

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação Lato Sensu em Arquitetura de Software Distribuído

Projeto Integrado

Relatório Técnico

TMS - Transport Management System

Edi Paulino da Silva

Franca SP

2023.

Projeto Integrado - Arquitetura de Software TMS

Sumário

1. Introdução	4
2. Especificação Arquitetural da solução	6
2.1. Restrições Arquiteturais	6
2.2. Requisitos Funcionais	7
2.3. Requisitos Não-Funcionais	9
2.4. Mecanismos Arquiteturais	9
3. Modelagem Arquitetural	10
3.1. Diagrama de Contexto	10
3.2. Diagrama de Container	11
3.3. Diagrama de Componentes	13
4. Avaliação da Arquitetura (ATAM)	14
4.1. Análise das abordagens arquiteturais	14
4.2. Cenários	15
4.3. Evidências da Avaliação	16
5. Avaliação Crítica dos Resultados	28
6. Conclusão	29
7. Referência	31
8. Link do projeto	31

Figuras

Figura 01:

Diagrama de componente da arquitetura atual do cliente _____ 11

Figura 02:

Diagrama de container da proposta de arquitetura _____ 12

Figura 03:

Diagrama de componente da proposta de arquitetura _____ 14

Figura 04:

Endpoint obter pedido por número e empresa da aplicação data-calculate-freight_ 19

Figura 05:

Execução do endpoint obter pedido por número e empresa da aplicação
data-calculate-freight _____ 19

Figura 06:

Configuração de conexão com banco de dados PostgreSQL _____ 22

Figura 07:

Endpoint Health Check da aplicação data-calculate-freight _____ 24

Figura 08:

Execução do endpoint Health Check da aplicação data-calculate-freight _____ 24

Figura 09:

Execução do endpoint Health Check da aplicação data-calculate-freight em caso de
falha _____ 25

Figura 10:

Endpoint Health Check da aplicação calculate-freight _____ 25

Figura 11:

Execução do endpoint Health Check da aplicação calculate-freight _____ 25

Figura 12:

Execução do endpoint Health Check da aplicação calculate-freight em caso de
falha _____ 26

Figura 13:

Endpoint de cálculo de frete da aplicação calculate-freight _____ 28

Figura 14:

Execução do endpoint de cálculo de frete da aplicação calculate-freight _____ 28

Figura 15:

Endpoint de cadastro de pedido com cálculo de frete da aplicação
calculate-freight _____ 28

Figura 16:

Execução do endpoint de cadastro de pedido com cálculo de frete da aplicação
calculate-freight _____ 29

Figura 17:

Criando imagem docker para a aplicação calculate-freight _____ 31

Figura 18:

Criando container docker para a aplicação calculate-freight _____ 31

Figura 19:

Criando imagem docker para a aplicação data-calculate-freight _____ 32

Figura 20:

Criando container docker para a aplicação data-calculate-freight _____ 32

Figura 21:

Execução dos testes unitários da aplicação data-calculate-freight _____ 33

Figura 22:

Execução dos testes unitários da aplicação calculate-freight _____ 33

1. Introdução

Um sistema de gerenciamento de transporte (*TMS*, na sigla em inglês) é uma ferramenta tecnológica utilizada para otimizar e automatizar as operações de transporte de uma empresa, melhorando a eficiência e reduzindo os custos.

Um *TMS* fornece diversos recursos valiosos como relatórios e análises para ajudar a tomar decisões estratégicas, como quais rotas são mais eficientes, quais são os custos sobre o transporte, quais veículos estão precisando de manutenção e como maximizar a capacidade de carregamento.

Permite a integração com outros sistemas, como o de gestão de estoques e de vendas, para garantir que todos os aspectos da cadeia de suprimentos sejam considerados, evitando assim falhas e atrasos no processo de entregas.

Proporciona mecanismos que podem ajudar a rastrear e monitorar as mercadorias enquanto elas estão em trânsito, garantindo que elas cheguem ao destino final em boas condições.

O *TMS* é uma ferramenta valiosa para garantir que as operações de transporte sejam eficientes, rentáveis e que a empresa alcance suas metas de negócios. Garante a precisão e a pontualidade das entregas, melhorando a satisfação do cliente aumentando a competitividade no mercado.

Nessa abordagem o sistema do cliente detém de uma arquitetura monolítica, onde todas as funcionalidades estão integradas e conectadas, consistindo em uma única unidade de *software*. Devido a essas condições está relatando problemas relacionados a performance, escalabilidade e custo.

A aplicação realiza o cálculo de frete de aproximadamente 100 mil pedidos por minuto, executa a conciliação de frete de 80 mil faturamentos em 4 horas, acarretando lentidão nas demais funcionalidades.

Em consequência da sobrecarga os problemas relacionados a essas condições estão afetando o todo, resultando em uma alta complexidade sobre o procedimento de escalamento somados aos custos excessivos, e proporcionando atrasos relacionados à sustentação e entrega de novas funcionalidades.

O sistema foi concebido sendo uma *API Rest*, porém como já dito sendo uma única unidade de *software*, então por mais que proporcione um forma de comunicação, muitas rotinas utilizam da comunicação banco a banco gerando brechas de segurança. Dessa forma, como não houve um estudo durante a

concepção da aplicação, não foi previsto os picos de processamento proporcionando os inúmeros problemas.

Devido aos pontos elencados a melhor alternativa consiste em desmembrar a aplicação utilizando de uma arquitetura de microsserviços que irá permitir uma escalabilidade melhorada, pois cada serviço pode ser escalado independentemente dos outros, o que permite que o cliente adapte sua infraestrutura de acordo com as necessidades do negócio. A flexibilidade é aumentada, pois os microsserviços podem ser desenvolvidos, testados e implantados de forma independente, permitindo que as equipes de desenvolvimento trabalhem de forma mais ágil e eficiente. E a facilidade de manutenção é maior, pois cada serviço é menor e mais simples, o que torna mais fácil identificar e corrigir problemas.

Também será necessário realizar a migração gradual do *SGBD Oracle* para o *PostgreSQL*, reduzindo o custo sobre o armazenamento de dados.

2. Especificação Arquitetural da solução

Após a realização de uma análise da operação do cliente é visto que a aplicação detém de um fluxo sequencial, onde tem influência direta sobre a entrada da venda durante a cotação de frete e consumação da entrega.

1. Cotação de frete, caso aprovado o pedido será cadastrado;
2. Motor de frete, vincula o cálculo do frete ao pedido;
3. Romaneio de embarque com pedidos calculados e aptos a transporte;
4. Inicia o tracking do pedido com o histórico de transporte;
5. Após entregue é realizado o faturamento com o comparativo do cálculo gerado pela transportadora versus o realizado no TMS.
6. Por fim, o usuário do sistema decide qual cálculo será pago.

Depois deste entendimento o cliente relata que os pontos mais cruciais são o cálculo de frete e o faturamento e para essa solução será abordado o cálculo de frete.

2.1. Restrições Arquiteturais

Segue levantamento das restrições arquiteturais posteriormente ao levantamento da estrutura.

ID	Descrição
RA - 01	Utilizar do frontend atual que é um single page application feito em Angular JS na versão 4. O mesmo não está na última versão do framework porém ainda irá suprir os novos backend (APIs).
RA - 02	Prover meios para reduzir custos sobre o armazenamento de dados.
RA - 03	Manter os outros recursos do monolito em funcionamento.
RA - 04	Migrar do ambiente on-premises para AWS cloud, visando mais abrangência de recursos de máquina.
RA - 05	Manter o FTP em funcionamento e conectar a um bucket S3 para armazenar os arquivos Eletronic Data Interchange (EDI), visando economia de disco e melhor gerenciamento.

2.2. Requisitos Funcionais

Segue levantamento dos requisitos funcionais após entrevista com o cliente e demais pessoas envolvidas na operação.

Legenda: B = Baixa, M = Média e A = Alta.

ID	Descrição	Dificuldade (B/M/A)	Prioridade (B/M/A)
RF - 01	Realizar cálculo de frete durante as cotações	M	A
RF - 02	Realizar cálculo de frete para entrada de pedido	M	A
RF - 03	Realizar atualização de cálculo sobre o pedido	M	A
RF - 04	Listar o cálculo de frete realizados para ser consumida pelo ERP	M	A
RF - 05	Consultar o cálculo realizado individualmente	M	A
RF - 06	Listar consultar os pedidos	M	M
RF - 07	Listar os pedidos individualmente e por empresa	M	M
RF - 08	Cadastrar e atualizar empresa referente ao pedido	M	M
RF - 09	Listar, consultar individualmente e por filtro a empresa	M	M
RF - 10	Remover cadastro empresa	M	M

RF - 11	Cadastrar e atualizar transportadora referente ao pedido	M	M
RF - 12	Listar, consultar individualmente e por filtro a transportadora	M	M
RF - 13	Remover cadastro transportadora	M	M
RF - 14	Cadastrar e atualizar rota de frete referente ao pedido	M	M
RF - 15	Listar, consultar individualmente e por filtro a rota de frete	B	M
RF - 16	Remover cadastro rota de frete	B	M
RF - 17	Cadastrar e atualizar range de frete referente ao pedido	B	M
RF - 18	Listar, consultar individualmente e por filtro a range de frete	B	M
RF - 19	Remover cadastro range de frete	B	M
RF - 20	Cadastrar e atualizar tipo de entrega referente ao pedido	B	B
RF - 21	Listar, consultar individualmente e por filtro a tipo de entrega	B	B
RF - 22	Remover cadastro tipo de entrega	B	B

2.3. Requisitos Não-Funcionais

Em meados de 2015 na história do cliente houve uma migração da aplicação que foi construída em *Oracle Forms* para uma única *Spring API*, que devido a essa circunstância não foi configurado um *API Gateway*. Também foi mantido o servidor *on-premises* que detinha do FTP para comunicação de arquivos do tipo *Electronic Data Interchange* (EDI).

Legenda: B = Baixa, M = Média e A = Alta.

ID	Descrição	Dificuldade (B/M/A)	Prioridade (B/M/A)
RNF - 01	Não permitir comunicação banco a banco com o ERP	A	A
RNF - 02	Não utilizar do SGBD Oracle visando redução de custo.	A	A
RNF - 03	Implementar recurso para monitorar as aplicações	M	A
RNF - 04	Criar um único ponto no sistema para realizar as operações de cálculo de frete	M	M
RNF - 05	Docker - containerização das aplicações	M	M
RNF - 06	Implementar testes de unidade	M	A

2.4. Mecanismos Arquiteturais

Segue levantamento dos mecanismos arquiteturais pertinente a solução que será construída.

Análise	Design	Implementação
Persistência	ORM	Hibernate
Frontend	Single Page Application	Angular JS
Backend	Microserviço	Java
Integração	API RESTful	Spring Boot
Log do sistema	No SQL	Mongodb
Teste de sistema	CI (Continuous Integration)	Git Lab
Deploy	CD (Continuous Deploy)	Git Lab

3. Modelagem Arquitetural

Neste capítulo serão apresentados alguns diagramas, que irão demonstrar qual a situação arquitetural e o que será desenvolvido como solução para o cliente. Com esses recursos é possível obter um entendimento do comportamento e fluxo dos sistemas, junto a seus problemas e futuras atualizações.

O diagrama de contexto representa uma visão macro da estrutura, exemplificando o funcionamento da comunicação entre os outros recursos.

No diagrama de container será demonstrado as duas soluções propostas para suprir a necessidade de performance durante o processo de cálculo de frete.

E o diagrama de componente apresenta o detalhamento dos recursos que serão implementados junto a seus comportamentos e objetivos.

3.1. Diagrama de Contexto

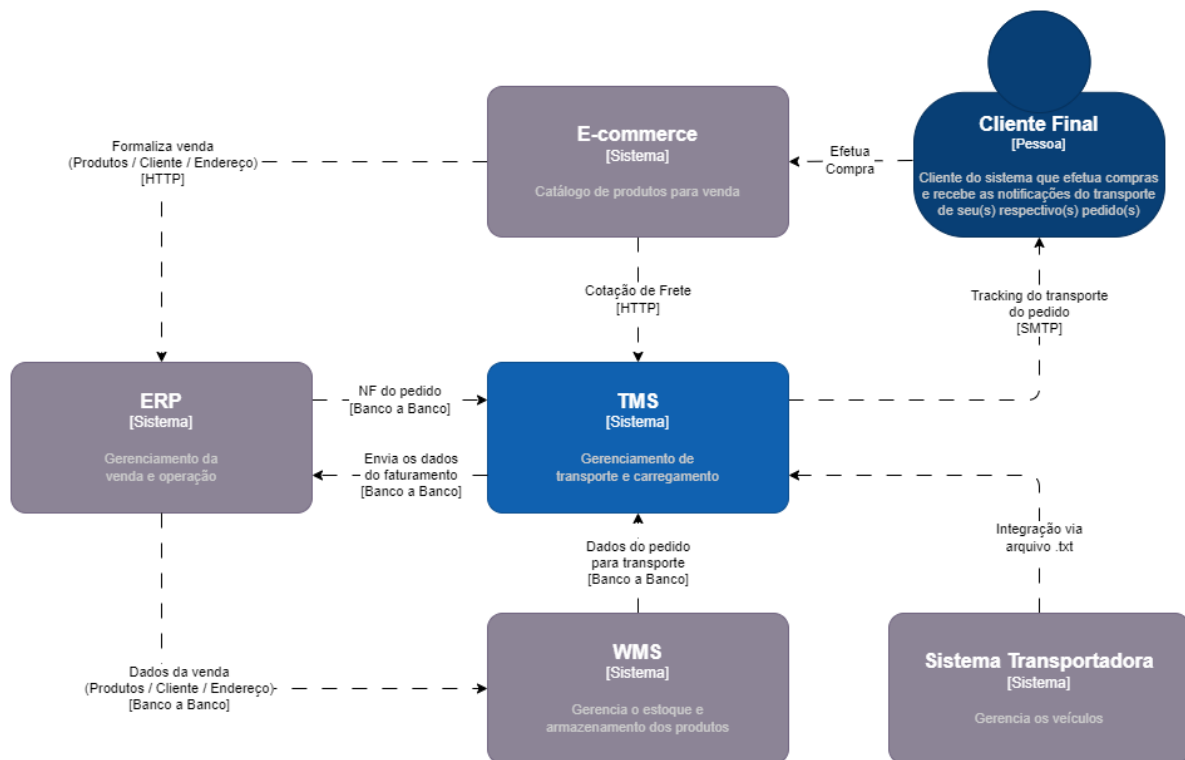


Figura 01: Diagrama de componente da arquitetura atual do cliente.

Na arquitetura apresentada vemos o papel do cliente final sobre o ato de realizar as compras no *e-commerce*, o tráfego dos dados da compra sendo enviados ao ERP para gestão dessas informações e emissão da nota fiscal, a identificação do produto no estoque com suas respectivas características de cubagem e peso no WMS e, a gestão do transporte executada pelo TMS.

Desta maneira é possível compreender que o TMS durante a cadeia de suprimentos, detém de uma influência direta que no início do fluxo é responsável pela cotação de frete e no fim executa a entrega do produto.

3.2. Diagrama de Container

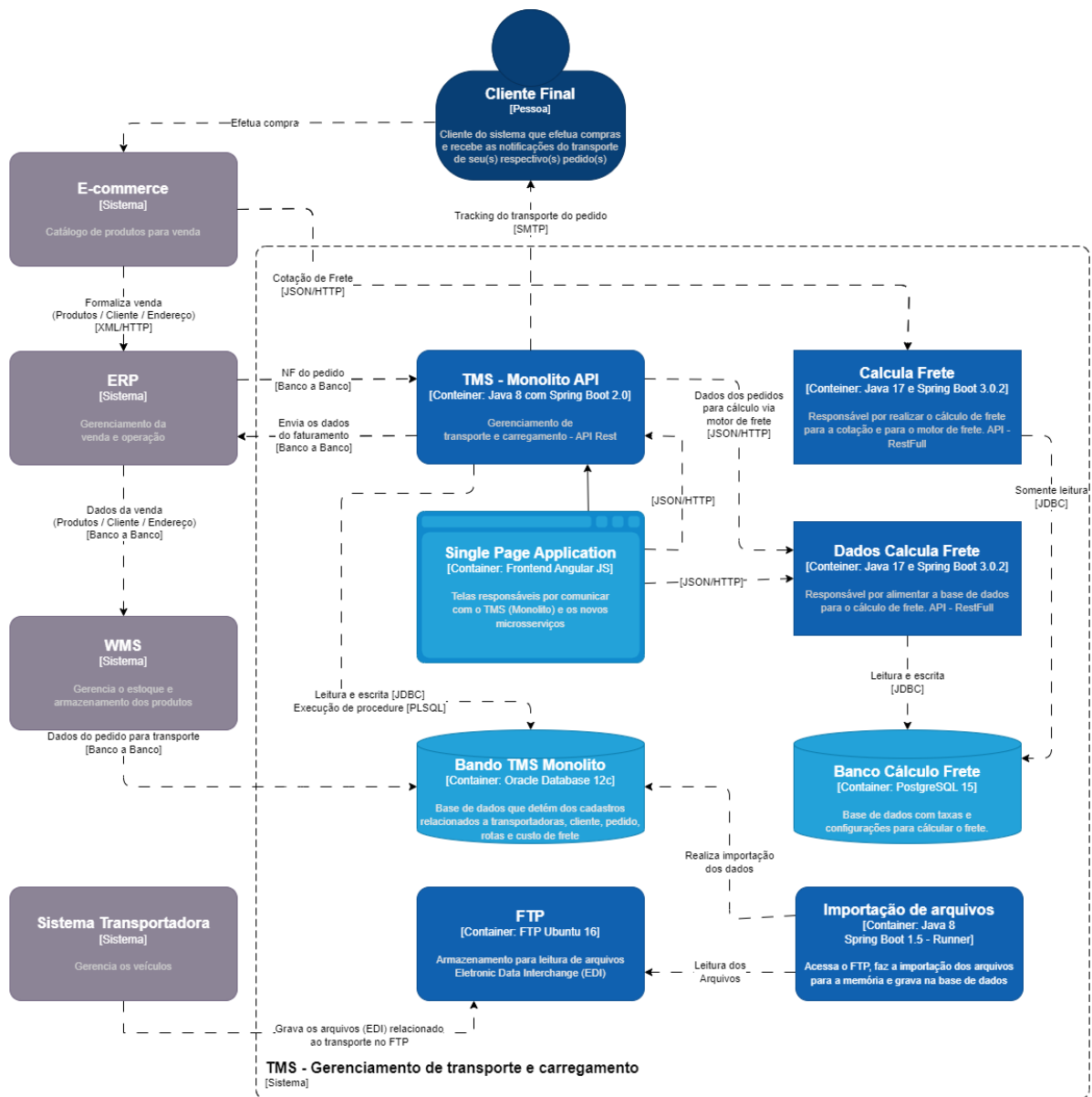


Figura 02: Diagrama de container da proposta de arquitetura.

Nesta arquitetura além dos containers também há uma demonstração parcial do fluxo da operação do cliente.

Em uma visão geral, como já mencionado o e-commerce representa o catálogo de produtos que realiza a cotação e envia os dados das compras ao ERP. O ERP realiza a gestão dessas informações, emite as notas fiscais e identifica o pedido que foi vendido ao WMS. O WMS dispõe de mecanismos para gerir a

quantidade de itens em estoque, validade, e manuseio dos produtos de forma mais performática.

O container TMS - Monolito API no momento ainda é a principal aplicação, como a mesma segue operante não pode ser descontinuada já nesse primeiro momento. Então a maior parte das parametrizações e configurações ainda serão geridas por esse recurso, e gradualmente será fatiado em outros microsserviços.

Junto ao monolito é notado o container de banco de dados do TMS, responsável pelo armazenamento dos dados de transporte, algumas lógicas implementadas em PLSQL e que no momento ainda está ativo a comunicação banco a banco com o WMS, posteriormente esse formato de integração será removido.

Além do WMS também é visto o *container* Importação de Arquivo, que tem a função de ler os arquivos .txt que estão no container FTP inseridos pelo container do sistema da transportadora. Esse é respectivo a integração de arquivos posicionais seguindo o formato *Electronic Data Interchange* (EDI).

O container *Single Page Application* (SPA), são as telas que consomem os endpoints do TMS monolito e do container Dados Calcula Frete. O Dados Calcula Frete terá o objetivo de ser a camada que irá persistir os dados de cálculo de frete na base. Ao lado deste container o Calcula Frete será o recurso responsável exclusivamente por realizar os cálculos quando necessário.

3.3. Diagrama de Componentes

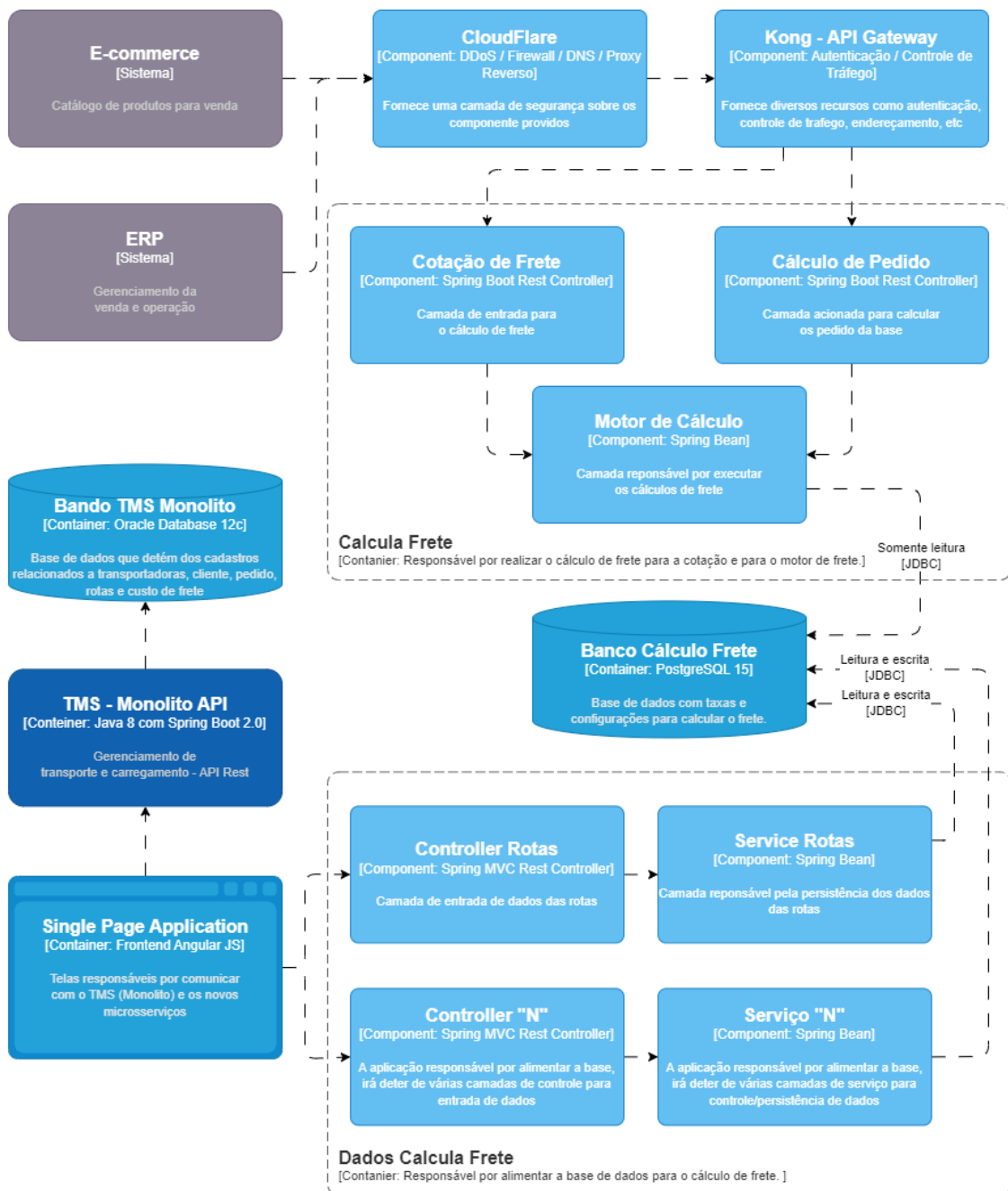


Diagrama de componente

Provê detalhamento sobre as aplicações Calcula de Frete e Dados Calcula Frete

Figura 03: Diagrama de componente da proposta de arquitetura.

No diagrama de componentes é visto algumas aplicações que fazem parte do ecossistema do cliente como por exemplo o e-commerce que tem como objetivo ser o catálogo de produtos que irá prover as vendas.

O SPA (*Single Page Application*) que detém de todas as telas e efetua a comunicação e consumo das APIs providas pelo monolito e os demais microsserviços que serão criados.

E nesta estrutura também é notável alguns pontos interessantes, como por exemplo a utilização do *CloudFlare*. Esse é um serviço terceiro, contratado com o objetivo de ser uma camada de segurança, impedindo ataques como DDoS a partir de suas políticas e configurações. E com sua adesão também será possível ter um melhor controle das requisições que realmente devem alcançar as aplicações, fazendo um melhor controle da *cloud* e de seus respectivos custos.

Em relação ao API Gateway o recurso escolhido foi o *Kong*, uma aplicação *Open Source* que proporciona inúmeros recursos com a instalação de alguns *plugins*, tais como autenticação, autorização, controle de tráfego que provê uma orquestração de requisições que podem ser direcionadas para todas as aplicações ou para uma especificamente.

E os componentes Calcula Frete e Dados Calcula Frete são as aplicações que serão desacopladas do monolito provendo escalabilidade horizontal e por consequência mais performance.

O Calcula Frete será responsável por deter dois *endpoints* principais, onde um realiza a cotação de frete para atender o e-commerce, e o outro irá executar o cálculo para os pedidos que entraram na base.

Como o componente Calcula Frete detém basicamente a responsabilidade de executar os cálculos, o Dados Calcula Frete é um conjunto de APIs dotada do objeto de realizar o cadastramento das informações necessárias para realizar os cálculos, como rotas, faixas de peso e valor, transporte entre outros. Deste modo promove uma camada resiliente sobre o fluxo de realizar ou configurar o cálculo, permitindo o escalonamento direcionado conforme for necessário.

4. Avaliação da Arquitetura (ATAM)

Com base no artigo Method for Architecture Evaluation publicado em agosto de 2000 pelos autores Rick Kazman, Mark Klein e Paul Clements, o ATAM (*Architecture Tradeoff Analysis Method*) é uma técnica voltada para arquitetura de *software* que tem o objetivo identificar e avaliar os atributos de qualidade levantando

os possíveis *trade-offs*, permitindo a tomada de decisão sobre o negócio e restrições técnicas.

4.1. Análise das abordagens arquiteturais

Segue levantamento dos atributos de qualidade respectivos aos requisitos não funcionais elencados anteriormente.

Legenda: B = Baixa, M = Média e A = Alta.

Atributos	Cenários	Importância	Complexidade
Interoperabilidade	1 - O sistema deve prover um meio de comunicação para o ERP consultar os pedidos calculados.	M	A
Interoperabilidade	2 - O sistema não deve utilizar o SGBD Oracle para salvar os dados.	A	A
Monitoramento	3 - O sistema deve prover meios para realizar o monitoramento do mesmo.	A	M
Manutenibilidade	4 - O sistema deve prover um único ponto para realizar as operações de cálculo de frete.	A	M
Portabilidade	5 - A aplicação deve ser capaz de ser executada em container.	M	M
Qualidade	6 - O sistema deve ter testes unitários de suas rotinas.	M	M

4.2. Cenários

Na listagem abaixo é demonstrado o detalhamento dos cenários especificados conforme os atributos de qualidade.

Cenário 1 - Interoperabilidade: Para o ERP estar coeso com as informações de cálculo o mesmo deve conseguir realizar uma chamada via HTTP *RESTful* e obter os dados de um produto X da empresa Y.

Cenário 2 - Interoperabilidade: Independente da quantidade de aplicações que serão criadas, as mesmas não devem fazer uso do SGBD *Oracle* para realizar o armazenamento de dados. Deste modo se faz necessário adotar um outro SGBD, como por exemplo o *PostgreSql*.

Cenário 3 - Monitoramento: Para realizar o monitoramento do sistema o mesmo deve fornecer algum recurso que consiga determinar que o mesmo está em funcionamento.

Cenário 4 - Manutenibilidade: Ao avaliar o monolito do cliente é possível identificar que o cálculo de frete é realizado em duas APIs diferentes e também via banco de dados quando um novo pedido é cadastrado. Por tanto, o sistema deve possuir um único ponto para realizar o cálculo de frete, reduzindo a redundância de código e problemas de inconsistência.

Cenário 5 - Portabilidade: Para facilitar o *deploy* nos ambientes de desenvolvimento, qualidade e produção, o sistema deve ter a capacidade de ser executado em formato de *container*.

Cenário 6 - Qualidade: Para assegurar que os processos estão sendo executados corretamente, suprimindo a necessidade de engenharia e negócio, o sistema deve possuir testes unitários sobre as rotinas implementadas.

4.3. Evidências da Avaliação

Os quadros a seguir provêm o detalhamento das evidências da avaliação para cada cenário, respectivamente e, após os quadros segue os testes realizados.

Atributo de qualidade:	Interoperabilidade
Requisito de qualidade:	O sistema deve prover um meio de comunicação para o ERP consultar os pedidos calculados.
Preocupação:	
O sistema não deve permitir comunicação banco a banco.	
Cenário 1:	
Para o ERP estar coeso com as informações de cálculo o mesmo deve conseguir realizar uma chamada via HTTP RESTful e obter os dados de um produto X da empresa Y.	
Ambiente:	
Sistema operando sem falhas	
Estímulo:	
O sistema recebe uma requisição em um serviço REST.	
Mecanismo:	
Criar um serviço REST para atender às requisições do ERP sobre obter o cálculo realizado aos pedidos	
Medida de resposta:	
Retornar os dados requeridos no formato JSON.	
Considerações sobre a arquitetura:	
Riscos:	A instabilidade na rede, na aplicação ou na base de dados poderá ocasionar lentidão ou perda de pacotes.

Pontos de sensibilidade:	Não há.
Trade-off:	Não há.

Data Calculate Freight

[Base URL: localhost:8080/]

https://localhost:8080/v2/api-docs?group=Version_1

Aplicação responsável por alimentar a base de cálculo de frete.

calculation-freight-controller	Calculation Freight Controller	>
calculation-type-range-freight-controller	Calculation Type Range Freight Controller	>
company-controller	Company Controller	>
freight-route-controller	Freight Route Controller	>
health-check-controller	Health Check Controller	>
orders-controller	Orders Controller	▼
GET	/orders/v1/ Pedidos listagem	
GET	/orders/v1/{id} Pedido por id	
GET	/orders/v1/order/{orderNumber}/company/{companyId} Pedido por numero e empresa	

Figura 04: Endpoint obter pedido por número e empresa da aplicação data-calculate-freight

Curl

```
curl -X GET "http://localhost:8080/orders/v1/order/1/company/1" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/orders/v1/order/1/company/1
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "company": { "id": 1, "name": "Melissa e Renato Financeira ME", "document": "33662514000161", "postalCode": "13952110", "active": false, "dateUpdate": "11/03/2023 02:31:09" }, "calculationFreight": { "id": 2, "destinyPostalCode": "14000001", "width": 3, "height": 4, "length": 3, "cubage": 36, "weight": 2, "freightValue": 2.2, "dateCreate": "08/05/2023 04:40:51", "dateUpdate": "10/05/2023 09:01:57" }, "number": "1", "dateUpdate": "10/05/2023 09:02:11" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Wed, 24 May 2023 11:52:14 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Figura 05: Execução do endpoint obter pedido por número e empresa da aplicação data-calculate-freight

Uma má prática da arquitetura atual do cliente é manter integração banco a banco entre alguns sistemas.

E fazendo uso desse formato de integração podemos ter vários problemas, como por exemplo executar consultas complexas e pesadas ferindo com o desempenho do sistema, sincronização frequente de dados sobrecarrega a rede negativamente. Caso haja algum problema em um banco ao atualizar algum campo corretamente, irá gerar inconsistência dos dados.

Por fim, caso as configurações de tráfego de dados não estiverem adequadas, manter a comunicação banco a banco proporciona abertura para falhas de segurança e privacidade.

Dessa forma, implementar o *endpoint* para estabelecer a comunicação entre o TMS e o ERP proporciona uma comunicação segura e consistente.

Atributo de qualidade:	Interoperabilidade
Requisito de qualidade:	O sistema não deve utilizar o SGBD Oracle para salvar os dados.
Preocupação:	
Redução de custo somado a cultura da empresa.	
Cenário 2:	
Independente da quantidade de aplicações que serão criadas, as mesmas não devem fazer uso do SGBD Oracle para realizar o armazenamento de dados. Deste modo se faz necessário adotar um outro SGBD, como por exemplo o PostgreSQL.	
Ambiente:	
Sistema operando sem falhas.	
Estímulo:	
O sistema recebe um montante de dados para realizar persistência na base.	

Mecanismo:	
Configurar a aplicação para se comunicar e fazer uso do SGBD PostgreSQL.	
Medida de resposta:	
Retornar status 200 ao realizar cadastros ou operações a partir dos serviços REST que executam procedimentos DML.	
Considerações sobre a arquitetura:	
Riscos:	Instabilidade da base de dados
Pontos de sensibilidade:	Confiabilidade
Trade-off:	Manutenibilidade: Ao realizar a adesão do SGBD PostgreSQL irá implicar em redução de custos, mas sempre será preciso realizar um planejamento sobre os objetos do banco, porque poderá ocasionar em falta de performance sobre as operações do mesmo.

```

1  spring:
2    mvc:
3      pathmatch:
4        matching-strategy: ant-path-matcher
5    flyway:
6      enabled: true
7      baseline-on-migrate: true
8    datasource:
9      hikari:
10       connection-test-query: SELECT 1
11       minimum-idle: 1
12       maximum-pool-size: 10
13       auto-commit: false
14       connection-timeout: 3000
15   → url: "jdbc:postgresql://localhost:5432/data_calculate_freight"
16       username: postgres
17       password: 1234
18       type: com.zaxxer.hikari.HikariDataSource
19    jpa:
20      properties:
21        hibernate:
22          enable_lazy_load_no_trans: true
23          show_sql: false
24        jdbc:
25          lob:
26            non_contextual_creation: true
27   → database-platform: org.hibernate.dialect.PostgreSQL9Dialect

```

Figura 06: Configuração de conexão com banco de dados *PostgreSQL*

Fazendo a troca do SGBD *Oracle* para *PostgreSQL* o cliente será capaz de reduzir custos sobre o armazenamento de dados, tendo uma margem maior sobre a escalabilidade do banco caso necessário, mais facilidade de uso e flexibilidade já que o *PostgreSQL* é um dos bancos mais usados na atualidade.

Segue os links dos arquivos de configuração da base das aplicações:


- [data-calculate-freight](#)
- [calculate-freight](#)

Atributo de qualidade:	Monitoramento
Requisito de qualidade:	O sistema deve prover um meio para realizar o monitoramento do mesmo.
Preocupação:	
Saber que o sistema está ativo e em pleno funcionamento.	
Cenário 3:	
Para realizar o monitoramento do sistema o mesmo deve fornecer algum recurso que consiga determinar que o mesmo está em funcionamento.	
Ambiente:	
Sistema operando sem falhas.	
Estímulo:	
O sistema recebe uma requisição em um serviço REST para realizar o monitoramento.	
Mecanismo:	
Criar um serviço REST que irá receber requisições em um intervalo X para atender o monitoramento.	
Medida de resposta:	
Retornar status 200 e com mensagem de sucesso em objeto JSON e retornar	

status 500 quando houver falha.


Considerações sobre a arquitetura:

Riscos:	A instabilidade na rede, na aplicação ou na base de dados poderá ocasionar lentidão ou perda de pacotes.
Pontos de sensibilidade:	Disponibilidade/estabilidade do recurso de AMP.
Trade-off:	Latência: Baixa latência pode fornecer informações quase instantâneas, porém obter resultados precisos em tempo real pode exigir algoritmos de processamento complexos.

Data Calculate Freight 

[Base URL: localhost:8080/]
<http://localhost:8080/v2/api-docs?group=Version 1>

Aplicação responsável por alimentar a base de cálculo de frete.

- calculation-freight-controller** Calculation Freight Controller >
- calculation-type-range-freight-controller** Calculation Type Range Freight Controller >
- company-controller** Company Controller >
- freight-route-controller** Freight Route Controller >
- health-check-controller** Health Check Controller  >

GET /health-check healthCheck

Figura 07: Endpoint *Health Check* da aplicação data-calculate-freight

GET /health-check healthCheck

Parameters Cancel

No parameters

Execute Clear

Responses Response content type application/json


Curl

```
curl -X GET "http://localhost:8080/health-check" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/health-check
```

Server response

Code	Details
200 	<p>Response body</p> <pre>{ "status": "Ok" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Wed, 24 May 2023 11:04:02 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Download

Figura 08: Execução do endpoint *Health Check* da aplicação data-calculate-freight

TMS - Transport Management System

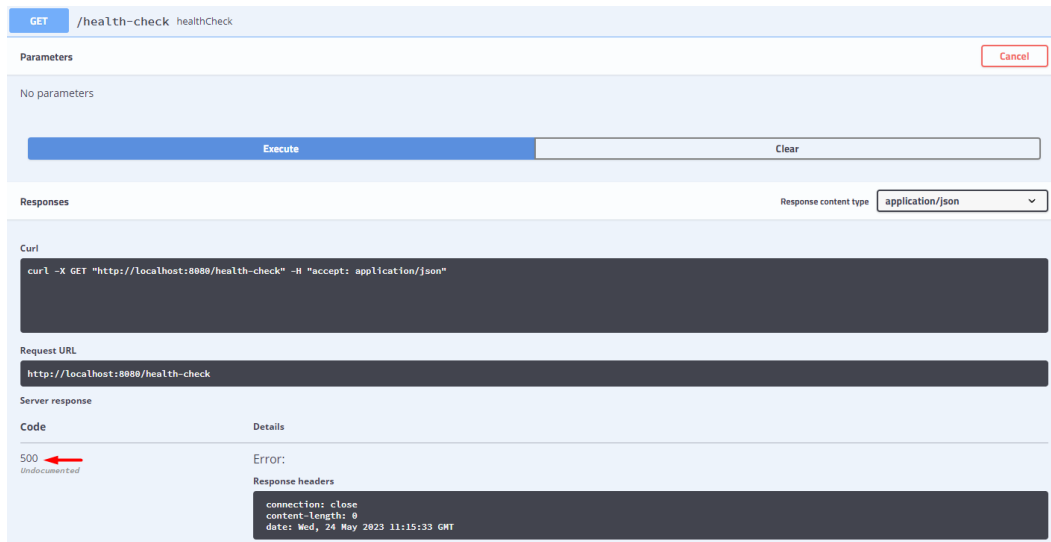


Figura 09: Execução do endpoint *Health Check* da aplicação data-calculate-freight em caso de falha

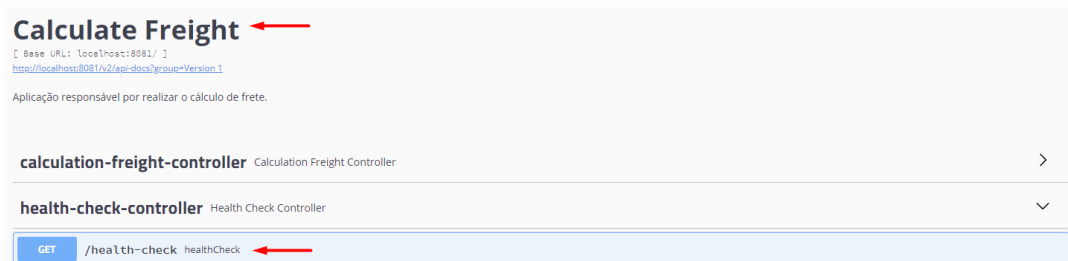


Figura 10: Endpoint *Health Check* da aplicação calculate-freight

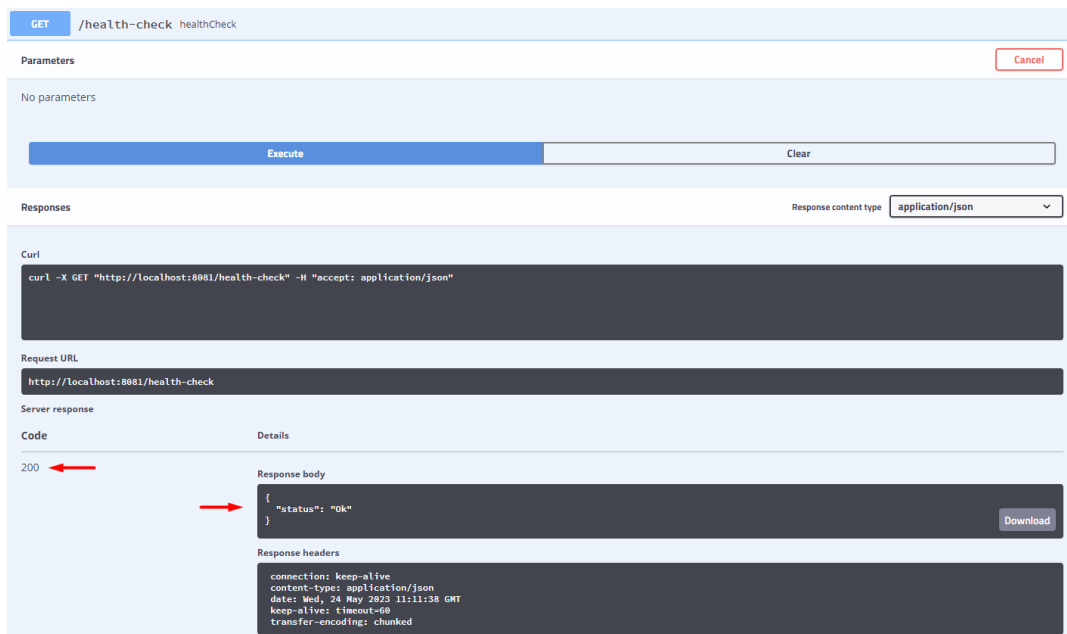


Figura 11: Execução do endpoint *Health Check* da aplicação calculate-freight

GET /health-check healthCheck

Parameters Cancel

No parameters

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X GET "http://localhost:8081/health-check" -H "accept: application/json"
```

Request URL

```
http://localhost:8081/health-check
```

Server response

Code	Details
500 <i>Undocumented</i>	<p>Error:</p> <p>Response headers</p> <pre>connection: close content-length: 0 date: Wed, 24 May 2023 11:17:41 GMT</pre>

Figura 12: Execução do endpoint *Health Check* da aplicação calculate-freight em caso de falha.

Com a implementação do endpoint *Health Check* para ambas aplicações será possível verificar a integridade das mesmas, permitindo que sistemas externos de AMP como por exemplo *New Relic*, *Grafana* e outros monitorem a saúde das APIs.

Este recurso permite o monitoramento de disponibilidade, ao verificar periodicamente o *status*, obtendo informações se as aplicações estão respondendo corretamente a partir de alertas de integridade, acionando rapidamente a equipe de sustentação.

Também é possível utilizar este meio para configurar o balanceamento de carga, permitindo um melhor controle e direcionamento do tráfego, avaliando os nós do aplicativo e garantindo que o usuário final receba respostas corretas sem gerar sobrecarga em nós indisponíveis.

Um outro recurso é empregar esse endpoint para realizar monitoramento de dependências críticas, como por exemplo o banco de dados, que é a forma que foi implementado para as APIs. Isso ajuda a identificar situações indesejadas logo no início, prevenindo que se tornem graves problemas.

Atributo de qualidade:	Manutenibilidade
Requisito de qualidade:	O sistema deve prover um único ponto para realizar as operações de cálculo de frete.

Preocupação:	
Deter de um único ponto de cálculo, que irá atender toda a aplicação se ocasionar inconsistência dos dados obtidos.	
Cenário 4:	
Ao avaliar o monolito do cliente é possível identificar que o cálculo de frete é realizado em duas APIs diferentes e também via banco quando um novo pedido é cadastrado. Por tanto, o sistema deve possuir um único ponto para realizar o cálculo de frete, reduzindo a redundância de código e problemas de inconsistência.	
Ambiente:	
Sistema operando sem falhas	
Estímulo:	
O sistema recebe uma requisição em um serviço REST para realizar o cálculo.	
Mecanismo:	
Criar um serviço REST que irá receber os dados necessários para realizar o cálculo de pedidos e cotação de frete.	
Medida de resposta:	
Retornar os dados requeridos no formato JSON.	
Considerações sobre a arquitetura:	
Riscos:	A instabilidade na rede, na aplicação ou na base de dados poderá ocasionar lentidão ou perda de pacotes.
Pontos de sensibilidade:	Legibilidade do Código: A complexidade pode provocar falta de legibilidade de código.
Trade-off:	Complexidade: A flexibilidade das rotinas poderá

ocasionar em mais complexidade.

Calculate Freight

[Base URL: localhost:8081/]
<http://localhost:8081/v2/api-docs?group=Version.1>

Aplicação responsável por realizar o cálculo de frete.

calculation-freight-controller Calculation Freight Controller

POST /calculation-freight/v1/ Cálculo de frete

Figura 13: Endpoint de cálculo de frete da aplicação calculate-freight.

Curl

```
curl -X POST "https://localhost:8081/calculation-freight/v1/" -H "accept: application/json" -H "Content-Type: application/json" -d '{"company": 1, "destinyPostalCode": "14080000", "height": 1, "length": 1.5, "weight": 0.5, "width": 1}'
```

Request URL

http://localhost:8081/calculation-freight/v1/

Server response

Code	Details
201	<p>Response body</p> <pre>{ "id": 3, "shippingCompany": { "id": 1, "name": "JP Log ME", "document": "88449952000138", "active": true, "dateCreate": "22/03/2023 10:15:34" }, "destinyPostalCode": "14080000", "width": 1, "height": 1, "length": 1.5, "cubage": 1.5, "weight": 0.5, "freightValue": 2, "dateCreate": "24/05/2023 08:54:09", "typeDelivery": "EXPRESS", "deliveryDay": "03/06/2023" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Wed, 24 May 2023 23:54:10 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Figura 14: Execução do endpoint de cálculo de frete da aplicação calculate-freight.

Calculate Freight

[Base URL: localhost:8081/]
<http://localhost:8081/v2/api-docs?group=Version.1>

Aplicação responsável por realizar o cálculo de frete.

calculation-freight-controller Calculation Freight Controller

health-check-controller Health Check Controller

orders-controller Orders Controller

POST /orders/v1/ Cadastro pedido

Figura 15: Endpoint de cadastro de pedido com cálculo de frete da aplicação calculate-freight.

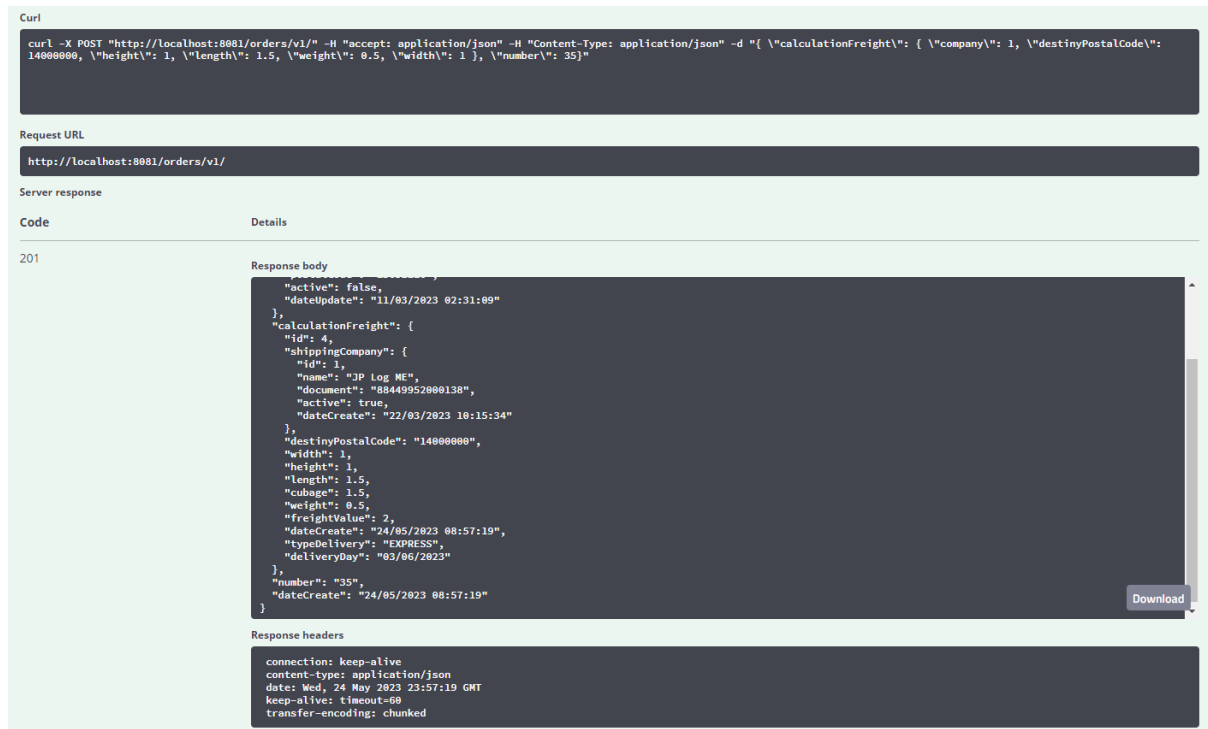


Figura 16: Execução do endpoint de cadastro de pedido com cálculo de frete da aplicação calculate-freight.

Na estrutura atual da aplicação há vários pontos que detêm a responsabilidade de executar o cálculo de frete, provocando que ao implementar as evoluções ou sustentação das rotinas, sempre se faz necessário replicar as modificações em todos os pontos.

No formato que foi construído o cálculo sempre será executado pelo *endpoint [POST] /calculation-freight/v1/* representado na figura 13. Quando o pedido já existir na base do ERP, utilizar do endpoint *[POST] /orders/v1/* mostrada na figura 15, que tem o papel de gerenciar os dados do pedido na base do TMS e também executar o cálculo através da rotina do *endpoint [POST] /calculation-freight/v1/*.

Atributo de qualidade:	Portabilidade
Requisito de qualidade:	A aplicação deve ser capaz de ser executada em container.
Preocupação:	
A não utilização de containers pode provocar falhas de configuração da aplicação sobre o sistema operacional do servidor.	
Cenário 5:	
Para facilitar o deploy nos ambientes de desenvolvimento, qualidade e produção. O sistema deve ter a capacidade de ser executado em formato de container.	
Ambiente:	
Sistema operando sem falhas.	
Estímulo:	
Ao executar um comando docker o container deve ser criado.	
Mecanismo:	
Criar o arquivo DockerFile na raiz da aplicação com o script de criação de container.	
Medida de resposta:	
Criar um container docker	
Considerações sobre a arquitetura:	
Riscos:	DockerFile do projeto de forma inadequada.
Pontos de sensibilidade:	Gerenciamento de Configuração: DockerFile do projeto de forma inadequada.
Trade-off:	Não há

Fazendo uso de *containers docker* as aplicações da solução ou o monolito terão um ganho considerável relacionado a portabilidade, devido o docker ser independente de sistemas operacionais. Os containers também proporcionar um isolamento dos recursos para as aplicações, facilidade na configuração bastando criar um arquivo *Dockerfile* com os parâmetros necessários, proporciona escalabilidade ao replicar o mesmo container diversas vezes, controle de versão, um gerenciamento mais simplificado e eficiência de recursos porque o kernel do sistema operacional é compartilhado para todas as instâncias, tornando mais eficiente em comparação a outros formatos de virtualização.

E sobre as aplicações desenvolvidas para fazer uso dos containers basta executar os comandos da listagem:

```
> docker build -t <image_name> .
> docker run -p 8080:8080 --name=<container_name> --network=<network_name>
<image_name>
```

```
$ docker build -t cf-img .
[+] Building 3.0s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 181B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:17
=> [auth] library/openjdk:pull token for registry-1.docker.io
=> CACHED [1/2] FROM docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb7d8
=> [internal] load build context
=> => transferring context: 45.70MB
=> [2/2] COPY build/libs/*.jar calculate-freight.jar
=> exporting to image
=> => exporting layers
=> => writing image sha256:83f1f59d61a7b664b05b464191b409ccfd4b913b4116d142b983501e8b6b4dd0
=> => naming to docker.io/library/cf-img
```

Figura 17: Criando imagem docker para a aplicação calculate-freight.

```
$ docker run -p 8081:8081 --name=cf-app --network=database-docker_postgres-compose-network cf-img
:: Spring Boot :: (v2.7.1)

2023-05-31 22:16:50.380 INFO 1 --- [main] b.c.j.c.CalculateFreightApplication : Starting CalculateFreightApplication using Java 17.0.2 on d156b332de71 with PID 1 (/calculate-freight.jar started by root in /)
2023-05-31 22:16:50.383 INFO 1 --- [main] b.c.j.c.CalculateFreightApplication : No active profile set, falling back to 1 default profile: "default"
2023-05-31 22:16:51.456 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-05-31 22:16:51.547 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 77 ms. Found 8 JPA repository interfaces.
2023-05-31 22:16:52.275 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8081 (http)
2023-05-31 22:16:52.286 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-05-31 22:16:52.287 INFO 1 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.64]
2023-05-31 22:16:52.362 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-05-31 22:16:52.363 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1985 ms
2023-05-31 22:16:52.494 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-05-31 22:16:52.716 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-05-31 22:16:52.781 INFO 1 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2023-05-31 22:16:52.842 INFO 1 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.6.9.Final
2023-05-31 22:16:53.009 INFO 1 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2023-05-31 22:16:53.112 INFO 1 --- [main] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.PostgreSQL9Dialect
2023-05-31 22:16:53.889 INFO 1 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-05-31 22:16:53.908 INFO 1 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-05-31 22:16:55.342 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http) with context path ''
2023-05-31 22:16:55.451 INFO 1 --- [main] b.c.j.c.CalculateFreightApplication : Started CalculateFreightApplication in 6.829 seconds (JVM running for 6.884)
2023-05-31 22:17:07.092 INFO 1 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-05-31 22:17:07.093 INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-05-31 22:17:07.096 INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
```

Figura 18: Criando container docker para a aplicação calculate-freight.

```
$ docker build -t dcf-img .
[+] Building 1.8s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 191B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:17
=> [internal] load build context
=> => transferring context: 46.64MB
=> [1/2] FROM docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb7d8
=> CACHED [2/2] COPY build/libs/*.jar data-calculate-freight.jar
=> exporting to image
=> => exporting layers
=> => writing image sha256:313816f5d6f15b679d75d45bc320cca3802ea958b1ebc4ef4897d20c02d425da
=> => naming to docker.io/library/dcf-img
```

Figura 19: Criando imagem docker para a aplicação data-calculate-freight.

```
$ docker run -p 8080:8080 --name=dcf-app --network=database-docker_postgres-compose-network dcf-img
:: Spring Boot :: (v2.7.10-SNAPSHOT)

2023-05-31 22:03:47.497 INFO 1 --- [main] b.c.j.d.DataCalculateFreightApplication : Starting DataCalculateFreightApplication us
)
2023-05-31 22:03:47.500 INFO 1 --- [main] b.c.j.d.DataCalculateFreightApplication : No active profile set, falling back to 1 de
2023-05-31 22:03:48.605 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories
2023-05-31 22:03:48.707 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in
2023-05-31 22:03:49.510 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2023-05-31 22:03:49.523 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-05-31 22:03:49.523 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2023-05-31 22:03:49.617 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplication
2023-05-31 22:03:49.618 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2023-05-31 22:03:49.791 INFO 1 --- [main] o.f.c.internal.license.VersionPrinter : Flyway Community Edition 8.5.13 by Redgate
```

Figura 20: Criando container docker para a aplicação data-calculate-freight.

Atributo de qualidade:	Qualidade
Requisito de qualidade:	O sistema deve ter testes unitários de suas rotinas.
Preocupação:	
Prover coesão sobre as rotinas do sistema e garantir o funcionamento das mesmas com qualidade.	
Cenário 6:	
Para assegurar que os processos estão sendo executados corretamente, suprimindo a necessidade de engenharia e negócio, o sistema deve possuir testes unitários sobre as rotinas implementadas.	
Ambiente:	

Sistema operando sem falhas.	
Estímulo:	
O sistema deve ser capaz de executar testes de unidade sobre suas rotinas.	
Mecanismo:	
Criar testes unitários para as rotinas do sistema	
Medida de resposta:	
Com as rotinas codificadas de forma correta junto aos testes de unidade, assim que executado, os testes devem retornar com sucesso.	
Considerações sobre a arquitetura:	
Riscos:	Criar testes onde o resultado possa representar falsos positivos.
Pontos de sensibilidade:	Confiabilidade sobre o resultado obtido durante a execução dos testes.
Trade-off:	Desempenho: Conforme o sistema for evoluindo os testes de unidade irão aumentar, que por consequência poderá haver impacto sobre a eficiência do recurso.

```

✓ Tests passed: 72 of 72 tests – 1 sec 428 ms
Starting Gradle Daemon...
Gradle Daemon started in 1 s 780 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
BUILD SUCCESSFUL in 12s
4 actionable tasks: 1 executed, 3 up-to-date
22:12:09: Execution finished ':test'.

```

Figura 21: Execução dos testes unitários da aplicação data-calculate-freight.

```

✓ Tests passed: 18 of 18 tests – 1 sec 28 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
BUILD SUCCESSFUL in 6s
4 actionable tasks: 1 executed, 3 up-to-date
22:27:04: Execution finished ':test'.

```

Figura 22: Execução dos testes unitários da aplicação calculate-freight.

Durante a codificação das aplicações foram implementados um total de 90 testes unitários, sendo 72 para data-calculate-freight e 18 para calculate-freight.

Com essa abordagem é possível assegurar a consistência da aplicação, validando os fluxos de persistência de dados e regras de negócio, visando qualidade de código e facilitando a manutenção dos sistemas.

5. Avaliação Crítica dos Resultados

Após o detalhamento da arquitetura no decorrer deste relatório técnico, segue o levantamento de alguns pontos positivos e negativos dessa abordagem, demonstrando quais serão os benefícios ao utilizar da arquitetura de microserviços e qual será o impacto na utilização dessa estrutura.

Legenda: P = Positivo e N = Negativo.

Ponto	Contexto	Descritivo
P	Microserviço	Sustentação/evolução seja realizada de forma isolada por serviço.
P	Microserviço	Menos impacto durante as entregas/deploy.
P	Microserviço	Permite o escalonamento de recurso de máquina de forma mais assertiva.
P	Microserviço	Aplicação dedicada a um único contexto/objetivo.
P	Microserviço	Testes sendo realizados a nível granular.
P	Segurança	Comunicação realizada somente via API Rest, sem fazer uso de integrações banco a banco.
P	Custos	Redução de custos com armazenamento de dados no SGBD PostgreSQL.
N	Gerenciamento	Aumento gradativo da complexidade de gerenciamento dos serviços.
N	Gerenciamento	Aumento gradativo de monitoramento dos componentes da arquitetura.

N	Microserviço	Aumento no tráfego de rede interno das aplicações.
N	Microserviço	Complexidade ao realizar testes no contexto total do ecossistema.
N	Microserviço	Dados isolados por contexto, sendo necessário utilizar de mecanismo para centralizar as informações.
N	Custos	Por mais que o SGBD Oracle proporcione altos custos, o mesmo é uma ferramenta formidável, ainda mais quando houver necessidade de realizar análise de dados para tomada de decisão.

6. Conclusão

No decorrer deste relatório técnico foi executado o levantamento dos problemas do cliente que no momento faz uso de um TMS com arquitetura monolítica, e está tendo dificuldades durante a execução das operações de cálculo de frete, cotação de frete e faturamento.

Somado às questões voltadas à execução, também temos pontos relacionados a custo da aplicação e base de dados, integração banco a banco e, regras desenvolvidas dentro do SGBD.

Após essa análise, a arquitetura contida nesse documento apoia o desmembramento da aplicação, propondo uma solução para o cálculo e cotação de frete utilizando do SGBD *PostgreSQL*.

Deste modo foram implementados duas aplicações sendo APIs RESTful, uma com o foco em armazenamento e gestão dos dados e outra para os cálculos e cotação. As aplicações detêm de testes para apoiar a qualidade de *software* e assegurar a execução das rotinas e regras de negócio de forma coesa. Também foram criados endpoints que provêem um meio de monitorar o funcionamento e a saúde das mesmas.

Durante a execução das validações do projeto alguns *trade-offs* foram encontrados como por exemplo, ao utilizar do SGBD *PostgreSQL* por mais que irá resultar em redução de custo sobre o armazenamento de dados, poderá surgir problemas com a manutenibilidade das aplicações caso não seja realizado um

planejamento adequado sobre os objetos de banco, juntamente a não realizar o desenvolvimento de regras de negócio na própria base de dados.

Com a implementação dos endpoints de *health check* é possível configurar APMs de monitoramento, como o New Relic por exemplo, e obter dados como se as aplicações estão adequadas e sem nenhum problema em suas respectivas operações. Entretanto, por mais que esse recurso ofereça vários ganhos, caso não haja baixa latência, não será possível obter os dados de forma imediata, provocando demora nos resultados e informações incoerentes nos monitoramentos.

Ao observar a aplicação atual do cliente, o cálculo e cotação de frete são realizados em vários pontos, e quando há necessidade de realizar uma atualização o trabalho se torna complexo, porque são vários lugares que será preciso executar o ajuste. Uma alternativa para corrigir essas situações e tornar a gestão do produto mais produtiva é centralizar todas as operações que detêm de similaridade em um único local, aplicando assim o reaproveitamento de código.

Mas considerando todas as possibilidades de cálculo de frete, que por mais que existem as taxas padrões, as mesmas podem ser atualizadas ou até mesmo acrescentado novos formatos de cobrança, portanto com a evolução dos sistemas poderá haver problemas de complexidade provocando que a flexibilidade do algoritmo torne o fluxo de manutenção complicado.

Para apoiar as regras de negócio e plenitude dos processos, o indicado é utilizar de testes, que executam a validação de cada parte do código, seja ele o menor possível ou até mesmo do sistema por completo, abordando todos os andares da pirâmide de teste. Mas no decorrer da evolução das aplicações que implica no crescimento das mesmas, os testes e validações irão tender a crescer em paralelo, e se não houver um planejamento sobre essa evolução, poderá ocasionar em problema de desempenho, no ato da execução dos testes e autenticação que tudo está correto.

Ao longo da concretização deste relatório se torna nítido o aprendizado relacionado aos pontos correlatos ao método de levantar e avaliar uma arquitetura de software. Com a adesão da arquitetura de microsserviços o processo de desenvolvimento e planejamento das aplicações sobre a separação de responsabilidade, no qual os serviços irão deter funções específicas, análogo a identificar os pontos de acoplamento e coesão, e determinar onde serão os limites de contexto.

Por mais que os microsserviços ofereçam várias vantagens, essa estratégia tem de ser aderente às necessidades do cliente e do sistema, porque a arquitetura monolítica também tem seus benefícios, simplicidade, complexidade operacional e facilidade de implementação por exemplo.

Os principais fatores para avaliar a melhor estratégia estão relacionados ao orçamento do cliente e qual o objetivo do sistema, levando sempre em conta que no futuro o objetivo pode mudar e, então, fazer análises e relatórios técnicos similares a esse.

E como próximos passos, pode ser implementado o desenvolvimento dos microsserviços do segmento de faturamento, que irão ser responsáveis por processar a conciliação de frete.

No repositório que está associado a este material tem o levantamento total da proposta arquitetural juntamente com um fluxograma do TMS.

7. Referência

Cássio Luís Rodrigues, Uma Proposta de Metodologia para Implementação de Arquiteturas de Microsserviços, 2018.

Paulo Henrique Monteiro Borba, Design de Arquiteturas de Microsserviços: Uma Revisão Sistemática, 2017.

Rafael Serapilha Durelli, Desafios na Migração de uma Arquitetura Monolítica para uma Arquitetura de Microsserviços, 2018.

F. D. Santana, Um estudo sobre o uso de sistemas TMS no formato embarcador: Uma análise das funcionalidades e benefícios, 2019.

Rick Kazman, Mark Klein e Paul Clements, ATAM: Method for Architecture Evaluation, 08/2000.

8. Link do projeto

Ao acessar o link do projeto, será direcionado a um repositório do *Git Hub*. No mesmo há alguns diretórios com os diagramas, o link dos repositórios das aplicações desenvolvidas e com o descritivo da arquitetura total junto a um fluxograma do TMS.

<https://github.com/edipsilva/puc-minas-arquitetura-de-software>

9. Vídeos

Fase 1:

1. Opção: [link](#)
2. Opção: [link](#)

Fase 2:

1. Opção: [link](#)
2. Opção: [link](#)