Documento de Arquitetura de Software (DAS)

Sistema de Controle e Consolidação de Fluxo de Caixa

Autor: Fernando Pinheiro Gomes

Data: 02/09/2025

Versão: 2.0

# 1. Introdução

Este documento apresenta a arquitetura de software proposta para o controle e consolidação do fluxo de caixa de um comerciante. O sistema permitirá o registro de lançamentos financeiros diários (débitos e créditos) e a geração de relatórios consolidados com o saldo diário. A solução deve ser precisa, escalável, resiliente e segura, atendendo aos requisitos funcionais e não funcionais definidos.

# 2. Objetivos e Escopo

# O objetivo principal é desenvolver um sistema que: - Permita o controle de lançamentos financeiros (débito e crédito). - Disponibilize relatórios de saldo consolidado diário. - Garanta resiliência entre os serviços de lançamento e consolidação. - Seja escalável para suportar picos de até 50 requisições/segundo. - Implemente práticas de segurança, monitoramento e auditoria. 2.1 Microserviços propostos

O escopo contempla a implementação de um conjunto de microserviços desacoplados que se comunicam de maneira assíncrona por meio de um barramento de mensageria, garantindo que o serviço de lançamentos não dependa da disponibilidade imediata do serviço de consolidação.

A arquitetura proposta é baseada em microsserviços orientados a eventos. O **Serviço de Lançamentos** será responsável por registrar os débitos e créditos informados pelos usuários, persistindo-os em um banco de dados transacional e publicando eventos em um barramento de mensageria. Esses eventos serão consumidos pelo **Serviço de Consolidação**, que processará as informações recebidas e calculará o saldo consolidado do dia, armazenando o resultado em uma base analítica destinada a relatórios e consultas posteriores. Para disponibilizar os resultados ao comerciante exporá uma API de consulta, possibilitando a integração com dashboards ou sistemas de terceiros. Complementarmente, o **Serviço de Notificações** (proposto mas não implementado no desafio técnico) será responsável por informar o comerciante sobre o fechamento do saldo diário por meio de e-mail, webhooks ou mensageria, enquanto o **Serviço de Autenticação e Autorização** (proposto mas não implementado no desafio técnico) será simulado para assegurar o acesso seguro ao sistema, definindo papéis específicos como administrador, operador financeiro e auditor. Por fim, o **Serviço de Monitoramento e Observabilidade** (implementado com promethus e grafana) garantirá a coleta de métricas, logs e rastreamento distribuído, viabilizando a gestão proativa do ambiente com ferramentas especializadas.

Os principais microserviços que compõem a solução são:

* **Serviço de Lançamentos**: responsável pelo registro de débitos e créditos e publicação de eventos.
* **Serviço de Consolidação**: realiza o processamento dos lançamentos e gera o saldo consolidado.
* **Serviço de Notificações**: comunica o fechamento do saldo diário por e-mail, webhook ou mensageria.
* **Serviço de Autenticação e Autorização**: garante acesso seguro e controle de permissões.
* **Serviço de Monitoramento e Observabilidade**: coleta métricas, logs e rastreamento distribuído. Serão utilizadas as ferramentas disponibilizadas pelo próprio framework para coleta das métricas

# 3. Justificativas Técnicas e de Negócio

A escolha por microserviços orientados a eventos foi feita para garantir desacoplamento, resiliência e capacidade de reprocessamento de eventos. Uma arquitetura monolítica, embora mais simples, não atenderia à escalabilidade nem à resiliência exigida. O uso de Apache Kafka foi preferido em relação a RabbitMQ ou SQS pela sua robustez em cenários de alto throughput e retenção de eventos.  
  
O PostgreSQL foi escolhido pelo equilíbrio entre custo, performance e confiabilidade. Redis foi adicionado como cache para dados de alta consulta, reduzindo a carga no banco relacional. A AWS foi escolhida pela capacidade de escalar horizontalmente, suporte a multi-AZ e ecossistema maduro de serviços. Do ponto de vista de negócio, essas escolhas asseguram que o comerciante tenha um sistema confiável, com continuidade operacional mesmo em caso de falhas parciais. Para

# 4. Análise Arquitetural Crítica

Trade-offs:  
- Eventos assíncronos: garantem resiliência, mas aumentam a complexidade de troubleshooting.  
- PostgreSQL: garante consistência forte, mas pode ter limitações de escala vertical.  
- Redis: garante alto desempenho em consultas, mas pode perder dados se não configurado com persistência.  
  
Riscos:  
- Sobrecarga no Kafka: mitigada com particionamento e ajuste de consumidores.  
- Falhas de segurança: mitigadas com WAF, autenticação forte e segregação de redes.  
- Custos de nuvem: mitigados com autoscaling, instâncias reservadas e monitoramento de consumo.  
  
Plano de evolução:  
- Suporte a múltiplas lojas e integrações contábeis.  
- Uso de machine learning para previsão de fluxo de caixa.  
- Sharding do banco de dados e expansão de tópicos Kafka.

# 5. Arquitetura Proposta (Visão Geral)

A arquitetura segue o padrão de microserviços orientados a eventos. Os serviços principais são:  
- API Gateway: entrada segura para clientes. (Simulado com nginx, mas pode ser usado KONG, WSO2)  
- Serviço de Lançamentos: registra débitos e créditos.  
- Serviço de Consolidação: processa lançamentos e gera saldo.  
- Banco de Dados Transacional: armazena lançamentos. (Implementado com H2 mas sugerido o PostgreSQL)  
- Serviço de Notificação: envia relatórios ou expõe APIs. (Não implementado no desafio)  
- Serviço de Autenticação/Autorização: autentica e autoriza usuários nos padrões de mercado OAuth2/OpenID. (Não implementado no desafio)   
- Monitoramento e Observabilidade: coleta métricas e logs.  
  
Descrição substitutiva de diagramas:  
- C1 (Contexto): comerciante → sistema de fluxo de caixa → integrações externas (e-mail, dashboards).  
- C2 (Contêineres): Frontend Angular, Backend (Spring Boot), Kafka, Redis, PostgreSQL, API Gateway.  
- C3 (Componentes): Serviço de Lançamentos (controller, service, producer Kafka), Serviço de Consolidação (consumer Kafka, calculador de saldo, persistência analítica).  
- C4 (Código): Pacote em Java para o Serviço de Lançamentos (controller → service → repository).  
  
Numa representação livre, poderia a arquitetura poderia ser descripta da seguinte forma:

Diagrama

O conteúdo gerado por IA pode estar incorreto.

# 6. Tecnologias e Frameworks

- Frontend: Angular + TypeScript  
- Backend: Java 21 + Spring Boot 3.x  
- Banco de Dados: PostgreSQL  
- Mensageria: Apache Kafka (Amazon MSK)  
- Cache: Redis  
- Infraestrutura: Kubernetes (EKS) + Docker

# 7. Infraestrutura

Os serviços serão containerizados com Docker e orquestrados via Kubernetes (EKS). Será adotada arquitetura multi-AZ para garantir resiliência e alta disponibilidade. API Gateway, Kafka e bancos estarão distribuídos entre zonas de disponibilidade. Foi implementado no desafio apenas o Docker por ser mais simples na execução local.

# 8. Escalabilidade, Alta Disponibilidade e Resiliência

* Escalabilidade horizontal via autoscaling em Kubernetes.
* Banco de dados com replicação e failover automático.
* Kafka com retenção de eventos para reprocessamento.
* Redis para otimização de consultas de alta frequência.

A arquitetura permitirá escalabilidade horizontal dos microserviços com autoscaling em Kubernetes. O serviço de lançamentos funcionará mesmo se o serviço de consolidação estiver indisponível. O banco de dados terá replicação e failover automático, e o Kafka assegura retenção de eventos para reprocessamento. Para rodar localmente a solução, será disponibilizadado um docker compose, que por sis só não garante alta resiliência e disponibilidade. Para ambientes de testes e desenvolvimento, isto não será necessário.

# 9. Segurança

Medidas adotadas:

- Autenticação e autorização via OAuth2/OpenID Connect (Keycloak ou Cognito).  
- Criptografia em trânsito (TLS 1.2+) e em repouso (KMS).  
- Proteção contra ataques comuns com WAF.  
- Segregação de redes (VPC, sub-redes públicas e privadas).

# 10. Monitoramento e KPIs

Serão monitorados KPIs como:  
- Latência de publicação/consumo de eventos.  
- Tempo médio de consolidação.  
- Falhas de autenticação.  
- Custo por requisição.  
  
Alertas proativos serão configurados, ex.: saldo não consolidado até 23h59 → alerta automático.

# 11. DevSecOps

CI/CD implementado com GitHub Actions ou GitLab CI.  
- Build e testes automatizados.  
- Análise de qualidade com SonarQube.  
- Segurança de containers com Trivy.  
- Deploy automatizado em Kubernetes.  
- Observabilidade com Prometheus/Grafana.

# 12. Custo

Observação: valores aproximados (em USD), baseados em preços médios da AWS em **us-east-1** em 2025. Custos variam conforme região, descontos de reserva e uso real.

**🔹 Componentes Principais e Custos**

**1. Kubernetes (EKS)**

* **EKS Control Plane**: ~$74/mês (taxa fixa).
* **Nós de Aplicação**: 3 × t3.medium (~$0.0416/h cada).  
  → ~$90/mês por nó × 3 = **$270/mês**.  
  👉 **Subtotal EKS: $344/mês**

**2. Banco de Dados (Amazon RDS – PostgreSQL)**

* Instância db.t3.medium (2 vCPU, 4GB RAM).
* ~ $0.067/h → ~$50/mês.
* Armazenamento 100 GB (gp3 SSD): ~$8/mês.  
  👉 **Subtotal RDS: $58/mês**

**3. Mensageria (Amazon MSK – Kafka)**

* Cluster básico com 3 brokers m5.large ($0.11/h cada).
* ~$240/mês.
* Armazenamento 100 GB: ~$10/mês.  
  👉 **Subtotal MSK: $250/mês**

**4. Cache (Redis – Amazon ElastiCache)**

* 1 nó cache.t3.small → ~$20/mês.  
  👉 **Subtotal Redis: $20/mês**

**5. Armazenamento Analítico (Amazon S3)**

* 100 GB armazenados: ~$2/mês.  
  👉 **Subtotal S3: $2/mês**

**6. Monitoramento e Logs**

* **CloudWatch** logs + métricas (~50GB/mês): ~$25/mês.  
  👉 **Subtotal Observabilidade: $25/mês**

**7. Rede e Transferência de Dados**

* Tráfego médio de 100 GB/mês para fora da AWS: ~$9/mês.  
  👉 **Subtotal Rede: $9/mês**

**🔹 Estimativa Consolidada**

| **Componente** | **Custo Mensal (USD)** |
| --- | --- |
| EKS (Cluster + Nós) | $344 |
| RDS PostgreSQL | $58 |
| MSK (Kafka) | $250 |
| Redis (ElastiCache) | $20 |
| S3 (Storage) | $2 |
| CloudWatch/Logs | $25 |
| Rede (Transferência) | $9 |
| **Total Estimado** | **~ $708/mês** |

**🔹 Observações**

* Esse valor (~**$708/mês**) cobre **produção pequena/média** com resiliência (multi-AZ, 3 nós de aplicação, 3 brokers Kafka).
* Em **dev/teste**, pode ser reduzido drasticamente (~$200/mês) usando instâncias menores, 1 nó por serviço e MSK Serverless.
* Para **produção crítica**, os custos podem subir (ex.: RDS Multi-AZ ~$100/mês, Redis clusterizado ~$80/mês, mais tráfego de rede).