logging: Utilização do stack ELK (Elasticsearch, Logstash, Kibana) com logs estruturados em JSON e correlation IDs. Retenção de 90 dias para auditoria.

Tracing: Implementado com Jaeger para rastreamento distribuído, identificação de gargalos e atribuição de falhas.

10.2 Disaster Recovery

A estratégia de recuperação de desastres inclui:

- **RTO** (Recovery Time Objective): 1 hora.
- **RPO** (Recovery Point Objective): 15 minutos.

A plataforma mantém replicação de banco de dados PostgreSQL e Redis em múltiplas regiões, além de backups automáticos diários. Testes de recuperação são realizados mensalmente.

11. EVOLUTION & ROADMAP

11.1 Technical Debt

Item	Priority	Effort	Timeline
Migrate from REST to GraphQL	Medium	8 semanas	Q2 2025
Implement proper Circuit Breakers	High	4 semanas	Q1 2025
Add comprehensive integration tests	High	4 semanas	Q1 2025

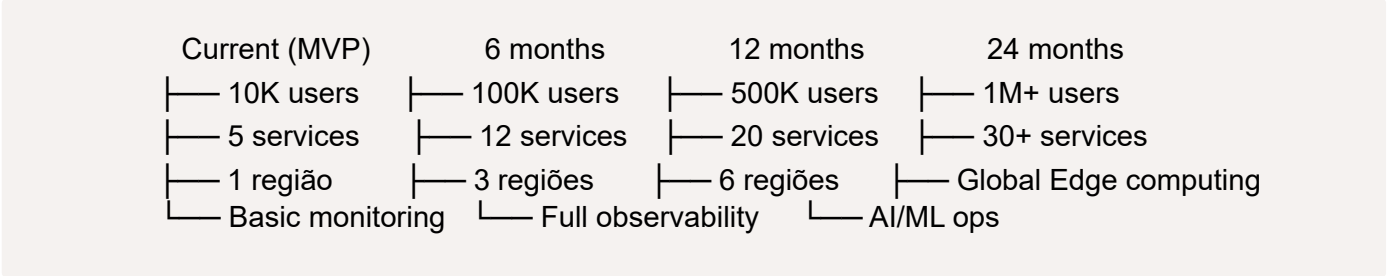
11.2 Future Enhancements

Phase 2: Machine Learning para alocação preditiva de GPU/servidores

Phase 3: Integração com dispositivos IoT (smart TVs, consoles portáteis)

Phase 4: Blockchain para integridade de ativos digitais (licenças de jogos)

11.3 Scalability Roadmap



12. RISKS & MITIGATION

Risk	Probability	Impact	Mitigation
Downtime de API de Terceiros	Média	Alto	Implementação de uma **estratégia multi-cloud** (AWS, GCP, Azure) para GPUs, com um *fallback* para infraestrutura *bare-metal* (própria ou dedicada) em caso de falha de provedores.
Violação de Dados (Data Breach)	Baixa	Muito Alto	Adotar segurança em **múltiplas camadas** (criptografia, controle de acesso estrito), realização regular de **pentests** (testes de intrusão) e **auditorias** de segurança.
Degradação de Performance do Banco de Dado	Média	Alto	Implementação de **otimização de queries** contínua, uso estratégico de **Redis** para *caching* de dados de alta leitura e, se necessário, adoção de **sharding** para distribuir a carga do PostgreSQL.
Indisponibilidade de Membro Chave da Equipe	Alta	Média	Criação de **documentação** completa e atualizada para todos os serviços (ADRs, Runbooks) e prática de **rotação de responsabilidades** (cross-training) para garantir o conhecimento distribuído.