PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído

Projeto Integrado

Relatório Técnico

Sistema de Convênio Bancário

Leonardo Santarém Rottoli

Belo Horizonte
Agosto de 2023.

Projeto Integrado – Arquitetura de Software Distribuído

Sumário

1. Introdução

O avanço das soluções tecnológicas tem aberto caminho para que as empresas melhorem a arquitetura de seus sistemas, resultando em soluções mais eficientes, escaláveis e resilientes. Os arquitetos de software frequentemente se deparam com desafios complexos que requerem soluções impactantes para o negócio, sem causar impactos negativos nos outros componentes da arquitetura e nas relações com os clientes e parceiros externos.

Empresas já consolidadas no mercado, com sistemas robustos muitas vezes baseados em arquiteturas legadas, enfrentam o desafio constante de atender prontamente às demandas dos clientes e partes interessadas. Nesse contexto, é comum que parcerias empresariais sejam viabilizadas por meio de integrações que envolvem o envio e recebimento de arquivos de integração, um padrão que tem sido adotado pela indústria há várias décadas. Contudo, o crescimento operacional pode potencialmente transformar o processamento desses arquivos em um obstáculo, especialmente em função das escolhas arquiteturais adotadas. O processamento de arquivos de grande volume pode resultar em um consumo mais elevado de memória durante o processamento das informações, agravando também o risco de falhas durante o processo, que podem levar à necessidade de reprocessamento total ou até mesmo gerar dados inconsistentes, em casos mais extremos.

Com o foco nas empresas inseridas nesse contexto, este trabalho busca oferecer uma solução otimizada para o processamento de arquivos, utilizando abordagens provenientes da arquitetura de software distribuído. A adoção de tecnologias como Apache Flink e Apache Kafka não apenas proporciona ganhos significativos em termos de desempenho, escalabilidade e resiliência para processos críticos, mas também se mostra especialmente relevante para organizações que buscam resolver desafios de desempenho no processamento de arquivos de forma direcionada, sem a necessidade de reestruturar completamente sua arquitetura.

Este estudo concentra-se em um cenário específico, onde um banco necessita realizar integrações por meio de arquivos com empresas parceiras, visando a recepção de pedidos de débito em conta, também conhecidos como "convênio bancário". É esperado que essa abordagem prática e específica possa trazer vantagens tanto para o setor bancário quanto para outras áreas que enfrentam desafios semelhantes no processamento eficiente de informações em colaborações entre organizações.

O objetivo deste trabalho é desenvolver uma solução otimizada para o processamento de arquivos em empresas que necessitam de integrações eficientes, utilizando uma arquitetura de software distribuído.

Os objetivos específicos propostos são:

- Realizar uma análise detalhada das tecnologias de processamento de dados distribuído, com ênfase nas ferramentas como Apache Flink e Apache Kafka.
- Identificar os desafios específicos enfrentados pelas empresas no processamento de arquivos e nas integrações, enfatizando os gargalos de desempenho e as questões de escalabilidade.
- Projetar uma arquitetura de software distribuído que incorpore efetivamente as tecnologias selecionadas, visando melhorar o desempenho e a eficiência do processamento de arquivos em cenários de integração.
- Desenvolver um protótipo funcional da solução proposta, demonstrando sua aplicabilidade e validando sua eficácia em lidar com os desafios identificados.
- Avaliar quantitativamente o desempenho da solução em comparação com abordagens tradicionais, por meio de métricas como tempo de processamento, consumo de recursos e escalabilidade.
- Elaborar diretrizes e recomendações para a implementação bem-sucedida da arquitetura proposta em ambientes empresariais reais.

Através desses objetivos específicos, pretende-se alcançar o objetivo maior de oferecer uma solução abrangente e eficaz para o processamento de arquivos em integrações empresariais, contribuindo para uma melhor eficiência operacional e otimização dos recursos tecnológicos.

2. Especificação Arquitetural da solução

Nesta seção, as orientações arquiteturais e os critérios que norteiam o desenvolvimento da solução proposta são delineados. O início é marcado pela apresentação das restrições arquiteturais, que estabelecem os limites e diretrizes fundamentais. Em sequência, os requisitos funcionais e não funcionais são detalhados, definindo as funcionalidades específicas e as qualidades intrínsecas a serem atendidas. Por fim, os mecanismos arquiteturais são descritos, delineando as estratégias adotadas para a consecução dos objetivos estabelecidos. Cada aspecto contribui para a compreensão integral do design arquitetural e seus elementos-chave..

2.1 Restrições Arquiteturais

As restrições arquiteturais descritas a seguir delimitam o escopo da solução e estabelecem parâmetros obrigatórios que precisam ser seguidos para garantir a eficácia e a conformidade com as necessidades do projeto.

R1: A solução deve ser desenvolvida utilizando as tecnologias Apache Flink e Apache Kafka para o processamento distribuído de dados.
R2: A arquitetura deve ser projetada para lidar com arquivos de diversos formatos e tamanhos, garantindo eficiência e escalabilidade
R3: A solução deve ser compatível com sistemas legados que utilizam integração via envio e recebimento de arquivos de integração.
R4: A arquitetura deve ser capaz de suportar um aumento gradual no volume de dados processados, assegurando que a performance não seja degradada.
R5: A solução deve ser projetada considerando requisitos de segurança, garantindo a integridade e confidencialidade dos dados durante todo o processo de processamento de arquivos.

2.2 Requisitos Funcionais

A seguir, são apresentados os requisitos funcionais que delineiam as funcionalidades específicas da solução proposta. Esses requisitos são fundamentais para o desdobramento do fluxo de processamento, desde a chegada dos arquivos de cobrança até a geração de arquivos de retorno e a interação com o core bancário. Cada requisito destaca uma etapa crucial do fluxo, contribuindo para a eficiência, a segurança e o cumprimento das metas do projeto.

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	O sistema deve receber arquivos de pedido de cobrança provenientes da integração via ETF File Gateway.	B	A
RF02	Os arquivos de pedido de cobrança recebidos devem ser armazenados no Amazon S3 por meio de função serverless.	B	M
RF03	Os pedidos de cobrança contidos nos arquivos devem ser processados de forma eficiente pelo microsserviço Apache Flink.	M	A
RF04	Linhas vazias oriundas do arquivo devem ser desconsideradas.	B	B
RF05	Registros inconsistentes no arquivo não devem ocasionar a interrupção do processamento.	B	A
RF06	Cada registro processado deve ser convertido em um modelo de evento. Em caso de sucesso, o evento deve conter as informações de pedido de cobrança. Em caso de erro, o evento deve conter um objeto de erro contendo uma breve descrição do problema encontrado.	M	A
RF07	Os registros processados devem ser publicados em um tópico Kafka como foco secundário da solução.	B	A
RF08	Os eventos contendo os pedidos de cobrança devem ser consumidos por um microsserviço Spring Boot responsável por aplicar regras de negócio para validação.	M	A
RF09	O microsserviço de validação deve publicar eventos de pedido de cobrança para o core bancário tentar efetuar o débito em conta.	B	A
RF10	O resultado da tentativa de cobrança, seja sucesso ou falha por saldo insuficiente ou conta bloqueada, deve ser consumido por um microsserviço Spring Boot.	B	A
RF11	O microsserviço de retorno deve armazenar temporariamente os resultados das tentativas de cobrança em um banco de dados.	B	A
RF12	A partir dos registros armazenados temporariamente, o microsserviço de retorno deve gerar arquivos de retorno contendo as informações pertinentes.	B	A
RF13	Os arquivos de retorno gerados devem ser disponibilizados no Amazon S3 para consulta posterior.	B	A
RF14	Uma função Lambda deve ser responsável por monitorar a chegada de arquivos de retorno no Amazon S3 e movê-los para o ETF.	M	B

RF15	O sistema deve fornecer logs detalhados das atividades realizadas pelos microsserviços, a fim de possibilitar o rastreamento e posterior análise.	A	M

*B=Baixa, M=Média, A=Alta.

2.3 Requisitos Não-funcionais

A seguir são destacados os requisitos não funcionais que definem os padrões essenciais que a solução deve alcançar. Isso abrange desde o desempenho até a segurança, garantindo a eficiência no processamento e a adaptabilidade da arquitetura. Cada requisito ressalta uma área fundamental para o êxito da implementação.

ID	Descrição	Prioridade B/M/A
RNF01	A solução deve ser capaz de processar arquivos de diferentes tamanhos e formatos de maneira eficiente, garantindo tempos de processamento adequados mesmo para volumes elevados de dados.	A
RNF02	A arquitetura deve ser dimensionável de forma horizontal, permitindo aumentar a capacidade de processamento conforme a demanda, sem comprometer a eficiência.	A
RNF03	A solução deve garantir alta disponibilidade, minimizando o tempo de inatividade e assegurando que as operações de processamento estejam disponíveis quando necessário.	B
RNF04	A arquitetura deve adotar medidas robustas de segurança, incluindo autenticação, autorização e criptografia de dados, a fim de proteger a integridade e confidencialidade das informações.	M
RNF05	A solução deve ser resiliente a falhas, sendo capaz de lidar com erros, quedas de nós ou componentes sem interromper o fluxo de processamento e, quando possível, recuperar automaticamente.	A

2.4 Mecanismos Arquiteturais

Nesta seção serão abordados os principais mecanismos arquiteturais que sustentam a solução de processamento de arquivos, analisando como atendem às necessidades do sistema nos estágios de análise, design e implementação. Serão explorados desde a persistência de dados até a comunicação assíncrona entre componentes, revelando como esses mecanismos garantem eficácia, escalabilidade e funcionalidade.

Análise	*Design*	Implementação
Persistência	Mecanismo de acesso a dados	Spring Data MongoDB
Integração com sistemas legados	Arquitetura de Microsserviços	Spring Boot
Integração	Comunicação assíncrona	Apache Kafka
Log do sistema	Captura de logs	Graylog
Teste de Software	Testes unitários	JUnit e Mockito
Deploy	Arquitetura em camadas	Docker e Kubernetes
Processamento de dados	Data streaming	Apache Kafka

3. Modelagem Arquitetural

A Modelagem Arquitetural proporciona uma visão abrangente da estrutura subjacente à solução de processamento de arquivos, representando sua interação com o ambiente, a organização dos componentes e a relação entre eles. Por meio dos diagramas de contexto, container e componentes, este capítulo oferecerá uma compreensão clara das fronteiras do sistema, das divisões lógicas dos módulos e das relações entre os elementos, contribuindo para uma visualização precisa e detalhada da arquitetura proposta.

3.1 Diagrama de Contexto

O diagrama de contexto proporciona uma visão panorâmica da solução de processamento de arquivos, destacando suas interações essenciais com atores externos e sistemas relacionados. Ao ilustrar as entradas e saídas fundamentais do sistema, esse diagrama estabelece um alicerce para compreender como a solução se conecta e colabora com o ambiente circundante, enriquecendo nossa percepção das trocas de informações e operações cruciais.

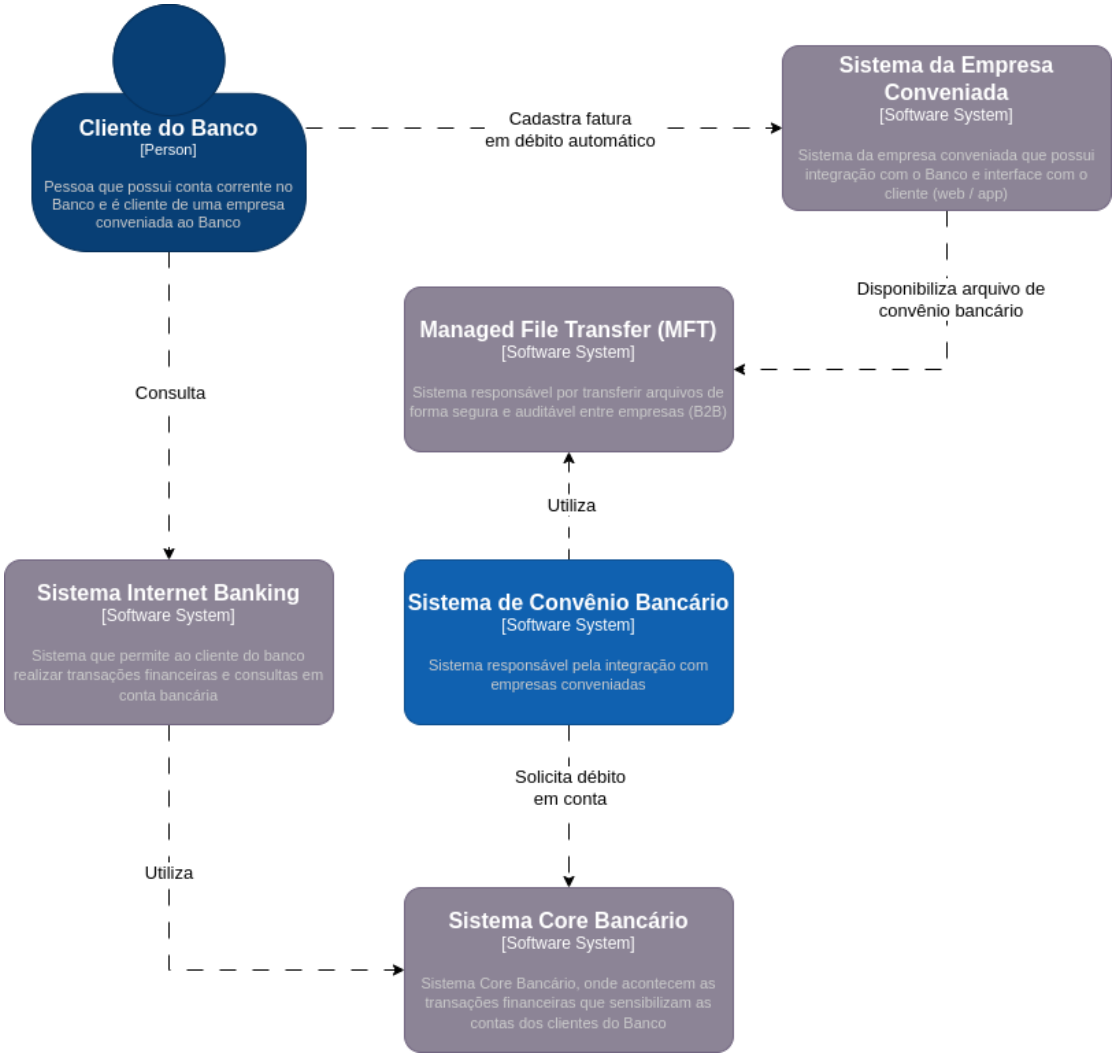

Figura 1 - Visão Geral da Solução. Elaborado pelo autor.

A figura 1 ilustra, de forma abrangente, como a solução de processamento de arquivos (aqui abstraída na figura do “Sistema de Convênio Bancário”) interage com os demais sistemas do contexto proposto. O ator principal é um usuário que é ao mesmo tempo cliente de um banco e de uma empresa conveniada ao banco. Ao comunicar à empresa parceira que deseja cadastrar sua fatura em débito em conta, essa empresa precisa estabelecer uma integração com o banco. Neste estudo estão sendo tratadas as integrações via arquivo, que são comumente estabelecidas entre empresas parceiras através de soluções de MTF (Managed Filte Transfer), como o IBM Sterling File Gateway. O Sistema de Convênio Bancário processa esse arquivo e dispara uma requisição de débito em conta para o sistema de core bancário, que é um sistema legado que possui interfaces de comunicação síncronas e assíncronas.

O retorno dessa transação é obtido novamente pelo Sistema de Convênio Bancário, e um arquivo de retorno é gerado e enviado de volta para a empresa

conveniada. O usuário consegue consultar o resultado dessa transação tanto no sistema de Internet Banking do banco quanto no sistema de gestão da empresa parceira, da qual ele também é cliente.

3.2 Diagrama de Container

O diagrama de container oferece uma representação visual da estrutura de alto nível da solução de processamento de arquivos, destacando os principais componentes e suas interações. Ao descrever as diferentes camadas e seus relacionamentos, esse diagrama proporciona uma compreensão clara da arquitetura em termos de containers, tais como aplicativos, bancos de dados e serviços externos, auxiliando na identificação das dependências e na visualização da organização global do sistema.

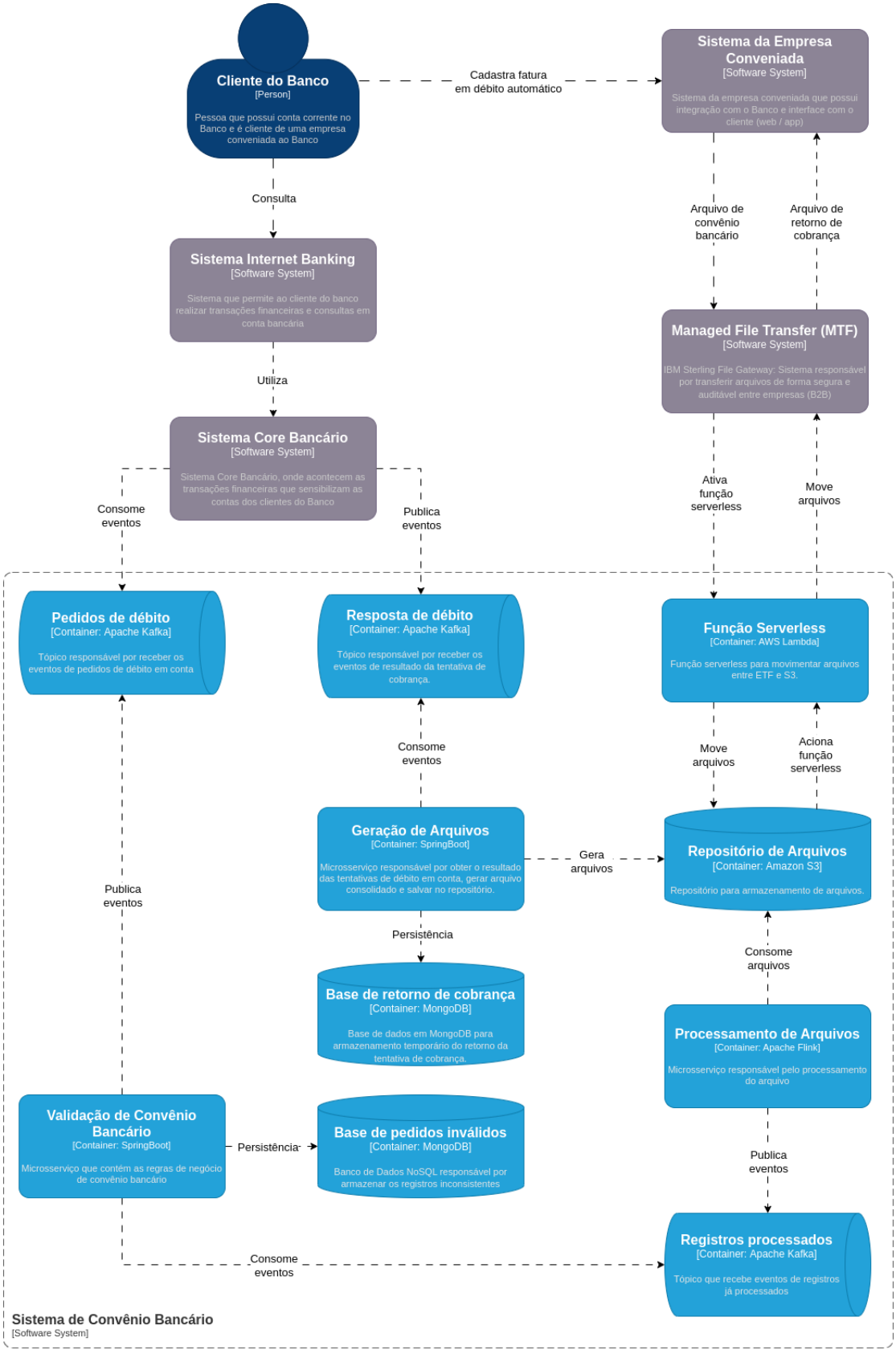

Figura 2 – Diagrama de container. Elaborado pelo autor.

A figura 2 apresenta os containers da aplicação e a forma como são integrados. Uma vez que um arquivo é disponibilizado pela empresa parceira no MTF, uma função AWS Lambda é disparada e movimenta o arquivo para um bucket S3. O microsserviço Apache Fink tem uma rotina agendada para verificar periodicamente esse bucket e consumir os arquivos para processamento. A arquitetura poderia fazer uso exclusivamente de microsserviços Spring Boot, porém, a decisão pelo uso do Apache Flink como solução de processamento de arquivo se deve aos seguintes fatores:

- Processamento Eficiente de Grandes Arquivos
- Processamento em Tempo Real e Contínuo
- Segmentação de Dados
- Janelas de Tempo e Descarte de Dados
- Gerenciamento de Estado Eficiente
- Economia de Recursos
- Paralelismo e Desempenho
- Baixa Latência e Resposta Rápida

De forma mais abrangente, o Flink é capaz de escalar horizontalmente, distribuindo o processamento de um único arquivo em instâncias distintas. Em uma aplicação Spring Boot, isso não seria possível, pois um arquivo teria que ser carregado inteiramente em memória por uma instância da aplicação, para que então fosse processado. Ao final do processamento, eventos são gerados em um tópico Apache Kafka, contendo o registro de pedido de débito, ou um resumo do erro resultante no processamento daquele registro individual. Importante destacar que, em caso de falha, o Flink consegue retomar de onde parou, ou seja, o arquivo não precisa ser reprocessado por completo novamente.

Os eventos são consumidos por um microsserviço Spring Boot, que realizará a verificação de elegibilidade da empresa que solicitou o débito em conta. Caso a empresa não esteja elegível, o pedido não vai para o core bancário, e um relatório é gerado ao final do dia para que a equipe responsável entre em contato com os representantes da empresa. Caso a empresa seja elegível, um evento é produzido em um tópico Kafka para consumo do sistema de core bancário. O banco então tentará efetuar o débito em conta, e publicará em um tópico Kafka o resultado dessa tentativa, que obrigatoriamente será um desses três status:

- Cobrança efetuada com sucesso

- Falha na cobrança: saldo insuficiente
- Falha na cobrança: conta bloqueada

Um novo microsserviço Spring Boot será responsável por consumir os eventos gerados pelo retorno da cobrança e persistir esses eventos em uma base de dados MongoDB, de forma temporária para que, ao final do dia, um arquivo seja gerado contendo o resultado de cada tentativa de cobrança. Uma vez gerado o arquivo, a base de dados é expurgada, para evitar consumo desnecessário de recursos. O arquivo gerado é disponibilizado no bucket S3, onde existe um gatilho para acionar a função Lambda que moverá esse arquivo para a ferramenta MTF, onde a empresa parceira terá acesso para obter o arquivo e realizar as operações pertinentes.

3.3 Diagrama de Componentes

O diagrama de componente revela uma visão detalhada da estrutura interna da solução de processamento de arquivos, decompondo os principais módulos e suas interconexões. Por meio dessa representação visual é possível compreender melhor as relações funcionais e técnicas entre os componentes, contribuindo para uma análise mais aprofundada da arquitetura.

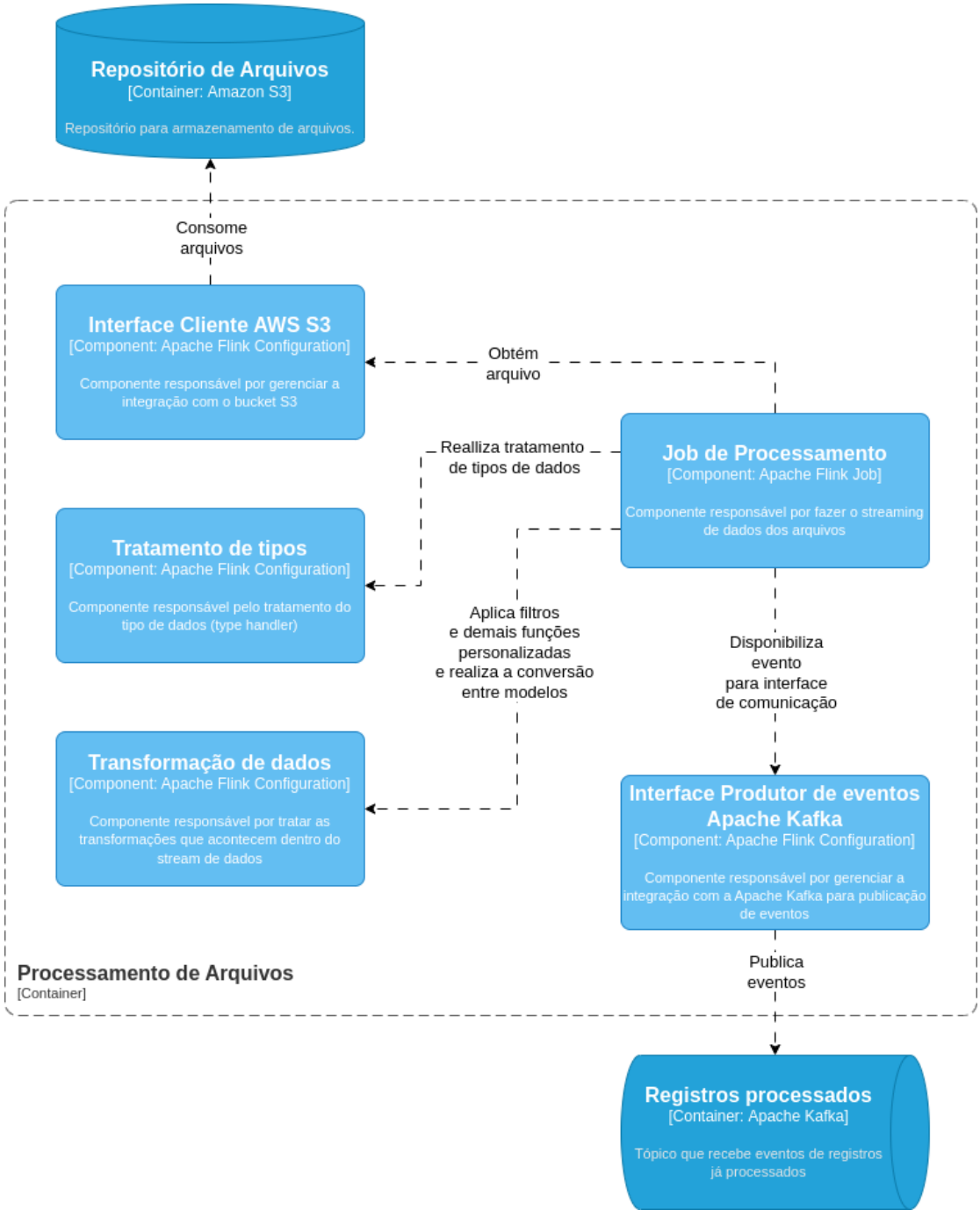

Figura 3 – Diagrama de Componentes. Elaborado pelo autor.

A figura 3 apresenta em detalhes os componentes que formam o container “Processamento de Arquivos” (Apache Flink), que é o foco principal do presente estudo. Sob a perspectiva desse container, o fluxo inicia com a chegada de um ou mais arquivos no bucket S3. O componente “Interface Cliente AWS S3” possui as configurações necessárias para estabelecer a conexão com o S3, e essa conexão é estabelecida através de uma rotina agendada, que periodicamente acessa o bucket em busca de novos arquivos. Quando há novos arquivos, o componente “Job de

Processamento” inicia o fluxo, gerando um stream de dados a partir do arquivo recebido. Na pipeline desse stream são acionados os tratamentos de dados básicos, que no contexto desse microsserviço, dizem respeito à consistência dos registros com relação ao layout de arquivo esperado. Para cada registro tratado pelo stream, o job aciona os seguintes componentes:

- Tratamento de tipos: componente responsável pela configuração dos tipos básicos que serão utilizados no processamento, como por exemplo, a formatação esperada para campos de datas e moedas.
- Transformação de dados: componente responsável por abrigar as regras de saneamento e transformação que serão aplicadas a cada registro. Podem ser aplicadas regras como:
    - Ignorar linhas vazias (FilterFunction)
    - Ignorar cabeçalhos e rodapés (FilterFunction)
    - “Parse” do registro e conversão para o POJO / modelo de dados adequado (FlatMapFunction)
    - Captura de erros (FlatMapFunction)
    - Definição de chave única para agrupamento de registros duplicados (KeySelector)
    - Agregação dos registros e conversão para o modelo de dados do evento a ser gerado (WindowFunction)

A etapa final da pipeline consiste da publicação de eventos para um tópico Apache Kafka, que ocorre mediante configuração definida no componente “Interface Produtor de eventos Apache Kafka”. Os eventos são publicados em um tópico que recebe os registros processados, os quais posteriormente serão consumidos por outro microsserviço, pertencente ao container “Validação de Convênio Bancário”.