# **Documento de Arquitetura de Microserviços: Controle de Fluxo de Caixa**

### 1. Introdução

Este documento descreve a arquitetura baseada em microsserviços proposta e desenvolvida, por Yan Esteves, para o sistema de Controle de Fluxo de Caixa.

A arquitetura segue todos os requisitos funcionas e não funcionais, todos os microsserviços foram desenvolvidos utilizando DDD, SOLID e boas praticas, todos contam com testes unitários, inclusive os componentes do front-end.

Foi desenvolvida com base nos requisitos funcionais e não funcionais apresentados pela empresa, utilizando tecnologias .NET para os serviços, React com Tailwind CSS para o front-end, e Docker para facilitar a implantação.

### 2. Visão Geral

### 2.1 Design de Arquitetura

### Diagrama Descrição gerada automaticamente

* **Tipo de Arquitetura**: Microserviços (simples)
* **Containers**: Docker
* **Mensageria**: RabbitMQ
* **Api Gateway**: Nginx
* **Serviços**: .NET Core
* **Banco de Dados**: PostgreSQL
* **Frontend**: React.JS + Tailwind CSS
* **Escalabilidade**: Horizontal

## 2.2. Microserviços

**Serviço de Controle de Lançamentos**:

* Responsável pelo registro de débitos e créditos.
* Alta cobertura de testes unitários.
* Utilizando DDD e boas práticas de código e arquitetura.
* Desenvolvido em .NET com banco de dados Postgresql e Entityframework Core.
* Dockerizado para facilitar a implementação.
* Possui uma dependência com o serviço de Banco de Dados do Controle de Lançamentos.
* Utiliza RabbitMQ para comunicação assíncrona entre serviços.
* Configuração de ambiente:
  + Variáveis de ambiente para conexão com o banco de dados.
  + Fila no RabbitMQ para comunicação assíncrona.

**Serviço de Banco de Dados do Controle de Lançamentos:**

* Container PostgreSQL para armazenamento de dados do Controle de Lançamentos.
* Exposto na porta 5432.
* Configuração do ambiente:
  + Nome do banco de dados: controle\_lancamentos\_db.
  + Usuário: admin
  + Senha: pass

**Serviço de Consolidado Diário:**

* Calcula o saldo diário consolidado com base nos lançamentos.
* Utiliza cache para otimizar consultas.
* Alta cobertura de testes unitários.
* Utilizando DDD e boas práticas de código e arquitetura.
* Dockerizado para facilitar a implementação.
* Possui uma dependência com o serviço de Redis e o serviço de Banco de Dados do Controle de Lançamentos.
* Utiliza RabbitMQ para comunicação assíncrona entre serviços.
* Configuração de ambiente:
  + Host do Redis: consolidado-report-redis.
  + Fila no RabbitMQ para comunicação assíncrona.

**Serviço de Autenticação**

* Responsável pela autenticação de usuários.
* Utilizando DDD e boas práticas de código e arquitetura.
* Desenvolvido em .NET com banco de dados Postgresql e Entityframework Core.
* Dockerizado para facilitar a implementação.
* Possui uma dependência com o serviço de Banco de Dados de Autenticação.
* Utiliza RabbitMQ para comunicação assíncrona entre serviços.
* Configuração de ambiente:
  + Variáveis de ambiente para conexão com o banco de dados.

**Serviço de Redis para Consolidado Diário:**

* Container Redis para cache do serviço de Consolidado Diário.
* Exposto na porta 6379.

**Serviço de Banco de Dados de Autenticação**:

* Container PostgreSQL para armazenamento de dados de autenticação.
* Exposto na porta 5433.
* Configuração do ambiente:
* Nome do banco de dados: autenticacao\_db.
  + Usuário: admin
  + Senha: pass

**Serviço de RabbitMQ**

* Container RabbitMQ para gestão de filas e mensagens.
* Exposto nas portas 5672 e 15672 (interface de gerenciamento).
* Configuração do ambiente:
  + Usuário: admin
  + Senha: pass
  + Volume "rabbitmq\_data" para persistência dos dados.

**Serviço de API Gateway:**

* Container Nginx funcionando como um gateway.
* Ele também valida a autenticação utilizando um script em **Lua** para poder validar se o token está de acordo com a secret, se a necessidade de chegar no microsserviço.
* Exposto na porta 80.
* Configurações customizadas no arquivo nginx.conf.
* Utiliza o arquivo jwt-auth.lua para autenticação.
* Dependências nos serviços de Controle de Lançamentos, Consolidado Diário, Autenticação e RabbitMQ.
* Construído a partir de um Dockerfile com a configuração JWT\_SECRET definida como "4fcfc25be07f50a0f0f6214df8b495c0".
* Redireciona tráfego para os serviços apropriados.

## **Frontend**

* Desenvolvido em React com Tailwind CSS.
* Consumirá os serviços de microserviços.
* Implementação de testes de unidade e integração.

## **Infraestrutura**

* Banco de Dados: Microsoft SQL Server para armazenamento dos lançamentos.
* Message Broker: RabbitMQ para comunicação assíncrona entre serviços.
* Orquestração: Kubernetes para gerenciamento e orquestração de contêineres.
* Balanceamento de Carga: Utilização de um serviço para distribuir o tráfego.

## **Ferramentas Adicionais**

* Logging: ELK Stack (Elasticsearch, Logstash, Kibana).
* Monitoramento: Application Insights para métricas de desempenho.

### 3. Testes

### Foram realizados testes de estresse com JMetter, com o cenário dos requisitos não funcionais. A aplicação apresentou a resiliência necessária, de acordo com os 5% de perda de requisições estipulados no desafio.

### Texto Descrição gerada automaticamenteTela de computador com texto preto sobre fundo branco Descrição gerada automaticamente

### *Testes executados no Jmetter.*

### 4. Considerações Finais

Esta arquitetura foi projetada para atender aos requisitos apresentados, proporcionando uma solução robusta, escalável e de fácil manutenção.