

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Arquitetura de Software Distribuído

Paulo Roberto Gonçalves

SISTEMA DE GESTÃO DE SERVIÇOS DE LOGÍSTICA

Belo Horizonte

2022

Paulo Roberto Gonçalves

SISTEMA DE GESTÃO DE SERVIÇOS DE LOGÍSTICA

Trabalho de Conclusão de Curso de
Especialização em Arquitetura de Software
Distribuído como requisito parcial à obtenção
do título de especialista.

Orientador(a): Pedro Alves de Oliveira

Belo Horizonte

2022

RESUMO

O trabalho tem como objetivo apresentar uma arquitetura distribuída para a transportadora Boa Entrega, a arquitetura denominada GSL (Gestão de Serviços de Logística) contempla um conjunto de módulos afim de atender as necessidades levantadas, principalmente em questões de integração, seja com sistemas externos ou sistemas legados da empresa. Devido a alta competitividade do setor de logística e o cenário da pandemia COVID-19, os serviços de logística estão cada vez mais requisitados, e os sistemas atuais da empresa não dão suporte a esse crescimento, portanto, a arquitetura GSL é importante para cumprir as metas traçadas. Os requisitos mais relevantes serão utilizados para o desenvolvimento de uma POC, com o objetivo de avaliar a arquitetura proposta ao decorrer do trabalho.

Palavras-chave: arquitetura de software, projeto de software, requisitos arquiteturais, microserviços, logística.

SUMÁRIO

1. Apresentação	6
1.1. Problema	6
1.2. Objetivo do trabalho	6
1.3. Definições e Abreviaturas	8
2. Especificação da Solução	9
2.1. Requisitos Funcionais	9
2.2. Requisitos Não Funcionais	12
2.3. Premissas	15
2.4. Restrições Arquiteturais	15
2.5. Mecanismos Arquiteturais	15
3. Modelagem Arquitetural	18
3.1. Macroarquitetura	18
3.2. Descrição Resumida dos Casos de Uso	21
3.3. Visão Lógica	22
3.3.1. Diagrama de Classes	22
3.3.2. Diagrama de Componentes	28
3.3.3. Diagrama de Implantação	32
3.3.4. Modelo de Dados	33
4. Prova de Conceito (POC) e Protótipo Arquitetural	36
4.1. Implementação	36
4.1.1. Casos de uso para implementação	37
4.1.2. Requisitos não funcionais para avaliação	37
4.1.3. Tecnologias utilizadas na implementação	38
4.1.4. Códigos	39
4.1.5. Vídeo de apresentação	43
4.2. Interfaces e APIs	43
5. Avaliação da Arquitetura	47
5.1. Análise das abordagens arquiteturais	47
5.2. Cenários	47

5.3. Evidências da Avaliação	49
5.3.1. Cenário 1	49
5.3.1.1. Evidências	51
5.3.2. Cenário 2	57
5.3.2.1. Evidências	59
5.3.3. Cenário 3	65
5.3.3.1. Evidências	69
5.3.4. Cenário 4	83
5.3.4.1. Evidências	86
5.4. Resultados	104
6. Conclusão	109
REFERÊNCIAS	112
APÊNDICES	115

1. Apresentação

No contexto atual da pandemia, serviços de logística se tornaram essenciais para a sobrevivência de muitas empresas. A venda é feita principalmente online, necessitando de transporte rápido e de qualidade, dos diversos produtos disponíveis, como comida, medicamentos, etc. A Boa Entrega é uma transportadora de grande porte, que atua em todo o território brasileiro, atendendo clientes de diferentes segmentos. O investimento em tecnologia é imprescindível para o sucesso do negócio, principalmente em um cenário tão competitivo como a área de logística.

1.1. Problema

A empresa já possui alguns sistemas implantados, de diversas áreas de aplicação. No entanto, alguns sistemas não possuem facilidade de integração e são de difícil manutenção. Para que as metas da empresa sejam alcançadas é importante que melhorias ocorram, principalmente melhorias tecnológicas. A empresa pretende melhorar os processos de entrega, e ao mesmo tempo expandir a sua atuação para outros municípios com a ajuda de parceiros. Além de informações analíticas é importante que a modernização dos sistemas ocorra, para que integrações com parceiros seja viável e que outras metas traçadas sejam alcançadas.

1.2. Objetivo do trabalho

O objetivo geral do trabalho é apresentar um projeto arquitetural aderente aos requisitos, com foco no baixo custo da solução e flexibilidade para incorporação de funcionalidades dos sistemas legados através de novos microsserviços ou existentes.

A solução proposta é chamada de Gestão de Serviços de Logística (GSL), no qual será desenvolvida pela equipe técnica da Boa Entrega. Essa solução possui quatro módulos:

1. Módulo de Informações Cadastrais: consiste em obter e manter informações de clientes, fornecedores, depósitos e mercadorias. Dentre essas destacam-se: identificação, dados de localização, dados complementares e informações necessárias ao negócio da empresa;
2. Módulo de Serviços ao Cliente: tem como escopo prover uma solução de *workflow*, com o uso de *Business Process Management* - BPM. Por meio deste módulo é possível desenhar, analisar e acompanhar todos os processos de atendimento ao cliente existentes na empresa - tanto os já existentes quanto os que ainda serão implantados, desta forma melhorando o desempenho e a eficiência desses processos;
3. Módulo de Gestão e Estratégia: tem como escopo prover a gestão estratégica de todas as atividades da empresa, com indicadores das entregas realizadas e a realizar, na forma de indicadores, representados na forma de *cockpit*. Para este módulo será utilizado uma ferramenta de gestão corporativa adquirida no mercado;
4. Módulo de Ciência de Dados: este módulo deve utilizar ferramentas adequadas para obtenção, guarda, recuperação e utilização dos dados corporativos pertinentes, com recursos para tratamento de dados massivos (*Big Data*), mineração dos dados para apoio às tomadas de decisão. Todos os dados deste módulo são obtidos de planilhas e bancos de dados, relacionais ou NoSQL. O uso de recursos de um *Date Warehouse* (DW), nesse contexto, é essencial para o sucesso desta iniciativa.

Os objetivos específicos são:

1. Realizar um estudo e escolha de tecnologias adequadas conforme os requisitos e restrições arquiteturais;
2. Desenvolver uma POC contemplando alguns microsserviços do módulo de Informações Cadastrais e microsserviços auxiliares, para validar as principais tecnologias escolhidas e demais elementos essenciais para uma arquitetura de microsserviços;

3. Explorar algumas integrações com sistemas legados via chamadas a APIs REST e integração via banco de dados.

1.3. Definições e Abreviaturas

- **AKS:** Azure Kubernetes Service;
- **API:** Application Programming Interface;
- **AWS:** Amazon Web Services;
- **BI:** Business Intelligence;
- **BPM:** Business Process Management;
- **CDC:** Change Data Capture;
- **CI/CD:** Continuous Integration/Continuous Delivery;
- **DTO:** Data Transfer Object;
- **EKS:** Elastic Kubernetes Service;
- **ER:** Entity Relationship;
- **ETL:** Extract Transform Load;
- **GSL:** Gestão de Serviços de Logística;
- **JSON:** JavaScript Object Notation;
- **JWT:** JSON Web Token;
- **LDAP:** Lightweight Directory Access Protocol;
- **ORM:** Object-Relational Mapping;
- **POC:** Proof of Concept;
- **REST:** Representational State Transfer;
- **RPC:** Remote Procedure Call;
- **SAF:** Sistema Administrativo-Financeiro;
- **SFC:** Sistema de Faturamento e Cobrança;
- **SGE:** Sistema de Gestão de Entregas;
- **SOAP:** Simple Object Access Protocol.

2. Especificação da Solução

Esta seção descreve os requisitos contemplados neste projeto arquitetural, divididos em dois grupos: funcionais e não funcionais.

Para as ferramentas de mercado a coluna “Dificuldade” foi desconsiderada. A coluna “Prioridade” foi considerada olhando a importância da funcionalidade para a arquitetura GSL e resolução dos problemas levantados.

2.1. Requisitos Funcionais

1. Módulo de Informações Cadastrais:

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	<p>O sistema deve permitir que o usuário consulte um pedido pelo seu código.</p> <p>Se o pedido existe é retornado para o usuário as informações do pedido, que são: código, data de emissão, data de entrega prevista, custo do frete, nome do destinatário, documento do destinatário, endereço do destinatário, volumes, responsável pela entrega, situação, nome do remetente e endereço do remetente.</p> <p>Se o pedido não existe é retornado uma mensagem de erro.</p>	B	A
RF02	<p>O sistema deve permitir que o responsável pela entrega realize a comprovação da entrega. Primeiramente o responsável faz a consulta dos dados do pedido, usando a funcionalidade descrita no RF01. Com os dados é feito a confirmação com o destinatário.</p> <p>Se a confirmação permite a entrega, é realizado a coleta de assinatura do destinatário e submetido ao sistema, finalizando a comprovação da entrega. Nesse momento o pedido tem sua situação alterada para “ENTREGA_CONCLUIDA”.</p> <p>Se a confirmação não permite a entrega, o responsável deve informar o motivo de forma obrigatória e de forma opcional uma observação. Nesse momento o pedido tem sua situação alterada para “ENTREGA_NAO_CONCLUIDA” e o motivo e observação são adicionados a entrega do pedido.</p>	M	A
RF03	O sistema deve permitir que parceiros consultem os pedidos	M	A

	disponíveis para continuação do processo de entrega. Os dados retornados são os mesmos descritos na RF01, com a adição da localização, que seria um centro de distribuição ou depósito.		
RF04	O sistema deve permitir que um parceiro marque um ou mais pedidos para continuação do processo de entrega. Ao submeter ao sistema os pedidos marcados são alterados para a situação "EM_TRANSPORTE". E o parceiro fica como responsável pela entrega.	B	A
RF05	O sistema deve permitir que os usuários realizem a simulação do custo de frete. Para isso devem informar ao sistema os dados: CEP de origem, CEP de destino, lista de volumes e serviços opcionais. O volume possui os dados: formato (Caixa ou Envelope), peso (Kg ou g), dimensão (Altura, Largura e Comprimento) e quantidade. Serviços opcionais possui os dados: Seguro (Se marcado é necessário informar o Valor Declarado). Se o serviço de seguro for adicionado é realizado a comunicação com o SGE, pois o cálculo do seguro é realizado no sistema legado. Para realizar o cálculo do seguro é necessário os dados: CEP de origem, CEP de destino e Valor Declarado.	A	A
RF06	O sistema deve permitir que o cliente realiza a simulação do custo de frete, considerando possíveis descontos. As informações submetidas são as mesmas descritas na RF05. É aplicado os descontos do cliente no custo de frete calculado.	A	A
RF07	O sistema deve permitir que o cliente consulte todos os seus pedidos e faça filtros pela situação.	B	M
RF08	O sistema deve permitir que o cliente crie um pedido. Para a criação do pedido deve ser informado os dados: documento do destinatário, nome do destinatário, endereço do destinatário, volumes, nome do remetente, endereço do remetente e serviços opcionais. O pedido é criado com a situação "PEDIDO_CRIADO".	M	A
RF09	O sistema deve permitir que o cliente consulte os seus endereços, para que sejam selecionados durante a criação do pedido.	B	M
RF10	O sistema deve permitir que usuários criem depósitos, informando o nome e endereço do mesmo.	B	B

RF11	O sistema deve permitir que usuários criem novos parceiros, informando o nome do parceiro e os depósitos em que ele pode pegar pedidos.	B	B
RF12	O sistema deve permitir que uma entrega seja buscado pelo número do pedido. Deve ser apresentado os dados da entrega, como código do pedido, responsável pela entrega, data da entrega, situação da entrega, motivo, observação e dados de geolocalização. Deve ser possível baixar a assinatura da entrega também.	B	B
RF13	O sistema deve permitir que usuários adicionem pedidos a um determinado depósito. Isso permite que parceiros possam atribuir esse pedido a si para a continuação do processo de entrega. Quando o pedido é adicionado a um depósito a situação dele é alterada para “PEDIDO RECEBIDO”.	B	A

2. Módulo de Serviços ao Cliente:

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF14	A ferramenta deve permitir importar e exportar arquivos de processos	-	A
RF15	A ferramenta deve permitir que usuários autorizados visualizem os processos implantados	-	A
RF16	A ferramenta deve possibilitar a criação de Dashboards com os processos ativos	-	A
RF17	A ferramenta deve permitir integração com APIs para utilização nos processos implantados	-	A
RF18	A ferramenta deve fornecer funcionalidade ou ferramenta externa compatível para desenho dos processos	-	M

3. Módulo de Gestão e Estratégia:

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF19	A ferramenta deve permitir a criação de novos projetos e importação de dados (Arquivo ou API) para popular os projetos	-	A
RF20	A ferramenta deve permitir visualizar indicadores dos projetos através de um Cockpit	-	A
RF21	A ferramenta deve permitir traçar metas para os projetos existentes	-	A

RF22	A ferramenta deve permitir a criação de Dashboards para visualização de vários projetos e seus indicadores	-	A
RF23	A ferramenta deve permitir a exportação de dados dos projetos, seja via API ou arquivos	-	A

4. Módulo de Ciência de Dados:

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF24	A ferramenta deve permitir a coleta de dados das bases de dados existentes e planilhas ou outro formato de arquivo	-	A
RF25	A ferramenta deve ser capaz de tratar dados relevantes para o negócio, fazendo ajustes conforme programado e quando necessário	-	A
RF26	A ferramenta deve ser capaz de enviar notificação ou relatórios por e-mail	-	M
RF27	A ferramenta deve permitir a visualização de informações, através de relatórios, Dashboards ou outros	-	A
RF28	A ferramenta deve fornecer funcionalidade ou ferramenta externa para construção de relatórios	-	A
RF29	A ferramenta deve permitir a extração de dados para formatos conhecidos, como planilhas, CSV, html, txt, dentro outros	-	M
RF30	A ferramenta deve fornecer controle de acesso para relatórios	-	A

2.2. Requisitos Não Funcionais

ID	Descrição	Prioridade B/M/A
RNF01	O sistema deve possuir características de aplicação distribuída	A
RNF02	O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades	A
RNF03	O sistema deve ser de fácil implantação	A
RNF04	O sistema deve suportar ambientes web e móveis	A
RNF05	O sistema deve ser recuperável no caso da ocorrência de erro	A
RNF06	O sistema deve utilizar recursos adequados para integração	A
RNF07	O sistema deve utilizar recursos de gestão de configuração, com integração contínua	A
RNF08	O sistema deve apresentar bom desempenho	A

A seguir os cenários de estímulo resposta para cada requisito não funcional:

RNF01 - O sistema deve possuir características de aplicação distribuída	
Estímulo	Uma requisição é feita pelo usuário
Fonte de estímulo	Usuário usando o sistema
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Microsserviços dependentes para efetuar a requisição do usuário
Resposta	O sistema deve se comunicar com todos os microsserviços necessários e produzir uma resposta única para o usuário
Medida da resposta	A resposta produzida não possui erros e é possível rastrear todo o fluxo de comunicação usando um Traceld gerado pelo mecanismo de log distribuído

RNF02 - O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades	
Estímulo	Uma requisição é feita por um usuário sem acesso ao recurso
Fonte de estímulo	Usuário sem acesso ao recurso fazendo a requisição
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Qualquer microsserviço com controle de acesso aos recursos
Resposta	O sistema deve verificar as permissões de acesso ao recurso para o usuário e retornar um erro informando que o usuário não tem acesso ao recurso
Medida da resposta	O sistema não deve permitir que o usuário acesse o recurso

RNF03 - O sistema deve ser de fácil implantação	
Estímulo	Uma nova tag é gerada para a branch x do repositório do microsserviço e é feito o agendamento do deploy para o ambiente y
Fonte de estímulo	Desenvolvedor criando tag no GitLab e agendando deploy
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Qualquer repositório de microsserviço com pipeline criado no GitLab
Resposta	Pipeline gera artefato validado do microsserviço e imagem Docker que será usada como deploy no Kubernetes. O deploy da imagem é efetuado no Kubernetes e o microsserviço fica disponível
Medida da resposta	Deploy do microsserviço com a quantidade correta de réplicas é feito conforme definido nos arquivos de deployment do Kubernetes e a situação do microsserviço é “UP” conforme leitura via API de monitoramento

RNF04 - O sistema deve suportar ambientes web e móveis	
Estímulo	Uma requisição é feita por outro software
Fonte de estímulo	Outro software se comunicando com o sistema
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Microsserviços dependentes para efetuar a requisição solicitada
Resposta	O sistema deve se comunicar com todos os microsserviços necessários e produzir uma resposta única no formato JSON
Medida da resposta	A resposta produzida não possui erros e está no formato JSON. A resposta pode ser lida pelo software que fez a requisição. O software deve ser o Browser usando JavaScript e uma aplicação móvel utilizando um cliente HTTP

RNF05 - O sistema deve ser recuperável no caso da ocorrência de erro	
Estímulo	Uma réplica de determinado microsserviço sendo utilizado é derrubada

	propositamente
Fonte de estímulo	Time de DevOps
Ambiente	Em funcionamento no ambiente de teste com diversos usuários utilizando o microserviço que será derrubado
Artefato	Microserviço que será derrubado
Resposta	Os usuários de teste continuam recebendo retorno de sucesso nas requisições efetuadas, mesmo com uma réplica derrubada
Medida da resposta	A réplica restante deve atender todas as requisições e deve constar no Kubernetes que existe uma réplica fora do ar. A réplica que foi derrubada deve subir automaticamente em no máximo um minuto e começar a receber requisições assim que a situação do microserviço for “UP” conforme leitura via API de monitoramento. A distribuição de requisições deve ocorrer normalmente quando mais de uma réplica existir

RNF06 - O sistema deve utilizar recursos adequados para integração	
Estímulo	Uma mensagem que deve ser processada pelo sistema legado, independente se o sistema legado está no disponível ou não, é enviada a uma fila do RabbitMQ (Ferramenta de mensageria)
Fonte de estímulo	Microserviço produtor da mensagem
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Microserviço produtor da mensagem e microserviço middleware (consumidor) que realiza integração entre GSL e sistema legado necessário
Resposta	A mensagem é consumida da fila do RabbitMQ e processada pelo middleware, acessando o sistema legado com o padrão de integração necessário
Medida da resposta	A mensagem produzida é recebida na fila correta do RabbitMQ e consumida pelo middleware correto, ao ser consumida a mensagem não aparece mais na fila. O middleware realiza o processamento da mensagem acessando o sistema legado com o padrão de integração adequado

RNF07 - O sistema deve utilizar recursos de gestão de configuração, com integração contínua	
Estímulo	Uma alteração na configuração de um microserviço é realizada no repositório que agrupa as configurações dos microserviços. A alteração na configuração é feita para determinada profile
Fonte de estímulo	Time de DevOps
Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Microserviço que está utilizando a configuração e profile que será alterada
Resposta	O microserviço atualiza automaticamente, sem precisar de reinicialização, a configuração alterada e passa a responder com essa nova configuração
Medida da resposta	O servidor de configuração (Spring Cloud Config Server) faz a leitura do arquivo de configuração no repositório de configurações, logo após a configuração ser alterada. A leitura da nova configuração fica disponível via API no servidor de configuração e é gerado uma mensagem para o RabbitMQ, que descreve uma alteração na configuração de determinado microserviço e profile, essa mensagem é consumida pelo microserviço que usa a configuração, fazendo a atualização da configuração

RNF08 - O sistema deve apresentar bom desempenho	
Estímulo	Uma requisição a um recurso cacheado é realizada
Fonte de estímulo	Usuário acessando o sistema

Ambiente	Em funcionamento com carga normal no ambiente de teste
Artefato	Qualquer microserviço que possui um recurso cacheado
Resposta	O sistema deve retornar os dados atualizados e sem apresentar erros
Medida da resposta	A requisição não deve executar queries no banco de dados MySQL e deve constar uma chamada ao cache configurado para o recurso. A requisição deve ser retornado em menos de 1 segundo

2.3. Premissas

- O sistema legado SGE utiliza o banco de dados MySQL.

2.4. Restrições Arquiteturais

- R1: A solução deve possuir baixo custo;
- R2: A solução deve se integrar com os sistemas legados sem a necessidade de substituição dos legados;
- R3: Deve ser utilizado uma arquitetura de microsserviços para os módulos novos;
- R4: A solução deve ser desenvolvida com a linguagem de programação Java e Framework Spring Boot;
- R5: Não deve ser alterado os sistemas legados para realizar novas integrações, deve ser utilizado meios já existentes para se integrar;
- R6: Deve ser considerado uma base de dados única para os microsserviços e BI, separada em schemas conforme a necessidade;
- R7: Deve ser utilizado uma ferramenta adquirida no mercado para os módulos de Serviços ao Cliente, Gestão e Estratégia e Ciência de Dados;
- R8: O sistema deve permitir hospedagem em nuvem híbrida;
- R9: A arquitetura deve permitir a incorporação de microsserviços dos sistemas legados, escritos em qualquer linguagem;

2.5. Mecanismos Arquiteturais

Análise	Design	Implementação
Linguagem Back-end	Linguagem de programação	Java 11
Framework Web Back-end	Framework Web	Spring Boot

Ferramenta Build Back-end	Ferramenta de Build	Maven
Documentação de APIs	Documentação	Swagger
Testes Back-end	Testes	JUnit, Mockito, Spring Framework
Banco de dados GSL/BI	Database	MySQL
Framework Persistência	ORM	Hibernate
Banco de Dados Cache	Cache	Redis
Mensageria/Integração assíncrona	Message Broker	RabbitMQ
Integração síncrona entre microsserviços	HTTP Client	Spring Cloud OpenFeign
Autenticação e autorização	Autenticação e autorização	OAuth2 e Keycloak
Servidor de configuração	Configuração centralizada	Spring Cloud Config
Empacotamento aplicações	Virtualização	Docker
Alta disponibilidade	Orquestração de containers	Kubernetes
Registro e descoberta de serviços	Registro e descoberta de serviços	Kubernetes
API Gateway	API Gateway	Spring Cloud Gateway e Nginx Ingress Controller no Kubernetes
Logs do sistema	Gerenciamento de logs	Filebeat e ELK (Elasticsearch, Logstash e Kibana)
Versionamento de código	Versionamento de código	Git/GitLab
Deploy	CI/CD	GitLab CI
Ferramenta para CDC	CDC	Debezium
Ferramenta para ETL	ETL	Airflow
Ferramenta para BI	BI	PowerBI
Ferramenta para BPM	Workflow com BPM	Camunda
Ferramenta para gerenciamento de projetos	Gestão e Estratégia	Asana
Circuit Breaker	Resiliência	Spring Cloud Circuit Breaker
Padronização integração com legados	Padrões de integração	Spring Integration
Log distribuído	Rastreabilidade logs	Spring Cloud Sleuth
Gerenciamento clusters Kubernetes	Gerenciamento cluster	Rancher
Monitoramento das aplicações	Monitoramento	Grafana e Prometheus
Linguagem Front-end	Linguagem de programação	TypeScript

Bibliotecas e Frameworks Front-end	Bibliotecas e Frameworks	ReactJS e Bootstrap
Servidor de aplicação Front-end	Servidor de aplicação	Nginx
Testes Front-end	Testes	Jest
Testes e2e	Testes	Cypress

Algumas tecnologias apresentadas na tabela acima podem ser alterados ou descartadas conforme a necessidade durante a implementação da arquitetura.

3. Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural da solução proposta, de forma a permitir seu completo entendimento visando à implementação da prova de conceito.

3.1. Macroarquitetura

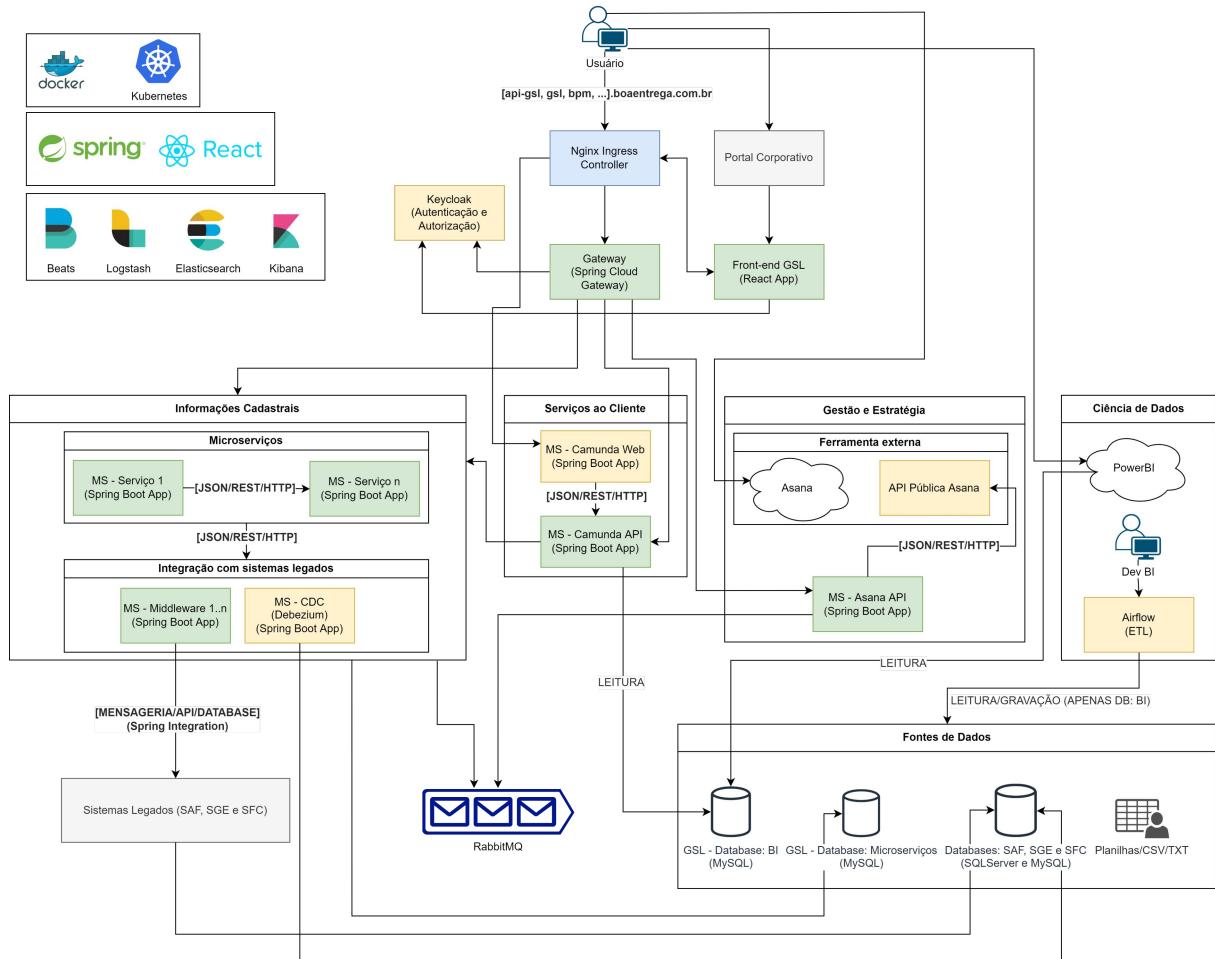


Figura 1 - Visão Geral da Solução. Fonte: Autor.

A Figura 1 mostra um diagrama com a visão geral da solução proposta. Detalhes específicos de cada componente serão apresentados no diagrama de componentes, portanto, será apresentado apenas uma visão geral de como os componentes interagem.

O usuário pode acessar diferentes aplicações dentro da arquitetura GSL, algumas delas serão construídas pela equipe da Boa Entrega e outras adquiridas e/ou integradas. O usuário pode ser uma pessoa ou um software que pertence a um parceiro que pretende se comunicar com as APIs do GSL. Na solução proposta foi considerado o Portal Corporativo, que faria apenas um redirecionamento para a aplicação Front-end do GSL, isso não impede o usuário de acessar diretamente a aplicação.

O usuário que pretende acessar o Front-end do GSL ou a API, precisa realizar uma autenticação. Todo usuário possui um perfil ou mais atribuídos, o qual determinadas permissões a ele. No caso da solução, é apresentado a possibilidade de acesso a API, Front-end e interface Web do Camunda (Software Workflow BPM). O acesso as aplicações de mercado, como Asana e PowerBI, são controlados pelo provedor do serviço.

O Front-end se comunica com a API do GSL, para prover as funcionalidades aos usuários. O agrupamento dos diversos endpoints e outras questões pertinentes fica a cargo do API Gateway. Ele seria a API do GSL, a qual seria utilizada pelo Front-end e parceiros.

O conjunto de microsserviços apresentados na solução é separado em quatro módulos, os quais já foram citados anteriormente. O módulo de informações cadastrais agrupa um conjunto maior de microsserviços, visto que ele gerencia os cadastros de diferentes entidades e algumas regras de negócio. Como apresentado na solução, existem os microsserviços que realizam a implementação de uma regra de negócio, e outros responsáveis pela integração com os sistemas legados. Os microsserviços de integração são chamados de middlewares. Existe um middleware para cada sistema legado, visto que a integração é diferente para cada um e é melhor especializar, devido a diferença de tecnologias. A integração com os sistemas legados é realizada utilizando os padrões de integração mais adequados.

Além dos middlewares é considerado uma solução de CDC, essa solução é temporário, e seria utilizada apenas enquanto existir funcionalidades que não foram integradas ou incorporadas na arquitetura GSL. Para o CDC seria utilizado o software Debezium, que é responsável por ler os logs de diferentes bancos de

dados e gerar mensagens que serão consumidas por algum microserviço, essas mensagens descrevem as modificações que foram realizadas em determinadas tabelas, isso possibilita que uma alteração feita em algum sistema legado possa ser refletida no sistema GSL, sem que o sistema legado faça a integração com o GSL. Caso a integração do sistema legado com o GSL for possível, não seria utilizado o CDC.

A comunicação entre os microsserviços é realizada de forma síncrona utilizando o padrão REST com JSON, por ser uma tecnologia mais estabelecida no mercado e possibilitar que exista uma integração com microsserviços dos sistemas legados de forma facilitada, como o caso do SGE, que é escrito em PHP. A comunicação assíncrona é realizada por meio de um Message Broker, na solução foi escolhido o RabbitMQ.

Os módulos de Serviços ao Cliente e Gestão e Estratégia são muito parecidos, ambos disponibilizam uma ferramenta que não será desenvolvida pela equipe da Boa Entrega, mas possuem microsserviços responsáveis pela integração com essas ferramentas. No caso do módulo de Serviços ao Cliente a ferramenta se comunica com uma API, isso é necessário no uso dos diagramas BPM. O módulo de Gestão e Estratégia possui uma API que faz a integração com a API pública da ferramenta escolhida, que é o Asana.

Também é apresentado no diagrama o módulo de Ciência de Dados, esse módulo contempla um BI e uma ferramenta de ETL. A solução de BI escolhida foi o PowerBI. A ferramenta de ETL escolhida foi o Airflow. Essa ferramenta de ETL se comunica com as diversas fontes de dados disponíveis, no caso da solução seriam os bancos de dados dos microsserviços, sistemas legados e arquivos dos mais diversos, como planilhas, csv, dentre outros. Todos esses dados são coletados, transformados e adicionados na base de dados do BI, que possui os dados necessários para utilização no PowerBI.

A solução proposta não considera que cada microserviço possui uma instância do MySQL, mas apenas um schema para cada um, no caso do MySQL um database. Isso permite o uso de uma instância do MySQL, e caso seja necessário

migrar no futuro seria mais fácil, visto que os microsserviços já suportam essa separação.

Como apresentado no diagrama, os microsserviços da solução GSL usam Spring Boot e são desenvolvidos em Java. Todos esses microserviços serão disponibilizados como containers Docker, que serão gerenciados pelo Kubernetes. O Kubernetes fornece recursos úteis para microsserviços, portanto, em alguns casos é utilizado soluções do Spring Cloud e quando necessário do Kubernetes. Como no caso da descoberta e registro de serviços, na solução proposta será utilizado o Kubernetes e não o Eureka do Spring Cloud.

O Front-end deverá utilizar ReactJS e outras bibliotecas e frameworks descritos nos mecanismos arquiteturais.

Também, é apresentado que a centralização de logs e gerenciamento será feita com o uso das ferramentas: Beats, Logstash, Elasticsearch e Kibana.

3.2. Descrição Resumida dos Casos de Uso

UC01 – INTEGRAÇÃO DE PARCEIROS NO PROCESSO DE ENTREGA	
Descrição	Integração de parceiros no processo de entrega
Atores	Cliente, Parceiro
Prioridade	A
Requisitos associados	RF08, RF07, RF01, RF03, RF04, RF13, RF11, RF10
Fluxo Principal	<p>O cliente deve conseguir criar um pedido, informando os dados necessários. O pedido criado deve aparecer na listagem de pedidos do cliente. O cliente poderá detalhar os dados do pedido se necessário.</p> <p>O parceiro deve conseguir consultar os pedidos que estão disponíveis para a continuação do processo de entrega. O pedido fica disponível após ser adicionado em um depósito que o parceiro possui acesso.</p> <p>Com base nos pedidos disponíveis o parceiro deve conseguir marcar um ou mais pedidos para continuar o processo de entrega. Os pedidos marcados não devem aparecer mais como disponíveis para continuação do processo de entrega.</p>

UC02 – COMPROVAÇÃO DE ENTREGA	
Descrição	Comprovação de entrega
Atores	Parceiro
Prioridade	A

Requisitos associados	RF02, RF03, RF01, RF12
Fluxo Principal	<p>O parceiro deve conseguir submeter a comprovação de entrega para determinado pedido que ele é responsável. Se o pedido não pertence a ele deve ser gerado uma mensagem de erro e o pedido não teve ter a situação alterada.</p> <p>O pedido não deve aparecer mais como disponível para continuação do processo de entrega após a comprovação ser submetida, e a situação do pedido deve ser “ENTREGA_CONCLUIDA” em caso de sucesso.</p> <p>O parceiro deve conseguir submeter uma comprovação informando que a entrega não foi possível. A situação do pedido deve ser “ENTREGA_NAO_CONCLUIDA” e deve constar o motivo no cadastro da entrega do pedido.</p>

UC03 – SIMULAÇÃO DO CUSTO DE FRETE	
Descrição	Simulação do custo de frete
Atores	Cliente
Prioridade	A
Requisitos associados	RF06, RF05
Fluxo Principal	<p>O cliente deve conseguir simular o custo de frete após informar os dados necessários. O cliente pode optar de forma opcional pelo seguro, se marcado, deve ser realizado uma requisição ao sistema legado SGE para completar o cálculo.</p> <p>Antes de apresentar o resultado do frete, deve ser aplicado possíveis descontos disponíveis para o cliente.</p>

3.3. Visão Lógica

Esta seção mostra a especificação dos diagramas da solução proposta, com todos os seus componentes, propriedades e interfaces. Nas subseções a seguir são apresentados os diagramas de Classes, Componentes, Implantação e Entidade-Relacionamento.

3.3.1. Diagrama de Classes

Segue abaixo os diagramas de classe de alguns microserviços importantes e que centralizam mais regras de negócio. Alguns componentes foram omitidos do

diagrama, pois não são relevantes do ponto de vista da implementação, como DTOs e algumas classes de configuração.

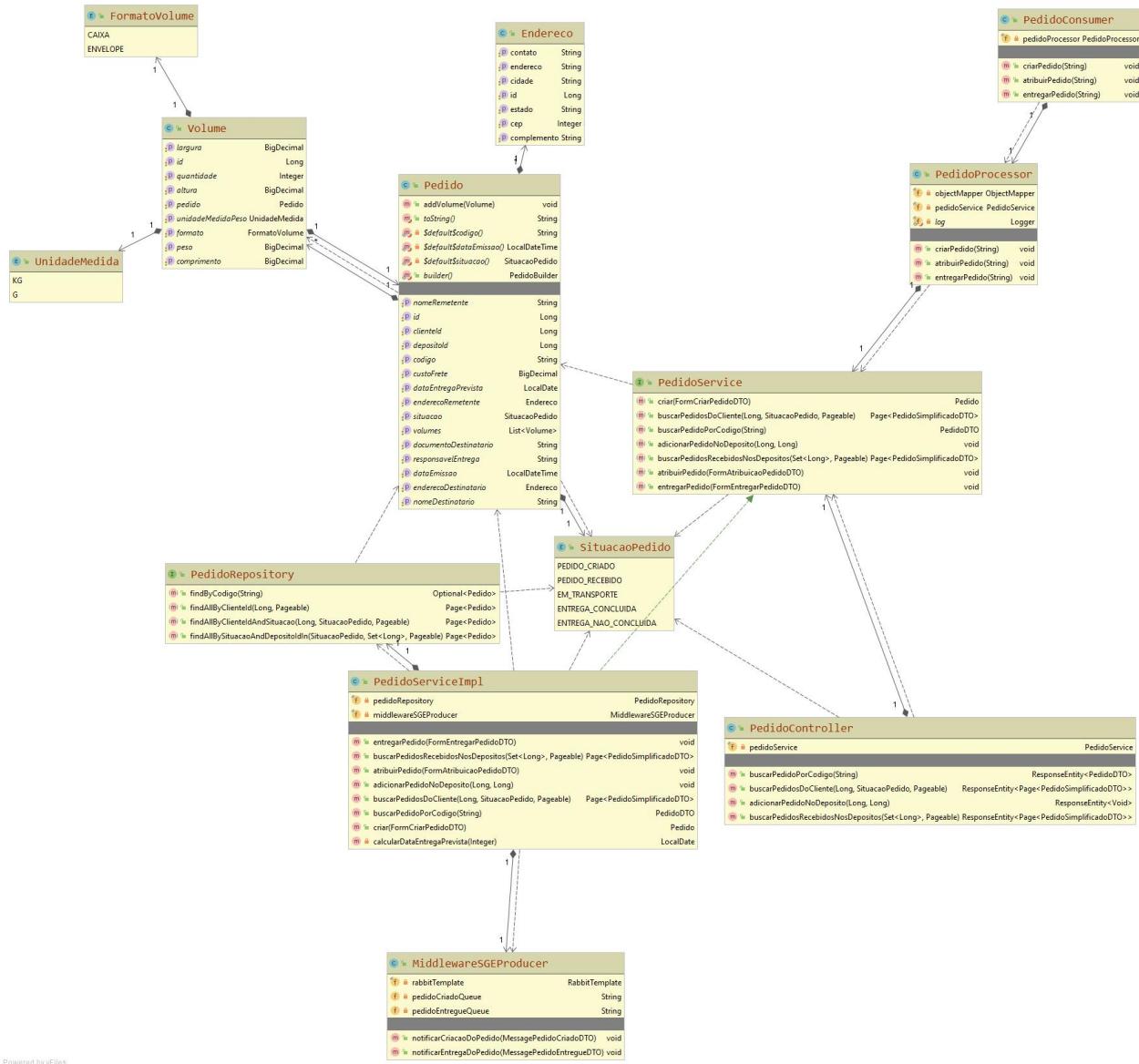


Figura 2 - Diagrama de classes do microserviço “pedido”. Fonte: Autor.

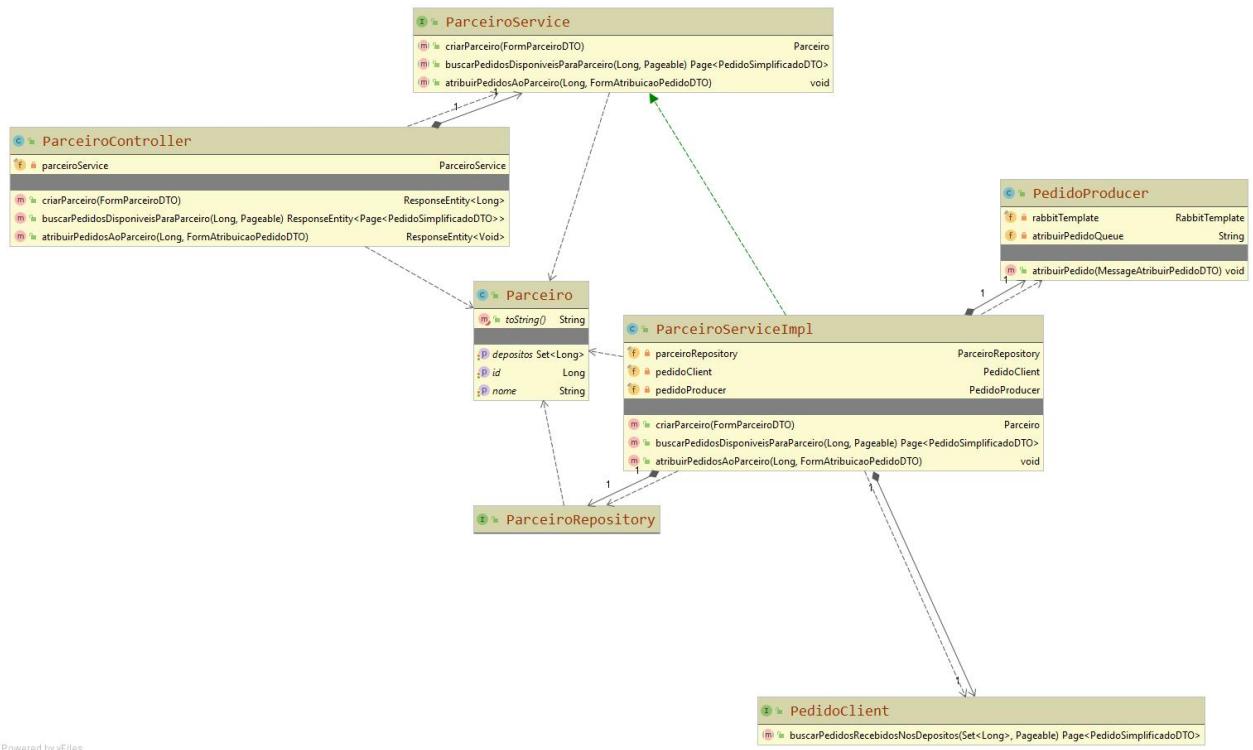


Figura 3 - Diagrama de classes do microserviço “parceiro”. Fonte: Autor.

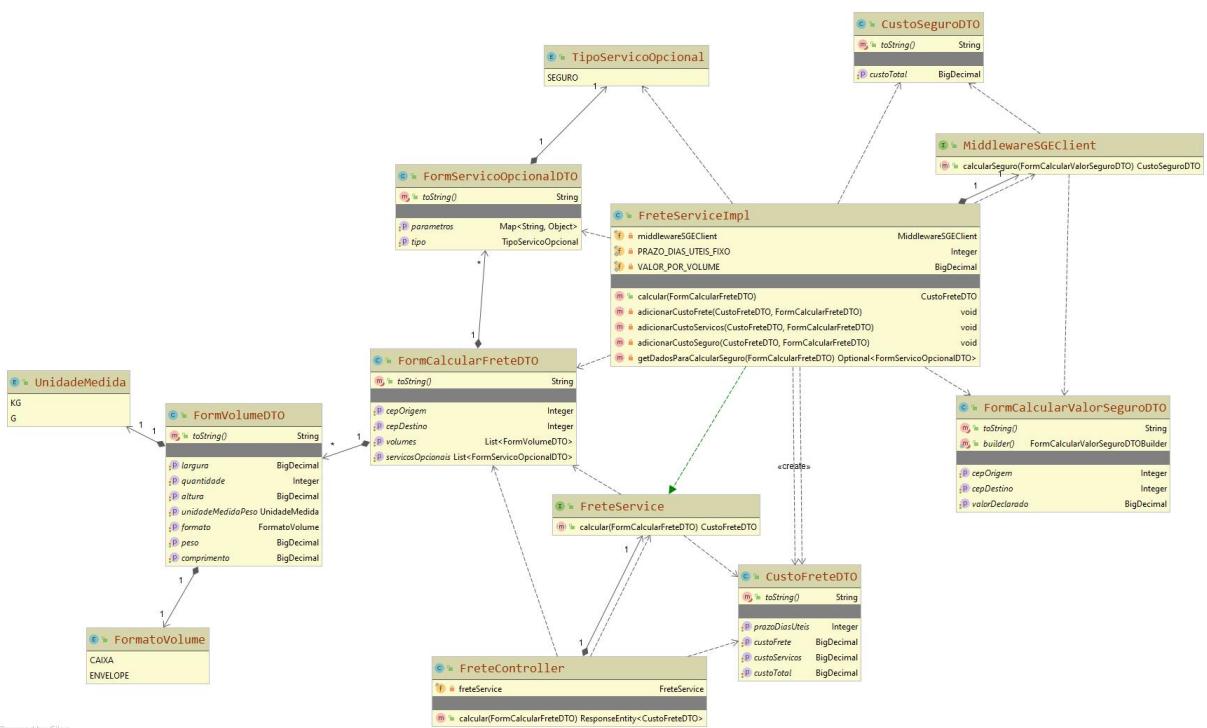


Figura 4 - Diagrama de classes do microserviço “frete”. Fonte: Autor.

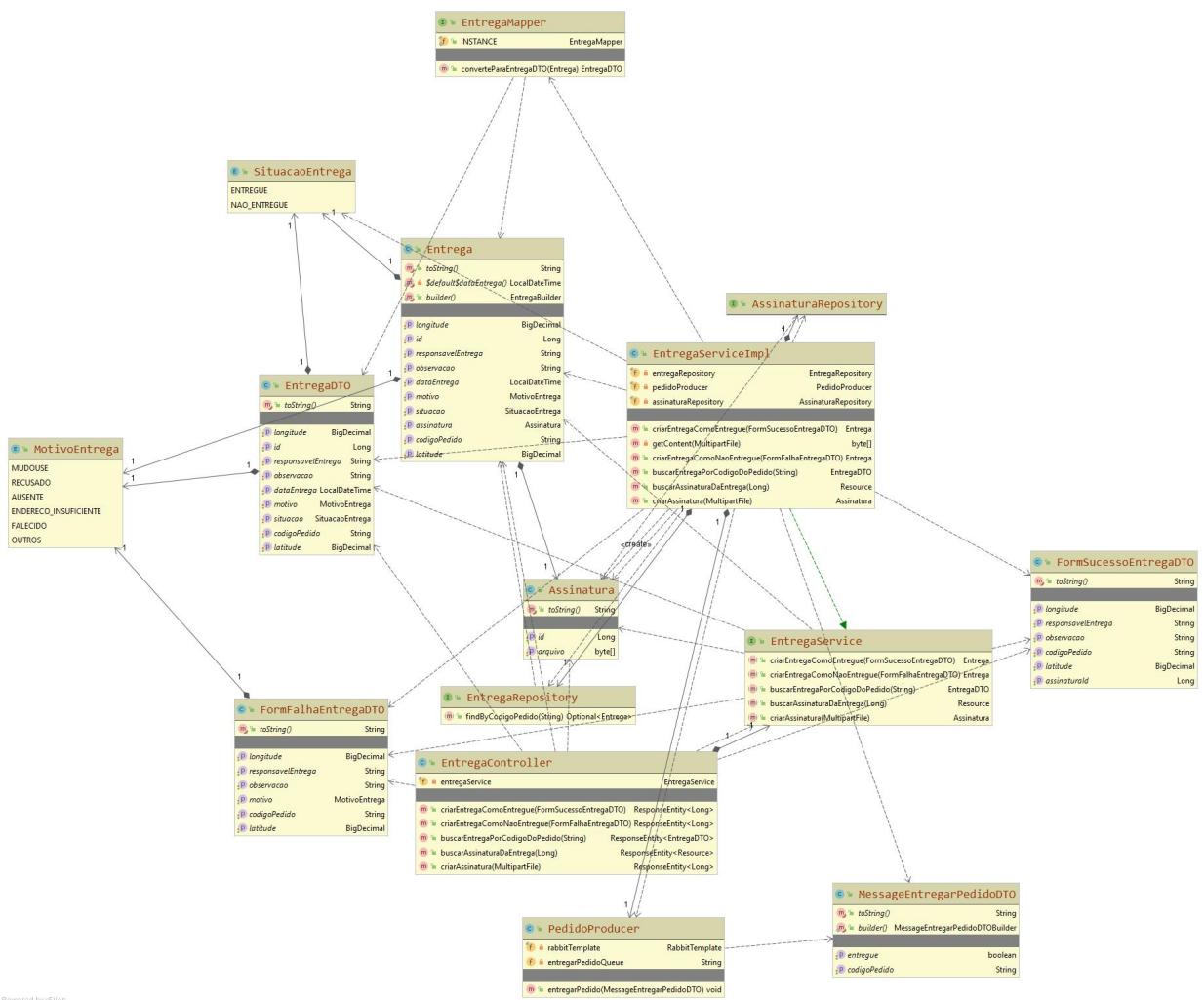


Figura 5 - Diagrama de classes do microserviço “entrega”. Fonte: Autor.

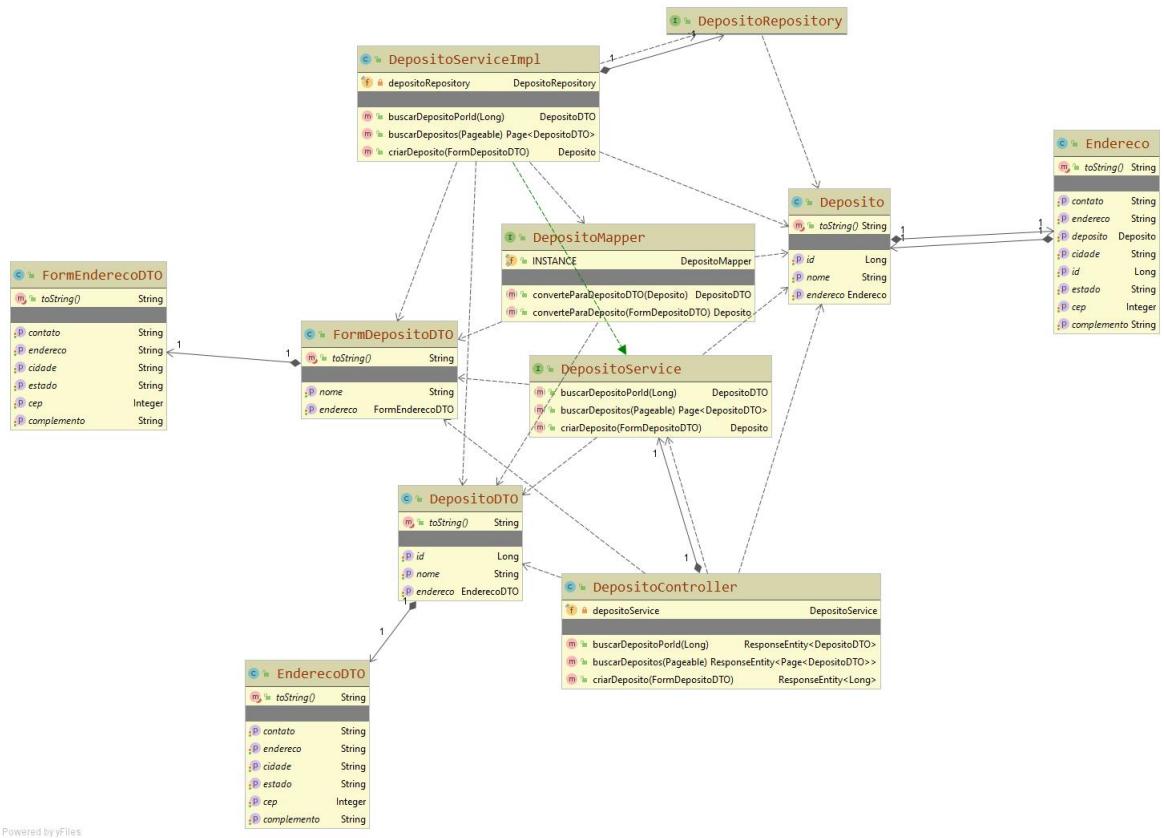


Figura 6 - Diagrama de classes do microserviço “deposito”. Fonte: Autor.

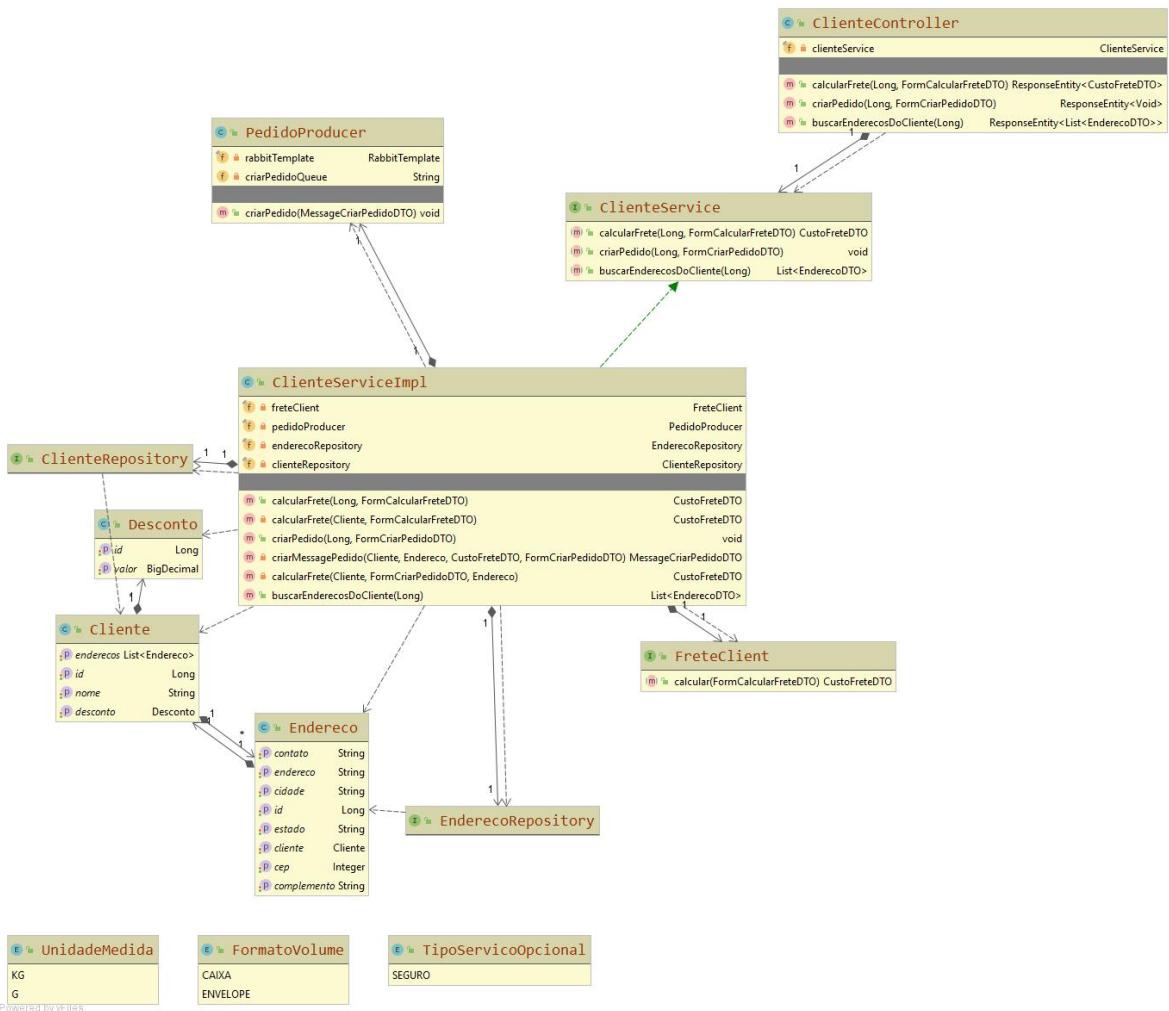


Figura 7 - Diagrama de classes do microserviço “cliente”. Fonte: Autor.

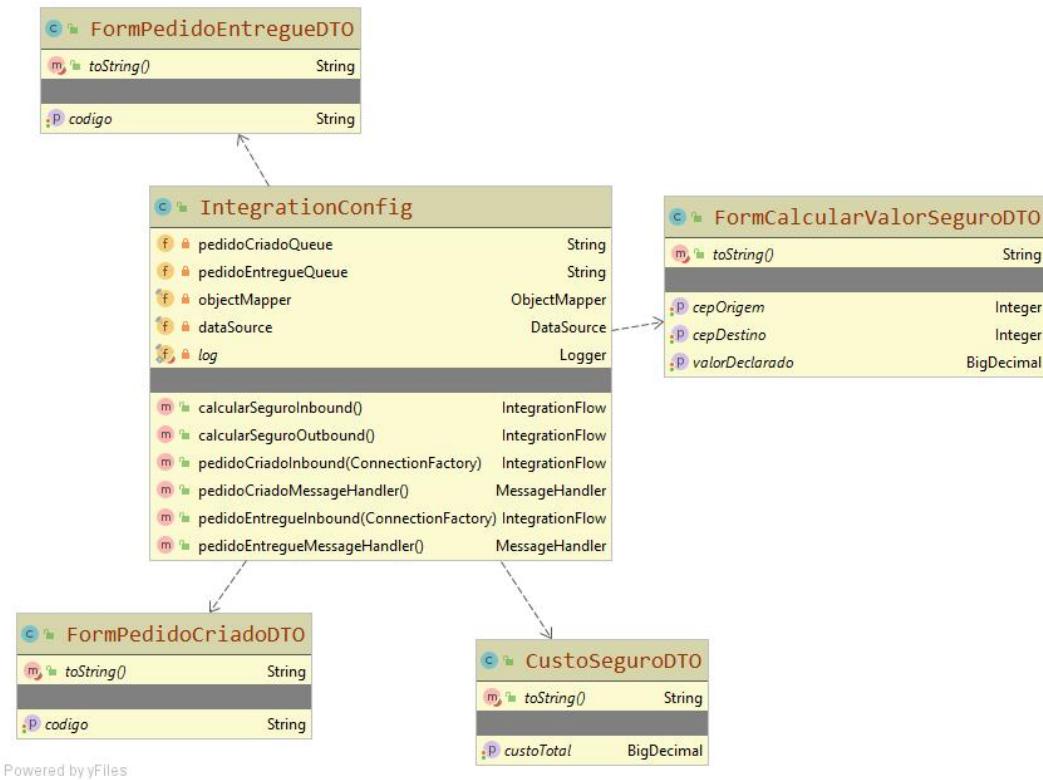


Figura 8 - Diagrama de classes do microserviço “middleware-sge”. Fonte: Autor.

3.3.2. Diagrama de Componentes

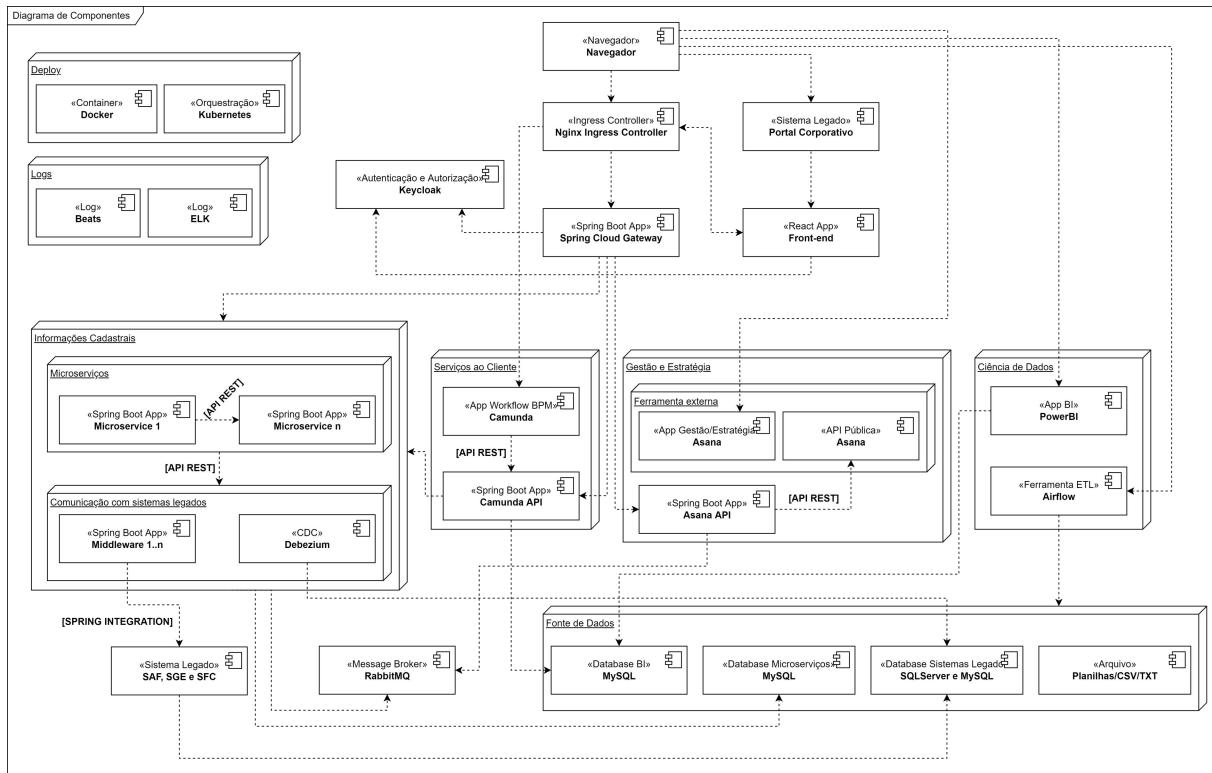


Figura 9 - Diagrama de componentes. Fonte: Autor.

Conforme diagrama apresentado na Figura 9, as entidades participantes da solução são:

- **Navegador** - Navegadores WEB como Google Chrome, Microsoft Edge, Safari e outros. Utilizado pelo usuário para acessar o Front-end do sistema GSL e outras aplicações internas e externas;
- **Nginx Ingress Controller** - Responsável por enviar as requisições do navegador para o serviço que está dentro do cluster Kubernetes, usa o host informado pelo usuário no navegador para saber em qual serviço deve ser encaminhado. Funciona como um Proxy Reverso;
- **Portal Corporativo, SAF, SGE e SFC** - Representam os sistemas legados da Boa Entrega;
- **Keycloak** - Aplicação responsável por gerenciar questões relacionadas a autenticação e autorização, tanto no Front-end quanto na API. Usa o padrão o OAuth2 e gera Tokens que serão utilizados para identificar o usuário e carregar as suas permissões. Também permite o gerenciamento de usuários com as suas autorizações. A carga dos usuários pode ser feita através de um LDAP ou ser armazenada diretamente na aplicação;
- **Spring Cloud Gateway** - Aplicação Spring Boot que usa a biblioteca Spring Cloud Gateway para atuar como Gateway. Essa aplicação representa a API que o Front-end deve consumir. Também é um cliente OAuth2, portanto, deve se autenticar no Keycloak;
- **Front-end** - Aplicação React que apresenta a interface visual para o usuário, utiliza várias APIs para fornecer as funcionalidades do sistema GSL ao usuário. Faz uso da API através do Nginx Ingress Controller, da mesma forma que o navegador acessa as aplicações;
- **Microservice 1 e Microservice n** - Aplicação Spring Boot que representa um microserviço de qualquer módulo. A comunicação síncrona é feita com HTTP no padrão REST e usando JSON;

- **Middleware 1..n** - Aplicação Spring Boot que representa um microserviço com responsabilidade de integração com algum sistema legado. A integração é feita utilizando padrões adequados de integração, sejam via APIs REST, mensageria ou acesso direto aos bancos de dados. Usa a biblioteca Spring Integration para padronizar e facilitar a implementação das integrações;
- **Debezium** - Aplicação Spring Boot que possui a dependência da ferramenta Debezium, que realiza o processo de CDC no sistema GSL. O objetivo desse componente é realizar a comunicação dos sistemas legados com o sistema GSL, para que alterações nos sistemas legados possam refletir no sistema GSL, sem que exista uma chamada de API ou geração de mensagem em um Message Broker. Esse componente pode ser tratado como temporário na arquitetura, visto que deve ser inutilizado quando os sistemas legados foram totalmente integrados ou incorporados na arquitetura GSL;
- **RabbitMQ** - Solução de Message Broker, utilizado principalmente na comunicação assíncrona entre os microsserviços e comunicação com sistemas legados ou sistemas externos;
- **Camunda** - Ferramenta de Workflow BPM que será utilizada no sistema GSL. A ferramenta é uma aplicação Spring Boot que disponibiliza uma interface Web para os usuários, essa interface é acessada pelo navegador passando pelo Ingress Controller. Essa ferramenta permite todo o gerenciamento de diagramas BPM e a integração com APIs REST para visualização do diagrama em tempo real ou com dados do BI;
- **Camunda API** - Aplicação Spring Boot que fornece diversas APIs para a ferramenta Camunda, essas APIs são utilizadas nos diagramas BPM durante a modelagem e execução dos mesmos;
- **Asana** - Ferramenta de gestão e estratégia que será utilizada no sistema GSL. Essa ferramenta não é hospedada na infraestrutura do GSL, é uma ferramenta adquirida no mercado e possibilita diversas integrações e importação de dados;
- **Asana API** - Aplicação Spring Boot que faz a integração com o Asana, acessando a API pública disponibilizada pelo mesmo. Essa aplicação pode

consumir mensagens do RabbitMQ, que são geradas por outros microserviços;

- **PowerBI** - Ferramenta de BI que será utilizada no sistema GSL. Essa ferramenta não é hospedada na infraestrutura do GSL, é uma ferramenta adquirida no mercado e possibilita a manipulação dos dados do banco de dados BI, para geração de relatórios e extrações;
- **Airflow** - Ferramenta de ETL que será utilizada no sistema GSL. Essa ferramenta é gratuita e será utilizada para coletar os diversos dados presentes nas bases de dados e arquivos, esses dados serão gravados no banco de dados do BI;
- **Databases** - O banco de dados escolhido para o sistema GSL é o MySQL, será utilizado uma instância do MySQL, mas com um database para cada microserviço;
- **Docker** - Ferramenta para realizar o empacotamento das aplicações em containers. Os containers são utilizados para fazer o deployment da aplicação no Kubernetes;
- **Kubernetes** - Ferramenta para realizar a orquestração dos containers. Fornece recursos importantes para o ambiente de microserviços, como o registro e descoberta de aplicações, configurações externalizadas, Self-Healing, dentre outros;
- **Beats** - Conjunto de ferramentas para coletar os logs dos containers. Os logs são enviados ao ELK, para indexação e apresentação nos Dashboards;
- **ELK** - Elasticsearch, Logstash e Kibana. Ferramentas responsáveis pelo recebimento dos logs, indexação e visualização.

3.3.3. Diagrama de Implantação

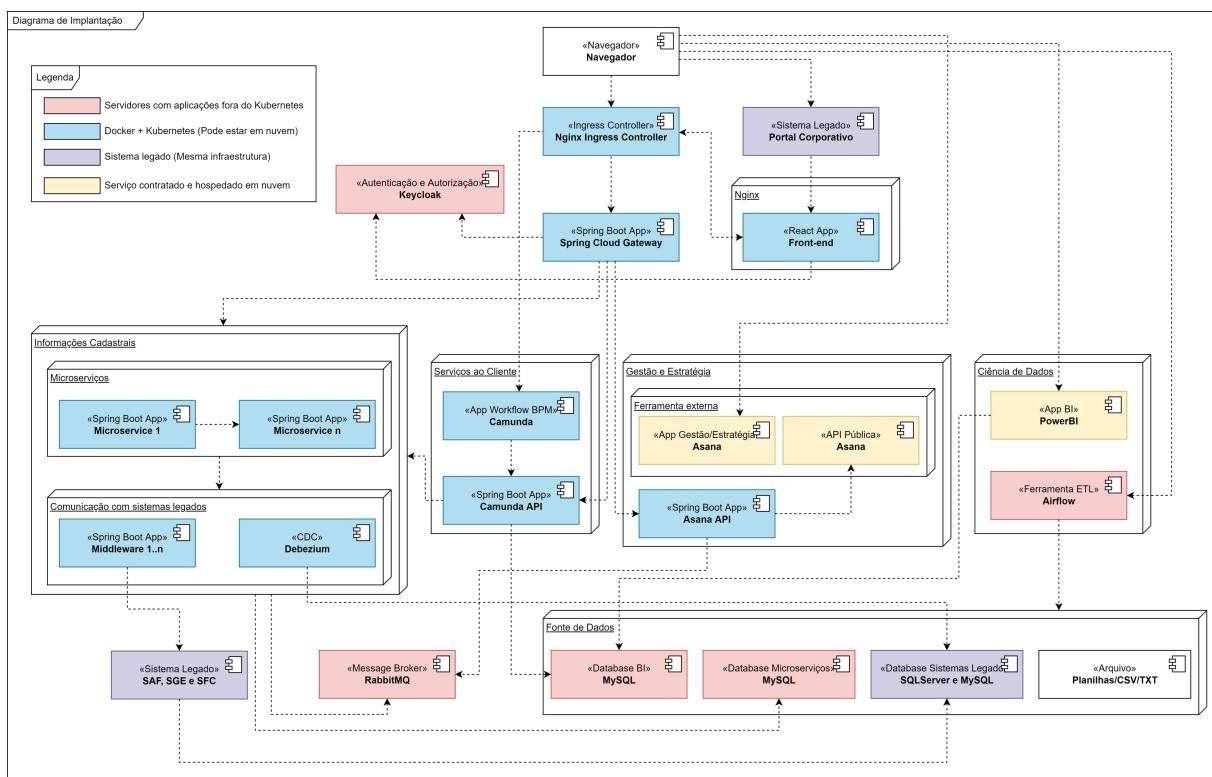


Figura 10 - Diagrama de Implantação. Fonte: Autor.

A Figura 10 apresenta o diagrama de implantação, os componentes apresentados possuem uma cor que representa a forma de implantação. Essas implantações estão detalhadas abaixo:

- **Servidores com aplicações fora do Kubernetes** - Essas aplicações são hospedadas fora do cluster Kubernetes, são aplicações que guardam estados, como bancos de dados, ferramenta de ETL, Message Broker e Keycloak. Essas aplicações estariam hospedadas na infraestrutura da Boa Entrega;
- **Docker + Kubernetes (Pode estar em nuvem)** - Essas aplicações se concentram nos microsserviços stateless que estão presentes nos diferentes módulos do sistema GSL, essas aplicações podem ser reinicializadas em qualquer momento e pode existir diferentes réplicas das mesmas. Essas aplicações ficam dentro do cluster Kubernetes, portanto, podem estar hospedadas na infraestrutura da Boa Entrega ou em nuvem, usando serviços como EKS da AWS ou AKS da Azure. As aplicações dentro do cluster não são acessadas externamente de forma individual, apenas as que estão

expostas através de um serviço do tipo LoadBalancer, como é o caso do IngressController que atua como um proxy reverso para o Gateway, Front-end e interface Web do Camunda, que estão dentro do cluster e não são acessados diretamente;

- **Sistema legado (Mesma infraestrutura)** - As aplicações legadas, como Portal Corporativo, SAF, SGE e SFC continuam na mesma infraestrutura. Essas aplicações podem ser adicionadas no cluster Kubernetes conforme necessidade, se incorporando na arquitetura do sistema GSL e portanto fazendo parte do sistema GSL;
- **Serviço contrato e hospedado em nuvem** - Alguns serviços não são hospedados na infraestrutura da Boa Entrega, como é o caso do Asana e PowerBI, essas aplicações são de terceiros e contratadas, portanto ficam hospedadas em nuvem. Os usuários fazem acesso a essas aplicações e a autorização e autenticação é gerenciada pelas mesmas.

3.3.4. Modelo de Dados

Os seguintes diagramas ER foram considerados na implementação da POC:

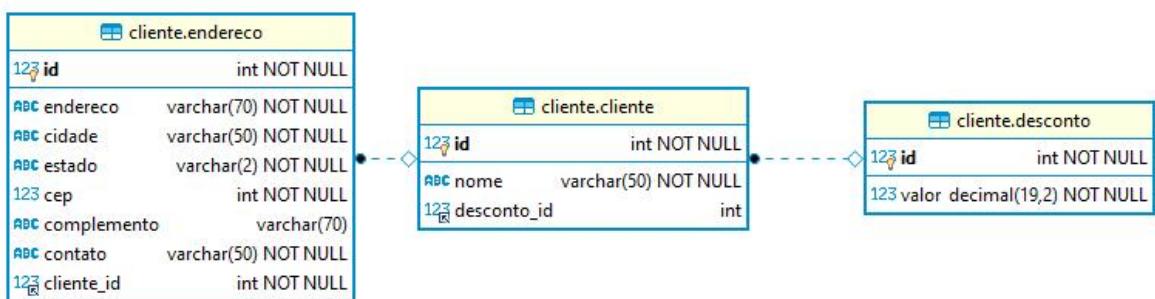


Figura 11 - Diagrama ER banco “cliente”. Fonte: Autor.

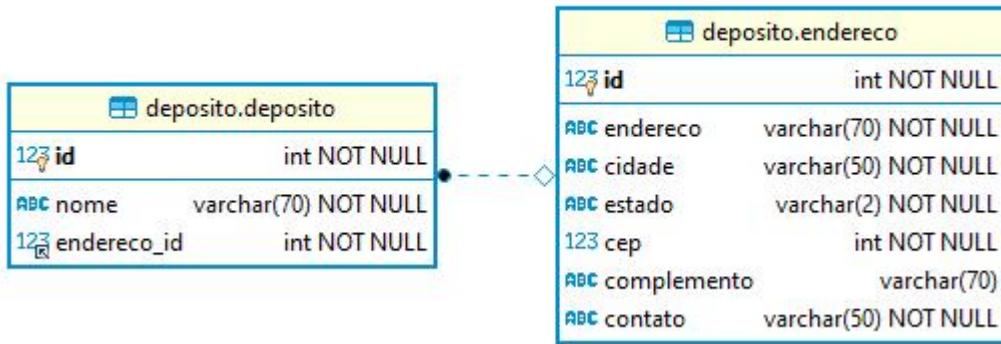


Figura 12 - Diagrama ER banco “deposito”. Fonte: Autor.

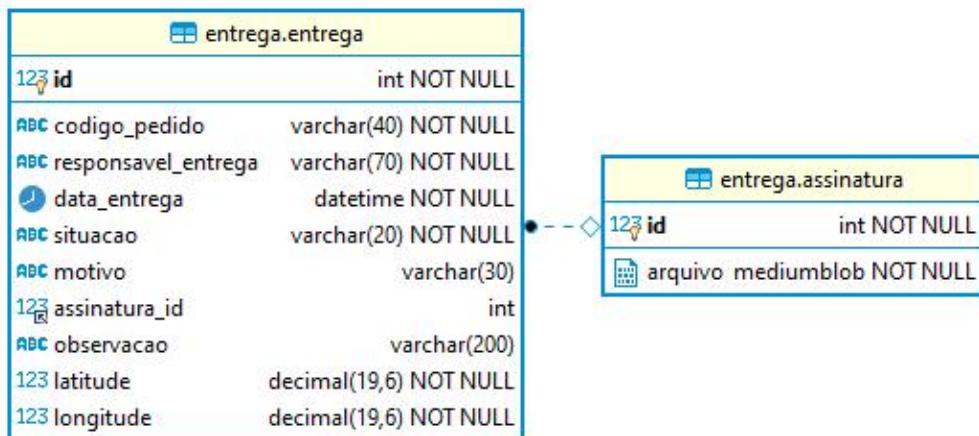


Figura 13 - Diagrama ER banco “entrega”. Fonte: Autor.



Figura 14 - Diagrama ER banco “parceiro”. Fonte: Autor.



Figura 15 - Diagrama ER banco “pedido”. Fonte: Autor.

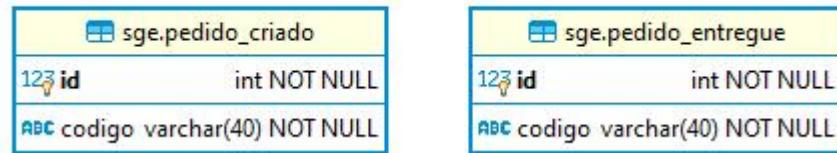


Figura 16 - Diagrama ER banco “sge”. Fonte: Autor.

4. Prova de Conceito (POC) e Protótipo Arquitetural

A seguir é apresentada a POC desenvolvida com o intuito de atender aos requisitos especificados. O repositório contendo os códigos da POC pode ser acessado através do link: <https://github.com/paulooorg/tcc-asd>.

4.1. Implementação

A implementação da POC envolve a criação de alguns microserviços, os quais serão descritos de forma mais detalhada ao decorrer do trabalho. Todos os microserviços foram implementados em Java com Spring Boot:

- **gateway**: Microserviço que utiliza o Spring Cloud Gateway para atuar como um API Gateway, expondo as APIs dos outros microserviços. Também centraliza questões de autenticação através do Keycloak;
- **middleware-sge**: Microserviço que utiliza o Spring Integration para fazer a integração com o sistema legado SGE;
- **documentacao-api**: Microserviço que utiliza o Swagger e centraliza a documentação dos microserviços;
- **cliente**: Microserviço que centraliza regras de negócio relacionadas ao cliente;
- **deposito**: Microserviço que centraliza regras de negócio relacionadas aos depósitos;
- **entrega**: Microserviço que centraliza regras de negócio relacionadas entregas de pedidos;
- **frete**: Microserviço que centraliza regras de negócio relacionadas ao cálculo do frete;
- **pedido**: Microserviço que centraliza regras de negócio relacionadas ao gerenciamento de pedidos;

- **parceiro:** Microserviço que centraliza regras de negócio relacionadas a integração com parceiros.

4.1.1. Casos de uso para implementação

Os seguintes casos de uso foram implementados na POC:

- **UC01 – INTEGRAÇÃO DE PARCEIROS NO PROCESSO DE ENTREGA;**
- **UC02 – COMPROVAÇÃO DE ENTREGA;**
- **UC03 – SIMULAÇÃO DO CUSTO DE FRETE.**

4.1.2. Requisitos não funcionais para avaliação

- **RNF01 - O sistema deve possuir características de aplicação distribuída;**

Critérios de aceite:

- Os microserviços devem se comunicar de forma síncrona através de REST APIs;
- Os microserviços devem ser comunicar de forma assíncrona através de mensageria utilizando RabbitMQ.

- **RNF02 - O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades;**

Critérios de aceite:

- Os usuários devem se autenticar no sistema para utilizar os endpoints;
- Um usuário deve acessar somente os endpoints liberados para o seu perfil.

- **RNF05 - O sistema deve ser recuperável no caso da ocorrência de erro;**

Critérios de aceite:

- Um microserviço que caiu sem processar as mensagens de uma fila no RabbitMQ deve ser capaz de processar as mensagens logo que estiver disponível novamente;
- Uma réplica de um microserviço que caiu, dentro do cluster do Kubernetes, deve se recuperar assim que possível de forma automática.

- **RNF06 - O sistema deve utilizar recursos adequados para integração.**

Critérios de aceite:

- Os microserviços devem fornecer APIs REST e devolver os dados no formato JSON;
- Os microsserviços devem trabalhar de forma stateless, podendo ser derrubados a qualquer momento, sem impacto na autenticação dos usuários.

4.1.3. Tecnologias utilizadas na implementação

As seguintes tecnologias, mais relevantes, foram utilizadas na implementação da POC:

- **Java 11** - Linguagem de programação;
- **Spring Boot** - Framework Web;
- **Spring Cloud** - Conjunto de bibliotecas do Spring Cloud, como Spring Gateway, Spring Sleuth, Spring Integration, Spring Security, Spring OAuth2 Client e Resource Server;
- **Docker** - Construção dos containers das aplicações;

- **Docker Compose** - Gerenciamento dos containers como banco de dados, mensageria e outras ferramentas (Apenas para a POC);
- **Kubernetes** - Orquestração de containers;
- **Minikube** - Implementação do Kubernetes (Utilizado para iniciar um cluster local);
- **MySQL** - Banco de dados dos microserviços;
- **Swagger** - Documentação das APIs;
- **RabbitMQ** - Message Broker;
- **Keycloak** - Serviço de autenticação e autorização;
- **Wiremock** - Ferramenta para fazer mock de APIs;
- **AB Tool** - Ferramenta da Apache para realizar Benchmark em APIs;
- **DBeaver** - Ferramenta para acesso as bases de dados.

4.1.4. Códigos

Todos os microserviços foram desenvolvidos em Java com Spring Boot. A arquitetura de código segue o padrão MVC. Para os microserviços que possuem alguma regra de negócio mais relevante é possível observar a segmentação de pastas, como é o caso do microserviço “pedido”:

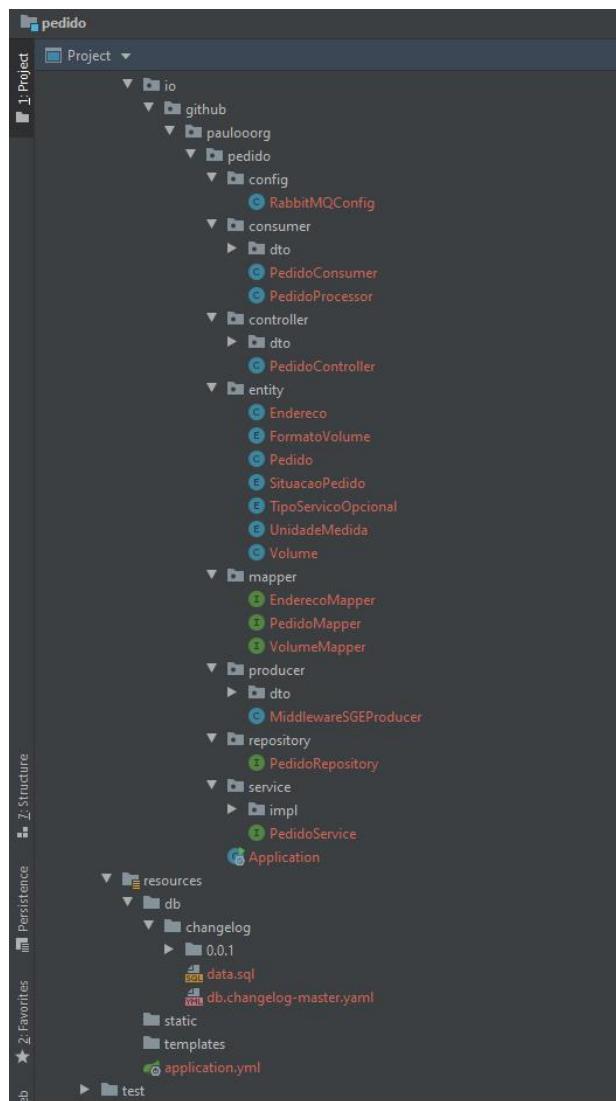


Figura 17 - Estrutura de pastas do microseviço “pedido”. Fonte: Autor.

Todos os microserviços que possuem regra de negócio relevante, como o microserviço de “pedido”, “entrega”, “parceiro”, dentre outros, devem seguir essa estrutura de pastas e o padrão MVC.

Nessa estrutura de pastas é possível observar a seguinte segmentação:

- **config** - Possui classes responsáveis pela configuração de componentes, como RabbitMQ, Logging, Filtros, dentre outros;

- **consumer** - Possui classes responsáveis por consumir e processar as mensagens consumidas do RabbitMQ. O processamento pode ser delegado para os serviços;
- **controller** - Possui classes responsáveis por expor endpoints no padrão REST. Também possui sub pacotes com os DTOs de request e response;
- **entity** - Possui classes responsáveis por representar o modelo do negócio, geralmente representam tabelas do banco de dados e podem conter regras de negócio que não devem ficar nos serviços;
- **mapper** - Possui classes responsáveis por fazer a conversão de dados entre entidades e DTOs e vice-versa;
- **producer** - Possui classes responsáveis por produzir mensagens para o RabbitMQ. Os serviços utilizam esses producers;
- **repository** - Possui classes responsáveis por fazer o acesso e manipulação dos dados no banco de dados configurado;
- **service** - Possui classes responsáveis por centralizar grande parte das regras de negócio do microserviço;

Alguns microserviços podem fugir desse padrão, pois são microserviços auxiliares, como é o caso do gateway e documentacao-api. Abaixo está uma foto da estrutura de pastas do microserviço documentacao-api:

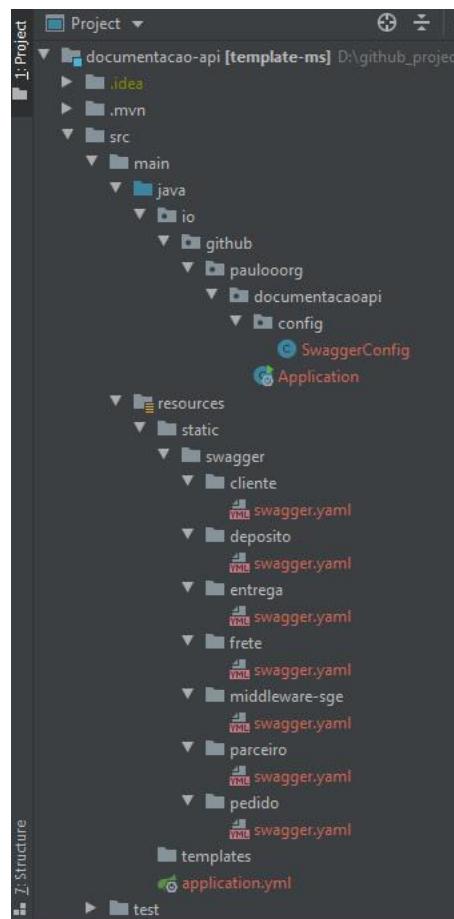


Figura 18 - Estrutura de pastas do microserviço “documentacao-api”. Fonte: Autor.

O microserviço gateway também tem uma estrutura mais enxuta:

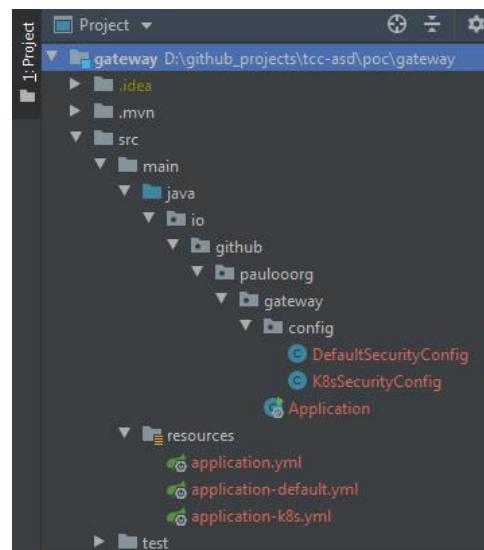


Figura 19 - Estrutura de pastas do microserviço “gateway”. Fonte: Autor.

Os demais microserviços apresentam a mesma segmentação de pastas que foi apresentada para o microserviço “pedido”, portanto, não será apresentado as imagens aqui. O código deles pode ser conferido no repositório da POC.

4.1.5. Vídeo de apresentação

Resumo POC: <https://vimeo.com/699897135>

Macroarquitetura: <https://vimeo.com/699897067>

Detalhado POC: <https://vimeo.com/699896974>

Os vídeos também estão disponíveis no repositório do GitHub.

4.2. Interfaces e APIs

A documentação das APIs foi realizada utilizando o Swagger. Essa documentação fica centralizada em um microserviço chamado “documentacao-api”. O microserviço agrupa todos os arquivos swagger.yaml de cada microserviço e disponibiliza na interface gráfica do Swagger.

As seguintes documentações foram implementadas:

Clients API 0.0.1 OAS3
/documentacao-api/swagger/cliente/swagger.yaml

Servers
http://localhost:8080/api/clientes - localhost - gateway

cálculo frete

POST /{id}/frete/calcular Calcula custo do frete

endereços

GET /{id}/enderecos Busca endereços do cliente

pedidos

POST /{id}/pedidos Cria um pedido para o cliente

Figura 20 - Documentação do microserviço “cliente”. Fonte: Autor.

Depósitos API 0.0.1 OAS3

/documentacao-api/swagger/deposito/swagger.yaml

Servers
http://localhost:8080/api/depositos - localhost - gateway ▾

depósitos

- GET** /{id} Busca um depósito por id ▾
- GET** / Busca todos os depósitos ▾
- POST** / Cria um depósito ▾

Figura 21 - Documentação do microserviço “deposito”. Fonte: Autor.

Entregas API 0.0.1 OAS3

/documentacao-api/swagger/entrega/swagger.yaml

Servers
http://localhost:8080/api/entregas - localhost - gateway ▾

entregas

- GET** /pedido Busca entrega por código do pedido ▾
- GET** /{id}/assinatura Busca assinatura da entrega ▾
- POST** /entregar/situacao/entregue Cria uma entrega com situação ENTREGUE ▾
- POST** /entregar/situacao/nao-entregue Cria uma entrega com situação NAO_ENTREGUE ▾
- POST** /assinatura Cria assinatura para adicionar na entrega ▾

Figura 22 - Documentação do microserviço “entrega”. Fonte: Autor.

Frete API 0.0.1 OAS3

/documentacao-api/swagger/frete/swagger.yaml

Servers
http://localhost:8080/api/fretes - localhost - gateway ▾

cálculo frete

- POST** /calcular Calcula custo do frete ▾

Figura 23 - Documentação do microserviço “frete”. Fonte: Autor.

Middleware SGE API 0.0.1 OAS3
 /documentacao-api/swagger/middleware-sge/swagger.yaml

Servers
 http://localhost:8080/api/middleware-sge - localhost - gateway

seguros

POST /seguros/calcular Calcula seguro

Figura 24 - Documentação do microserviço “middleware-sge”. Fonte: Autor.

Parceiros API 0.0.1 OAS3
 /documentacao-api/swagger/parceiro/swagger.yaml

Servers
 http://localhost:8080/api/parceiros - localhost - gateway

parceiros

GET /{id}/pedidos/disponiveis Busca pedidos disponíveis para o parceiro

POST / Cria um parceiro

POST /{id}/atribuir/pedidos Atribui pedidos ao parceiro

Figura 25 - Documentação do microserviço “parceiro”. Fonte: Autor.

Pedidos API 0.0.1 OAS3
 /documentacao-api/swagger/pedido/swagger.yaml

Servers
 http://localhost:8080/api/pedidos - localhost - gateway

pedidos

GET /codigo Busca um pedido pelo código

GET /clientes/{clienteId} Busca pedidos do cliente

GET /situacao/recebido/depositos Busca pedidos recebidos nos depósitos informados

GET /{pedidoId}/deposito/{depositoId} Adiciona pedido ao depósito

PATCH /{pedidoId}/deposito/{depositoId}

Figura 26 - Documentação do microserviço “pedido”. Fonte: Autor.

Os endpoints apresentados nas documentações acima são utilizados entre os microserviços e possíveis sistemas externos que venham a se integrar na arquitetura GSL.

Além das REST APIs, também foi criado algumas filas no RabbitMQ:

Queues

All queues (5)

Pagination

Page 1 of 1 - Filter: Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
middleware-sge-pedido-criado-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
middleware-sge-pedido-entregue-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
pedido-atribuir-pedido-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
pedido-criar-pedido-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
pedido-entregar-pedido-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	

Figura 27 - Filas criadas no RabbitMQ. Fonte: Autor.

Alguns microserviços da arquitetura GSL utilizam componentes comuns entre os microserviços, portanto, foi criado um projeto chamado “commons”, que tem como objetivo centralizar componentes comuns entre os microserviços. Para essa POC foi considerado apenas um componente de log de requisições. A estrutura de pastas pode ser visualizada abaixo:

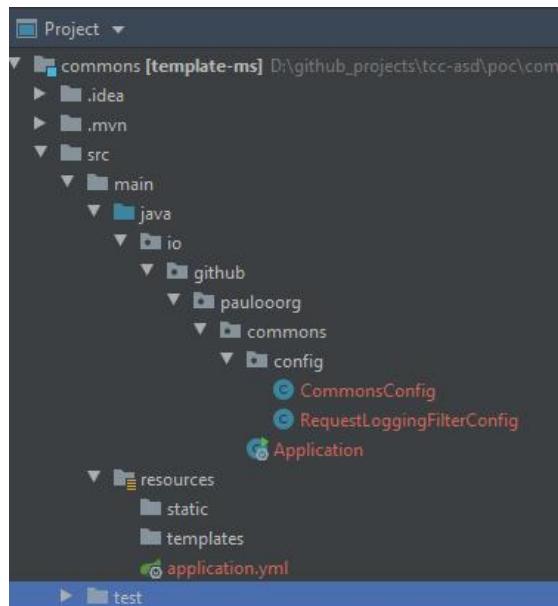


Figura 27 - Estrutura de pastas do projeto “commons”. Fonte: Autor.

5. Avaliação da Arquitetura

A avaliação da arquitetura desenvolvida é abordada nesta seção, visando avaliar se ela atende ao que foi proposto.

5.1. Análise das abordagens arquiteturais

A seguir estão listados os atributos de qualidade mais relevantes para o projeto, classificados por importância e complexidade e atrelados a um cenário que será avaliado.

Atributos de Qualidade	Cenários	Importância (B,M,A)	Complexidade (B,M,A)
Aplicação Distribuída	Cenário 1: O sistema deve possuir características de aplicação distribuída	M	A
Segurança	Cenário 2: O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades	A	A
Confiabilidade	Cenário 3: O sistema deve ser recuperável no caso da ocorrência de erro	A	M
Interoperabilidade	Cenário 4: O sistema deve utilizar recursos adequados para integração	A	A

5.2. Cenários

Para este projeto foram avaliados quatro cenários:

- **Cenário 1 - Aplicação Distribuída:** Os microsserviços devem se comunicar de forma síncrona utilizando APIs REST com JSON. A comunicação assíncrona deve ser realizada por meio de mensageria, as mensagens também devem estar no formato JSON. Toda a comunicação entre os diferentes microsserviços deve ser rastreável através dos logs das aplicações;

- **Cenário 2 - Segurança:** Os usuários do sistema devem se autenticar por meio de login com usuário e senha para poder utilizar os endpoints disponíveis. Cada usuário possui um grupo de perfis, que da acesso a determinados endpoints. Apenas usuários que possuem o perfil esperado podem executar requisições nos endpoints;
- **Cenário 3 - Confiabilidade:** Os microsserviços devem utilizar duas réplicas, que devem ser acessadas conforme a estratégia de load balancer configurada. Se um microserviço cair o outro ativo deve responder normalmente, de forma transparente para o usuário. Se os microsserviços envolvidos que consomem mensagens do RabbitMQ não estão ativos, as mensagens devem permanecer na fila e serem consumidas logo que uma réplica estiver disponível;
- **Cenário 4 - Interoperabilidade:** Os microsserviços do tipo middleware, responsáveis pela integração com os sistemas legados, devem receber mensagens através de uma fila do RabbitMQ. Essas mensagens devem estar no formato JSON e serem consumidas pelos middlewares corretos. O middleware deve fazer a integração com o sistema legado da forma mais adequada, seja por acesso ao banco de dados, REST APIs, mensageria ou outros. Os middlewares também podem receber requisições a REST APIs disponibilizadas, que fazem o acesso aos sistemas legados de forma síncrona, essas APIs devem utilizar o formato de dados JSON. Os microsserviços devem ser stateless, e caso algum caia não é necessário realizar a autenticação novamente.

5.3. Evidências da Avaliação

5.3.1. Cenário 1

Atributo de Qualidade:	Aplicação Distribuída
Requisito de Qualidade:	O sistema deve possuir características de aplicação distribuída
Preocupação:	Os microsserviços devem se comunicar de forma síncrona utilizando APIs REST com JSON. A comunicação assíncrona deve ser realizada por meio de mensageria, as mensagens também devem estar no formato JSON. Toda a comunicação entre os diferentes microsserviços deve ser rastreável através dos logs das aplicações
Cenário(s):	Cenário 1
Ambiente:	Sistema em operação normal
Estímulo:	Uma requisição a um microserviço é realizada por um usuário utilizando o sistema
Mecanismo:	Os microsserviços devem se comunicar utilizando APIs REST para a comunicação síncrona. Quando necessário uma comunicação assíncrona e/ou garantia de execução, independente se o serviço está disponível ou não, deve ser utilizado mensageria com RabbitMQ. Todo o fluxo de requisições deve ser rastreável através de um Traceld. As requisições que o usuário faz sempre passam pelo API Gateway, que deve tratar questões de autenticação e roteamento. A comunicação entre os microsserviços, exceto o API Gateway, deve ser feita sem a utilização do Gateway, ou seja, cada microserviço se comunicando com o outro através do seu nome/endereço (Utilização de registro e descoberta de serviços dentro do Kubernetes)
Medida de resposta:	<ul style="list-style-type: none"> * A resposta produzida para o usuário não possui erros * Deve ser possível consultar o pedido criado e essa requisição de consulta deve retornar os dados no formato JSON

- * A produção da resposta é feita pela comunicação síncrona e assíncrona com diferentes microsserviços
- * A comunicação entre os diversos microsserviços é rastreável
- * Os microsserviços devem utilizar APIs REST e mensageria

Fluxo para as evidências:

Usuário com perfil de Cliente acessa o microserviço “cliente” e realiza uma requisição utilizando o endpoint de criação de pedido. O usuário seleciona o seguro para o pedido.

O microserviço “cliente” deve acessar internamente o microserviço de “frete” de forma síncrona utilizando a API REST disponível. O microserviço “frete” deve acessar o microserviço “middleware-sge” de forma síncrona utilizando a API REST disponível para consultar os dados do seguro. Após esse fluxo o microserviço “cliente” deve montar uma mensagem contendo os dados do pedido, essa mensagem deve ser enviada para uma fila do RabbitMQ. Ao ser enviada o usuário recebe um retorno de sucesso para a requisição feita e o processo continua de forma assíncrona.

O microserviço “pedido” deve consumir as mensagens geradas referentes a criação de pedidos e gravar o pedido no banco de dados.

Todas as requisições, síncronas e assíncronas, geram logs e um Traceld, que faz referência a requisição inicial feita pelo usuário. Deve ser possível rastrear esse Traceld em todos os microserviços que participaram da requisição.

Considerações sobre a arquitetura:

Riscos:	A comunicação com os middlewares pode deixar a requisição lenta, visto que é necessário acesar os sistemas legados para consultar determinados dados. Algum microserviço participante da requisição pode ficar indisponível.
---------	---

	O serviço de mensageria pode estar indisponível.
Pontos de Sensibilidade:	Microsserviços participantes da requisição e mensageria devem estar disponíveis.
Tradeoff:	A tecnologia escolhida para comunicação síncrona entre os microsserviços foi APIs REST utilizando JSON. Pode ser utilizado RPC com gRPC para melhorar a performance. A escolha de APIs REST com JSON se deu pela grande utilização dessas tecnologias no mercado e pela facilidade de encontrar bibliotecas estáveis. A integração de serviços legados na arquitetura GSL é um ponto importante na decisão, visto que os serviços do SGE estão escritos em PHP e a reescrita de alguns pode não ser necessária, portanto, a comunicação entre os microsserviços deve ocorrer sem maiores problemas e utilizando tecnologias padronizadas e estáveis entre os diferentes microsserviços e linguagens.

5.3.1.1. Evidências

As requisições para os endpoints dos microsserviços será realizada utilizando a documentação do Swagger. Existe um microserviço chamado “documentacao-api” que centralizada a documentação dos microsserviços da arquitetura GSL. Abaixo é apresentado a documentação do endpoint de criação de pedido para o cliente, no microserviço “cliente”:

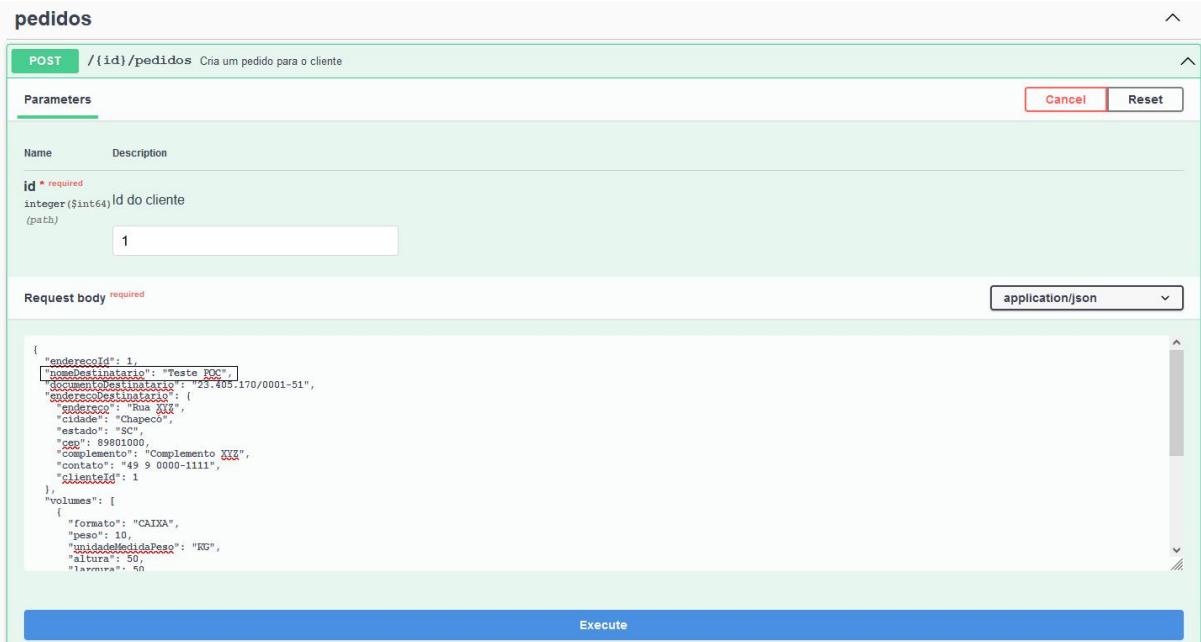


Figura 28 - Documentação do microserviço “pedido” - Endpoint POST /{id}/pedidos. Fonte: Autor.

Com esses dados apresentados na imagem o usuário realiza a execução da requisição. Essa requisição passa primeiro pelo Gateway, que é um microserviço com Spring Cloud Gateway, que centraliza detalhes de segurança e roteamento. Na imagem abaixo é possível identificar que o Gateway foi requisitado e fez o roteamento da rota “<http://localhost:8080/api/clientes/1/pedidos>” para o microserviço respondendo em “<http://localhost:8086/api/clientes>”:

```
2022-04-06 01:51:45.326 DEBUG 18180 --- [ctor-http-nio-3] o.s.c.g.h.RoutePredicateHandlerMapping : Mapping [Exchange[POST http://localhost:8080/api/clientes/1/pedidos]] to
Route[id='cliente', uri=http://localhost:8086/api/clientes, order=0, predicatePaths: [/api/clientes/**], match trailing slash: true, gatewayFilters=[[org.springframework.cloud
.gateway.filter.factory.TokenRelayGatewayFilterFactory$Lambda$1057/0x0000000800684440@38dd2306, order = 1], [[RemoveRequestHeader name = 'cookie', order = 1]], metadata={}]]
```

Figura 29 - Logs do microserviço “gateway”. Fonte: Autor.

É possível identificar nos logs do microserviço “cliente” que a requisição foi recebida:

Figura 30 - Logs do microserviço “cliente”. Fonte: Autor.

No log apresentado a biblioteca Spring Cloud Sleuth adicionou um Traceld a requisição, esse Traceld se repete nos outros microserviços, para que seja possível rastrear as requisições realizadas. No caso da centralização de logs isso é muito útil para identificar o fluxo de uma requisição. Utilizando mensageria com RabbitMQ o Traceld também é adicionado. No caso do log acima o Traceld gerado é: **a16cc5dae2e78717**. O formato é Nome da aplicação, Traceld e SpanId.

O microserviço “cliente” consulta o microserviço “frete” para calcular o valor de frete para o pedido, é possível identificar isso verificando os logs:

```
2022-04-06 01:51:45.500 DEBUG [frrete,ai6cc5dae2e78717,9482476bbffaa26] 19440 --> [nio-8085-exec-2] o.s.w.s.f.CommonsRequestLoggingFilter : After request [POST /api/fretes/calcular, headers->x-b3-traceid:16cc5dae2e78717, x-b3-spanid:9482476bbffaa26, x-b3-parentid:1, x-b3-sampled:1, accept:/*/*, user-agent:Java/13.0.1, host:localhost:8085, connection:'keep-alive', content-length:244, content-type:application/json;charset=UTF-8], payload=[{"cepOrigem":89801000, "cepDestino":89801000, "volumes": [{"formato": "CAIXA", "peso": 10, "unidadeMedidaPeso": "KG", "altura": 50, "largura": 50, "comprimento": 50, "quantidade": 1}], "servicosOpicionais": [{"tipo": "SEGURU", "parametros": {"valorDeclarado": 150}}]}]
```

Figura 31 - Logs do microserviço “frete”. Fonte: Autor.

Perceba que o Traceld se mantém o mesmo gerado no microserviço “cliente”. O microserviço “frete” faz uma consulta de forma opcional ao microserviço “middleware-sge”. Essa consulta é realizada apenas quando o pedido possui seguro, nesse teste foi considerado. Abaixo é possível visualizar os logs do microserviço “middleware-sge”:

```
2022-04-06 01:51:45.494 DEBUG [middleware-sge_a16cc5dae2e78717,eb8b8e64c30f6d75] 21500 --- [io-8090-exec-10] o.s.w.f.CommonsRequestLoggingFilter : After request [POST /api/middleware-sge/seguros/calcular, headers[x-b3-traceid:"a16cc5dae2e78717", x-b3-spandid:"eb8b8e64c30f6d75", x-b3-parentspaid:"9428476bbfffa26", x-b3-sampled:"1", accept:"/*", user-agent:"Java/13.0.1", host:"localhost:8090", connection:"keep-alive", content-length:"65", Content-Type:"application/json; charset=UTF-8"], payload=[{"cepOrigem":89801000, "cepDestino":89801000, "valorDeclarado":150}]]
```

Figura 32 - Logs do microserviço “middleware-sge”. Fonte: Autor.

O Traceld se mantém. Com essa requisição do “middleware-sge” devolvida ao microserviço “frete” e posteriormente ao microserviço “cliente” é possível fazer a criação do pedido. A criação do pedido pelo microserviço “cliente” envolve a criação e envio de uma mensagem para o RabbitMQ, essa mensagem é consumida pelo microserviço “pedido”. Abaixo é possível visualizar a mensagem gerada que ficou na fila esperando ser consumida (O microserviço “pedido” foi parado para que a mensagem ficasse assim):

Queues									
All queues (3)									
Pagination									
Page 1 of 1 - Filter: <input type="text"/>		<input type="checkbox"/> Regex ?							
Overview	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
pedido-atribuir-pedido-queue	classic	D	idle	0	0	0			
pedido-criar-pedido-queue	classic	D	idle	1	0	1	0.00/s	0.00/s	0.00/s
pedido-entregar-pedido-queue	classic	D	idle	0	0	0			

Figura 33 - Fila do RabbitMQ com mensagem pendente. Fonte: Autor.

Consultando a mensagem podemos identificar que se trata da requisição do usuário:

Message 1
The server reported 0 messages remaining.

Exchange	(AMQP default)
Routing Key	pedido-criar-pedido-queue
Redelivered	0
Properties	<ul style="list-style-type: none"> priority: 0 delivery_mode: 2 headers: __TypeId__: io.github.pauloorg.cliente.producer.dto.MessageCriarPedidoDTO b3: a16cc5dae2e78717-baa08e720666e673-1 content_encoding: UTF-8 content_type: application/json
Payload	586 bytes Encoding: string { "clienteId": 1, "prazoDiasUteis": 10, "custoFrete": 7.89, "nomeDestinatario": "Teste POC", "documentoDestinatario": "23.405.170/0001-51", "endereçoDestinatario": "Rua XYZ", }

Figura 34 - Payload da mensagem pendente no RabbitMQ. Fonte: Autor.

É possível identificar na mensagem que o campo “nomeDestinatario” carrega o mesmo valor que foi preenchido pelo usuário. Também podemos identificar que nos headers da mensagem existe o Traceld gerado. A mensagem está no formato JSON.

Quando o microserviço “pedido” estiver disponível novamente a mensagem é consumida. Deixando o microserviço disponível o log abaixo é gerado:

```
2022-04-06 01:59:52.802 INFO [[pedido,a16c5dae2e78717,cf73440f0b998126]] 19292 --- [ntContainer#1-1] i.g.p_pedido.consumer.PedidoProcessor : Mensagem recebida em criarPedido(... -> FormCriarPedidoDTO(clienteId=1, prazoDiasUteis=10, custoFrete=7.89, nomeDestinatario=Teste POC, documentoDestinatario=23.405.170/0001-51, enderecoDestinatario=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89901000, complemento=Complemento XYZ, contato=49 9 0000-1111), volumes=[FormVolumeDTO (formato=CAIXA, peso=10, unidadeMedidaPeso=KG, altura=50, largura=50, comprimento=50, quantidade=1)], nomeRemetente=Cliente 1, enderecoRemetente=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89901000, complemento=Complemento XYZ, contato=49 9 0000-1111))
```

Figura 35 - Logs do microserviço “pedido”. Fonte: Autor.

O Traceld se mantém. A mensagem que estava no RabbitMQ não aparece mais na fila:

Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
pedido-atribuir-pedido-queue	classic	D	idle	0	0	0			
pedido-criar-pedido-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
pedido-entregar-pedido-queue	classic	D	idle	0	0	0			

Figura 36 - Fila do RabbitMQ com mensagem consumida. Fonte: Autor.

Também podemos verificar que o pedido foi gravado no banco de dados do microserviço “pedido”:

```
select * from pedido where nome_destinatario = 'Teste POC';
```

id	cliente_id	codigo	data_emissao	data_entrega_prevista	custo_frete	nome_destinatario
12	1	b1270f9b-1fe2-48ff-acd2-4a33622-04-06 04:59:53	2022-04-06	7.89	Teste POC	

Figura 37 - Resultado da consulta SQL para a tabela “pedido”. Fonte: Autor.

O pedido também fica disponível através do endpoint abaixo:

The screenshot shows a REST API endpoint for a pedido. The URL is `http://localhost:8080/api/pedidos/codigo?codigo=b1270f9b-1fe2-48ff-acd2-4a33606913d7`. The response body is a JSON object:

```
{
  "codigo": "b1270f9b-1fe2-48ff-acd2-4a33606913d7",
  "dataAmissao": "2022-04-06T01:59:53",
  "dataEntregaPrevista": "2022-04-06",
  "custoFrete": 7.89,
  "nomeDestinatario": "Teste POC",
  "enderecoDestinatario": "23.405.170/0001-51",
  "enderecoDestinatario": {
    "endereco": "Rua XYZ",
    "cidade": "Chapéco",
    "estado": "SC",
    "cep": 89801000,
    "complemento": "Complemento XYZ",
    "contato": "49 9 0000-1111"
  }
}
```

Figura 38 - Consulta do pedido no endpoint GET /api/pedidos/codigo?. Fonte: Autor.

Como apresentado na imagem acima, os dados são devolvidos no formato JSON.

Podemos identificar com essas evidências que a aplicação possui um conjunto de microserviços que trabalham juntos para completar uma determinada requisição do usuário. Esses microserviços se comunicam entre si utilizando REST APIs ou mensageria. Também, podemos visualizar que o fluxo das requisições pode ser rastreado através do Traceld gerado para a primeira requisição.

5.3.2. Cenário 2

Atributo de Qualidade:	Segurança
Requisito de Qualidade:	O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades
Preocupação:	Os usuários do sistema devem se autenticar por meio de login com usuário e senha para poder utilizar os endpoints disponíveis. Cada usuário possui um grupo de perfis, que da acesso a determinados endpoints. Apenas usuários que possuem o perfil esperado podem executar requisições nos endpoints
Cenário(s):	Cenário 2
Ambiente:	Sistema em operação normal
Estímulo:	Dois usuários com perfis distintos tentam se autenticar na API e realizar requisições para determinado endpoint
Mecanismo:	Toda requisição passa pelo API Gateway, que realiza o roteamento da mesma para o microserviço adequado. Além disso, o Gateway centralizada detalhes da autenticação, funcionando como um OAuth2 Client para o serviço Keycloak. O Keycloak gerencia os usuários e suas permissões, podendo gerar Tokens de acesso após um login bem sucedido. Os Tokens gerados carregam as permissões do usuário.
Medida de resposta:	A requisição que foi repassada pelo Gateway ao microserviço adequado carrega o Token gerado, e o microserviço atua como um OAuth2 Resource Server, que consegue consultar esse Token e tratar detalhes de segurança para cada endpoint. Esses endpoints podem ser protegidos, restringindo o acesso a usuários autenticados e com determinadas permissões.
* Os usuários precisam se autenticar no sistema para utilizar os endpoints expostos pelo API Gateway	

- * Quando não autenticado o usuário não consegue fazer requisições e deve ser redirecionado para a tela de login do Keycloak caso esteja no navegador
- * Os usuários autenticados possuem perfis e podem fazer requisições para endpoints liberados para o seu perfil
- * Se o usuário tentar fazer uma requisição a um endpoint bloqueado para o seu perfil deve ser retornado uma mensagem de erro
- * O Token de acesso gerado para o usuário deve conter no payload as permissões

Fluxo para as evidências:

Usuário “parceiro” com perfil “parceiro” tenta acessar a documentação da API sem autenticação, o acesso deve ser bloqueado e o usuário redirecionado a tela de login do Keycloak. O usuário precisa informar os dados de login corretos e após um sucesso no login um Token de acesso é gerado para o usuário. É possível identificar nas sessões do Keycloak que o usuário está logado. O Token gerado para o usuário deve estar no formato JWT e o payload do mesmo deve carregar as permissões do usuário. Para o usuário em questão a permissão deve ser o perfil “parceiro”.

O usuário “parceiro”, já autenticado, tenta fazer uma requisição ao endpoint de criação de pedidos, e deve ser bloqueado, pois o endpoint permite a criação de pedidos apenas por usuários com o perfil “cliente”.

O usuário “cliente” com perfil “cliente”, já autenticado, tenta fazer uma requisição ao endpoint de criação de pedidos, a requisição não deve ser bloqueada, e o usuário consegue criar um pedido.

Considerações sobre a arquitetura:

Riscos:	Se os Tokens não forem armazenados em um local seguro os mesmos podem ser expostos a um usuário malicioso. A utilização de HTTPS nas APIs é importante também para evitar esse vazamento
Pontos de Sensibilidade:	Utilização de HTTPS nas APIs expostas

	ao usuário e armazenar corretamente os Tokens no Front-end
Tradeoff:	Não há

5.3.2.1. Evidências

O Keycloak foi utilizado como sistema de gerenciamento de identidade, nele conseguimos gerenciar um Realm, que é um espaço dentro do Keycloak que armazena clientes, usuários, permissões dos usuários e outras configurações. O Keycloak realiza a geração de Tokens para os usuários. No contexto da POC o API Gateway é um cliente do Keycloak, que realiza a conexão com o Keycloak para poder gerar Tokens para os usuários. Os Tokens gerados estão no formato JWT.

Para a POC apenas o API Gateway foi considerado como cliente, mas podem existir mais clientes, como o Front-end ou um parceiro, que precisa se integrar na API do GSL. Os demais microsserviços atuam como OAuth2 Resource Server, pois recebem do API Gateway o Token utilizado na requisição, podendo assim, fazer tratamentos de segurança nos endpoints.

Para realizar os testes dois usuários foram cadastrados no Keycloak:

ID	Username	Email	Last Name	First Name
4137cd9-9b49-4b8a-a906-b5d7f9d...	cliente		X	Cliente
349e92d2-377b-4e4c-8c86-356b423...	parceiro		X	Parceiro

Figura 39 - Usuários cadastrados no Keycloak. Fonte: Autor.

O usuário “cliente” possui o perfil “cliente” e o usuário “parceiro” possui o perfil “parceiro”:

Roles > cliente

Cliente

Details Attributes Users in Role

Username	Last Name	First Name
cliente	X	Cliente

Roles > parceiro

Parceiro

Details Attributes Users in Role

Username	Last Name	First Name
parceiro	X	Parceiro

Figura 40 - Perfis para os usuários “cliente” e “parceiro”. Fonte: Autor.

O usuário “parceiro” tenta acessar a documentação da API usando a URL: <http://localhost:8080/documentacao-api/docs.html>, mas por não estar autenticado é redirecionado a tela de login do Keycloak:

SISTEMAGSL

Sign in to your account

Username or email

Password

Sign In

Status: 302 Found

Método: GET

Dominio: localhost:9000

Arquivo: auth?response_type=code&client_id=gateway&state=FG3p5Y-Ov63ltv5vAEDc5_NM

Iniciador: document

Tipo: html

Transferido: 3,88 KB

Tamanho: 3,58 KB

Cabeçalhos: Filterar cabeçalhos

Requisição: GET http://localhost:8080/documentacao-api/docs.html

Resposta: Status: 302 Found

Versão: HTTP/1.1

Transferido: 3,88 KB (tamanho 3,58 KB)

Figura 41 - Tela de login do Keycloak após tentativa de acesso sem autenticar. Fonte: Autor.

É possível identificar pelos logs de Rede da imagem acima que a URL descrita foi acessada primeiro, mas o usuário não estando autenticado o status retornado foi 302 e ocorre um redirecionamento para a URL de login do Keycloak. O API Gateway está executando na porta 8080, enquanto o Keycloak na 9000.

O usuário “parceiro” precisa informar o login e senha, no teste foi utilizado “parceiro” e “123”. Quando logado podemos identificar que o redirecionamento ocorre normalmente para a URL requisitada inicialmente:

Status	Método	Domínio	Arquivo	Iniciador	Tipo	Transferido	Tam...
302	POST	localhost:9000	authenticate?session_code=xJjgeOlbboIckyG5is_8pfri5YSrjhqk1tV8sM30&execution=79c	document	html	4,40 KB	1,43 KB
302	GET	localhost:8080	keycloak/state=Fg5pST-Ov63ltv5vAEDc5_NMaKGSyZGfFimr7Zedp8t=&session_state=717C	document	html	1,80 KB	1,43 KB
302	GET	localhost:8080	docs.html	document	html	1,89 KB	1,43 KB
200	GET	localhost:8080	index.html?configUrl=/documentacao-api/v3/api-docs/swagger-config	document	html	1,88 KB	1,43 KB
200	GET	localhost:8080	swagger-ui.css	stylesheet	css	140,74 KB	140,2...
200	GET	localhost:8080	swagger-ui-bundle.js	script	js	1,05 MB	1,05 ...
200	GET	localhost:8080	swagger-ui-standalone-preset.js	script	js	329,09 KB	328,6...
200	GET	localhost:8080	favicon-16x16.png	FaviconLoader.jsm:191 (img)	png	1,10 KB	665 B
200	GET	localhost:8080	swagger-config	swagger-ui-bundle.js:2 (fetch)	json	1,24 KB	865 B
200	GET	localhost:8080	swagger.yaml	swagger-ui-bundle.js:2 (fetch)	octet-stream	6,31 KB	5,91 KB

Figura 42 - Requisição de login enviada ao Keycloak. Fonte: Autor.

Na imagem acima é possível visualizar que foi realizado um POST para o Keycloak, repassando os dados de login do usuário. O Keycloak validou esses dados e autenticou o usuário, gerando um Token para ele. É possível visualizar que o usuário possui uma sessão ativa no Keycloak:

User	From IP	Session Start
parceiro	172.25.0.1	Apr 6, 2022 11:54:30 PM

Figura 43 - Sessão do usuário “parceiro” no Keycloak. Fonte: Autor.

Ativando o nível de log TRACE no API Gateway conseguimos identificar que o mesmo recebeu a requisição do usuário “parceiro” e seu Token de acesso:

```
2022-04-07 00:12:50.948 TRACE 25028 --- [ctor-<http-nio-2> o.s.http.codec.json.Jackson2JsonDecoder : [177cdab] [a8f98701-01, l:/127.0.0.1:58356 - R:localhost/127.0.0.1:9000]
Decoded {[<access_token>=eyJhbGciOiJSUzIiNiIsInRcCigoiAisIdUwiwia2kIiA6ICjXcUhKNGrzVksQRURCNfFcHpdGJSVzjYkZn0VRcehVrkpJNw9ZmRjIn0
,eyJleHAiojE2ND0kzMD0NzAsImhdIC6MTy0tMwMTE3MCwiYXVoF90aw1ljoXnjQ5MzAwMDcwLCjdgk1o14ZTcsY2RLmc00NzE1LTQ1NGItYmJmNy0yZGZjMDQ3M2U1YWMiLcJpc3M1o1JodHRwO18vbG9jYwxbob3N0OjkwMDA
vYXVoF90aw1c9yZWFsbXMu21zdGvTyUdTCIsImF1ZC16ImFjY291bn0iCjzdW10i1zN0lhotJkMi0zNzdiLTr1NGMtOGM4Ni0zNTz1NDiZNGNiYmUiCj0eXai0i1cZWFyZXiiCjhenAi0iJnYXRld2F5TiwiC2VzC2lv19zdGf0ZS1
6ijdmN2M3MjilThlyWtNG1iYs1h2mQ1l0tkyMTj3Lvh0tR1z1iisMjci0i6ijAicLjcyZwfbsv9h2Nlc3MiOnsicm9sZXMi0lsib2ZmbGluvshy2Nlc3MiLCjwYxjZwlyby1sInVtYv9hdXRob3JpemF0a9uUiwiZGVyXVsdc1
yb2xlcylzaXN0Zwh3NsIl19LcJyZXvndxjzV9h2Nlc3MionsiyWnjb3vudCteyjzb2xlcylgyjWmzDUT0tIXmVmzWE5NzWmIiwlwmdvawPzWqoInRydwsUsIm5hWuijoiQxjZwlyby1yIwiCh1lZmVycmVxVzXJuWm1ljoicGfyY2Vpcm8iCjnaXz
lbl9uWw1ljoiuGFyy2Vpcm8iLcjmYw1pbhfbmFTzsI61glgfQ
,`.Vtp4MyQhQzeoSTLB-Vz1pf_Q7PrwMs1Yo8Cz190RcDKRz_oHiptY8f3FhvcFFhx50a2m3crd7TSefgPmcMcazz_ZyCr0j97WWhk5nXqcWAhzAgdvTjKbf_TrA_NF1bGNBZ8a
,`.Plwo4k_hHhPE50SOWYOSrUYxe_ApKn3h50EpQ0F
,`.Plwo4k_hHhPE50SOWYOSrUYxe_ApKn3h50EpQ0F
,Tg, expires_in=300, refresh_expires_in=1800, refresh_token=eyJhbGciOiIUz1N1iisInRscCigoiAisIdu1iia2lktiA61CjJmDEzyTAAC0C032DFjlTRkndEYTAZMS0iNj10B1MTg50dEifq
,`.eyJleHAiojE2ND0kzMD1NzAsImhdIC6MTy0tMwMTE3MCwianRpjoiM04ZTY1NzYzViin00MM1LwjmNGEtNxhKNT20Tj1iwiXNz1joiAhR0cDov2xvY2FsaaG9zd0b5MDAwL2F1dgGvcmBhGz1Lnpc3RlbwFHu0w
,`.iLCJhdQ1o1jodRw018bw9jYxob3N0ojkwMDAvYXXv0c9yZWFsbXxvU21zdGvTyUdTTci1iN1Y1i61jM00wESMmQyLTM3N2ItNGU0y94yzg2LTM1N1M0jM0Y21z1s1sInRscC161j1j1z1j1c2g1LjhenAi0iJnYxrld2f51i
ic2Vzc21vb19zdGf0ZS1Gj1dNmZm3MjilThlyWtNG1iYs1h2m01TkyT1lzmh0tC1z1s1sInRjbm81j1joiChJvZmlsZsB1bwFbcTsInRpzC161jdmw2M3z1lThlyWtNG1iYs1h2mQ1l0tkyMTj1ZmVh0tC1z1j9
,`.Px4Ch131wf0Rip3IafZgyPl0c9fd0MMF_ZEAT2cvo, token_type=Bearer, not-before-policy=0, session_state=7f7c7fbb-8eae-4b5a-afds-9212efea975f, scope=profile email]
```

2022-04-07 00:12:50.974 DEBUG 25028 --- [ctor-<http-nio-2> ebsessionServerSecurityContextRepository : Saved SecurityContext 'SecurityContextImpl' [Authentication=OAuth2AuthenticationToken [PrincipalName: [parceiro], Granted Authorities: [<ROLE_USER>, <SCOPE_email>, <SCOPE_profile>]], User Attributes: [<{name=parceiro, family_name=X}>], Credentials=[PROTECTED], Authenticated=true, details=null, Granted Authorities=[<ROLE_USER>, <SCOPE_email>, <SCOPE_profile>]]' in WebSession: 'org.springframework.web.server.session .InMemoryWebSessionStore\$InMemoryWebSession@5c61966c'

Figura 44 - Logs do microserviço “gateway” com Token de acesso. Fonte: Autor.

O Token de acesso está no formato JWT, e podemos visualizar o payload deste Token:

```
i0iI4ZTc5Y2R1MC00NzE1LTQ1NGItYmJmNy0yZG
ZjMDQ3M2U1YWMiLCJpc3Mi0iJodHRwO18vbG9jY
Wxob3N0OjkwMDAvYXV0aC9yZWFsbXvMvU21zdGVt
YUdTTCIsImF1ZC16ImFjY291bnQilCJzdW10iI
zND1h0TjkMi0zNzdiLTr1NGMtOGM4Ni0zNTz1ND
IzNGNiYmUiLCJ0eXai0iJCZWfyzXiilCJhenAi0
iJnYXR1ld2F5IiwiC2Vzc21vb19zdGF0ZSi6Ijdm
N2M3ZmJilThlyWtNG1iYs1h2mQ1LTkyMTj1ZmV
h0Tc1z1sImFjci16IjAiLCJyZWFsbv9h2N1c3
MiOnsicm9sZXMi0lsib2ZmbGluvshy2Nlc3MiL
CJwYXJjZwlybyIsInVtYV9hdXRob3JpemF0a9u
IiwiZGVmYXVsdC1yb2xlcylzaXN0Zw1h2Z3NsIl1
9LCJyZXNvdXjJzV9h2Y2N1c3Mi0nsiYWNjb3VudC
I6eyJyb2xlcyl6WyJtYw5h2Z2UtYWNjb3VudCisI
m1hbmFnZs1hY2NvdW50LWxpbtmtzIiwidm1ldy1w
cm9maWx1l19fSwic2NvcGUi0iJwcm9maWx1IGV
tYw1sIiwiC2lkIjoiN2Y3YzdmYmIt0GVhZs00Yj
VhLWFmZDUtOTiXmmVmZWE5NzVmIiwiZw1haWxf
mVyaWzPZwQoInRydwsUsIm5hbwUo1iQYXJjZwly
byBYIiwiChJ1ZmVycmVxK3VzXJuYw1l1joiUGFyY2Vpcm
8iLCJmYW1pbhfbmFTzsi6IlgifQ.YTbp4NyqHQ
zeOsTLB-
VziPf_Q7PrwMs1Yo8Cz190RcDKRzjmpNhC4Ayuh
5zNcXh8Kwi-
P061DPu_oHi1ptY8f3FhvcFFhx50a2m3crd7TS
eFgPmcMcazz_ZyCr0j97WWhk5nXqcWAhzAgdvTj
Kbf_TrA_NF1bGNBZ8a-
Plwo4k_hHhPE50SOWYOSrUYxe_ApKn3h50EpQ0F
```

Figura 45 - Payload do Token de acesso - Site <https://jwt.io/>. Fonte: Autor.

Na imagem acima é possível identificar que o Token é do usuário “parceiro” e que o mesmo possui o perfil “parceiro”. Esses dados de permissão são utilizados para fazer o bloqueio dos endpoints.

Com o usuário “parceiro” logado e o perfil “parceiro” atribuído conseguimos testar se os endpoints estão fazendo o bloqueio por permissão. Para esse teste vamos utilizar o endpoint <http://localhost:8086/api/clientes/{id}/pedidos> referente a criação de pedidos para um cliente. Apenas usuários com perfil “cliente” podem criar um pedido. Se o usuário “parceiro” tenta criar um pedido para o cliente de id 1, o seguinte erro ocorre:

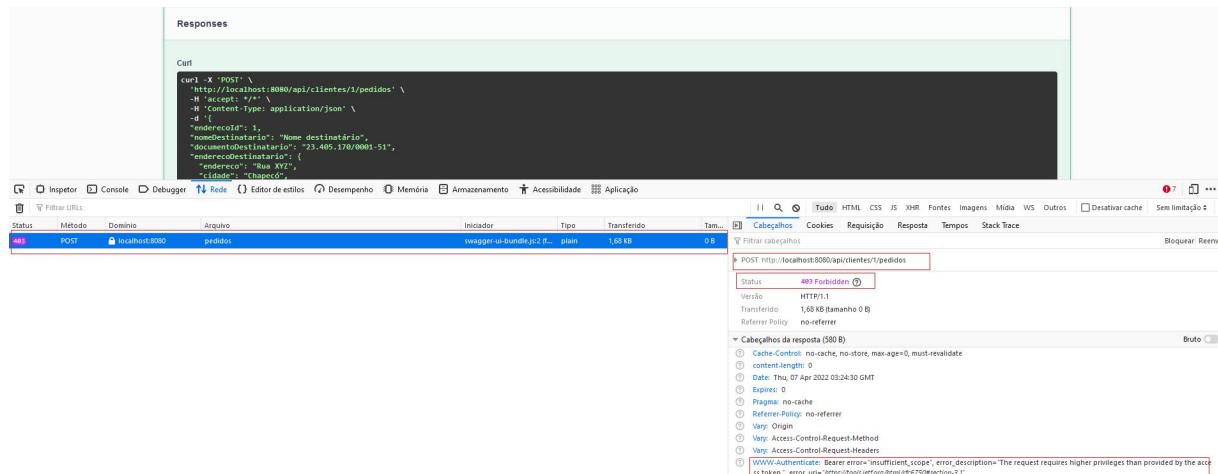


Figura 46 - Requisição negada por falta de permissão. Fonte: Autor.

É possível identificar que a requisição foi bloqueada e o status retornado é 403. Nos headers da response é possível visualizar que uma mensagem de erro é devolvida, informando que o usuário precisa de permissão para acessar o recurso. A nível de código a permissão é adicionado no método do controller:

```
@RolesAllowed({"cliente"})
@PostMapping("/{id}/pedidos")
```

Figura 47 - Trecho de código do controller “ClienteController” do microserviço “cliente”. Fonte: Autor.

Para conseguir realizar a requisição precisamos logar com o usuário “cliente”. O login do usuário “cliente” é realizado e a sessão no Keycloak criada:

User	From IP	Session Start
cliente	172.25.0.1	Apr 7, 2022 12:29:12 AM

Figura 48 - Sessão do usuário “cliente” no Keycloak. Fonte: Autor.

Realizando a requisição novamente temos outra resposta:

Status	Método	Dominio	Arquivo	Iniciador	Tipo	Transferida	Tam.
201	POST	localhost:8080	pedidos	swaggen-ui-bundle.js:2 f...	plain	1,48 KB	0 B

Figura 49 - Requisição com sucesso para POST /api/clientes/1/pedidos. Fonte: Autor.

É possível identificar na imagem acima que a requisição foi realizada e status de retorno é 201.

As evidências apresentadas mostraram que o API Gateway está protegido utilizando OAuth2 com o Keycloak e que usuários precisam se autenticar para utilizar as APIs expostas pelo Gateway. Também, foi apresentado que o tratamento de permissão de acesso aos recursos a nível de perfil é realizado, como apresentado no acesso ao endpoint de criação de pedidos para o cliente.

5.3.3. Cenário 3

Atributo de Qualidade:	Confiabilidade
Requisito de Qualidade:	O sistema deve ser recuperável no caso da ocorrência de erro
Preocupação:	<p>Os microserviços devem utilizar duas réplicas, que devem ser acessadas conforme a estratégia de load balancer configurada. Se um microserviço cair o outro ativo deve responder normalmente, de forma transparente para o usuário. Se os microserviços envolvidos que consomem mensagens do RabbitMQ não estão ativos, as mensagens devem permanecer na fila e serem consumidas logo que uma réplica estiver disponível</p>
Cenário(s):	Cenário 3
Ambiente:	Sistema em operação normal
Estímulo:	Usuário realiza uma requisição a um serviço que possui uma réplica indisponível ou um serviço dependente indisponível
Mecanismo:	<p>Os microserviços da arquitetura GSL devem estar containerizados por meio do Docker. O deploy desses containers deve ser feito no Kubernetes. O ideal é que cada Pod contenha apenas uma imagem. As réplicas desses Pods devem estar configuradas conforme a necessidade de cada microserviço.</p> <p>Todos os serviços Kubernetes referentes aos microserviços devem ser do tipo ClusterIP, exceto o API Gateway e Front-end ou algum outro serviço que deve ser disponibilizado para fora do cluster, esses serviços devem ser do tipo LoadBalancer. Deve ser utilizado o Nginx Ingress Controller, para atuar como um Proxy Reverso para esses serviços do tipo LoadBalancer.</p>
	<p>Alguns serviços, como banco de dados e outros, ficaram fora do cluster Kubernetes, reutilizando a infraestrutura atual da Boa Entrega, portanto, é necessário que seja configurado os serviços para mapear essa conexão para dentro do cluster.</p>

Questões de registro e descoberta de serviços devem ocorrer por meio do Kubernetes, portanto, não é necessário utilizar bibliotecas como o Spring Cloud Eureka. A ideia é manter um ambiente mais poliglota, pensando que serviços do sistema legado SGE, escritos em PHP, possam se integrar nessa arquitetura.

Questões de alta disponibilidade ficam a cargo do Kubernetes, podendo ser escalado os microsserviços conforme necessidade. É importante considerar o uso do Actuator, nos microsserviços co Spring Boot, para disponibilizar informações de análise dos microsserviços, podendo assim, ser verificado se os microsserviços estão disponíveis e coletar outras métricas.

O cluster Kubernetes deve conter apenas serviços stateless, serviços stateful devem ficar, preferencialmente, fora do cluster. Para que o impacto de uma queda em um microsserviço seja o menor possível.

Além das réplicas, os microsserviços devem utilizar mensageria com RabbitMQ nos casos onde é necessário uma comunicação assíncrona e garantia de execução das requisições realizadas pelo usuário.

Medida de resposta:

- * Um microsserviço dependente que está indisponível deve processar as mensagens pendentes na fila do RabbitMQ assim que estiver disponível
- * Uma réplica indisponível de algum microsserviço deve ser recriada de forma automática e estar disponível em no máximo 1 minuto (Considerando o tempo de inicialização da aplicação Spring Boot)
- * Um microsserviço operando com duas réplicas deve continuar atendendo, de forma transparente, as requisições dos usuários quando uma das réplicas ficar indisponível

Fluxo para as evidências (Mensageria com RabbitMQ):

Usuário logado com perfil de “cliente” realiza a criação de 3 pedidos, mas o microsserviço “pedido” está indisponível, nenhuma réplica pode processar as mensagens. Essas mensagens devem ficar no RabbitMQ pendentes de

processamento.

O microserviço “pedido” deve ser inicializado e as mensagens pendentes consumidas. Os pedidos criados devem ficar disponíveis para consulta assim que forem criados.

Fluxo para as evidências (Réplica no Kubernetes):

Um microserviço acessando recursos externos ao cluster deve estar disponível com duas réplicas. Deve ser possível visualizar que as duas réplicas estão ativas e visualizar os logs das mesmas.

O usuário deve realizar algumas requisições para o microserviço, essas requisições devem ser feitas passando pelo Ingress Controller, que deve repassar ao API Gateway e posteriormente ao microserviço que deve atender essa requisição.

Consideramos as seguintes réplicas:

R1 e R2

NR1 e NR2 (Representam novas réplicas recriadas)

Enquanto o usuário realiza requisições, a réplica R1 deve ser removida, o usuário deve continuar fazendo requisições e as requisições devem continuar sendo processadas pela réplica R2. Deve ser possível visualizar que a réplica R1 foi recriada e agora é NR1 e que em no máximo um minuto está disponível. Como o usuário está fazendo requisições para a réplica R2, deve ser removido a mesma e o usuário deve continuar fazendo as requisições e elas devem ser processadas pela réplica RN1.

Considerações sobre a arquitetura:

Riscos:

Um serviço externo ao cluster sem um gerenciamento adequado pode ficar indisponível e comprometer o sistema como um todo.

Se serviços stateful forem considerados

	e não configurados de forma adequada poderá ocorrer a perda de alguma informação para os usuários.
Pontos de Sensibilidade:	<p>Serviços externos ao cluster devem estar disponíveis.</p> <p>Microserviços devem ser stateless, para evitar maiores problemas durante a parada de algum microserviço.</p> <p>Ferramentas de gerenciamento são importantes para verificar a saúde dos microserviços e coletar outras métricas.</p>
Tradeoff:	O Kubernetes pode gerar uma complexidade para o gerenciamento dos microserviços, visto que é necessário pessoas especializadas em DevOps, entretanto, o uso dele é benéfico, pois é possível manter uma hospedagem em nuvem híbrida de forma facilitada e muitos cloud providers já fornecem a hospedagem com Kubernetes, como AWS e Azure. Questões de alta disponibilidade e escalabilidade são pontos importantes a serem considerados também. Outras questões como manter um ambiente poliglota também são pontos positivos, visto que é possível utilizar alguns recursos do Kubernetes para resolver problemas de descoberta e registro de serviços, centralização/externalização de configuração e questões relacionais a resiliência com utilização de um service mesh, sem adicionar esses elementos de infraestrutura no código da aplicação,

	pois eles ficam a nível do container e os microserviços centralizam principalmente regras de negócio sem muitos detalhes de infraestrutura.
--	---

5.3.3.1. Evidências

Mensageria com RabbitMQ:

O Message Broker escolhido para a arquitetura do sistema GSL foi o RabbitMQ. Para o cenário descrito foram criados dois microserviços, o microserviço de “cliente” e o de “pedido”. O microserviço de “cliente” permite que clientes criem pedidos, esses pedidos são enviados ao microserviço de “pedido” através de uma fila configurada no RabbitMQ. Essa fila pode ser visualizada na imagem abaixo:

Queues									
All queues (3)									
Pagination									
Page		1	-	of 1	- Filter:	<input type="text"/>	<input type="checkbox"/> Regex	?	
Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
pedido-atribuir-pedido-queue	classic	D	idle	0	0	0			
pedido-criar-pedido-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
pedido-entregar-pedido-queue	classic	D	idle	0	0	0			

Figura 50 - Fila “pedido-criar-pedido-queue” criada no RabbitMQ. Fonte: Autor.

Outras filas também foram criados, porém, para essa evidência será utilizado apenas a fila “pedido-criar-pedido-queue”.

Para que a mensagem fique pendente no RabbitMQ é necessário que o microserviço “pedido” esteja indisponível. Como podemos ver na imagem abaixo, consultando o Actuator, o microserviço está disponível:

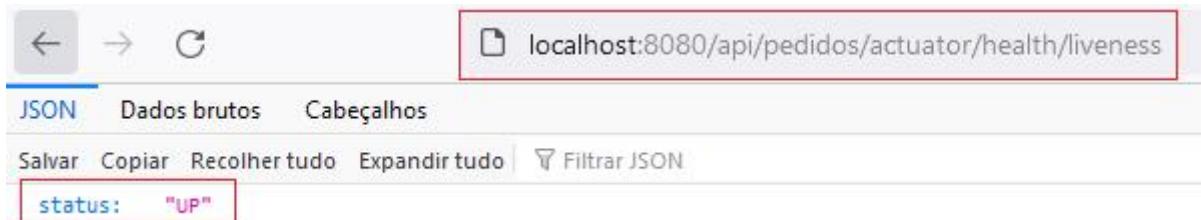


Figura 51 - Consulta a um endpoint do Actuator do microserviço “pedido”. Fonte: Autor.

Derrubando o microserviço podemos consultar essa mesma URL novamente e agora não é mais possível acessar o serviço:



Figura 52 - Consulta ao mesmo endpoint da Figura 51, mas sem resposta. Fonte: Autor.

Essa resposta da imagem acima é o resultado do API Gateway tentando acessar o serviço.

O microserviço “cliente” está disponível e será utilizado para a criação de três pedidos, todos contendo o mesmo request body:

The screenshot shows the Network tab of a browser's developer tools. A red box highlights the curl command in the top section, which is used to create a new order. Another red box highlights the first row in the table below, which shows a successful POST request with status 201.

```
Curl
curl -X 'POST' \
  'http://localhost:8080/api/clientes/1/pedidos' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "enderecold": 1,
    "nomeDestinatario": "Nome destinatário",
    "documentoDestinatario": "23.405.170/0001-51",
    "enderecoDestinatario": {
      "endereco": "Rua XYZ",
      "cidade": "Chapecó",
      "estado": "SC",
      "cep": 89801000,
      "complemento": "Complemento XYZ",
      "contato": "49 9 0000-1111",
      "clienteId": 1
    },
    "volumes": [
      {
        "formato": "CAIXA",
        "peso": 10,
        "unidadeMedidaPeso": "KG",
        "altura": 50,
        "largura": 50,
        "comprimento": 50,
        "quantidade": 1
      }
    ],
    "servicosOpcionais": [
      {
        "tipo": "SEGURO",
        "parametros": {
          "valorDeclarado": 150
        }
      }
    ]
}'
```

Status	Método	Domínio	Arquivo
201	POST	localhost:8080	pedidos
201	POST	localhost:8080	pedidos
201	POST	localhost:8080	pedidos

Figura 53 - Três requisições de criação de pedido no microserviço “cliente”. Fonte: Autor.

É possível identificar na imagem acima que os três pedidos foram criados, todas as respostas das requisições foram devolvidas com status http 201.

Nos logs do microserviço “cliente” é possível visualizar que as requisições foram recebidas e um Traceld foi gerada para cada uma:

```

2022-04-07 21:20:20.546 DEBUG [cliente,7619855afbd2b839,7619855afbd2b839] 22700 --- [nio-8086-exec-8] o.s.w.f.CommonsRequestLoggingFilter : After request [POST
/api/clientes/1/pedidos, headers=[user-agent:"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0", accept:"/*", accept-language:"pt-BR,pt;q=0.8,
2022-04-07 21:20:25.094 DEBUG [cliente,fa65e762ebbeb0e3,fa65e762ebbeb0e3] 22700 --- [nio-8086-exec-9] o.s.w.f.CommonsRequestLoggingFilter : After request [POST
/api/clientes/1/pedidos, headers=[user-agent:"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0", accept:"/*", accept-language:"pt-BR,pt;q=0.8,
2022-04-07 21:20:26.386 DEBUG [cliente,cd7a4282d46813ad,cd7a4282d46813ad] 22700 --- [io-8086-exec-10] o.s.w.f.CommonsRequestLoggingFilter : After request [POST
/api/clientes/1/pedidos, headers=[user-agent:"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0", accept:"/*", accept-language:"pt-BR,pt;q=0.8,

```

Figura 54 - Logs das requisições de criação de pedido com Traceld gerado. Fonte: Autor.

Visualizando a interface gráfica do RabbitMQ podemos ver que as três mensagens referentes a criação dos pedidos estão pendentes na fila:

Queues									
All queues (3)									
Pagination									
Page 1 of 1 - Filter: <input type="text"/> <input type="checkbox"/> Regex ?									
Overview									
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
pedido-atribuir-pedido-queue	classic	D	idle	0	0	0			
pedido-criar-pedido-queue	classic	D	idle	3	0	3	0.00/s	0.00/s	0.00/s
pedido-entregar-pedido-queue	classic	D	idle	0	0	0			

Figura 55 - Mensagens pendentes na fila “pedido-criar-pedido-queue”. Fonte: Autor.

Buscando as mensagens podemos identificar que elas possuem o mesmo Traceld nos headers:

Properties	priority: 0 delivery_mode: 2 headers: __TypeId__: io.github.paulooorg.cliente.producer.dto.MessageCriarPedidoDTO b3: 7619855afbd2b839-83c72541b02e4f26-1 content_encoding: UTF-8 content_type: application/json
Payload	595 bytes Encoding: string
Message 2	
The server reported 1 messages remaining.	
Exchange	(AMQP default)
Routing Key	pedido-criar-pedido-queue
Redelivered	0
Properties	priority: 0 delivery_mode: 2 headers: __TypeId__: io.github.paulooorg.cliente.producer.dto.MessageCriarPedidoDTO b3: fa65e762ebcd0e3-a7dc1bb28aafe59-1 content_encoding: UTF-8 content_type: application/json
Payload	595 bytes Encoding: string
Message 3	
The server reported 0 messages remaining.	
Exchange	(AMQP default)
Routing Key	pedido-criar-pedido-queue
Redelivered	0
Properties	priority: 0 delivery_mode: 2 headers: __TypeId__: io.github.paulooorg.cliente.producer.dto.MessageCriarPedidoDTO b3: cd7a4282d46813ad-8b480c7808f25661-1

Figura 56 - Mensagens pendentes da fila “pedido-criar-pedido-queue” com os Tracelds. Fonte: Autor.

Para que essas mensagens sejam processadas é necessário subir o microserviço “pedido”, com ele disponível as mensagens serão processadas. Após subir o microserviço as mensagens foram processadas e os logs abaixo gerados:

```
2022-04-07 21:38:18.330 INFO [pedido,761985afbd2b839,33d08470822273f7] 28804 --- [ntContainer#0-1] i.g.p.pedido.consumer.PedidoProcessor : Mensagem recebida em criarPedido(..) -> FormCriarPedidoDTO(clienteid=1, prazoDasUteis=10, custoFrete=7.89, nomeDestinatario=Nome destinatário, documentoDestinatario=23.405.170/0001-51, enderecoDestinatario=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111), volumes=[FormVolumeDTO(formato=CAIXA, peso=10, unidadeMedidaPeso=KG, altura=50, largura=50, comprimento=50, quantidade=1)], nomeRemetente=Cliente 1, enderecoRemetente=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111))

2022-04-07 21:38:18.446 INFO [pedido,f465e762ebbd0e3,187763628cd4336e] 28804 --- [ntContainer#0-1] i.g.p.pedido.consumer.PedidoProcessor : Mensagem recebida em criarPedido(..) -> FormCriarPedidoDTO(clienteid=1, prazoDasUteis=10, custoFrete=7.89, nomeDestinatario=Nome destinatário, documentoDestinatario=23.405.170/0001-51, enderecoDestinatario=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111), volumes=[FormVolumeDTO(formato=CAIXA, peso=10, unidadeMedidaPeso=KG, altura=50, largura=50, comprimento=50, quantidade=1)], nomeRemetente=Cliente 1, enderecoRemetente=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111))

2022-04-07 21:38:18.480 INFO [pedido,cd7a4282d46813ad,d9c3e73183014829] 28804 --- [ntContainer#0-1] i.g.p.pedido.consumer.PedidoProcessor : Mensagem recebida em criarPedido(..) -> FormCriarPedidoDTO(clienteid=1, prazoDasUteis=10, custoFrete=7.89, nomeDestinatario=Nome destinatário, documentoDestinatario=23.405.170/0001-51, enderecoDestinatario=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111), volumes=[FormVolumeDTO(formato=CAIXA, peso=10, unidadeMedidaPeso=KG, altura=50, largura=50, comprimento=50, quantidade=1)], nomeRemetente=Cliente 1, enderecoRemetente=FormEnderecoDTO(endereco=Rua XYZ, cidade=Chapecó, estado=SC, cep=89801000, complemento=Complemento XYZ, contato=49 9 0000-1111))
```

Figura 57 - Processando das mensagens pendentes pelo microserviço “pedido”. Fonte: Autor.

Na imagem acima é possível visualizar que as três mensagens foram recebidas e possuem o mesmo Traceld. Esses três pedidos podem ser consultados pela API e consequentemente estão gravados no banco de dados:

The screenshot shows a MySQL Workbench interface. In the top-left, there's a code editor with the SQL query:

```
select * from pedido order by id desc limit 3;
```

In the bottom-right, the results of the query are displayed in a table:

	id	cliente_id	codigo	data_emissao
1	16	1	1df141e6-fa7c-4532-9368-f4e822fa50ef	2022-04-08 00:38:18
2	15	1	f9d5a51a-5979-4070-a607-2b426305c056	2022-04-08 00:38:18
3	14	1	fbcc71dc-354f-4aab-bae7-d32f6bcda318	2022-04-08 00:38:18

Figura 58 - Resultado da consulta SQL com os três pedidos criados. Fonte: Autor.

Réplica no Kubernetes:

Os microserviços da arquitetura GSL são containerizados e o deploy é realizado no Kubernetes. Para coletar os dados para essa evidência foi utilizado uma implementação do Kubernetes chamada Minikube, essa implementação permite subir localmente um cluster de Kubernetes de forma fácil. O Minikube fornece um Dashboard, onde podemos visualizar de forma facilitada as informações sobre os Pods, Serviços, Logs dos serviços, dentre outras informações.

Os serviços envolvidos nessa evidência são os seguintes:

The screenshot shows the Kubernetes Services page. On the left, there's a sidebar with 'Workloads' (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets) and 'Service' (N). Below that is an 'Ingresses' section. The main area is titled 'Services' and lists five entries:

Name	Namespace	Labels	Type
k8s-poc-documentacao-api	default	-	ClusterIP
k8s-poc-deposito	default	-	ClusterIP
mysqlDb	default	-	ExternalName
k8s-poc-gateway	default	-	LoadBalancer
kubernetes	default	component: apiserver provider: kubernetes	ClusterIP

Figura 60 - Serviços ativos no cluster Kubernetes. Fonte: Autor.

A seguir uma descrição mais detalhada dos serviços:

- **k8s-poc-documentacao-api**: Microserviço em Spring Boot que agrupa a documentação de todos os outros microserviços;
- **k8s-poc-deposito**: Microserviço em Spring Boot responsável pelo gerenciamento de depósitos;
- **mysqlDb**: Serviço que faz o mapeamento de um serviço externo (Banco de dados MySQL do GSL) para dentro do cluster Kuberentes;
- **k8s-poc-gateway**: Microserviço em Spring Boot que atua como API Gateway.

Os serviços do tipo ClusterIP não são acessados fora do cluster. Os serviços do tipo LoadBalancer podem ser acessados fora do cluster após a atribuição de um endereço IP. Os serviços do tipo ExternalName realizam o mapeamento de um serviço externo para dentro do cluster Kubernetes.

O serviço externo, no caso o banco de dados MySQL, está executando fora do cluster em um container do Docker:



Figura 60 - Container ativo com bancos de dados da arquitetura GSL. Fonte: Autor.

Também foi criado um Ingress Controller com Nginx Ingress Controller:

Host	Path	Path Type	Service Name	Service Port
api.boaentrega.com.br	/	Prefix	k8s-poc-gateway	8080
desenvolvedores.boaentrega.com.br	/	Prefix	k8s-poc-gateway	8080

Figura 61 - Ingress Controller com duas rules criadas. Fonte: Autor.

O Ingress Controller está atuando como um Proxy Reverso, fazendo o roteamento para os serviços que estão dentro do cluster Kubernetes. Foi realizado o mapeamento de dois endereços: api.boaentrega.com.br e desenvolvedores.boaentrega.com.br. Esses dois hosts foram configurados no arquivo hosts da máquina local. O endereço api.boaentrega.com.br aponta para o API Gateway, enquanto o desenvolvedores.boaentrega.com.br aponta para o microserviço de documentação da API, que é mapeado pelo API Gateway. Outros serviços como Front-end teriam um mapeamento no mesmo formato.

Podemos visualizar no Dashboard do Minikube que existem dois Pods para o serviço “k8s-poc-deposito”:

Pods								
Name	Namespace	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)
k8s-poc-documentacao-api-5c756696f-9ltsh	default	poc-documentacao-api:latest	app: k8s-poc-documentacao-api pod-template-hash: 5c756696f	minikube	Running	0	-	-
k8s-poc-deposito-77f884658fb8svn	default	poc-deposito:latest	app: k8s-poc-deposito pod-template-hash: 77f884658f	minikube	Running	1	-	-
k8s-poc-deposito-77f884658fs2fmv	default	poc-deposito:latest	app: k8s-poc-deposito pod-template-hash: 77f884658f	minikube	Running	1	-	-
k8s-poc-gateway-65bf96db8crhh09	default	poc-gateway:latest	app: k8s-poc-gateway pod-template-hash: 65bf96db8c	minikube	Running	0	-	-

Figura 62 - Pods disponíveis - Dois Pods para o microserviço “deposito”. Fonte: Autor.

Ambos foram iniciados no mesmo instante, pois nas configurações de Deployment é considerado duas réplicas. Podemos visualizar que os demais serviços possuem apenas um Pod, pois foi considerado apenas uma réplica.

Com os serviços funcionando, podemos acessar a documentação da API usando a URL mapeada no Ingress Controller: <http://desenvolvedores.boaentrega.com.br/>:

Figura 63 - Documentação do microserviço “deposito”. Fonte: Autor.

A documentação apresentada na imagem acima é referente ao microserviço “deposito”, nele podemos consultar todos os depósitos cadastrados. Um depósito já foi previamente cadastrado:

```
select * from deposito;
```

	id	nome	endereco_id
1	1	Depósito 1	1

Figura 64 - Resultado da consulta SQL com os depósitos. Fonte: Autor.

Também podemos consultar esse depósito utilizando a API documentada no Swagger:

```
curl -X 'GET' \
'http://api.boaentrega.com.br/api/depositos/' \
-H 'accept: */*'
```

Request URL
<http://api.boaentrega.com.br/api/depositos/>

Server response

Code	Details
200	Response body

```
{
  "content": [
    {
      "id": 1,
      "nome": "Depósito 1",
      "endereco": {
        "endereco": "Rua XYZ",
        "cidade": "Chapecó",
        "estado": "SC",
        "cep": "89801000",
        "complemento": "Complemento XYZ",
        "contato": "49 9 0000-1111"
      }
    }
  ],
  "pageable": {
    "sort": {}
  }
}
```

Elements Console Recorder Network Performance Memory Application Security Lighthouse AdBlock

Preserve log Disable cache No throttling

Filter Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies Blocked Requests 3rd-party req

5 ms 10 ms 15 ms 20 ms 25 ms 30 ms 35 ms 40 ms 45 ms 50 ms 55 ms 60 ms 65 ms

Name	Headers	Preview	Response	Initiator	Timing
depositos/	General	Request URL: http://api.boaentrega.com.br/api/depositos/ Request Method: GET Status Code: 200 OK			

Figura 65 - Resultado da busca no endpoint GET /api/depositos. Fonte: Autor.

Essas requisições geram logs no microserviço “deposito”, esses logs podem ser visualizados pelo Dashboard do Minikube:

```

Logs from poc-deposito in k8s-poc-deposito...
[...]
2022-04-08 01:41:07.736 DEBUG [deposito,935df56e6/aa207d,935df56e6/aa207d] 1 --- [nio-8088-exec-2] o.s.w.f.CommonsRequestLoggingFilter : After request [GET /api/depositos/, headers=[x-request-id: 3117d59c4d76dbac55078f034ec0, x-real-ip: "172.17.0.1", x-forwarded-host: "api.boaentrega.com.br", api.boaentrega.com.br", x-forwarded-scheme: "http", x-scheme: "http", accept: "*/*", user-agent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36", accept-encoding: "gzip, deflate", accept-language: "pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7", forwarded: "proto=http;host=api.boaentrega.com.br;for=172.17.0.1:60806", host: "k8s-poc-deposito.default.svc.cluster.local", content-length: "0"]]
2022-04-08 01:41:07.736 DEBUG [deposito,4fcfc78206338f,4fcfc78206338f] 1 --- [nio-8088-exec-2] o.s.w.f.CommonsRequestLoggingFilter : Before request [GET /api/depositos/, headers=[x-request-id: 74bed65a364ed4cf8d6a7ddff5fd30ef, x-real-ip: "172.17.0.1", x-forwarded-host: "api.boaentrega.com.br", api.boaentrega.com.br", x-forwarded-scheme: "http", x-scheme: "http", accept: "*/*", user-agent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36", accept-encoding: "gzip, deflate", accept-language: "pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7", forwarded: "proto=http;host=api.boaentrega.com.br;for=172.17.0.1:60806", host: "k8s-poc-deposito.default.svc.cluster.local", content-length: "0"]]

```

Figura 66 - Logs do microserviço “deposito” em um dos Pods disponíveis. Fonte: Autor.

Os logs da imagem acima são referentes a um único Pod, mas os dois Pods do microserviço “deposito” estão operacionais. Sabendo que os Pods estão operacionais e conforme a imagem acima, recebendo requisições e registrando nos logs, é possível realizar o teste.

Para esse teste será utilizado a ferramenta Apache Benchmark Tool (AB) <https://httpd.apache.org/docs/2.4/programs/ab.html>. Com essa ferramenta podemos realizar requisições por um determinado período e verificar se essas requisições foram completadas com sucesso. Durante esse período podemos parar os dois Pods disponíveis do microserviço “deposito”, um Pod de cada vez. O resultado esperado é que os Pods sejam recriados de forma automática e que fiquem disponíveis em no máximo 1 minuto. Também, que as requisições que o AB fizer sejam completadas com sucesso durante esse período. Um grande número de requisições falhadas não é tolerável, um número baixo sim, visto que o usuário pode realizar uma retentativa.

No momento do teste os seguintes Pods do microserviço “deposito” estão ativos:

NAME	READY	STATUS	RESTARTS	AGE
k8s-poc-deposito-77f884658f-kxn2t	1/1	Running	0	3m47s
k8s-poc-deposito-77f884658f-wdx85	1/1	Running	0	3m8s
k8s-poc-documentacao-api-5c756696f-9ltsh	1/1	Running	0	4h14m
k8s-poc-gateway-65bf96db8c-rhh9	1/1	Running	0	4h15m

Figura 67 - Resultado do comando “kubectl get pods --watch” que mostra os Pods ativos. Fonte: Autor.

Para execução do teste com a ferramenta AB foi utilizando o seguinte comando:

```
ab -s 40 -c 2 -t 120 http://api.boaentrega.com.br/api/depositos/
```

Nesse comando os parâmetros informados significam:

- **-s:** Tempo máximo de espera para cada request;
- **-c:** Número de requests feitas a cada momento (Concorrência);
- **-t:** Tempo do teste em segundos.

Após a execução do teste via terminal, a seguinte saída é apresentada:

```
PS C:\Users\paulo> ab -s 40 -c 2 -t 120 http://api.boaentrega.com.br/api/depositos/
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking api.boaentrega.com.br (be patient)
Finished 3340 requests

Server Software:
Server Hostname:      api.boaentrega.com.br
Server Port:          80

Document Path:         /api/depositos/
Document Length:       487 bytes

Concurrency Level:    2
Time taken for tests: 120.067 seconds
Complete requests:   3340
Failed requests:      1
          (Connect: 0, Receive: 0, Length: 1, Exceptions: 0)
Non-2xx responses:   1
Total transferred:   2992337 bytes
HTML transferred:    1626237 bytes
Requests per second: 27.82 [/sec] (mean)
Time per request:   71.896 [ms] (mean)
Time per request:   35.948 [ms] (mean, across all concurrent requests)
Transfer rate:        24.34 [Kbytes/sec] received

Connection Times (ms)
              min     mean[+/-sd] median     max
Connect:        0      0.3      0       2
Processing:    22     52.7     70     1740
Waiting:       21     51.5     32     1740
Total:         23     52.7     70     1741

Percentage of the requests served within a certain time (ms)
  50%    70
  66%    80
  75%    80
  80%    83
  90%   100
  95%   120
  98%   142
  99%   170
100%  1741 (longest request)
PS C:\Users\paulo> |
```

Figura 68 - Relatório da ferramenta AB Tool após execução do comando. Fonte: Autor.

Podemos visualizar na imagem acima que o teste levou 120 segundos e executou 3340 requisições na URL GET <http://api.boaentrega.com.br/api/depositos/>. Desse total apenas 1 requisição falhou, o que é tolerável.

Durante a execução desse teste com a ferramenta AB foi realizado o monitoramento dos Pods com o comando:

```
kubectl get pods --watch
```

Ao final do teste esse comando apresenta a seguinte saída:

NAME	READY	STATUS	RESTARTS	AGE
k8s-poc-deposito-77f884658f-kxn2t	1/1	Running	0	3m47s
k8s-poc-deposito-77f884658f-wdx85	1/1	Running	0	3m8s
k8s-poc-documentacao-api-5c756696f-9ltsh	1/1	Running	0	4h14m
k8s-poc-gateway-65bf96db8c-rhhhd9	1/1	Running	0	4h15m
k8s-poc-deposito-77f884658f-kxn2t	1/1	Terminating	0	4m57s
k8s-poc-deposito-77f884658f-4vgm6	0/1	Pending	0	0s
k8s-poc-deposito-77f884658f-4vgm6	0/1	Pending	0	0s
k8s-poc-deposito-77f884658f-kxn2t	1/1	Terminating	0	4m57s
k8s-poc-deposito-77f884658f-4vgm6	0/1	ContainerCreating	0	0s
k8s-poc-deposito-77f884658f-kxn2t	0/1	Terminating	0	4m58s
k8s-poc-deposito-77f884658f-kxn2t	0/1	Terminating	0	4m58s
k8s-poc-deposito-77f884658f-kxn2t	0/1	Terminating	0	4m58s
k8s-poc-deposito-77f884658f-4vgm6	0/1	Running	0	2s
k8s-poc-deposito-77f884658f-4vgm6	1/1	Running	0	41s
k8s-poc-deposito-77f884658f-wdx85	1/1	Terminating	0	5m12s
k8s-poc-deposito-77f884658f-mtrqr	0/1	Pending	0	0s
k8s-poc-deposito-77f884658f-mtrqr	0/1	Pending	0	0s
k8s-poc-deposito-77f884658f-wdx85	1/1	Terminating	0	5m12s
k8s-poc-deposito-77f884658f-mtrqr	0/1	ContainerCreating	0	0s
k8s-poc-deposito-77f884658f-wdx85	0/1	Terminating	0	5m14s
k8s-poc-deposito-77f884658f-wdx85	0/1	Terminating	0	5m14s
k8s-poc-deposito-77f884658f-wdx85	0/1	Terminating	0	5m14s
k8s-poc-deposito-77f884658f-mtrqr	0/1	Running	0	2s
k8s-poc-deposito-77f884658f-mtrqr	1/1	Running	0	41s

Figura 69 - Resultado do comando “kubectl get pods --watch” que mostra os Pods ativos. Fonte: Autor.

Na saída apresentada na imagem acima podemos identificar que no início os dois Pods disponíveis eram **k8s-poc-deposito-77f884658f-kxn2t** e **k8s-poc-deposito-77f884658f-wdx85**. Durante a execução dos testes com a ferramenta AB foi realizado a exclusão do Pod **k8s-poc-deposito-77f884658f-kxn2t** e isso gerou a recriação de um novo Pod com o nome **k8s-poc-deposito-77f884658f-4vgm6**. Quando esse novo Pod ficou disponível foi realizado a exclusão do Pod **k8s-poc-**

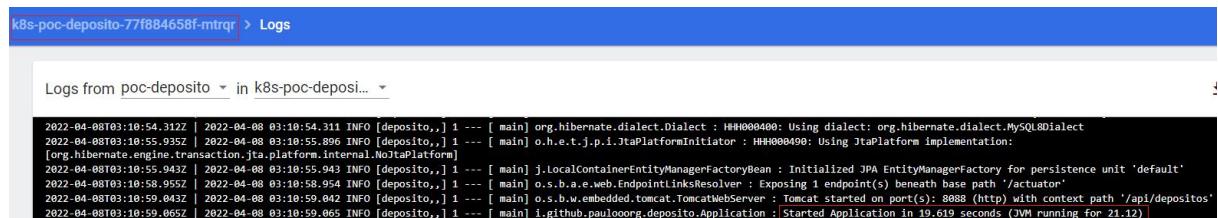
deposito-77f884658f-wdx85 e isso gerou a recriação de um novo Pod com o nome **k8s-poc-deposito-77f884658f-mtrqr**.

Podemos verificar no Dashboard do Minikube que esses dois novos Pods aparecem como ativos:

Pods						
	Name	Namespace	Images	Labels	Node	Status
●	k8s-poc-deposito-77f884658f-mtrqr	default	poc-deposito:latest	app: k8s-poc-deposito pod-template-hash: 77f884658f	minikube	Running
●	k8s-poc-deposito-77f884658f-4vgm6	default	poc-deposito:latest	app: k8s-poc-deposito pod-template-hash: 77f884658f	minikube	Running

Figura 70 - Pods do microserviço “deposito” ativos. Fonte: Autor.

Podemos visualizar o log do Pod **k8s-poc-deposito-77f884658f-mtrqr** para identificar o tempo de inicialização:



```

Logs from poc-deposito > in k8s-poc-deposito-77f884658f-mtrqr
Logs from poc-deposito in k8s-poc-deposito-77f884658f-mtrqr

2022-04-08T03:10:54.312Z | 2022-04-08 03:10:54.311 INFO [deposito,,] 1 --- [ main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2022-04-08T03:10:55.935Z | 2022-04-08 03:10:55.896 INFO [deposito,,] 1 --- [ main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-04-08T03:10:55.943Z | 2022-04-08 03:10:55.943Z INFO [deposito,,] 1 --- [ main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-04-08T03:10:58.952Z | 2022-04-08 03:10:58.954 INFO [deposito,,] 1 --- [ main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'
2022-04-08T03:10:59.043Z | 2022-04-08 03:10:59.042Z INFO [deposito,,] 1 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8088 (http) with context path '/api/depositos'
2022-04-08T03:10:59.065Z | 2022-04-08 03:10:59.065Z INFO [deposito,,] 1 --- [ main] i.github.pauloorg.deposito.Application : Started Application in 19.619 seconds (JVM running for 21.12)

```

Figura 71 -Log do Spring Boot indicando o tempo de inicialização do microserviço. Fonte: Autor.

Conforme a imagem acima, o Pod foi recriado e a aplicação iniciou em 19 segundos.

Consideração sobre as duas evidências apresentadas:

Conforme apresentado nas evidências a arquitetura permite que requisições possam ser processadas mesmo que um microserviço esteja fora, contanto que a mensagem seja enviado ao RabbitMQ. Também foi apresentado que com a utilização do Kubernetes é possível gerenciar réplicas para os microserviços e que

as mesmas são recriadas de forma automática e recebem as requisições dos usuários assim que estiverem disponíveis.

5.3.4. Cenário 4

Atributo de Qualidade:	Interoperabilidade
Requisito de Qualidade:	O sistema deve utilizar recursos adequados para integração
Preocupação:	Os microsserviços do tipo middleware, responsáveis pela integração com os sistemas legados, devem receber mensagens através de uma fila do RabbitMQ. Essas mensagens devem estar no formato JSON e serem consumidas pelos middlewares corretos. O middleware deve fazer a integração com o sistema legado da forma mais adequada, seja por acesso ao banco de dados, REST APIs, mensageria ou outros. Os middlewares também podem receber requisições a REST APIs disponibilizadas, que fazem o acesso aos sistemas legados de forma síncrona, essas APIs devem utilizar o formato JSON. Os microsserviços devem ser stateless, e caso algum caia não é necessário realizar a autenticação novamente.
Cenário(s):	Cenário 4
Ambiente:	Sistema em operação normal
Estímulo:	Usuário parceiro realiza uma requisição para o cálculo de frete ou entrega de pedido utilizando as APIs disponibilizadas para integração de parceiros
Mecanismo:	Os microserviços que expõem APIs que devem ser utilizadas por parceiros, sistemas externos e Front-end do GSL precisam estar no padrão REST e devolver os dados no formato JSON. Com esse formato de dado é possível realizar facilmente a integração com outros sistemas e desenvolver aplicações Front-end ou móveis.
	Para a arquitetura GSL foi definido que a comunicação entre microsserviços

internos deve ser no padrão REST e formato de dados JSON.

Durante a integração de um parceiro no sistema GSL, não deve ocorrer problemas de autenticação por queda nos microserviços, o parceiro, depois de autenticado, não deve se autenticar novamente caso o microserviço caia. A autenticação é necessário somente quando o Token de acesso é inválido/expirado ou não existe.

Além da integração de parceiros, a arquitetura GSL deve se integrar com os sistemas legados, cada sistema legado é único e fornece meios diferentes de integração, portanto, é importante que seja mantido o padrão de um middleware para cada sistema legado, com o objetivo de especializar esses microserviços e centralizar os mecanismos de integração em cada um.

Para a integração com os sistemas legados, os middlewares devem expor APIs REST ou consumir mensagens do RabbitMQ. Os middlewares também podem através de rotinas programadas fazer a leitura de dados dos sistemas legados para atualizar algum microserviço da arquitetura GSL. Esses middlewares devem utilizar o Spring Integration para abstrair detalhes de integração de sistemas, visto que a integração pode ser via conexão com banco de dados, chamada em uma API REST/SOAP/RPC, dentre outros.

Medida de resposta:

- * Os microserviços devem oferecer APIs públicas no padrão REST e devolver os dados no formato JSON
- * Os microserviços devem trabalhar de forma stateless sem impacto na autenticação
- * Os middlewares devem consumir mensagens do RabbitMQ ou receber requisições nas APIs REST e realizar a integração com os sistemas legados da forma mais adequada

Fluxo para as evidências (API Públcas para integração de parceiros):

Usuário logado com perfil de “parceiro” deve acessar o microserviço “parceiro” e realizar uma requisição ao endpoint de busca de pedidos disponíveis para atribuição (continuação do processo de entrega). Esses pedidos estão disponíveis

no microserviço de “pedido” e são buscados através do cadastro do parceiro, que possui um conjunto de depósitos em que ele atua.

Os dados devolvidos pelo endpoint de busca de pedidos disponíveis para atribuição deve estar no formato JSON, esse endpoint deve utilizar o verbo GET e ser possível consultar os dados utilizando o navegador, da mesma forma que é feito ao acessar uma URL de um site qualquer.

Fluxo para as evidências (Middleware e integração com sistema legado):

Integração com REST API:

Os clientes devem conseguir calcular o custo do frete conforme alguns parâmetros, esse cálculo é específico para entregas realizadas pela Boa Entrega. Para o cálculo do frete o microserviço “cliente” disponibiliza uma API REST, que internamente faz a integração com o microserviço “frete”, onde é calculado o custo do frete e dos serviços opcionais. O serviço opcional pode ser um seguro para a entrega, o valor desse seguro não é calculado nos microserviços da arquitetura GSL, mas nos sistemas legados, portanto, é necessário fazer a integração com o sistema SGE para que seja realizado o cálculo do valor do seguro. Essa integração deve ser realizada via API REST e utilizar Spring Integration, visto que para esse cálculo do seguro o SGE expõe uma API. Ao final do processo de cálculo do seguro, o microserviço de “cliente” deve aplicar possíveis descontos que o cliente possui e devolver os dados no formato JSON.

Integração com banco de dados:

Os parceiros devem conseguir realizar a comprovação de uma entrega, essa comprovação é feita pelo microserviço “entrega”. Antes de realizar a comprovação os parceiros precisam se atribuir o pedido, o qual, fica de responsabilidade do parceiro para continuação do processo de entrega.

A comprovação de entrega pode ser bem sucedida ou não, mesmo assim, essa comprovação de entrega deve refletir no sistema SGE, para isso deve ser realizado

a integração com o sistema SGE, para que seja gravado a informação de que o pedido foi entregue.

Para essa integração o microserviço “middleware-sge” deve consumir mensagens referentes a entrega do pedido, nessa mensagem deve constar os dados necessários para serem gravados no sistema SGE. O middleware deve ler a mensagem com o Spring Integration e realizar a integração via banco de dados com o sistema SGE, gravando as informações da entrega nas tabelas necessárias.

Considerações sobre a arquitetura:

Riscos:	Os riscos envolvem a disponibilidade dos sistemas legados e da facilidade de integração. Se os sistemas legados estiverem fora do ar ou apresentando lentidão, isso irá afetar o sistema GSL. A impossibilidade de integração ou mudanças recorrentes nas integrações pode também ser um ponto de risco, visto que seria necessário atualizar o middleware toda vez.
Pontos de Sensibilidade:	Disponibilidade dos sistemas legados. Atenção ao tipo de integração para evitar um alto acoplamento e alto custo de manutenção.
Tradeoff:	Não há

5.3.4.1. Evidências

Os middlewares da arquitetura GSL são responsáveis por fazerem a integração com os sistemas legados SAF, SGE e SFC. Deve existir um middleware para cada sistema legado, assim, é possível manter uma especialização para cada middleware e reaproveitar os mecanismos de integração, visto que cada sistema legado é único e os meios de integração são diferentes.

Esses middlewares recebem requisições através de REST APIs ou fazem a leitura de mensagens do RabbitMQ. Também, é possível ter uma comunicação dos sistemas legados com o sistema GSL, através de técnicas de CDC com Debezium ou rotinas agendas nos middlewares que fazem consultas nos sistemas legados e geram mensagens ao RabbitMQ, essas mensagens seriam consumidas por microserviços da arquitetura GSL.

Para a implementação das integrações foi escolhido o Spring Integration, pois ele fornece uma abstração de diversas técnicas de integração, isso acelera e facilita o desenvolvimento das integrações com os sistemas legados.

Integração com REST API:

Para essa evidência as seguintes aplicações foram consideradas:

- **API Gateway:** Microserviço utilizando Spring Cloud Gateway;
- **Microserviço “cliente”:** Microserviço que expõe o endpoint de cálculo do frete para os clientes. Acessa o microserviço “frete”;
- **Microserviço “frete”:** Microserviço que expõe o endpoint para cálculo de frete conforme alguns parâmetros;
- **Microserviço “middleware-sge”:** Microserviço que representa um middleware para fazer a integração com o sistema legado SGE. Expõe um endpoint para realizar o cálculo do seguro;
- **Sistema SGE (Mock com Wiremock):** Para representar uma REST API do sistema SGE foi utilizado um mock com Wiremock. Esse mock devolve um valor fixo que representa o custo do seguro.

Como estamos considerando o API Gateway e ele possui as configurações de autenticação, precisamos primeiro se autenticar no sistema para utilizar o endpoint do microserviço “cliente”. Para essa evidência será utilizado o usuário “cliente”.

Será verificado antes de apresentar o endpoint de cálculo do seguro se um usuário autenticado permanece autenticado, mesmo após a substituição de deploy do API Gateway.

Após o usuário se autenticar, é possível observar que uma sessão para o usuário foi criada no Keycloak:

IP Address	Started	Last Access	Clients	Action
172.25.0.1	Apr 10, 2022 3:41:19 PM	Apr 10, 2022 3:41:19 PM	gateway	Logout

Figura 71 - Sessão do usuário “cliente” no Keycloak. Fonte: Autor.

Vamos derrubar o API Gateway, para verificar se o Gateway guarda algum estado relacionado a autenticação, é esperado que o usuário não precise se autenticar novamente, pois isso prejudicaria a integração de parceiros no sistema GSL, visto que a troca de deploys deve ser algo comum no dia a dia, independente do horário.

Na imagem abaixo é possível identificar que o API Gateway está disponível e o usuário possui um sessão:

The screenshot shows the Network tab of a browser developer tools interface. A successful GET request is listed for the URL `localhost:8080/api/clientes/1/enderecos`. The response status is 200 OK, and the response body contains the following JSON data:

```

{
    "id": 1,
    "endereco": "Bua XYZ",
    "cidade": "Chapéco",
    "estado": "SC",
    "cep": "89001000",
    "complemento": "Complemento XYZ",
    "contato": "+49 9 0000-1111"
}

```

Figura 72 - Requisição com sucesso ao endpoint `api/clientes/1/enderecos`. Fonte: Autor.

Após derrubar o API Gateway o usuário tenta realizar uma nova requisição, essa requisição não é atendida, pois não existe um serviço disponível:

The screenshot shows the Network tab of a browser developer tools interface. A failed GET request is listed for the URL `localhost:8080/api/clientes/1/enderecos`. The response status is 503 Service Unavailable, and the response body contains the message "Não foi possível conectar".

The response body also includes the following text: "O Firefox não conseguiu estabelecer uma conexão com o servidor localhost:8080. • Este site pode estar temporariamente fora do ar ou sobrecarregado. Tente de novo em alguns instantes. • Se você não conseguir carregar nenhuma página, verifique a conexão de rede do computador. • Se a rede ou o computador estiver protegido por um firewall ou proxy, verifique se o Firefox está autorizado a acessá-la.".

Figura 73 - Requisição com falha ao endpoint `api/clientes/1/enderecos`. Fonte: Autor.

É possível perceber na imagem acima que o Cookie SESSION permanece o mesmo, o usuário continua tentando realizar uma requisição para o microserviço “cliente”. No Keycloak a sessão do usuário permanece ativa:

The screenshot shows the Sessions tab for a user named "cliente" in the Keycloak interface. The table displays the following session information:

IP Address	Started	Last Access	Clients	Action
172.25.0.1	Apr 10, 2022 3:41:19 PM	Apr 10, 2022 3:41:19 PM	gateway	Logout

Figura 74 - Sessão do usuário “cliente” no Keycloak. Fonte: Autor.

O usuário tenta realizar a requisição novamente após o API Gateway estar disponível:

The screenshot shows a browser developer tools interface with the Network tab selected. A successful GET request is listed with the following details:

- Status: 200 OK
- Método: GET
- Dominio: localhost:8080
- Arquivo: enderecos
- Iniciador: document
- Tipo: vnd.mozilla..
- Transferido: 528 B
- Tamanho: 139 B
- Cabeçalhos: GET http://localhost:8080/api/clientes/1/enderecos
- Resposta: Status 200 OK, Version HTTP/1.1, Transferido 544 B (tamanho 139 B), Referer Policy no-referrer, Cabeçalhos da resposta (403 B), Cabeçalhos da requisição (543 B).

The response body contains JSON data representing an address:

```

{
  "id": 1,
  "endereco": "RUA XYZ",
  "cidade": "Chapecó",
  "estado": "SC",
  "cep": "89001000",
  "complemento": "Complemento XYZ",
  "contato": "+59 9 0000-1111"
}

```

Figura 75 - Requisição com sucesso ao endpoint /api/clientes/1/enderecos. Fonte: Autor.

É possível identificar pela imagem acima que a requisição agora foi atendida, o usuário não precisou realizar uma nova autenticação, mas foi gerado um novo Cookie de sessão para ele, isso pode ser verificado pelas outras chamadas realizadas antes da chamada com status 200.

No Keycloak a sessão permanece ativa, porém, foi atualizado a data de último uso:

The screenshot shows the 'Sessions' tab for the 'cliente' user in the Keycloak 'Users' section. The table displays session information:

IP Address	Started	Last Access	Clients	Action
172.25.0.1	Apr 10, 2022 3:41:19 PM	Apr 10, 2022 3:44:51 PM	gateway	Logout

Figura 76 - Sessão do usuário “cliente” no Keycloak com último acesso atualizado. Fonte: Autor.

Com o usuário autenticado e essa evidência apresentada podemos verificar a integração via REST API com o sistema legado. Para isso vamos utilizar o endpoint de cálculo do seguro disponível no microserviço “cliente”:

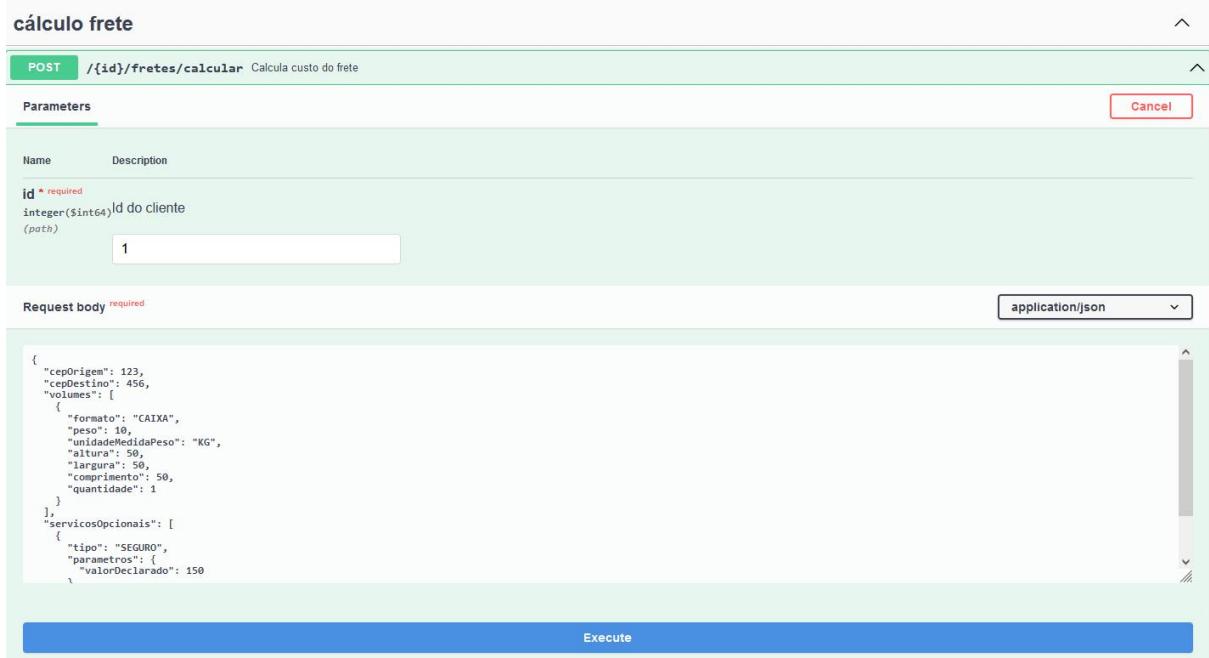


Figura 77 - Documentação do microserviço “cliente” com endpoint de cálculo de frete. Fonte: Autor.

Esse endpoint realiza a comunicação com o microserviço “frete”, que realiza de fato o cálculo do frete. Esse microserviço recebe alguns serviços opcionais, com destaque para o serviço de seguro. Nessa evidência vamos considerar que o cálculo do frete possui o serviço de seguro, assim, é realizada a integração com o sistema legado SGE, pois é ele que possui os dados de seguro para concluir o cálculo.

A seguir o endpoint de cálculo de frete do microserviço “frete”:

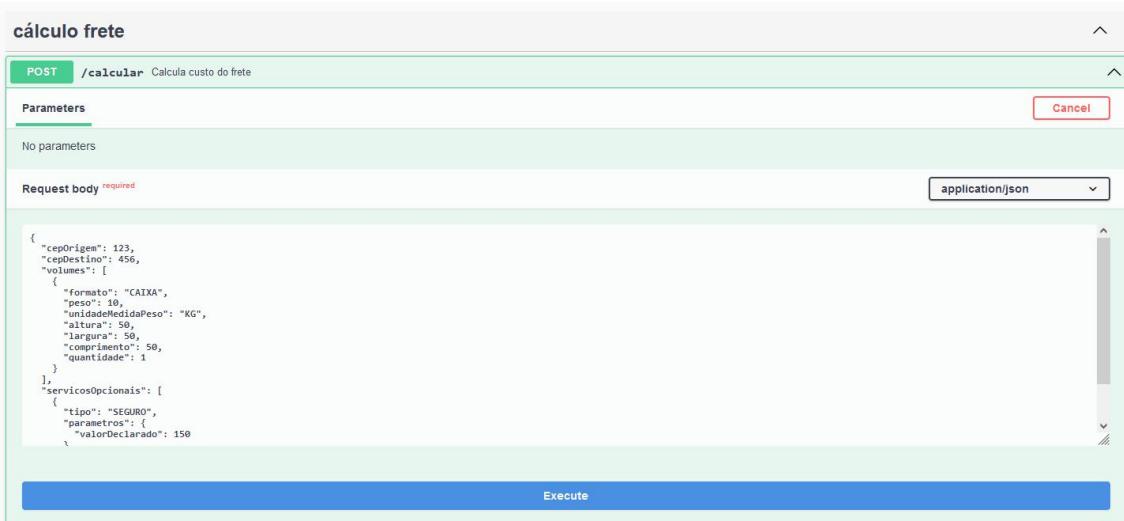


Figura 78 - Documentação do microserviço “frete” com endpoint de cálculo de frete. Fonte: Autor.

Perceba que nesse endpoint não é recebido o id do cliente, pois os descontos do cliente são aplicados no microserviço “cliente”. O microserviço de “frete” chamada o microserviço “middleware-sge” para realizar o cálculo do seguro, para isso é utilizado o endpoint a seguir:

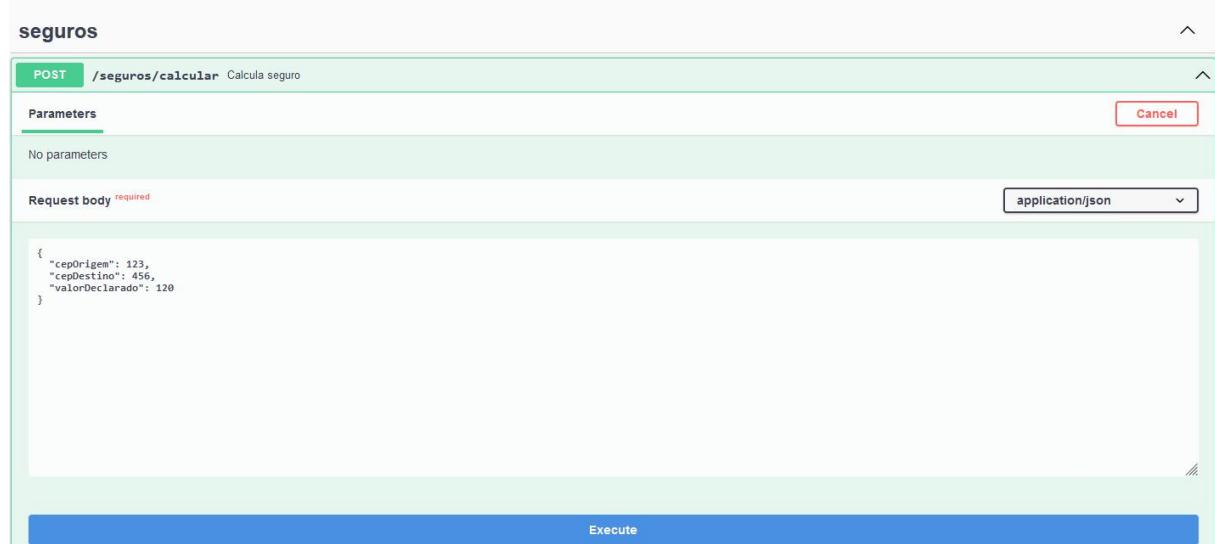


Figura 79 - Documentação do microserviço “middleware-sge” com endpoint de cálculo de seguros.

Fonte: Autor.

Esse endpoint do microserviço “middleware-sge” é exposto através do Spring Integration, isso pode ser verificado pelo código abaixo:

```

@IntegrationConfig.java
    ...
    @Bean
    public IntegrationFlow calcularSeguroInbound() {
        return IntegrationFlows
            .from(
                Http
                    .inboundGateway( ...path: "/seguros/calcular" )
                    .requestMapping(m -> m.methods(HttpMethod.POST))
                    .requestPayloadType(FormCalcularValorSeguroDTO.class)
            )
            .channel("calcularSeguro.httpRequest")
            .get();
    }

    @Bean
    public IntegrationFlow calcularSeguroOutbound() {
        return IntegrationFlows
            .from( messageChannelName: "calcularSeguro.httpRequest" )
            .handle(
                Http
                    .outboundGateway( uri: "http://localhost:9090/sge/seguros/calcular" )
                    .httpMethod(HttpMethod.POST)
                    .expectedResponseType(CustoSeguroDTO.class)
                    .mappedRequestHeaders("")
            )
            .get();
    }
}

```

Figura 80 - Trecho de código referente a integração com SGE no microserviço “middleware-sge”.

Fonte: Autor.

Perceba que no código acima é exposto o endpoint do verbo POST com o path “/seguros/calcular” e ele faz uma chamada a URL: “<http://localhost:9090/sge/seguros/calcular>”. Essa URL é um serviço mockado, que utiliza Wiremock. As configurações desse mock são as seguintes:

```

mock-sge > mappings > ① calcularSeguro.json > [ ] mappings > ② ③ response > ④ headers
1 ↴ {
2 ↴   "mappings": [
3 ↴     {
4 ↴       "metadata": {
5 ↴         "description": "Mapping Calcular Seguro"
6 ↴       },
7 ↴       "request": {
8 ↴         "url": "/sgc/seguros/calcular",
9 ↴         "method": "POST"
10      },
11     "response": {
12       "status": 200,
13       "bodyFileName": "responseCalcularSeguro.json",
14       "headers": [
15         {
16           "Content-Type": "application/json"
17         }
18       ]
19     }
20   }
① responseCalcularSeguro.json U ×
mock-sge > _files > ① responseCalcularSeguro.json > ...
1 {
2   "custoTotal": 4.8
3 }

```

Figura 80 - Arquivos de mock do Wiremock referente ao mock do sistema SGE. Fonte: Autor.

Esse mock é utilizado para representar a integração via REST API com o sistema legado SGE.

Fazendo a requisição de cálculo do seguro via Swagger, no microserviço “cliente”, podemos verificar que a mesma devolve os dados no formato JSON:

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/clientes/1/fretes/calcular' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "cepOrigem": 123,
    "cepDestino": 456,
    "volumes": [
      {
        "formato": "CAIXA",
        "peso": 10,
        "unidadeMedidaPeso": "KG",
        "altura": 50,
        "largura": 50,
        "comprimento": 50,
        "quantidade": 1
      }
    ],
    "servicosOpcionais": [
      {
        "tipo": "SEGURO",
        "parametros": {
          "valorDeclarado": 150
        }
      }
    ]
}'
```

Request URL

<http://localhost:8080/api/clientes/1/fretes/calcular>

Server response

Code	Details
200	Response body
	<pre>{ "prazoDiasUteis": 10, "custoTotal": 7.31, "custoFrete": 2.89, "custoServicos": 4.8, "descontoCliente": 0.38 }</pre>

Figura 81 - Resultado m JSON da requisição ao endpoint /api/clientes/1/fretes/calcular. Fonte: Autor.

Isso possibilita que parceiros e clientes possam realizar a integração com o sistema GSL de forma facilitada, visto que a leitura desse formato de dado por uma aplicação Front-end ou aplicação móvel é fácil de ser realizada.

Também conseguimos identificar que o Wiremock recebeu a requisição do microserviço “middleware-sge”:

```

2022-04-10 16:04:40.871 Request received:
127.0.0.1 - POST /sgc/seguros/calcular

Accept: [application/json, application/*+json]
Content-Type: [application/json; charset=UTF-8]
User-Agent: [Java/11.0.5]
Host: [localhost:9090]
Connection: [keep-alive]
Content-Length: [55]
{"cepOrigem":123,"cepDestino":456,"valorDeclarado":150}

Matched response definition:
{
    "status" : 200,
    "bodyFileName" : "responseCalcularSeguro.json",
    "headers" :
        "Content-Type" : "application/json"
}
}

Response:
HTTP/1.1 200
Content-Type: [application/json]
Matched-Stub-Id: [25386d88-7f4b-485e-8cc6-b3b2337578fc]

```

Figura 82 - Logs do Wiremock apresentando que a requisição chegou a ele. Fonte: Autor.

A imagem acima comprova que o “middleware-sge” está conseguindo acessar o mock que representa o sistema legado SGE. E que a integração via API REST é possível com Spring Integration.

Integração com banco de dados:

Algumas integrações podem não ser possíveis via APIs REST, e em alguns casos a integração via banco de dados será a única opção. Por esse motivo foi considerado uma integração via banco de dados usando Spring Integration. No qual será apresentado as evidências a seguir.

Para essa evidência foi considerado principalmente os microserviços “parceiro”, “entrega” e “middleware-sge”. Toda entrega realizada deve ser notificada ao sistema SGE, como o sistema SGE pode estar fora no momento de ser notificado é necessário que essa notificação seja enviado antes para uma fila do RabbitMQ e que posteriormente seja processada pelo microserviço “middleware-sge”, que fará a integração com o sistema SGE via banco de dados.

O parceiro realiza a entrega de pedidos em que ele é responsável. Para consultar pedidos disponíveis para o parceiro realizar entregas é possível utilizar o endpoint abaixo:

```
Curl
curl -X 'GET' \
'http://localhost:8080/api/parceiros/1/pedidos/disponiveis?size=10&page=0' \
-H 'accept: application/json'

Request URL
http://localhost:8080/api/parceiros/1/pedidos/disponiveis?size=10&page=0

Server response
Code Details

200 Response body
{
  "content": [
    {
      "id": 3,
      "codigo": "df3df743-68fc-4a63-a7cc-20fa098f47a3",
      "dataEmissao": "2022-04-09T21:41:32",
      "dataEntregaPrevista": "2022-04-21",
      "custoFrete": 7.31,
      "enderecoDestinatario": {
        "endereco": "Rua XYZ",
        "cidade": "Chapecó",
        "estado": "SC",
        "cep": 89801000,
        "complemento": "Complemento XYZ",
        "contato": "49 9 0000-1111"
      },
      "responsavelEntrega": null,
      "situacao": "PEDIDO_RECEBIDO",
      "enderecoRemetente": {
        "endereco": "Rua XYZ",
        "cidade": "Chapecó",
        "estado": "SC",
        "cep": 89801000,
        "complemento": "Complemento XYZ",
        "contato": "49 9 0000-1111"
      }
    }
  ]
}
```

Figura 83 - Resultado da requisição ao endpoint /api/parceiros/1/pedidos/disponiveis. Fonte: Autor.

A imagem acima apresenta um pedido disponível para o parceiro continuar o processo de entrega, perceba que a situação do pedido está como “PEDIDO_RECEBIDO” e não existe responsável pela entrega. Para fazer atribuição de um pedido a um parceiro é utilizado o endpoint abaixo:

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/parceiros/1/atribuir/pedidos' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "pedidoids": [
      3
    ]
}'
```

Request URL

```
http://localhost:8080/api/parceiros/1/atribuir/pedidos
```

Server response

Code	Details
204	Response headers
	<pre>cache-control: no-cache,no-store,max-age=0,must-revalidate date: Sun,10 Apr 2022 19:33:11 GMT expires: 0 pragma: no-cache referrer-policy: no-referrer vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 1 ; mode=block</pre>

Figura 84 - Resultado da requisição ao endpoint POST /api/parceiros/1/atribuir/pedidos. Fonte: Autor.

O parceiro deve informar seu identificador e os pedidos que ele gostaria de se atribuir. Se consultarmos os pedidos do cliente podemos identificar que esse pedido agora é de responsabilidade do parceiro:

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/pedidos/clientes/1?situacao=EM_TRANSPORTE&size=10&page=0' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8080/api/pedidos/clientes/1?situacao=EM_TRANSPORTE&size=10&page=0
```

Server response

Code	Details
200	Response body
	<pre>{ "id": 3, "codigo": "df3df743-68fc-4a63-a7cc-20fa098f47a3", "dataEmissao": "2022-04-09T21:41:32", "dataEntregaPrevista": "2022-04-21", "custoFrete": 7.31, "enderecoDestinatario": { "endereco": "Rua XYZ", "cidade": "Chapecó", "estado": "SC", "cep": 89801000, "complemento": "Complemento XYZ", "contato": "49 9 0000-1111" }, "responsavelEntrega": "Parceiro", "situacao": "EM_TRANSPORTE", "enderecoRemetente": { "endereco": "Rua XYZ", "cidade": "Chapecó", "estado": "SC", "cep": 89801000, "complemento": "Complemento XYZ", "contato": "49 9 0000-1111" } }, "pageable": { "sort": { "id": 3 } }</pre>

Figura 85 - Resultado da requisição ao endpoint GET /api/pedidos/clientes/1. Fonte: Autor.

Perceba que a situação agora é “EM_TRANSPORTE” e o responsável está preenchido.

Para efetivar a entrega do pedido precisamos utilizar os endpoints disponíveis no microserviço “entrega”:

entregas		
GET	/pedido	Busca entrega por código do pedido
GET	/{id}/assinatura	Busca assinatura da entrega
POST	/entregar/situacao/entregue	Cria uma entrega com situação ENTREGUE
POST	/entregar/situacao/nao-entregue	Cria uma entrega com situação NAO_ENTREGUE
POST	/assinatura	Cria assinatura para adicionar na entrega

Figura 86 - Endpoints do microserviço “entrega” responsáveis pela comprovação da entrega. Fonte: Autor.

O parceiro realiza o upload de uma imagem que representa a assinatura do destinatário e confirma a entrega do pedido. O pedido pode ser entregue com sucesso ou não. Em ambos os casos é necessário que o sistema SGE seja notificado. Para representar essa notificação realizamos um insert em uma base de dados que representa a base de dados do sistema legado SGE.

Sabemos que o pedido a ser entregue é o de código **df3df743-68fc-4a63-a7cc-20fa098f47a3**, conforme apresentando nas imagens acima. Portanto, vamos utilizar o endpoint de entrega bem sucedida:

```
Curl

curl -X 'POST' \
  'http://localhost:8080/api/entregas/entregar/situacao/entregue' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "codigoPedido": "df3df743-68fc-4a63-a7cc-20fa098f47a3",
    "responsavelEntrega": "Parceiro",
    "assinaturaId": 3,
    "observacao": "Observação",
    "latitude": -25.1234,
    "longitude": -45.1465
}'

Request URL
http://localhost:8080/api/entregas/entregar/situacao/entregue

Server response

Code Details
201 Response body
2
```

Figura 87 - Requisição com sucesso ao endpoint de comprovação de entrega. Fonte: Autor.

É possível identificar pela imagem acima que o pedido agora está entregue e foi vinculado uma assinatura a ele. O registro de entrega criado é o de id 2 e podemos consultar ele no banco de dados:

```
select a.*, e.* from entrega e join assinatura a on e.assinatura_id = a.id
where e.codigo_pedido = 'df3df743-68fc-4a63-a7cc-20fa098f47a3';|
```

assinatura(+)

'select a*, e.* from entrega e join assinatura a on e.assinatura_id = a.id
where e.codigo_pedido = 'df3df743-68fc-4a63-a7cc-20fa098f47a3';| Enter a SQL expression to filter results (use Ctrl+Space)

	id	arquivo	id	codigo_pedido	responsavel_entrega	data_entrega	Value	Calc
1	3	6PNG IHDR c KPV... [1756]	2	df3df743-68fc-4a63-a7cc-20fa098f47a3	Parceiro	2022-04-10 19:41:00		

Figura 88 - Resultado da consulta SQL com os dados da entrega e assinatura do destinatário. Fonte: Autor.

Também podemos verificar no banco de dados do SGE, para validar se a notificação de entrega foi gravada:



```
select * from pedido_entregue where codigo = 'df3df743-68fc-4a63-a7cc-20fa098f47a3';
```

id	codigo
1	df3df743-68fc-4a63-a7cc-20fa098f47a3

Figura 89 - Resultado da consulta SQL com os dados da notificação ao sistema SGE (Integração via banco de dados). Fonte: Autor.

A imagem acima apresenta uma consulta a tabela “pedido_entregue” que representa uma tabela fictícia do sistema SGE, apenas para representar a integração via banco de dados utilizando o Spring Integration no microserviço “middleware-sge”.

Esse insert é realizado através da leitura de uma mensagem enviado ao RabbitMQ pelo microserviço “pedido”. Essa mensagem então é consumida pelo microserviço “middleware-sge” que gera o insert na tabela para o código de pedido recebido.

Essa mensagem é enviada para a fila apresentada na imagem abaixo:

queue	type	consumed	idle	new	dropped	rejected	redelivered	reject / ack	reject / requeue
middleware-sge-pedido-criado-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
middleware-sge-pedido-entregue-queue	classic	D	idle	1	0	1	0.00/s	0.00/s	0.00/s

Figura 90 - Fila “middleware-sge-pedido-entregue-queue” que recebe a notificação de entrega do pedido. Fonte: Autor.

Perceba que na imagem acima existe uma mensagem pendente. Foi realizado a entrega de mais um pedido e parado o microserviço “middleware-sge” para apresentar isso.

Se consultarmos essa mensagem podemos identificar que nela existe os dados de entrega do pedido:

Get Message(s)	
Message 1	
The server reported 0 messages remaining.	
Exchange	(AMQP default)
Routing Key	middleware-sge-pedido-entregue-queue
Redelivered	0
Properties	priority: 0 delivery_mode: 2 headers: __TypeId__: io.github.paulooorg.pedido.producer.dto.MessagePedidoEntregueDTO b3: 5baa5e7e56640b3f-1826c6044e33590e-1 content_encoding: UTF-8 content_type: application/json
Payload	{"codigo": "65e5e923-8cd2-4cf0-b6af-bf90478d3594"}
Encoding:	string

Figura 91 - Mensagem pendente na fila “middleware-sge-pedido-entregue-queue”. Fonte: Autor.

Essa mensagem é consumida pelo microserviço “middleware-sge” através dos códigos abaixo:

```

@IntegrationConfig.java
    ...
    @Bean
    public IntegrationFlow pedidoEntregueInbound(ConnectionFactory connectionFactory) {
        return IntegrationFlows
            .from(Amqp.inboundAdapter(connectionFactory, pedidoEntregueQueue))
            .handle(pedidoEntregueMessageHandler())
            .get();
    }

    @Bean
    public MessageHandler pedidoEntregueMessageHandler() {
        JdbcMessageHandler jdbcMessageHandler = new JdbcMessageHandler(
            dataSource,
            updateSql: "INSERT INTO pedido_entregue (codigo) VALUES(?)"
        );
        jdbcMessageHandler.setPreparedStatementSetter(
            (ps, m) -> {
                String payload = new String((byte[]) m.getPayload());
                try {
                    FormPedidoEntregueDTO formPedidoEntregue = objectMapper.readValue(
                        payload,
                        FormPedidoEntregueDTO.class
                    );
                    log.info("Realizando insert na tabela pedido_entregue com os dados -> {}", formPedidoEntregue);
                    ps.setString(1, formPedidoEntregue.getCodigo());
                } catch (JsonProcessingException e) {
                    throw new RuntimeException(e);
                }
            }
        );
        return jdbcMessageHandler;
    }
}

```

Figura 92 - Trecho de código do microserviço “middleware-sge” que realiza a integração via banco de dados com o sistema SGE. Fonte: Autor.

Os códigos apresentados acima utilizam Spring Integration, que consome uma mensagem do RabbitMQ e realiza o insert na tabela “pedido_entregue”.

Consideração sobre as evidências apresentadas:

Pelas evidências apresentadas podemos identificar que a integração com os sistemas legados é possível através dos middlewares e com Spring Integration. E as integrações com parceiros e clientes é possível através de APIs públicas do sistema GSL, as quais possuem autenticação, mas que não afetam a integração das mesmas em sistemas externos. Também, podemos visualizar que os dados devolvidos nessas APIs públicas estão no formato JSON, facilitando a integração com diferentes linguagens e aplicações.

5.4. Resultados

O objetivo da POC foi realizar a avaliação da arquitetura proposta com base em alguns requisitos não funcionais escolhidos:

Requisitos Não Funcionais	Testado	Homologado
RNF01: O sistema deve possuir características de aplicação distribuída	SIM	SIM
RNF02: O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades	SIM	SIM
RNF05: O sistema deve ser recuperável no caso da ocorrência de erro	SIM	SIM
RNF06: O sistema deve utilizar recursos adequados para integração	SIM	SIM

Todos os requisitos não funcionais apresentados na tabela acima foram implementados, testados e homologados, conforme descrito nas evidências dos cenários propostos. Durante o processo de implementação desses cenários foi identificado alguns pontos fortes da arquitetura e possíveis melhorias.

Abaixo os pontos fortes e possíveis melhorias para cada requisito não funcional apresentado na tabela:

- RNF01: O sistema deve possuir características de aplicação distribuída
 - Pontos fortes
 - A utilização de APIs REST com formato de dados JSON facilita a integração entre microserviços de diferentes linguagens. Por ser um padrão bem estabelecido no mercado é possível facilmente encontrar bibliotecas e frameworks que trabalham nesse formato
 - O Message Broker RabbitMQ possibilita a fácil configuração e utilização, tanto em ambiente local quanto em produção, e

permite que microserviços se comuniquem de forma assíncrona facilmente, independente da tecnologia que foram escritos. A utilização dessa tecnologia permite que mensagens não sejam perdidas caso algum microserviço esteja indisponível

- A utilização do Spring Framework e Spring Cloud possibilita a rápida construção de microserviços, visto que o Spring Cloud fornece um conjunto amplo de tecnologias essenciais no contexto de microserviços, como Spring Sleuth, Spring Integration, Spring Circuit Breaker, Spring Eureka, Spring Gateway, Spring Config Server, dentre outros. Alguns dos quais foram utilizados na arquitetura proposta
 - Melhorias
 - Para os microserviços internos, que não expõem APIs REST públicas seria possível utilizar um outro padrão de comunicação entre microserviços, como RPC com gRPC, isso permite um relacionamento mais forte entre os microserviços pois podemos trabalhar com schemas, e mais performance, visto que podemos trabalhar com dados binários e HTTP2
 - Para remover detalhes de infraestrutura dos microserviços poderia ser utilizado uma tecnologia de Service Mesh, já que estamos considerando o uso de containers e Kubernetes, assim, questões como resiliência poderiam ser tratadas por ferramentas como Istio, Linkerd, dentre outros. Isso possibilita um cenário mais poliglota dentro da arquitetura e o repasse de detalhes de infraestrutura ao time de DevOps, que possui uma visão mais geral do conjunto de microserviços
- RNF02: O sistema deve fornecer controle de acesso via perfil para determinadas funcionalidades
 - Pontos fortes
 - O Spring Gateway com as bibliotecas de cliente OAuth2 conseguem se integrar facilmente com o Keycloak, que

gerencia bem os usuários e suas permissões, permitindo a geração de Tokens de acesso que possuem no payload as permissões dos usuários

- O Spring Security permite de forma fácil bloquear endpoints conforme o perfil do usuário
- RNF05: O sistema deve ser recuperável no caso da ocorrência de erro
 - Pontos fortes
 - A utilização do Message Broker RabbitMQ permite que mensagens sejam armazenadas e processadas assim que um microserviço indisponível se torne disponível para processá-las
 - A biblioteca Spring Actuator permite expor diversas métricas sobre o microserviço, as quais podem ser utilizadas em Dashboards para verificar a saúde de cada microserviço e identificar problemas o mais rápido possível
 - O uso de Kubernetes permite a fácil configuração de réplicas para os microserviços, as quais são gerenciadas e recriadas de forma automática pelo Kubernetes
 - Por utilizarmos Spring Framework é possível utilizar bibliotecas como Spring Circuit Breaker para tratar casos de resiliência, como retentativas, fallbacks, dentre outros. Se não for utilizado um Service Mesh
 - Melhorias
 - Em uma arquitetura de microserviços o monitoramento dos mesmos é essencial, na POC não foi considerado ferramentas de monitoramento, como Prometheus, Grafana, Zabbix, dentre outras, mas foram considerados como sugestão nos mecanismos arquiteturais. Isso é um ponto em aberto, visto que já existem sistemas operando na Boa Entrega, e essas ferramentas de monitoramento já utilizadas podem ser reutilizadas para a arquitetura GSL. Caso não existam pode ser

utilizado as sugestões apresentadas nos mecanismos arquiteturais

- No cenário relacionado a esse requisito não funcional foi identificado que os microserviços com Spring Boot podem ser inicializados mais rapidamente, isso possibilita um atendimento maior de requisições durante a troca de deploys. Para isso é possível utilizar o projeto Spring Native, assim que o mesmo estiver estável. Com isso conseguimos inicializar os serviços mais rapidamente e consumir menos recursos
- RNF06: O sistema deve utilizar recursos adequados para integração
 - Pontos fortes
 - O uso da biblioteca Spring Integration permite uma maior facilidade para a integração com os sistemas legados, visto que ela fornece uma abstração e padronização do uso de padrões de integração
 - As APIs públicas no formato REST e trabalhando com JSON permitem que sistemas externos se integrem com facilidade
 - Melhorias
 - Devido a dificuldade de integração com os sistemas legados é importante que as integrações sejam feitas com cuidado, e que na medida do possível esses serviços legados sejam incorporados na arquitetura GSL, seja por meio da reescrita dos serviços ou incorporação dos mesmos da maneira que estão hoje
 - A ferramenta de CDC proposta pode ser utilizada com um Kafka ou de forma embutível em uma aplicação Spring Boot, mas estando em uma aplicação Spring Boot a disponibilidade do serviço se torna importante para o correto processamento das mensagens, portanto, devemos considerar esse componente como temporário na arquitetura, até que os serviços sejam

incorporados na arquitetura GSL ou meios de integração mais sofisticados sejam possíveis

6. Conclusão

Os objetivos propostos neste trabalho foram atingidos, a arquitetura proposta permite uma hospedagem On Premise, reutilizando a infraestrutura da Boa Entrega, e também a possibilidade de hospedagem em Cloud. Para prover essa flexibilidade foi escolhido a estratégia de containers para a distribuição das aplicações, esses containers seriam gerenciados pelo Kubernetes. Essa estratégia permite que microserviços de diferentes linguagens consigam incorporar a arquitetura proposta, sem que sejam necessariamente escritos na linguagem Java.

Caso a hospedagem em Cloud seja a escolha adequada financeiramente, é possível escolher alguns dos vários players disponíveis no mercado, como AWS, Azure, dentre outros, que fornecem serviços para hospedagem de containers com Kubernetes.

Além dos objetivos gerais foram considerados alguns objetivos específicos, que se concentraram no desenvolvimento da POC e da escolha de tecnologias. A arquitetura do sistema GSL é principalmente uma arquitetura de microserviços, portanto, elementos essenciais para essa arquitetura foram avaliados e descritos no decorrer do trabalho. Durante a implementação da arquitetura proposta é importante ter em mente que a adoção dessa arquitetura em um time de desenvolvimento envolve a estruturação de equipes qualificadas. O nível de organização e segmentação dos times é importante para o sucesso da implementação e sustentação dos microserviços. Também, é importante a consideração de times de DevOps capacitados, visto que a implementação de Kubernetes, e microserviços, envolve muita complexidade, as quais são de certa forma repassadas do desenvolver para a equipe de DevOps. O monitoramento e configuração desses ambientes, seja de ambiente de produção ou desenvolvimento, é extremamente importante para o sucesso da implementação.

Uma escolha considerada na arquitetura visando o baixo custo, foi a reutilização do banco de dados MySQL já presente no sistema legado SGE, com o objetivo de utilizar a mesma infraestrutura existente. Os microserviços não possuem

bancos de dados compartilhados, existe um banco de dados para cada microserviço, podemos imaginar isso como schemas. A instância do MySQL é a mesma. A longo prazo, caso seja necessário adotar mais instâncias do MySQL, ou até diferentes bancos de dados relacionais, fica mais fácil a migração.

Alguns sistemas adquiridos no mercado foram considerados para os módulos de Serviços ao Cliente, Gestão e Estratégia e Ciência de Dados. O objetivo das escolhas foi encontrar uma solução barata e que resolvesse os problemas apresentados.

O módulo de Serviços ao Cliente utilizará a ferramenta Camunda, que é uma ferramenta gratuita e permite que a hospedagem seja feita On Premise ou em Cloud, sendo a Cloud da própria Camunda ou Cloud contratada na AWS ou Azure, pois a aplicação Camunda pode ser embutida em uma aplicação Spring Boot. Além da aplicação Web, existe um modelador de processos BPM. Essa escolha vai de encontro com o objetivo de baixo custo da arquitetura, visto que a preocupação estaria relacionada apenas a hospedagem.

Para o módulo de Gestão e Estratégia também foi escolhido uma ferramenta de mercado já consolidada, no caso da proposta seria a Asana, a ferramenta permite a integração via APIs e fornece diversos templates para o gerenciamento de diferentes tipos de projetos. A Asana não seria hospedado na infraestrutura da Boa Entrega, seria realizado a contratação de alguns usuários na plataforma da própria Asana, como é uma aplicação que não seria acessada por todos os usuários da Boa Entrega, o custo por usuário não seria tão alto.

O módulo de Ciência de Dados teve como escolha uma ferramenta já consolidada no mercado e que resolvesse os problemas apresentados, para isso foi escolhido o PowerBI. Essa ferramenta não seria hospedada na infraestrutura da Boa Entrega, mas seria contratada. É possível utilizar ferramentas gratuitas de BI, como o Metabase, que seria com hospedagem On Premise, essa solução não foi considerada, pois a ideia é prover uma solução mais robusta e consolidada, portanto, a utilização do Metabase fica em aberto, caso o custo seja baixo e resolver os

problemas apresentados. Além do BI, foi considerado uma ferramenta de ETL, no caso o Airflow, essa ferramenta é gratuita e seria hospedada On Premise.

Todas essas ferramentas escolhidas resolvem os problemas apresentados e possibilitam uma hospedagem em Cloud e/ou On Premise, visando um baixo custo e alta entrega de valor.

Um dos pontos importantes para a implementação desta arquitetura é a integração com os sistemas legados, tanto que isso foi um dos objetivos específicos do trabalho, testar a integração via REST APIs e via banco de dados, pois a ideia a longo prazo é que o sistema GSL se torne um sistema único, centralizando, em diferentes microserviços, os serviços de logística da Boa Entrega.

Para que isso ocorra de forma viável, é importante que os sistemas legados sejam incorporados na medida do possível na arquitetura GSL, e os serviços que não podem realizar a incorporação a curto prazo, sejam integrados através dos middlewares, para que a longo prazo sejam reescritos. O Portal Corporativo deve a longo prazo ficar com menos funcionalidades de negócio, não existe motivo para replicar funcionalidades do sistema SGE ou qualquer outro sistema dentro do Portal Corporativo, esse sistema deve ser apenas um portal de fato, encaminhando os usuários para outros sistemas e centralizando questões específicas de um portal.

REFERÊNCIAS

APACHE, s/ autor. **Documentação AB Tool.** Apache, 2022. Disponível em: <https://httpd.apache.org/docs/2.4/programs/ab.html> - Acesso em 08/04/2022.

BAELDUNG, Tim Schimandle. **Spring Cloud Sleuth in a Monolith Application.** Baeldung, 2020. Disponível em: <https://www.baeldung.com/spring-cloud-sleuth-single-application> - Acesso em 21/03/2022.

REFACTOR FIRST, Amrut Prabhu. **Spring Cloud Gateway Keycloak OAuth2 OIDC Integration.** Disponível em: <https://refactorfirst.com/spring-cloud-gateway-keycloak-oauth2-openid-connect.html> - Acesso em 24/03/2022.

REFACTOR FIRST, Amrut Prabhu. **Spring Cloud Gateway — Resource Server with Keycloak RBAC.** Disponível em: <https://refactorfirst.com/spring-cloud-gateway-keycloak-rbac-resource-server.html> - Acesso em 24/03/2022.

KUBERNETES, s/ autor. **Instalar Ingress Controller no Minikube.** Kubernetes, 2022. Disponível em: <https://kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/> - Acesso em 28/03/2022.

SOSHACE, Erol Hira. **Spring Cloud Config Refresh Strategies.** SOSHACE, 2020. Disponível em: <https://soshace.com/spring-cloud-config-refresh-strategies/> - Acesso em 21/03/2022.

DEBEZIUM, s/ autor. **Documentação Debezium.** Debezium, 2022. Disponível em: <https://debezium.io/documentation/> - Acesso em 18/03/2022.

ASANA, s/ autor. **Documentação Asana.** Asana, 2022. Disponível em: <https://developers.asana.com/docs/> - Acesso em 17/03/2022.

G1, TV Globo. **Setor de logística cresce durante a pandemia e tem milhares de vagas de emprego abertas em SP.** G1, 2021. Disponível em: <https://g1.globo.com/sp/sao-paulo/noticia/2021/09/13/setor-de-logistica-cresce-durante-a-pandemia-e-tem-milhares-de-vagas-de-emprego-abertas-em-sp.ghtml> - Acesso em 03/03/2022.

G1, Paula Monteiro. **Logística cresce na pandemia com aumento de compras pela internet.** G1, 2021. Disponível em: <https://g1.globo.com/economia/pme/pequenas-empresas-grandes-negocios/noticia/2021/01/31/logistica-cresce-na-pandemia-com-aumento-de-compras-pela-internet.ghtml> - Acesso em 03/03/2022.

MEDIUM, Joshgun Huseynov. **Spring Cloud Config vs Kubernetes ConfigMap — Detailed Comparison.** Medium, 2021. Disponível em: <https://joshgunh.medium.com/spring-cloud-config-vs-kubernetes-configmap-detailed-comparison-bce64b594af8> - Acesso em 21/03/2022.

MICROSOFT, s/ autor. **Por que usar o Microsoft Power BI.** Microsoft, 2022. Disponível em: <https://powerbi.microsoft.com/pt-br/why-power-bi/> - Acesso em 18/03/2022.

SPRING, s/ autor. **Documentação Spring Integration.** Spring, 2022. Disponível em: <https://docs.spring.io/spring-integration/reference/html/> - Acesso em 25/03/2022.

CAMUNDA, s/ autor. **Documentação Camunda com Spring Boot.** Camunda, 2022. Disponível em: <https://docs.camunda.org/get-started/spring-boot/> - Acesso em 15/03/2022.

MEDIUM, Deepak Choudhary. **Solving the “Too many Swaggers” problem in a Microservice architecture.** Medium, 2020. Disponível em:

<https://deepakschoudhary.medium.com/solving-the-too-many-swaggers-problem-in-a-microservice-architecture-5194b723ca4e> - Acesso em 16/03/2022.

AIRFLOW, s/ autor. **Documentação Airflow.** Airflow, 2022. Disponível em:
<https://airflow.apache.org/docs/apache-airflow/stable/> - Acesso em 18/03/2022.

APÊNDICES

Repositório:

<https://github.com/paulooorg/tcc-asd>

Vídeos:

Resumo POC: <https://vimeo.com/699897135>

Macroarquitetura: <https://vimeo.com/699897067>

Detalhado POC: <https://vimeo.com/699896974>

Os vídeos também estão disponíveis no repositório do GitHub.