Pontifícia Universidade Católica de Goiás

João Vitor Ferreira da Silva

Pedro Nunes Marques Junior

Vitor Hugo Batista Pereira

Ariel Jorge da Silva

Leandro Batista de Sousa Galdino

PROJETO INTEGRADOR III-A

migração de arquitetura monolítica para microsserviço

Goiânia - GO

2025

INTRODUÇÃO

Este documento técnico apresenta, de forma abrangente e detalhada, a proposta de migração do sistema de vendas online da TechStore — atualmente estruturado como uma aplicação monolítica — para uma arquitetura distribuída baseada em microsserviços. A iniciativa surge a partir da necessidade de modernizar a plataforma tecnológica da organização, garantindo maior escalabilidade, flexibilidade e resiliência diante do crescimento contínuo do volume de transações e da complexidade operacional. À medida que o sistema atual evoluiu ao longo dos anos, tornou-se evidente que o modelo monolítico impõe limitações significativas, tais como dificuldade de manutenção, acoplamento excessivo entre componentes, ciclos de entrega mais lentos e riscos elevados associados a falhas pontuais que impactam toda a aplicação.

Nesse contexto, a transição para microsserviços é proposta como um caminho estratégico para aprimorar a capacidade de resposta da TechStore às demandas do mercado, facilitar a adoção de novas funcionalidades e reduzir o impacto de mudanças internas. A abordagem distribuída permite uma divisão clara de responsabilidades entre serviços independentes, cada um com escopo funcional específico, possibilitando ciclos de desenvolvimento isolados, implantação contínua e escalabilidade seletiva conforme as necessidades de uso. Além disso, a arquitetura orientada a serviços contribui para a melhoria da observabilidade, da resiliência e da tolerância a falhas, elementos essenciais para a operação de sistemas críticos em ambientes corporativos de médio e grande porte.

Este documento consolida as diretrizes técnicas necessárias para essa transformação, abrangendo desde a análise das restrições do modelo atual até a definição da arquitetura alvo, mecanismos de integração e estratégias de comunicação entre serviços. São também discutidas práticas de Segurança da Informação que asseguram o cumprimento dos pilares de Confidencialidade, Integridade e Disponibilidade (CID), garantindo que os dados e operações do sistema permaneçam protegidos durante todo o processo de migração e na

operação contínua do ambiente distribuído. Por fim, o relatório apresenta uma estratégia de migração incremental, incluindo táticas de rollback, mitigação de riscos e uso de padrões arquiteturais adequados para garantir uma transição segura, previsível e alinhada às melhores práticas do setor.

DESENVOLVIMENTO

O desenvolvimento desta proposta técnica parte de um cenário-base que simula a operação atual da TechStore, cujo sistema de vendas online foi construído e evoluiu ao longo dos anos como uma aplicação monolítica. Esse modelo, embora funcional e historicamente adequado para o estágio inicial da empresa, tem se mostrado progressivamente limitado diante do aumento do tráfego, da ampliação do portfólio de produtos e da necessidade de incorporar novas funcionalidades com maior agilidade. Nesta seção, descrevem-se em detalhe as características da arquitetura vigente, bem como os fatores que motivam a migração para um paradigma baseado em microsserviços.

Atualmente, o sistema da TechStore concentra toda a lógica de negócio em uma única base de código e em um único processo de execução. Os módulos responsáveis por autenticação, catálogo de produtos, gerenciamento de pedidos e processamento de pagamentos compartilham um mesmo repositório e um banco de dados relacional unificado. A implantação é realizada como um artefato único, o que exige ciclos completos de testes de integração e torna o processo de deploy mais rígido e arriscado.

As principais características observadas são:

- **Implantação monolítica:** todo o sistema é empacotado e distribuído como uma única aplicação, sem separação lógica ou operacional entre módulos.
- **Banco de dados único:** todas as entidades — usuários, produtos, pedidos e transações — residem em um único esquema relacional, dificultando independência evolutiva de dados.
- **Alta interdependência:** a comunicação entre módulos ocorre por chamadas internas altamente acopladas (tight-coupled), impossibilitando isolamento de falhas e dificultando modificações pontuais.
- **Escalabilidade vertical predominante:** a única forma prática de aumentar a capacidade de processamento é reforçar o servidor ou a máquina virtual que executa o monólito, limitando elasticidade.

- **Ciclo de desenvolvimento lento:** o compartilhamento de uma base de código extensa torna mais frequentes conflitos de merge, regressões inesperadas e dificuldades para manter pipelines de entrega contínua.
- **Observabilidade limitada:** o monitoramento, métricas e logs são centralizados, porém com pouca granularidade, dificultando identificar gargalos, falhas específicas e comportamentos anômalos por funcionalidade.

Com base nessas características, foram identificados riscos e limitações importantes:

- **Ponto único de falha:** qualquer indisponibilidade em um módulo pode comprometer a operação de toda a plataforma.
- **Impacto operacional no deploy:** mudanças ou correções exigem a atualização de todo o sistema, aumentando risco de indisponibilidade e necessidade de janelas de manutenção.
- **Barreiras tecnológicas:** a adoção de novas tecnologias, linguagens ou bancos de dados específicos por módulo é inviabilizada pelo acoplamento estrutural.
- **Escalabilidade ineficiente:** não é possível escalonar seletivamente apenas os componentes mais exigidos, resultando em uso subótimo de recursos.
- **Segurança e compliance:** a falta de segregação de dados e de controles granulares de acesso aumenta a complexidade para implementar políticas de segurança e atender requisitos regulatórios.

Identificação de Módulos do Sistema Atual

O sistema monolítico da TechStore foi concebido inicialmente para sustentar operações básicas de comércio eletrônico. À medida que a empresa expandiu seu catálogo de produtos, integrou-se a parceiros externos e passou a atender um volume significativamente maior de usuários simultâneos, o modelo monolítico começou a apresentar gargalos estruturais e restrições tecnológicas. Nesta seção, descrevem-se em profundidade os módulos que compõem o sistema atual,

destacando suas funções, pontos de interdependência e limitações que motivam a necessidade de migração para uma arquitetura distribuída.

1. **Módulo de Autenticação e Usuários:** Este módulo é responsável por funcionalidades críticas de controle de acesso e gestão de identidade, incluindo autenticação (login), cadastro de novos usuários, gerenciamento de permissões e perfis, redefinição de senha, expiração de sessão e validação de tokens de acesso. Por se tratar do ponto de entrada de praticamente todas as transações, é um dos componentes mais sensíveis do sistema.

   Principais problemas observados:

   a. **Forte acoplamento interno:** diversas rotinas deste módulo chamam diretamente funções de outros subsistemas, criando dependências rígidas e dificultando a evolução individual do componente.
   b. **Dependências circulares:** fluxos de autenticação acionam consultas ao catálogo de produtos ou ao módulo de pedidos para checagens secundárias, ampliando o risco de falhas em cascata.
   c. **Baixa escalabilidade em horários de pico:** o aumento simultâneo de logins, validações e renovações de sessão provoca lentidão perceptível.
   d. **Risco elevado de indisponibilidade geral:** falhas neste módulo, por ser centralizado e indispensável, têm grande probabilidade de derrubar todo o monólito.

2. **Módulo de Produtos e Estoque:** Responsável pelo coração da operação de e-commerce, este módulo mantém o catálogo de produtos, suas categorias, atributos, imagens, estoque disponível e histórico de movimentações. É o módulo mais consultado pelos usuários, especialmente em períodos de campanhas promocionais ou datas comemorativas.

   Principais limitações identificadas:

a. **Consultas pesadas ao banco:** buscas complexas e filtros aplicados diretamente no banco relacional geram alto consumo de CPU e I/O, especialmente sem índices otimizados.
b. **Ausência de cache:** a inexistência de mecanismos de caching (como Redis ou CDN para imagens) aumenta a latência e sobrecarrega o banco de dados.
c. **Impacto sistêmico na experiência do usuário:** lentidão neste módulo repercute diretamente na navegação, exibição do catálogo e construção do carrinho, afetando toda a jornada de compra.
d. **Dependências diretas:** o módulo é consumido por autenticação, pedidos e pagamentos, ampliando o impacto de qualquer falha ou degradação.

3. **Módulo de Pedidos:** Este módulo centraliza a lógica relacionada ao carrinho de compras, cálculo de frete, criação, atualização e monitoramento de pedidos. Inclui regras de preço, aplicação de descontos, etapas de checkout e integração direta com o módulo de pagamentos.

   Principais desafios e vulnerabilidades:

   a. **Mistura de regras de negócio com lógica de apresentação:** parte significativa da lógica está acoplada a templates e componentes visuais, dificultando manutenções e testes automatizados.
   b. **Acúmulo histórico de falhas:** por ser um dos módulos mais modificados, apresenta alto índice de regressões e código legado.
   c. **Dependência rígida de outros módulos:** qualquer oscilação nos módulos de produtos ou pagamentos causa falhas na criação e atualização dos pedidos.
   d. **Baixa resiliência:** não há mecanismos de retry, filas assíncronas ou compensações transacionais, tornando o fluxo de compra suscetível a interrupções.

4. **Módulo de Pagamentos:** Este é o módulo mais crítico do ponto de vista transacional. Ele integra com múltiplos gateways externos, processa

autorizações de pagamento, valida antifraude (quando aplicável) e registra transações com parceiros financeiros.

Principais pontos de atenção:

a. **Sensibilidade extrema a falhas externas:** latências e indisponibilidades dos provedores de pagamento são propagadas para dentro do sistema, tornando o monólito inteiro vulnerável.
b. **Ausência de isolamento:** uma simples falha na comunicação com um gateway pode bloquear o checkout e impactar as demais funcionalidades do sistema.
c. **Lentidão com impacto geral:** tempos de resposta elevados ocasionam filas de requisições e travamentos, refletindo em degradação do desempenho global.
d. **Necessidade urgente de desacoplamento:** por lidar com rotinas críticas e de alto risco operacional, este módulo é prioridade na transição para microsserviços.

Arquitetura Distribuída Proposta

A nova arquitetura proposta para a TechStore adota um modelo distribuído baseado em microsserviço, com foco em escalabilidade, isolamento de falhas, independência tecnológica e fortalecimento da segurança da informação. A transição para esse paradigma visa reduzir o acoplamento entre componentes, melhorar a capacidade de evolução do sistema e permitir que cada serviço seja desenvolvido, implantado e monitorado de forma autônoma. A seguir, detalham-se os principais serviços definidos, os mecanismos de comunicação entre eles e as estratégias de tolerância a falhas adotadas.

Detalhamento dos Microsserviços

A decomposição do monólito resultou na identificação de domínios funcionais independentes, cada um representado por um microsserviço com responsabilidades claramente delimitadas. Cada serviço pode adotar a tecnologia de persistência mais adequada ao seu domínio (polyglot persistence) e pode evoluir sem impactar diretamente os demais.

1. **Auth Service:** Responsável pela autenticação, autorização e gestão de identidades.

   Principais características:

   a. Implementação de **OAuth2** e emissão de **JWT** com escopos e claims personalizados.
   b. **Hashing seguro** de senhas utilizando bcrypt ou Argon2.
   c. Manejo de **refresh tokens** com expiração configurável.
   d. Controle granular de permissões e papéis (RBAC/ABAC).
   e. Logs específicos de autenticação e tentativas de acesso, reforçando auditoria e rastreabilidade.
   f. Isolamento do domínio de segurança evita que vulnerabilidades afetem outros serviços.

2. **Product Service:** Gerencia catálogo, informações de produtos, categorias, imagens e estoque.

   Características técnicas:

   a. Banco NoSQL **MongoDB**, adequado para estruturas flexíveis de produtos.
   b. Camada de **cache Redis** para consultas intensivas e redução da carga no banco primário.
   c. Atualização **assíncrona** dos níveis de estoque, consumindo eventos do Order Service.
   d. Capacidade de escalabilidade horizontal para suportar picos de acesso ao catálogo.

e. Indexação otimizada para melhorar desempenho de busca e filtragem.

3. **Order Service:** Responsável por todo o fluxo de criação, atualização e gerenciamento de pedidos.

   Principais elementos:

   a. Persistência em **PostgreSQL** para garantir transações robustas e consistência relacional.
   b. Implementação do padrão **SAGA** para orquestrar processos distribuídos (pedido → pagamento → atualização de estoque).
   c. Consumo e publicação de eventos em sistema de mensageria (Kafka ou RabbitMQ).
   d. Capacidade de reprocessamento automático em falhas específicas do fluxo de compra.
   e. Separação clara entre lógica de negócios e processamento assíncrono.

4. **Payment Service:** Gerencia transações financeiras e integrações com gateways externos.

   Características fundamentais:

   a. Processamento **assíncrono**, reduzindo latência perceptível pelo usuário.
   b. Mecanismo de **antifraude** interno ou integrado a fornecedores externos.
   c. **Retentativas automáticas** de pagamento com política de backoff.
   d. Sincronização periódica de status com gateways para consistência entre sistemas.
   e. Logs e trilhas de auditoria reforçadas devido ao caráter crítico do domínio.

5. **Notification Service:** Serviço responsável pela comunicação ativa com o usuário.

Funcionalidades:

a. Envio de notificações por **e-mail, push e SMS** baseado em eventos do sistema.
b. Integração com provedores externos por meio de webhooks e filas assíncronas.
c. Templates dinâmicos e rastreamento de mensagens entregues e falhas.
d. Isolamento garante que gargalos de envio de mensagens não afetem o fluxo de compra.

Comunicação Entre Serviços

A arquitetura distribuída exige protocolos otimizados e adequados a cada tipo de operação. A combinação de estratégias síncronas e assíncronas garante desempenho, resiliência e flexibilidade.

- **REST (HTTP/JSON):** Utilizado para consultas simples entre serviços, especialmente leituras eventuais que não exigem alta performance. Adequado para integração entre microsserviços internos ou chamadas de front-end.
- **gRPC:** Empregado em cenários que demandam **baixa latência**, streaming ou alto throughput. Ideal para comunicação entre serviços de backend com chamadas frequentes e de alto volume.
- **Kafka ou RabbitMQ (Mensageria Assíncrona):** Esse modelo permite **desacoplamento**, reprocessamento, escalabilidade e resiliência. Utilizados para publicação e consumo de eventos entre serviços:
    - atualizações de estoque
    - criação de pedidos
    - eventos de pagamento

- **Webhooks Externos:** Usados para integrações com provedores de pagamento, antifraude e serviços de notificação externos. Permitem comunicação reativa baseada em eventos.

Controle de Falhas e Resiliência

Para garantir disponibilidade e robustez do ambiente distribuído, são aplicados padrões e mecanismos avançados de tolerância a falhas:

- **Circuit Breaker:** Impede que falhas repetidas em um serviço provoquem degradação global, abrindo o circuito temporariamente.
- **Bulkhead Pattern:** Isola pools de recursos para evitar que falhas de um serviço esgotem recursos críticos de outros.
- **Timeouts configuráveis:** Garante que chamadas externas não bloqueiem o serviço indefinidamente.

  Retry com Backoff Exponencial: Repetição automática de operações críticas com controle progressivo de intervalos, evitando sobrecarga.
- **Dead Letter Queue (DLQ):** Destinação de mensagens que não puderem ser processadas após várias tentativas, mantendo integridade do fluxo.
- **Detecção de falhas e autoscaling:** Integrado a plataformas como Kubernetes, habilitando substituição automática de instâncias e expansão sob demanda.

Estratégia de Migração

A migração do sistema monolítico da TechStore para uma arquitetura distribuída de microsserviços deve ser conduzida de forma gradual, controlada e altamente observável, a fim de evitar interrupções na operação e permitir reversão rápida em caso de falhas. A seguir, descrevem-se as etapas sequenciais da estratégia, considerando impactos técnicos, dependências entre módulos, táticas de redução de risco e requisitos de segurança.

Etapa 1 – Implementação do API Gateway

O primeiro passo consiste na criação de um **API Gateway** que atuará como ponto único de entrada para as requisições externas. Antes da extração de qualquer módulo, o Gateway será configurado para rotear chamadas para o monólito, sem alterar o comportamento atual do sistema.

Principais objetivos:

- Centralizar autenticação, rate limiting, auditoria e logging.
- Estabelecer camada de roteamento e versionamento de APIs.
- Preparar o ambiente para transição gradual de endpoints do monólito para microsserviços.
- Criar base para observabilidade distribuída (correlação de trace IDs).

Benefício principal: permite migração **transparente** ao cliente, reduzindo impacto operacional.

Etapa 2 – Extração do Módulo de Produtos (Product Service)

Por possuir alto volume de acesso e baixo acoplamento com regras transacionais críticas, o módulo de produtos é o primeiro a ser extraído. A nova API será exposta pelo API Gateway, enquanto o monólito ainda consumirá dados legados até que a migração esteja estável.

Ações-chave:

- Implementação do Product Service com MongoDB e Redis.
- Sincronização inicial dos dados a partir do banco monolítico.
- Criação de adaptadores (anti-corruption layer) para permitir que o monólito consulte o novo serviço.
- Monitoramento de latência e consistência de dados.

Motivação: reduz carga no banco monolítico e elimina um dos gargalos mais significativos.

Etapa 3 – Migração do Auth Service

O módulo de autenticação é sensível e afeta todos os usuários. A abordagem deve assegurar transição segura para o novo modelo de sessões e tokens.

Etapas principais:

- Criação do Auth Service com OAuth2 e JWT.
- Migração gradual de sessões, permitindo coexistência temporária do sistema de login antigo com o novo (“dual auth mode”).
- Regeneração controlada de tokens para usuários ativos.
- Reconfiguração do API Gateway para delegar autenticação ao microsserviço.
- Implementação de logs detalhados para detecção de anomalias de acesso.

Risco crítico mitigado: falhas de autenticação deixam de derrubar o sistema inteiro, pois o novo serviço será isolado e escalável.

Etapa 4 – Criação do Order Service com Orquestração SAGA

Nesta etapa, inicia-se a extração do fluxo de pedidos, um dos processos mais complexos devido às dependências com produtos e pagamentos.

Atividades principais:

- Implementação do Order Service com PostgreSQL.
- Criação de orquestradores SAGA para lidar com criação de pedido, reserva de estoque e confirmação de pagamento.
- Publicação e consumo de eventos via Kafka/RabbitMQ.
- Ajuste do monólito para enviar e receber eventos temporariamente, garantindo compatibilidade durante a transição.
- Testes intensivos de consistência e rollback transacional distribuído.

Resultado esperado: pedidos passam a ser processados de forma resiliente, isolando falhas em serviços dependentes.

Etapa 5 – Migração do Payment Service (Fase Mais Sensível)

Por integrar com sistemas financeiros externos, esta é a etapa com maior criticidade operacional. A migração deve ser feita em modo “shadow” antes da ativação completa.

Passos principais:

- Construção do Payment Service com rotinas assíncronas e antifraude.
- Implementação de filas, retries, circuit breaker e backoff exponencial.
- Execução de um período de **execução paralela (shadow mode)** onde:
    - o monólito continua processando pagamentos oficialmente;
    - o novo serviço processa transações em paralelo apenas para validação.
- Testes comparativos de consistência e latência.
- Comutação gradual para o novo serviço com rollback automático via feature flags.

Foco crítico: evitar falhas financeiras que resultem em cobranças duplicadas, pedidos incorretos ou rejeições indevidas.

Etapa 6 – Desativação Progressiva do Monólito

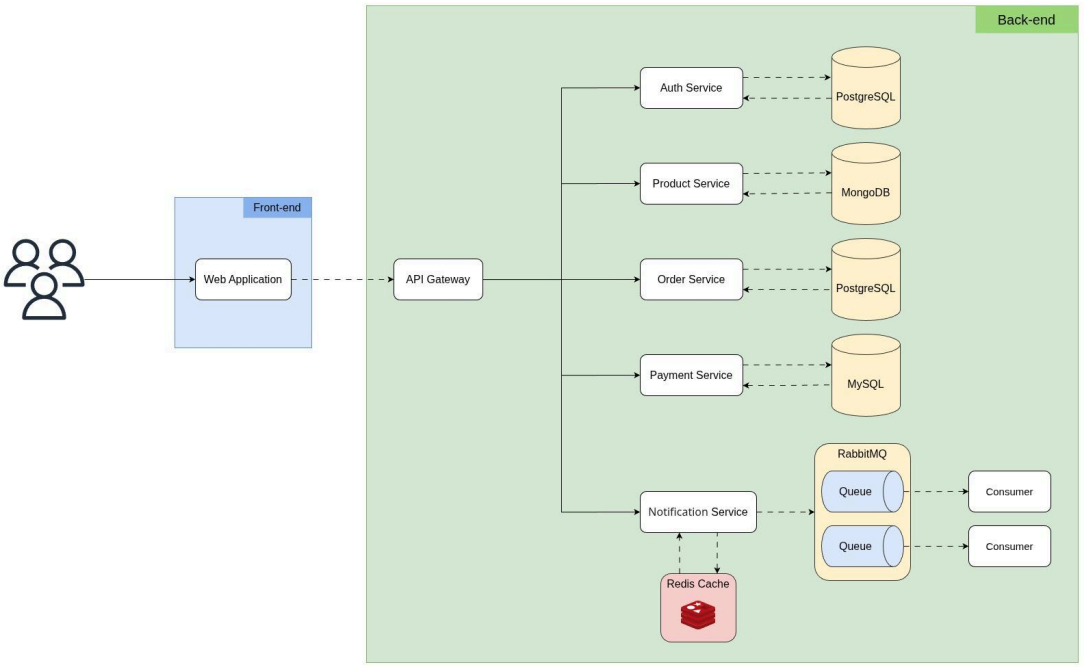
Após cada microsserviço demonstrar estabilidade operacional, inicia-se a desativação progressiva dos módulos remanescentes do monólito.

Atividades:

- Redução incremental de responsabilidades do monólito até restarem apenas funções residuais.
- Remoção de consultas internas ainda dependentes do legado.
- Migração final de dados históricos, se necessário.
- Desligamento faseado com janelas controladas e planos de rollback.
- Atualização das documentações, dashboards e rotinas de operação.

Resultado final: o monólito torna-se obsoleto e completamente desligado, concluindo a transição para a arquitetura distribuída.

Diagrama da Arquitetura



A arquitetura distribuída proposta para a TechStore pode ser representada por um diagrama lógico que destaca os principais componentes, suas interações e as camadas tecnológicas que sustentam o ecossistema de microsserviços. Embora a visualização gráfica não esteja incluída nesta seção, a descrição a seguir fornece uma visão completa e estruturada de todos os elementos que compõem o desenho arquitetural.

Visão Geral da Arquitetura

No centro da solução está o **API Gateway**, que funciona como ponto de entrada único para todos os clientes — aplicações web, mobile, integrações externas e parceiros. A partir dele, o tráfego é roteado para os diferentes microsserviços, que são executados como containers **Docker** e orquestrados por **Kubernetes** para garantir alta disponibilidade, escalabilidade e isolamento.

A comunicação assíncrona entre os serviços ocorre através de um barramento de eventos implementado com **Kafka**, permitindo fluxos desacoplados,

tolerância a falhas e alta vazão de mensagens. Para otimização de desempenho, serviços de alta demanda utilizam **Redis** como camada de cache distribuído. Cada microsserviço possui seu próprio banco de dados, seguindo o princípio de **banco independente por serviço** (polyglot persistence).

A observabilidade é assegurada por meio de **Prometheus** e **Grafana**, responsáveis por métricas, alertas, dashboards e monitoramento contínuo da saúde dos serviços. Além disso, todos os logs são centralizados em uma plataforma de log aggregation (como ELK/EFK), garantindo rastreabilidade, auditoria e análise de incidentes.

Segurança da Informação (CID)

A nova arquitetura adota o princípio de **Security by Design**, aplicando controles preventivos, detectivos e corretivos desde as fases iniciais de concepção dos microsserviços. A abordagem reforça a conformidade com requisitos internos da TechStore, boas práticas internacionais (NIST, OWASP, ISO 27001) e necessidades operacionais de um ambiente distribuído. Os pilares de **Confidencialidade, Integridade e Disponibilidade (CID)** são tratados de forma transversal, garantindo proteção contínua de dados, fluxos de comunicação e componentes críticos.

Confidencialidade

Para assegurar que informações sensíveis não sejam acessadas por entidades não autorizadas, a arquitetura integra controles modernos e camadas redundantes de proteção:

- **Criptografia de Trânsito:** uso obrigatório de **TLS 1.3** em toda comunicação externa e interna entre microsserviços, incluindo gRPC, REST e eventos.
- **Autenticação Federada:** implementação de **OAuth2 + OpenID Connect**, oferecendo segurança e padronização no ciclo de autenticação.
- **Proteção de Credenciais:** senhas e segredos tratados com **hashing bcrypt** e rotacionados periodicamente.
- **Criptografia de Dados em Repouso:** aplicação de **AES-256** para campos sensíveis em bancos e mensagens persistidas.
- **Gestão de Acessos:**
    - **RBAC (Role-Based Access Control)** para permissões operacionais.
    - **ABAC (Attribute-Based Access Control)** para decisões dinâmicas em microsserviços específicos.
- **Isolamento de Ambientes:** ambientes de dev, stage e produção segregados em clusters independentes.
- **Política Zero Trust:** verificação contínua de identidade, autorização e contexto, sem confiar em perímetros pré-existentes.

Integridade

A integridade garante que dados e transações não sejam alterados indevidamente. A arquitetura adota mecanismos preventivos e processuais que asseguram confiabilidade:

- **Assinaturas Digitais com ECDSA:** aplicadas em tokens, mensagens críticas e fluxos inter-serviços sensíveis.
- **Checksums e Hashes de Conteúdo:** usados para validar pacotes, arquivos, payloads e eventos em trânsito.

- **Logs Imutáveis (WORM):** armazenamento Write Once Read Many, permitindo trilhas de auditoria resistentes a alterações.
- **Auditoria Completa:** registro de acessos, ações administrativas e operações críticas com correlação via IDs de requisição.
- **Versionamento de APIs:** mantém compatibilidade e permite detecção de inconsistências entre versões.
- **Validação Estrita de Payloads:** schemas JSON e Protobuf com validação automática nos Gateways e serviços consumidores.

Disponibilidade

A arquitetura prioriza continuidade operacional, oferecendo resiliência contra falhas de rede, sobrecarga de tráfego e indisponibilidade de componentes externos:

- **Replicação Ativa/Ativa:** microsserviços executados em réplicas distribuídas para garantir tolerância a falhas.
- **Balanceadores NGINX/Envoy:** distribuem requisições, reduzem hotspots e realizam health checks contínuos.
- **Autoscaling Horizontal:** o Kubernetes ajusta dinamicamente a quantidade de réplicas com base em métricas (CPU, latência, filas).
- **Monitoramento Contínuo:** Prometheus + Grafana para métricas, alertas e detecção precoce de anomalias.
- **Backups Automáticos:** de bancos e tópicos críticos, com políticas de retenção e versionamento.
- **Testes Regulares de Restauração:** simulações de falhas e recovery para garantir confiabilidade dos backups.
- **Infraestrutura Multizona:** execução dos serviços em múltiplas zonas de disponibilidade para evitar indisponibilidade regional.

CONCLUSÃO

A migração da arquitetura monolítica para uma arquitetura distribuída baseada em microsserviços representa um avanço estratégico significativo para a TechStore. O novo modelo melhora a **escalabilidade**, permite **isolamento de falhas**, aumenta a **resiliência operacional** e possibilita adoção contínua de novas tecnologias. Além disso, a separação lógica dos domínios facilita evolução independente dos serviços, reduz riscos em implantações e aprimora a experiência do usuário final.

Com a inclusão de mecanismos robustos de **Segurança da Informação**, alinhados aos pilares **CID**, a empresa passa a operar com maior maturidade tecnológica, garantindo proteção, continuidade e confiabilidade em todo o ciclo de processamento de dados. A arquitetura distribuída estabelece uma base sólida para crescimento sustentável, inovação futura e capacidade de competir em ambientes de alta demanda.

REFERÊNCIAS

- **Newman, Sam.** *Building Microservices.*
- **Fowler, Martin.** *Microservices Patterns.*
- **Richards, Mark; Ford, Neal.** *Fundamentals of Software Architecture.*
- Documentação oficial: **NGINX**, **Kubernetes**, **Kafka**, **Prometheus**, **Grafana**.
- **RFC 7519 — JSON Web Tokens (JWT).**