INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Logistics as microservices From monolithic to microservices A case study Logistics

NELSON DANIEL MENDES FERREIRA
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Logistics as microservices

"From monolithic to microservices

A case study – *Logistics*"

Nelson Daniel Mendes Ferreira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Ana Maria Neves Almeida Baptista Figueiredo

Supervisor: Paulo Jorge Lopes de Araújo

Porto, Outubro 2020

Resumo

O desenvolvimento de *software* aos longos dos últimos anos tem evoluído dada a exigência do mercado e a sua necessidade de obter soluções com elevado desempenho e eficácia. As empresas visam atender os requisitos dos seus clientes, pretendendo assegurar soluções com elevado nível de desempenho, usabilidade, disponibilidade e escalabilidade enquanto reduzem os seus custos de desenvolvimento e implantação.

Assim, ao longo dos anos as empresas tendem a decompor os seus serviços em pequenos segmentos de código-fonte, denominados de microsserviços. Estes, permitem obter o máximo partido dos serviços disponibilizados pelos provedores de *cloud* e proporcionar soluções satisfatórias quer para as empresas quer para os clientes.

Neste trabalho procede-se à realização de uma migração arquitetural, decompondo uma aplicação monolítica da empresa Flowinn, Logistics, em utilização no quotidiano dos seus clientes, em diversos microsserviços e utilizar padrões comuns em arquiteturas baseadas em microsserviços. Após a migração são avaliados os impactos e observadas, ou não melhorias associadas à migração nas vertentes acima descritas.

Como resultado pretende-se reduzir os custos da empresa e tornar o produto mais competitivo no mercado em que se encontra.

Palavras-chave: Monolítica, Microsserviços, Evolução arquitetural*, Continuous Integration*, *Continuous Delivery*.

Abstract

Over the past few years, software development has evolved given the market's demand for solutions with high performance and efficiency. The companies aim to meet the requirements of their customers, aiming to ensure solutions with a high level of performance, usability, availability and scalability while reducing their development and deployment costs.

Over the years, companies have tended to break their services into small segments of source code, denominated microservices. These get the most out of the services provided by cloud providers and allow satisfactory solutions for companies and customers.

With this work, it is intended to do an architectural migration, decomposing a monolithic application of the company Flowinn, in use in the daily of its customers, in several microservices and to use common standards in microservices based architectures. Subsequently, it is intended to assess the impacts and improvements associated with migration in the areas described above.

All this work aims to reduce the company's costs and make the product more competitive in the market in which it is.

Keywords: Monolithic, Microservices, Architectural evolution, Continuous Integration, Continuous Delivery.

Agradecimentos

Aos meus pais, pelo carinho e motivação que me deram ao longo de todos os meus anos de estudo.

À minha família, por me ajudarem sempre que preciso.

À minha namorada, Ana Rita, por todo o apoio e dedicação demonstrado, estando ao meu lado em todas as minhas conquistas.

À minha orientadora, Ana Almeida, pela orientação prestada, disponibilidade e apoio que sempre demonstrou.

À empresa Flowinn, pela experiência, conhecimento e confiança transmitida desde o primeiro dia.

A todos muito obrigado!

Índice

1 Introdução 1

1.1 Contexto 1

1.2 Problema 2

1.3 Objetivos 2

1.4 Resultados esperados 3

1.5 Hipótese de investigação 4

2 Contexto teórico 5

2.1 Arquiteturas monolíticas 5

2.2 Arquiteturas baseadas em microsserviços 7

2.3 Domain-driven design 8

2.4 Base de dados em microsserviços 8

2.4.1 Base de dados partilhada 9

2.4.2 Base de dados para cada microsserviço 9

2.5 Padrão *Saga* 9

2.6 Padrão *CQRS* 10

2.7 Event sourcing 10

2.8 API Management 11

2.9 API Gateway 12

2.10 Descoberta de serviços (*Service discovery*) 12

2.11 Comunicação síncrona entre microsserviços 13

2.12 Comunicação assíncrona entre microsserviços 13

2.12.1 *RabbitMQ* 14

2.12.2 *Apache Kafka* 15

2.12.3 Comparação entre *RabbitMQ* e *Apache Kafka* 16

2.13 Continuous integration (CI) e continuous deployment (CD) 18

2.13.1 *Jenkins* 19

2.13.2 *Bitbucket Pipelines* 19

2.13.3 Comparação entre *Jenkins* e *Bitbucket Pipelines* 19

2.13.4 *Docker* 20

2.13.5 *Kubernetes* 21

2.13.6 *Helm* 22

2.14 Cloud computing 22

2.14.1 *Google Cloud Platform* 23

2.14.2 *Amazon Web Services* 23

2.14.3 *Microsoft Azure* 24

2.14.4 DigitalOcean 24

2.15 JMeter 24
2.16 Servidor de autenticação único 24

3 Migração entre arquiteturas 27

3.1 Estratégias de migração 27
3.2 Exemplos de Migrações 29
3.2.1 Wix 29
3.2.2 Netflix 30

4 Análise de valor 31

4.1 Novo Modelo de Desenvolvimento de Conceitos 31
4.1.1 Identificação da Oportunidade 32
4.1.2 Análise da Oportunidade 33
4.1.3 Geração e Enriquecimento de Ideias 34
4.1.4 Seleção da Ideia 34
4.1.5 Definição de conceito 34
4.2 Valor para o cliente e valor percebido pelo cliente 35
4.3 Business Model Canvas 35
4.4 Analytic Hierarchy Process 38

5 Abordagens tecnológicas 41

5.1 Gateway e JHipster Registry 41
5.2 Gateway e Consul 42
5.3 WSO2 API Manager 44
5.4 Avaliação de abordagens 45

6 Estudo de caso – Logistics 51

6.1 Aplicação monolítica – "As is" 51
6.1.1 Arquitetura 51
6.1.2 Componentes 53
6.1.3 Implantação 55
6.2 Arquitetura baseada em microsserviços – "To be" 56
6.2.1 Gateway e JHipster Registry 57
6.2.2 Base de dados para cada microsserviço 57
6.2.3 Apache Kafka 58
6.2.4 Bitbucket Pipelines 58
6.2.5 Docker 59
6.2.6 Google Cloud Platform 59

7 Análise e conceção 61

7.1 Requisitos 61
7.2 Representação do domínio 63
7.2.1 Gateway 63

7.2.2 MS1 – *Zones* 64
7.2.3 MS2 – Product 64
7.2.4 MS5 – Numbering Series 65
7.2.5 MS6 – Stock 66
7.2.6 MS8 – Reception 66
7.2.7 MS10 – Integrator 67
7.2.8 MS14 – Optimization 68
7.2.9 MS15 – Intelligence 68
7.3 Processos do armazém 69
7.3.1 Receção 69
7.3.2 Arrumação 72
7.3.3 Expedição 74
7.4 Processos de sincronização 78
7.5 Padrões utilizados 79
8 Implantação da nova solução 81
8.1 Bitbucket Pipelines 81
8.2 Kubernetes 84
8.2.1 Implantação na *Google* 85
8.2.2 *DigitalOcean* 88
9 Avaliação 91
9.1 Métricas 91
9.1.1 Desempenho e escalabilidade 91
9.1.2 Manutenibilidade, automatização e satisfação 98
10 Conclusão 105
10.1 Trabalho futuro 106
A Anexos 111
A.1 Microsserviços 111
A.2 Bitbucket Pipelines e Jira 114
A.2.1 *Pipeline* definido 114
A.2.2 Integração *Jira* 120
A.3 Ficheiro de configuração do Kubernetes 123
A.4 Avaliação 126
A.4.1 Desempenho e escalabilidade 127
A.4.2 Manutenibilidade, automatização e satisfação 129

Lista de Figuras

Figura 1 – *Service Registry Pattern*, baseado em (Chris Richardson, 2018c) 13
Figura 2 – "Arquitetura geral do *RabbitMQ* simplificada" (Pieter Humphrey, 2017) 15
Figura 3 – "Arquitetura global *Apache Kafka*" (*Kafka Architecture*, 2017) 16
Figura 4 – Etapas do padrão *Strangler Fig Application* (Samir Behara, 2018) 28
Figura 5 – Representação do Novo Modelo de Desenvolvimento de Conceitos (Koen et al., 2002) 31
Figura 6 – Diagrama de componentes utilizando *JHipster Registry,* baseado em (JHipster, 2017a) 41
Figura 7 – Diagrama de componentes utilizando *Consul*, baseado em (JHipster, 2017b) 43
Figura 8 – Diagrama de componentes utilizando *WSO API Manager*, baseado em (Skalena, 2018) 44
Figura 9 – Diagrama de componentes das aplicações monolíticas 52
Figura 10 – Diagrama de componentes da aplicação Logistics 54
Figura 11 – Diagrama de componentes da aplicação Logistics Intelligence 55
Figura 12 – Diagrama de implantação da aplicação monolítica 56
Figura 13 – Diagrama de componentes da solução a implementar 57
Figura 14 – Ferramentas a utilizar na automatização do processo de implantação 58
Figura 15 – Diagrama de casos de uso 62
Figura 16 – Microsserviços *Logistics* 63
Figura 17 – Representação do microsserviço *Zones* 64
Figura 18 – Representação do microsserviço *Product* 65
Figura 19 – Representação do microsserviço *Numbering Series* 66
Figura 20 – Representação do microsserviço *Stock* 66
Figura 21 – Representação do microsserviço *Reception* 67
Figura 22 – Representação do microsserviço *Integrator* 67
Figura 23 – Representação do microsserviço *Optimization* 68
Figura 24 – Representação do microsserviço *Intelligence* 68
Figura 25 – Diagrama de sequência de rececionamento de produtos 70
Figura 26 – Diagrama de sequência de finalização da receção 72
Figura 27 – Diagrama de sequência da arrumação de um produto 73
Figura 28 – Diagrama de sequência da aprovação da encomenda 75
Figura 29 – Diagrama de sequência da estimativa das encomendas 77
Figura 30 – Diagrama de sequência da importação de produtos 78
Figura 31 – *Pipeline* desenvolvido no *Bitbucket Pipelines* 82
Figura 32 – *Step* de implantação na *DigitalOcean* 84
Figura 33 – Diagrama de implantação de alta granularidade 85
Figura 34 – Diagrama de implantação na *Google Cloud Platform* 87
Figura 35 – Diagrama de implantação na *DigitalOcean* 89
Figura 36 – Tempo médio de resposta na criação de linhas de receção 93
Figura 37 – Tempo de resposta médio na finalização das receções 93

Figura 38 – Duração do processo de finalização das receções 94
Figura 39 – Tempo de resposta médio do processo de arrumação 95
Figura 40 – Duração média do processo de arrumação 95
Figura 41 – Duração média do processo de expedição 96
Figura 42 – Representação do microsserviço *Entities* 111
Figura 43 – Representação do microsserviço *Settings* 112
Figura 44 – Representação do microsserviço *Operation* 112
Figura 45 – Representação do microsserviço *NMVS* 113
Figura 46 – Representação do microsserviço *Stowage* 113
Figura 47 – Representação do microsserviço *Container* 113
Figura 48 – Representação do microsserviço *Shipping Order* 114
Figura 49 – *Steps* executados conforme o *branch* 115
Figura 50 – Definição dos steps de configuração do ambiente 115
Figura 51 – Definição do *step build* 116
Figura 52 – Definição do *step Docker Hub* 117
Figura 53 – *Step* de implantação da imagem no *Kubernetes* 118
Figura 54 – *Script* de implantação 119
Figura 55 – Fluxo do ciclo de vida de um *issue* existente na empresa (imagem cedida pela *Flowinn)* 120
Figura 56 – Ambientes de implantação definidos no *Bitbucket* 121
Figura 57 – Informação detalhada do ambiente de testes *DigitalOcean* 122
Figura 58 – *Issue* LWMS-15 representado Jira 123
Figura 59 – Detalhe das implantações do *issue* LWMS-15 123
Figura 60 – Configuração do *deployment* do microsserviço *Zones* 124
Figura 61 – Configuração do container presente no *pod* 124
Figura 62 – Ficheiro da implantação da base de dados do microsserviço zones 126
Figura 63 – QEF utilizado para avaliar a solução desenvolvido 130
Figura 64 – Questionário apresentado e 1ª pergunta 131
Figura 65 – 2ª pergunta do questionário 131
Figura 66 – 3ª pergunta do questionário 131
Figura 67 – 4ª pergunta do questionário 131
Figura 68 – 5ª pergunta do questionário 131
Figura 69 – 6ª pergunta do questionário 132
Figura 70 – Respostas obtidas na 1º pergunta 132
Figura 71 – Respostas obtidas na 2º pergunta 132
Figura 72 – Respostas obtidas na 3º pergunta 133
Figura 73 – Respostas obtidas na 4º pergunta 133
Figura 74 – Respostas obtidas na 5º pergunta 133
Figura 75 – Respostas obtidas na 6º pergunta 133

Lista de Tabelas

Tabela 1 – Comparação entre *RabbitMQ* e *Apache Kafka* 16
Tabela 2 – Comparação entre *Jenkins* e *Bitbucket Pipelines* 20
Tabela 3 – Business Model Canvas 36
Tabela 4 – Comparação dos critérios elegidos 46
Tabela 5 – Soma das colunas da comparação dos critérios 46
Tabela 6 – Matriz normalizada 46
Tabela 7 – Resultados do processo de sincronização das entidades 97
Tabela 8 – Resultados do processo de sincronização dos produtos 97
Tabela 9 – Avaliação do fator documentação 98
Tabela 10 – Avaliação do fator monitorização 99
Tabela 11 – Avaliação do fator processo de implantação 100
Tabela 12 – Objetivos e questões 101
Tabela 13 – Classificação das respostas 102
Tabela 14 – Avaliação do fator satisfação da empresa 103
Tabela 15 – Variáveis utilizadas no *step build* 116
Tabela 16 – Variáveis utilizadas no *step Docker Hub* 117
Tabela 17 – Variáveis utilizadas no *step* de implantação 119
Tabela 18 – DNS utilizado para cada comunicação 125
Tabela 19 – Criação das linhas de receção na aplicação monolítica 127
Tabela 20 – Criação das linhas de receção na aplicação baseada em microsserviços 127
Tabela 21 – Processo de expedição na aplicação monolítica 128
Tabela 22 – Processo de expedição na aplicação baseada em microsserviços 129

Acrónimos e Símbolos

Lista de Acrónimos

CD	*Continuous Deployment*
SPR	*Single Principle Responsibility*
CQRS	Command Query Responsibility Segregation
MB	*Message Broker*
AMQP	*Advanced Message Queuing Protocol*
CI	*Continuous Integration*
UI	*User Interface*
IaaS	*Infrastructure as a Service*
PaaS	*Platform as a Service*
SaaS	*Software as a Service*
AHP	*Analytic Hierarchy Process*
CTO	*Chief Technology Officer*
DNS	*Domain Name System*
HTTP	*Hypertext Transfer Protocol*
API	*Application Programming Interface*
ERP	*Enterprise Resource Planning*
HTML	*HyperText Markup Language*
REST	*Representational State Transfer*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
JDBC	*Java Database Connectivity*
GB	*Gigabyte*
QEF	*Quality evaluation framework*

JWT *Json Web Token*

1 Introdução

1.1 Contexto

Este trabalho, resulta da necessidade de melhorar o desempenho de uma aplicação denominada de *Logistics*, desenvolvida e comercializada pela empresa Flowinn que atua no mercado de Tecnologias de Informação para a Gestão. Esta empresa dedica-se ao desenvolvimento de aplicações de gestão com o intuito de otimizar os processos diários, de alguns tipos de indústrias, maioritariamente do ramo farmacêutico, mas também na indústria alimentar, metalomecânica pesada e de madeiras.

A referida aplicação, surgiu com o intuito de promover uma melhoria no funcionamento interno dos armazéns de distribuição da indústria farmacêutica, aumentando assim a sua eficiência. Os armazéns são espaços idealizados para armazenar produtos em grande quantidade, sendo necessário utilizar uma estrutura coerente e organizada permitindo, rececionar, manobrar e expedir os mais diversos tipos de produtos com um controlo moderado das condições ambientais e de segurança.

A gestão de armazéns inclui um conjunto de atividades a jusante e a montante, nomeadamente a receção e o armazenamento de produtos de acordo com as necessidades, a recolha de produtos de acordo com as encomendas dos clientes e a preparação dos mesmos para expedição. Para qualquer tipo de indústria, o processo de gestão de armazéns tem como principais objetivos, a redução de espaço ocupado e de tempo despendido, uma vez que agrega um conjunto de operações que, na cadeia de fornecimento (*Supply Chain*) do produto, não acrescentam valor. Além disso, em particular na indústria farmacêutica, é necessário dar uma rápida resposta aos pedidos efetuados, pelo que o fator tempo adquire uma elevada importância.

1.2 Problema

Atualmente, a aplicação *Logistics* apresenta uma arquitetura monolítica, implementada através da *framework JHipster*, utilizando *Spring* como *back-end* e *Angular* 8 como *front-end*.

Com o objetivo de otimizar as tarefas essenciais no armazém dos clientes, é fulcral obter uma rápida resposta aos pedidos efetuados, permitindo a sua satisfação atempada, por isso os clientes desta aplicação (maioritariamente da indústria farmacêutica), têm solicitado algumas funcionalidades, pelo que a aplicação tem sofrido algumas atualizações ao longo do tempo. As referidas atualizações foram sendo sucessivamente realizadas na aplicação existente, provocando, atualmente, inúmeros problemas tais como, manutenibilidade do código fonte, difícil implementação de novas funcionalidades, quebra de funcionalidades já implementadas aquando nova implementação e grande dimensão da aplicação tornando as *builds* extensas, sendo estas, por vezes, simples correções de uma única tarefa.

Além das questões referidas, a aplicação *Logistics* não implementa testes, sendo obrigatória a execução de testes manuais, por isso mais falíveis, podendo não detetar falhas tanto nas novas funcionalidades como nas previamente implementadas.

É de salientar que o processo de disponibilização das novas funcionalidades para cada um dos clientes é executado manualmente. Este, é suscetível a falhas de configuração ou erros humanos, que por sua vez podem paralisar o ambiente de produção, resultando em prejuízo para os clientes.

Assim pode considerar-se que existem basicamente quatro grandes problemas a resolver:

- Elevado tempo de resposta;
- Algum descontrolo nas atualizações;
- Realização de testes manualmente;
- Processo de implantação de forma manual.

1.3 Objetivos

Com base no problema descrito anteriormente, observa-se de forma imediata quatro objetivos:

- Redução do tempo de resposta;
- Controlo de atualizações;
- Realização de testes de forma automática;
- Disponibilização automática das novas funcionalidades desenvolvidas.

Com vista à concretização dos objetivos acima descritos, o trabalho a desenvolver nesta dissertação deverá implementar uma arquitetura que permita responder aos problemas referidos, melhorando o desempenho e escalabilidade da aplicação, relativamente à arquitetura monolítica atual. Assim, torna-se necessário efetuar uma análise dos requisitos pretendidos para se prosseguir com a escolha cuidada das tecnologias a utilizar.

Ao contrário do produto existente na empresa, pretende-se que, esta nova versão, implemente testes automatizados para as diversas funcionalidades, de modo a facilitar implementações futuras e aumentar a fiabilidade do código fonte.

Posteriormente às implementações efetuadas, é expectável a automatização do processo de implantação dos serviços nos respetivos clientes, aplicando o conceito de CD (*Continuous Deployment*), eliminando-se, assim, os possíveis erros causados pelos processos manuais, tornando o processo de implantação repetível e confiável (Jez Humble & David Farley, 2010).

Atualmente, a empresa Flowinn, disponibiliza um serviço de autenticação, que visa eliminar a necessidade dos clientes que possuem vários produtos fazerem *login* em cada um dos mesmos. Este serviço permite usufruir de uma sessão transversal aos produtos, uma vez que esta é armazenada no servidor de autenticação.

Assim, pretende-se que a aplicação a desenvolver, comunique com o serviço de autenticação referido.

1.4 Resultados esperados

É expectável implementar uma solução arquiteturalmente mais moderna, baseada em microsserviços, que cumpra os objetivos acima descritos e que visa solucionar os problemas atuais da empresa Flowinn. A solução desenvolvida, deverá prosseguir na

empresa e a sua implantação nos clientes deverá ser automática e independente para cada microsserviço. Com isto, é de esperar que os problemas como, difícil manutenção e implementação de novas funcionalidades, erros manuais e falhas na detenção de funcionalidades incorretamente implementadas, sejam reduzidos ou se possível eliminados.

1.5 Hipótese de investigação

No caso em apreço, será que a transformação da aplicação, que atualmente assenta numa arquitetura monolítica, para uma arquitetura baseada em microsserviços, trará vantagens no que respeita aos problemas identificados e objetivos a atingir acima descritos?? Será que se obterão os resultados esperados referidos na seção anterior?

A resposta poderá surgir a partir da transformação da aplicação de uma arquitetura monolítica, para uma arquitetura baseada em microsserviços, e posterior avaliação.

Avaliação esta relativa os benefícios da solução para a empresa Flowinn, nomeadamente ao nível do desempenho e manutenção da solução. Esta avaliação será efetuada através da comparação da solução atual e da solução desenvolvida após a conclusão deste trabalho.

Por fim, pode também avaliar-se os benefícios de implantação contínua no quotidiano da solução e o seu impacto no desenvolvimento.

2 Contexto teórico

Neste capítulo encontram-se descritas de forma sucinta as definições das arquiteturas monolítica e de microsserviços, indicando vantagens e desvantagens de cada uma delas, e ainda padrões a utilizar para se prosseguir com a transformação da primeira na segunda.

2.1 Arquiteturas monolíticas

Nesta secção, apresenta-se o conceito de arquitetura monolítica, descrevendo as suas vantagens e desvantagens. Uma grande parte dos sistemas utilizados por muitas organizações, têm por base uma arquitetura monolítica, formadas por vários módulos que são compilados separadamente e depois ligados, formando assim um grande sistema onde os módulos podem interagir. Por vezes estes sistemas são bastante complexos. Hipoteticamente numa situação estática, tal configuração não acarreta problemas, no entanto, pensando um modo evolutivo, a complexidade atinge um nível significativo, tornando-se uma tarefa muito difícil para os diferentes programadores destas aplicações, devido à complexidade e tamanho de código existente em cada um destes sistemas.

As arquiteturas monolíticas são compostas apenas por um segmento, onde se encontram presente todos os módulos necessários para o funcionamento da aplicação de *software*. Estes módulos são executados todos na mesma máquina, partilhando recursos de processamento e memória (Chris Richardson, 2018c).

Este segmento, geralmente apresenta uma grande dimensão, visto que tradicionalmente são constituídas pelas seguintes camadas (Daniel Viana, 2017):

- *Client side* ou *front-end*: primeira camada visível para o utilizador quando acede à aplicação. Esta contém componentes relacionados com a interface do utilizador, *design*, usabilidade e interação com o mesmo;

- *Server side* ou *back-end*: conjunto de camadas responsáveis por implementar as funcionalidades da aplicação, aceder a camadas de persistência de dados, entre outros.

Este tipo de arquitetura é das mais antigas e projeta aplicações sem modularidade externa, ou seja, não visam ser módulos de outras aplicações.

As vantagens e desvantagens (Chris Richardson, 2018c) desta arquitetura são as a seguir apresentadas.

Vantagens:

- Fácil desenvolvimento;
- Pode sofrer, facilmente, alterações radicais tanto a nível de código, como a nível de base de dados;
- Fácil de implantar;
- Fácil implementação de testes de ponta-a-ponta (*end-to-end*);
- Fácil de escalar horizontalmente o monólito.

Desvantagens:

- Manutenibilidade do código-fonte;
- Alta dependência dos componentes do código;
- Elevado tamanho da aplicação, em casos mais complexos, prejudica o processo de CD;
- Necessidade de implantar toda a aplicação quando existem novas alterações;
- Escalabilidade limitada, pois exige que todo o sistema seja replicado;
- Confiabilidade reduzida, dado que uma falha em um dos módulos, poderá levar à falha de todo o sistema;
- Ausência de flexibilidade, visto que exige o compromisso com a tecnologia escolhida inicialmente para a aplicação.

Nos últimos anos tem-se vindo a adotar uma abordagem de implementação de arquiteturas baseada em microsserviços, que surgiu como um padrão comum de

desenvolvimento de software a partir das melhores práticas de uma série de organizações líderes e o seu esforço na construção de aplicações de grandes dimensões e continuamente evolutivas, capazes de reagir rapidamente aos requisitos de software sempre em mudança (Cesar de la Torre et al., 2017).

2.2 Arquiteturas baseadas em microsserviços

As arquiteturas baseadas em microsserviços, contrariamente às arquiteturas monolíticas, utilizam uma abordagem à modularização de software, consistem num conjunto de aplicações constituídas por uma coleção de serviços, designados de microsserviços. Estes representam unidades que adotam o padrão SPR (*Single Principle Responsibility*), serviços que devem ter responsabilidade de apenas uma parte da aplicação, tornando assim cada unidade menos acoplada (Fanis Despoudis, 2017). Este desacoplamento permite que cada serviço possa ser desenvolvido por equipas diferentes, implementado e mantido de forma independente. A comunicação destes serviços ínfimos, permite resolver problemas de negócios complexos, no entanto apresentam algumas vantagens e desvantagens que devem ser consideradas antes da sua implantação (Chris Richardson, 2018c).

Algumas vantagens:

- Maior desacoplamento;
- Em caso de falha de um microsserviço, não há comprometimento da integridade total da aplicação;
- Permite experimentação e a utilização de diferentes tecnologias;
- Facilita o processo de CD de aplicações grandes e complexas, através da implantação dos diversos serviços;
- Constituído por pequenas porções de código em cada serviço, facilitando a sua manutenção ao longo do ciclo de vida;
- Existência de unidades independentes, permitindo assim, desenvolvimento, implantação e escalabilidade diferente entre si, inclusive estes processos podem ser efetuados por equipas distintas e independentes.

Desvantagens:

- A definição correta dos serviços é um processo complexo;

- Implementação de funcionalidades que necessitam de vários microsserviços, podem exigir coordenação entre equipas;
- Maior dificuldade de implementação de testes de integração;
- Processo de implantação exige a coordenação de um elevado número de serviços a implantar;
- É necessário um tratamento especial por parte dos programadores nos processos de falhas.

Atualmente, existem diversos padrões que visam solucionar problemas comuns em arquiteturas distribuídas baseadas em microsserviços, tais como, distribuição da base de dados e consistência da informação, descoberta e comunicação dos microsserviços, entre outros. Ao longo deste capítulo serão apresentados padrões e algumas boas práticas a ter em conta para a implementação deste tipo de arquiteturas.

2.3 Domain-driven design

O padrão DDD (*Domain-driven design*), segundo (Evans, 2003) consiste no alinhamento entre a implementação da solução tecnológica e os conceitos de negócio o que facilita a implementação de soluções presentes em domínios complexos.

A utilização deste padrão exige a perceção do domínio de negócio, de modo a que este seja refletido na implementação e estruturação do código implementado.

Este padrão apresenta vantagens que poderão beneficiar o desenvolvimento deste projeto, nomeadamente, facilita a comunicação entre os programadores e os analistas e permite definir os contextos do domínio e as suas características. Esta definição por sua vez, auxilia a implementação de serviços desacoplados o que permite a sua reutilização ao longo da solução.

2.4 Base de dados em microsserviços

A migração de uma arquitetura monolítica para uma arquitetura baseada em microsserviços, exige a seleção do padrão a utilizar relativamente à base de dados da nova solução.

Nas secções seguintes apresentam-se dois padrões distintos, base de dados partilhada e base de dados por microsserviço.

2.4.1 Base de dados partilhada

Este conceito consiste na utilização de uma base de dados partilhada entre vários microsserviços (Chris Richardson, 2018b). Assim, quando um microsserviço necessita de informação acede à base de dados, evitando comunicações com outros microsserviços. A implementação deste padrão é simples e as transações locais são efetuadas de forma atómica e consistente. No entanto, a utilização deste padrão apresenta inúmeras desvantagens (Chris Richardson, 2018b), tais como:

- Alto acoplamento no desenvolvimento e execução: Necessidade de coordenação das alterações, caso existam programadores a trabalhar em diferentes serviços que tenham acesso à mesma tabela;
- Quantidade de pedidos por parte de todos os microsserviços pode causar sobrecarga e falhas na base de dados única.

2.4.2 Base de dados para cada microsserviço

De modo a colmatar as desvantagens do padrão descrito na secção anterior, surge o padrão *database per service* (Chris Richardson, 2018a), este consiste na utilização de uma base de dados distinta para cada microsserviço.

Com a utilização deste padrão, cada microsserviço é responsável pelos seus dados, impossibilitando a alteração externa dos mesmos. Os microsserviços tornam-se menos acoplados garantindo-se que alterações à base de dados de um microsserviço, não afetem as restantes. Dado que cada base de dados é independe, a seleção da tecnologia a utilizar torna-se independente também, possibilitando o uso de diferentes tecnologias no armazenamento de dados conforme a necessidade do microsserviço. Contudo, este padrão apresenta algumas desvantagens (Chris Richardson, 2018a), que devem ser consideradas:

- Quando é pedida informação que necessita dos dados presentes em microsserviços distintos, é necessária a consulta de ambos;
- Transações que interajam com diversos microsserviços necessitam de ser controladas, com o intuito de manter coerente a informação de todas as bases de dados.

2.5 Padrão *Saga*

No cenário da existência de uma base de dados distinta para cada microsserviço, é necessário manter a sua coerência e assegurar as validações do pedido efetuado. Para controlar estas situações deve ser utilizado o padrão Saga (Umesh Ram Sharma, 2017).

Este padrão, consiste "na sequência de transações locais. Cada microsserviço atualiza a sua base de dados e publica um evento para ativar a próxima transação". No caso de a transação falhar, a Saga executa transações de modo a reverter as alterações anteriormente efetuadas. Este processo aumenta a complexidade do código implementado (Umesh Ram Sharma, 2017).

2.6 Padrão *CQRS*

O padrão CQRS (*Command Query Responsibility Segregation*), consiste na separação dos conceitos de leitura e escrita da base de dados. Estes conceitos apresentam diferentes responsabilidades, sendo o primeiro responsável por consultar e apresentar os dados pedidos, acedendo à base de dados em modo leitura e o segundo é responsável pela escrita dos dados (Dominic Betts et al., 2013; Edson Yanaga, 2017).

Este padrão permite a utilização da mesma ou diferentes bases de dados, podendo estas armazenar diferentes esquemas de dados. Assim, é possível tirar partido das seguintes vantagens (Dominic Betts et al., 2013):

- Separação dos conceitos aperfeiçoada;
- Escalabilidade independente: Permite escalar de forma independente os componentes de leitura e escrita;
- Permite a consulta de dados em implementações de *event sourcing*.

No entanto, este padrão apresenta também algumas desvantagens (Dominic Betts et al., 2013):

- Aumento da complexidade do sistema: Necessidade de desenvolver um mecanismo de sincronização quando existem várias bases de dados;
- Possíveis inconsistências dos dados: Os utilizadores podem aceder a dados que ainda não foram atualizados.

2.7 Event sourcing

Segundo Chris Richardson (Dominic Betts et al., 2013), o padrão *Event sourcing*, consiste no armazenamento de uma sequência de eventos, sendo que, cada evento representa uma alteração do estado da entidade. Através do histórico de eventos, é possível obter os diversos estados e reconstituir o percurso até ao estado atual da entidade.

Algumas vantagens deste padrão são:

- Preservação do histórico de eventos, facilitando a identificação em caso de falhas ou comportamentos indesejados do sistema;
- Possibilidade de verificar o estado de uma entidade em qualquer momento (Dominic Betts et al., 2013; Martin Fowler, 2005).

Tal como os anteriores também este padrão apresenta algumas desvantagens (Dominic Betts et al., 2013; Martin Fowler, 2005) como:

- Necessidade de aprendizagem de implementação do padrão;
- Aumento a complexidade do sistema.

2.8 API Management

O API *Management,* é uma plataforma centralizada de gestão das API constituintes de um produto. O objetivo desta plataforma é "*criar, analisar e gerir os API num ambiente seguro e escalável*" (Brajesh De, 2017).

Esta plataforma deve integrar as seguintes funcionalidades (Brajesh De, 2017):

- API *Gateway*: Ponto de entrada do produto, que será explicado no subcapítulo seguinte;
- Monitorização das API: Apresenta a informação do fluxo de acesso, as datas, os consumidores, o sucesso ou insucesso das respostas e o desempenho de cada API;
- Restrição das API: A utilização de uma API pode ser configurada, permitindo aplicar a determinados consumidores, limites ou políticas de uso;
- Portal dos desenvolvedores: Este portal permite a gestão das API existentes. A documentação de cada API deverá ser desenvolvida e guardada neste portal, juntamente com a própria;

- Autenticação e segurança das API;
- Gestão do ciclo de vida da API.

2.9 API Gateway

O padrão API *Gateway* é considerado o ponto de entrada para os clientes. Estes efetuam os seus pedidos para o *Gateway* que, por sua vez, reencaminha os pedidos para o(s) respetivo(s) microsserviço(s) reforçando aspetos de segurança, taxas de uso, validação e transformação da informação (Brajesh De, 2017).

A utilização deste padrão permite isolar os microsserviços implementados dos clientes, assim estes não necessitam de saber a localização de cada microsserviço, apenas a localização do API *Gateway*, o que contribui para a simplificação do código desenvolvido em cada cliente.

As desvantagens deste padrão, passam pelo aumento da complexidade e necessidade de adicionar o componente à solução. Este componente exige desenvolvimento, implantação e alta disponibilidade, para que possa receber sempre os pedidos dos clientes. Por fim, o tempo de resposta pode aumentar, uma vez que é adicionado um componente extra (Umesh Ram Sharma, 2017).

2.10 Descoberta de serviços (*Service discovery*)

"Em aplicações monolíticas implantadas em máquinas físicas, para efetuar o pedido de uma funcionalidade, não existe necessidade de descobrir a localização do serviço, uma vez que a sua localização maioritariamente é estática" (Chris Richardson, 2018c). Este cenário difere em arquiteturas implantadas na nuvem.

"As instâncias dos microsserviços são atribuídas dinamicamente em localizações da nuvem". Assim, surge a "necessidade de descobrir a localização dos serviços" (Chris Richardson, 2018c). Com esta necessidade, surgiu o padrão Service registry (Chris Richardson, 2016b), que consiste na implementação de um componente, Service registry, responsável por armazenar a instância do serviço e a sua localização. Este padrão relaciona-se com o padrão Self Registration, uma vez que, segundo (Chris Richardson, 2016a), as instâncias de microsserviços devem ser registadas ou removidas do componente nas seguintes circunstâncias:

- Quando inicializa deve registar-se;

- Quando termina, devido a um erro ou voluntariamente, deve ser eliminada;
- Deve renovar o seu registo periodicamente.

Através dos dois padrões acima descritos, quando é efetuado o pedido de uma funcionalidade, "deverá ser consultado o componente, Service Registry, com o intuito de obter a lista de instâncias e as respetivas localizações das mesmas", como ilustrado na Figura 1.

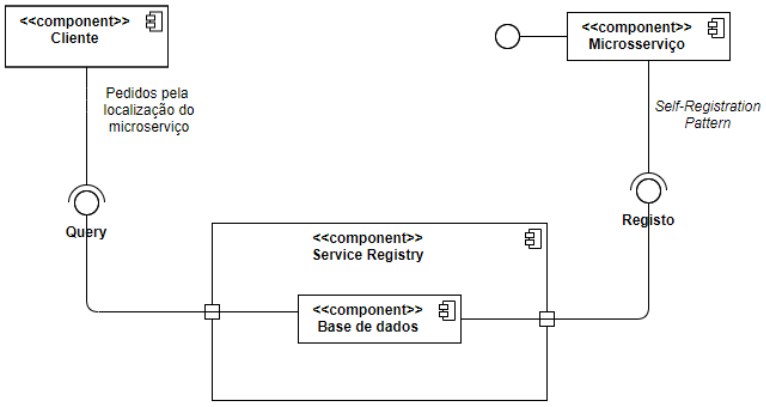

Figura 1 – *Service Registry Pattern*, baseado em (Chris Richardson, 2018c)

2.11 Comunicação síncrona entre microsserviços

A comunicação nos microsserviços pode ser efetuada de forma síncrona, constituída por um remetente e um destinatário. O remetente efetua o pedido e fica bloqueado à espera do momento em que o destinatário lhe responde.

A comunicação síncrona, pode ser facilmente implementada permitindo respostas em tempo real, em contrapartida esta exige conhecer a localização do microsserviço destinatário, sendo que, este tem obrigatoriamente, de estar disponível no momento da comunicação, caso contrário esta não resultará. Este tipo de comunicação oferece uma baixa disponibilidade, sendo difícil garantir que a funcionalidade pretendida é executada pelo sistema. Em comparação com a comunicação assíncrona, esta apresenta maior acoplamento ao longo do tempo. Considerando a necessidade de conhecimento do contrato do microsserviço destinatário, uma mudança deste contrato obriga o remetente a adaptar-se.

2.12 Comunicação assíncrona entre microsserviços

A comunicação assíncrona surgiu com o intuito de solucionar problemas existentes da comunicação síncrona, estes eram encontrados com elevada frequência em arquiteturas baseadas em microsserviços.

A utilização de um MB (*Message Broker*), permite solucionar os problemas acima descritos através da utilização de mensagens, proporcionando comunicação assíncrona entre microsserviços. Através do MB, o microsserviço remetente não necessita de saber a localização do destinatário e, de modo a realizar a comunicação, escreve a mensagem no MB e este entrega a mensagem ao(s) destinatário(s), eliminado o acoplamento entre microsserviços (Chris Richardson, 2018c).

A comunicação assíncrona entre microsserviços apresenta algumas vantagens e desvantagens (Chris Richardson, 2018c).

Vantagens:

- Nenhum acoplamento entre microsserviços (acima descrito);
- Maior confiabilidade: contrariamente à comunicação síncrona, o destinatário pode estar indisponível quando o recetor escreve a mensagem, esta será armazenada numa fila de mensagens até que o destinatário a possa processar;
- Aumento do desempenho dos microsserviços: após o envio da mensagem para a fila do MB, o microsserviço continuará disponível para responder aos pedidos efetuados, contrariamente ao que acontece quando comunica sincronamente, uma vez que fica temporariamente indisponível, à espera da resposta.

Desvantagens:

- Possível problema de desempenho: O MB deve apresentar um bom desempenho para possibilitar e acompanhar o crescimento do sistema;
- Possível ponto único de falha: em caso de falha do MB, a confiabilidade do sistema é afetada;
- Complexidade adicionada: "O MB é um componente que deve ser instalado e configurado".

2.12.1 *RabbitMQ*

O *RabbitMQ* "*é uma ferramenta leve e poderosa*", que permite a comunicação através de mensagens. Este é um MB *open-source*, desenvolvido em 2007, que por defeito usa o AMQP (*Advanced Message Queuing Protocol*) (Roy, 2017), porém suporta outros.

Este MB "destaca-se devido a sua facilidade de implementação, flexibilidade, disponibilidade de ferramentas complementares e API simples e intuitiva" (Vinicius Feitosa Pacheco, 2018).

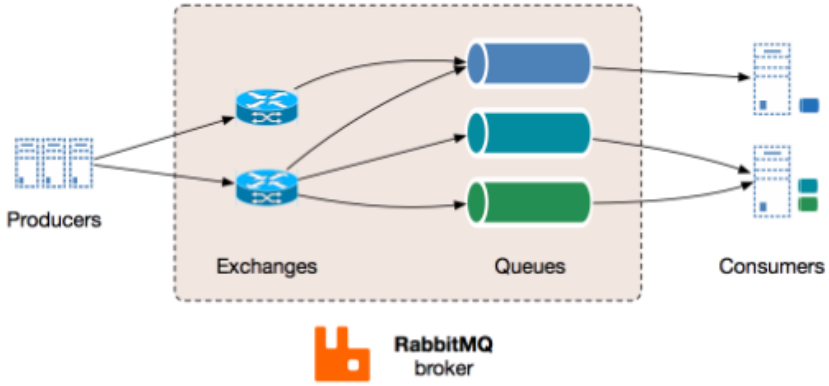

Figura 2 – "Arquitetura geral do *RabbitMQ* simplificada" (Pieter Humphrey, 2017)

Através da análise do diagrama da Figura 2, podem identificar-se os seguintes componentes:

- *Producers* (Produtores): componente que publica mensagens no *RabbitMQ*;
- Exchanges (Trocas): componente responsável por "receber as mensagens dos produtores e atribuí-las às respetivas filas, dependendo das regras definidas";
- *Queues* (Filas): componente que armazena as mensagens até que os consumidores as recebam;
- *Consumers* (Consumidores): componente que recebe mensagens.

2.12.2 *Apache Kafka*

O *Apache Kafka* surgiu em 2011, "*com o intuito de escapar às limitações dos MB tradicionais*" (Jakub Korab, 2017). Foi desenhado para suportar um elevado fluxo de mensagens, sendo considerado por (Vinicius Feitosa Pacheco, 2018), o MB "*com melhor desempenho e o mais escalável para a entrega de mensagens*".

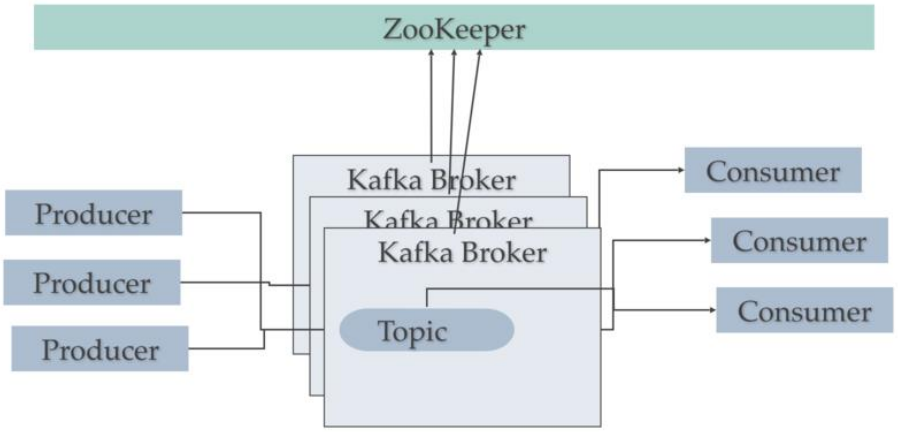

Figura 3 – "Arquitetura global *Apache Kafka*" (*Kafka Architecture*, 2017)

Contrariamente ao *RabbitMQ*, não funciona sem o servidor *ZooKeeper*, representado na Figura 3.

O *ZooKeeper* é um serviço *open-source*, desenvolvido pela *Apache Software Foundation*, que disponibiliza funcionalidades importantes para aplicações distribuídas, tais como, eleição de um líder, armazenamento de objetos através de uma chave única e coordenação de processos (Stephane Maarek, 2019).

Neste contexto, o *ZooKeeper* tem a responsabilidade de administrar o MB *Apache Kafka*, guardar os produtores e consumidores.

2.12.3 Comparação entre *RabbitMQ* e *Apache Kafka*

Na Tabela 1 pode observar-se uma comparação entre o MB *RabbitMQ* e o MB *Apache Kafka* relativamente a algumas características (Jakub Korab, 2017; Vinicius Feitosa Pacheco, 2018; *What Is Apache Kafka?*, 2019).

Tabela 1 – Comparação entre *RabbitMQ* e *Apache Kafka*

Características	RabbitMQ	Apache Kafka
Open-Source	Sim	Sim

Escalabilidade	Elevada, contudo, inferior ao *Apache Kafka*, "Vinte mil mensagens por segundo"	Elevada, "devido à partição de tópicos, permitindo assim, leitura paralela do tópico". Pode facilmente ser distribuído devido às suas partições. "Cem mil mensagens por segundo"
Disponibilidade	Alta	Alta
Desempenho	Elevada, contudo, necessita de mais recursos	Elevada
Modelo de Operação	"*Produtor envia a mensagem e o consumidor recebe as mensagens das respetivas filas que subscreveu*". "*Neste processo, o roteamento da mensagem é efetuado pelo MB*"	Produtor envia mensagem que fica armazenada no respetivo tópico. "*Os consumidores são responsáveis por rastrear e ler as mensagens*"
Histórico de mensagens	A mensagem é removida após ser entregue ao consumidor	A mensagem é persistida, por um determinado período configurável
Envio da mensagem para vários consumidores	Não permite, é necessário replicar a fila e enviar a mensagem para as duas filas	Permite
Dependências	*Erlang*	*Zookeeper*
Monitorização	Interface gráfica incluída	Necessário *software*
Protocolo	*AMQP*, por defeito, com suporte via plugins para outros	Binário
Tecnologias suportadas	Muitas, *Python*, *Java*, *JavaScript*, *Spring*, entre outras	Muitas, *Erlang*, *.NET*, *Java*, *Python*, entre outras
Ideal nos seguintes cenários	"*Necessidade de suportar múltiplos protocolos*" "*Integrar consumidores com roteamento complexo*"	"*Necessidade de replicação*", histórico de mensagens, *event-sourcing*

Através da análise da Tabela 1, é possível concluir que as alternativas apresentam uma excelente qualidade nos serviços que proporcionam. A escolha da tecnologia deverá ter em conta o cenário onde se pretende integrar, o modelo de operação e o protocolo

pretendido para a comunicação. Por fim, é de realçar que o MB *Apache Kafka*, apresenta um melhor desempenho e escalabilidade do que o *RabbitMQ*, podendo este ser um fator decisivo para a escolha da alternativa a implementar.

2.13 Continuous integration (CI) e continuous deployment (CD)

O objetivo de uma arquitetura baseada em microsserviços é acelerar o desenvolvimento de *software*. Para que seja realizado com sucesso os diferentes objetivos, devem definidos a nível organizacional, a definição de pequenas equipas, autónomas, bem como a nível de processo, com a utilização de *continuous delivery* e *continuous deployment*.

Assim, com o intuito de responder rapidamente e com qualidade à demanda dos clientes, é necessário automatizar os processos de implantação, minimizando os erros manuais e os entregues ao cliente. Assim, é aconselhável a prática de CI (*Continuous Integration*). Esta preconiza que os programadores devem integrar com frequência código fonte por eles desenvolvido no repositório partilhado, permitindo que, no caso de existência de erros, estes sejam descobertos e corrigidos rapidamente (Jez Humble & David Farley, 2010).

A frequência de integração das alterações permite evitar conflitos no momento da integração, já que o restante código fonte da equipa não será muito divergente do atual na nova funcionalidade.

A prática de CI torna mais rápido o processo de integração e entrega das novas funcionalidades. A qualidade do *software,* deve ser assegurada através de processos automáticos de compilação e testes ao código desenvolvido (Martin Fowler, 2006).

CD consiste na implantação autónoma das alterações efetuadas ao código-fonte, realizando testes automáticos à aplicação, de modo a validar se as alterações são corretas e garantir que a implantação efetuada fica estável e disponível. Este, tem como objetivo diminuir o tempo entre o momento de desenvolvimento do código e o de entrega ao cliente, através do processo automático de implantação no ambiente de produção (Jez Humble & David Farley, 2010).

Benefícios da prática de CD (Jez Humble & David Farley, 2010):

- Rápida resposta à demanda do cliente;

- Facilidade na entrega de *software* em novos ambientes: este processo, consiste na configuração do ambiente, indicando o mesmo nos *pipelines* automáticos de implantação;

- Redução de erros: o processo de implantação automático, torna a implantação repetível e reduz possíveis erros provocados por processos manuais;

- *Feedback* do cliente: a empresa receberá a opinião do cliente muito mais rápido, já que o tempo de desenvolvimento e entrega ao cliente é reduzido.

As abordagens de CI/CD, relacionam a prática de ambas, combinado os dois processos de modo a tirar partido das vantagens de ambos. Esta prática visa automatizar os processos de entrega e implantação, com o intuito de proporcionar rapidamente um produto de qualidade aos clientes.

2.13.1 *Jenkins*

O *Jenkins*, é um servidor de automatização *open-source* escrito em *Java*. Esta ferramenta, que surgiu em 2011, desenvolvida por *Kohsuke Kawaguchi*, permite automatizar o processo de desenvolvimento de *software*, através de CI e facilita o processo de CD (John Ferguson Smart, 2011).

2.13.2 *Bitbucket Pipelines*

O *Bitbucket Pipelines* é um serviço da *Atlassian* para projetos na *cloud*, disponibilizado por volta do ano 2017. Está integrado na UI (*User Interface*) do *bitbucket.* O intuito deste serviço, é facilitar os processos de CI e CD e integrar automaticamente o mesmo com o produto *Jira,* mostrando quais os problemas incluídos em cada implantação (Atlassian, 2019).

2.13.3 Comparação entre *Jenkins* e *Bitbucket Pipelines*

Na Tabela 2 pode observar-se uma comparação entre *Jenkins* e o *Bitbucket Pipelines* relativamente a algumas características (Atlassian, 2019; John Ferguson Smart, 2011).

Tabela 2 – Comparação entre *Jenkins* e *Bitbucket Pipelines*

Características	Jenkins	Bitbucket Pipelines
Open-Source	Sim	Não. Preço variado consoante o tempo mensal gasto em *builds*
Integração do código fonte	Necessita de configurar o repositório para efetuar a *build*	Executado no próprio repositório do projeto
Instalação	Configuração do servidor	Não necessita
Suporte de repositórios	*Git, Mercurial, Github*	*Git* e *Mercurial*
Escrita do pipeline	*Jenkinsfile* ou definição dos steps através da UI	Ficheiro *yml*
Suporte para implantação em *Dockers*	Sim	Sim
Integração com o *Jira*	Sim, configurável através de *plugin*	Sim, automática
Linguagens suportadas	*Python, Ruby, Java, Android, C/C++*	*Java, Javascript, PHP, Python, Ruby*
Comunidade	Extensa	Sem dados obtidos
Facilidade de uso	Fácil	Fácil
Monitorização	UI do servidor	UI do Bitbucket

2.13.4 *Docker*

O *Docker* é uma ferramenta *open-source* que surgiu em 2013, desenvolvida pela empresa *Docker, Inc*. Tem como objetivo auxiliar a implantação das aplicações através da utilização de *containers* (contentores). Estes são um ambiente isolado de *software* executável que agrupam o código e as suas dependências (*O que é Docker?*, 2018).

A utilização destes *containers* permite isolar a execução da aplicação, o que torna possível executar diversas aplicações de forma isolada num mesmo servidor, apresentando inúmeras vantagens (*O que é Docker?*, 2018), tais como:

- Dependências: a execução da aplicação é efetuada no *container*, a sua imagem inclui as dependências necessárias para uma execução bem-sucedida, sem a necessidade de configurações ou instalações na máquina ou no servidor onde se encontra;

- Redução de custos: a utilização dos *Docker,* permite a redução de custos de servidores, uma vez que não é necessária uma quantidade tão grande de espaço disponível para a execução da aplicação;

- Cópia de segurança e reversão da versão implantada: através das imagens, é possível reverter a aplicação para versões anteriores caso a imagem implantada não seja desejada;

- Sistematização: a imagem é construída através de ficheiros de configuração, permitindo replicarem-se sem qualquer diferença no ambiente;

- Integração com serviços *cloud*.

2.13.5 *Kubernetes*

Kubernetes é um projeto *open-source* desenvolvido pela *Cloud Native Computing Foundation* e disponibilizado em 2014. Este sistema permite a orquestração de *containers* e a automatização do processo de implantação dos mesmos.

A utilização de *Kubernetes* segundo (Sayfan, 2017), permite a implantação distribuída de software visando obter escalabilidade e implantações sem causar indisponibilidade do sistema. Através das suas funcionalidades, este permite, escalabilidade horizontal, escalabilidade automática conforme a utilização dos recursos onde se encontra a executar o microsserviço, controlo de disponibilidade do mesmo e reinício automático em caso de falha, implantação automática e reversão da mesma em caso de falha.

De acordo com (Sayfan, 2017), a implantação do *Kubernetes* resulta num *cluster* constituído por pelo menos um *node*. Este, consiste numa máquina física ou virtual responsável por hospedar os *pods*. Estes *nodes* permitem replicar rapidamente os *pods* neles hospedados, permitindo implantações sem inatividade da aplicação, tolerância à falha da mesma, disponibilidade, escalabilidade e elasticidade. Por fim, um *pod* é constituído por um ou mais *containers*. No caso de utilização de mais de um *container*

por *pod*, estes irão partilhar os recursos de armazenamento e rede, nomeadamente o endereço de rede será o mesmo para os *containers*.

2.13.6 *Helm*

Helm é uma ferramenta *open-source* desenvolvida por *Deis.* Esta visa auxiliar a instalação e configuração de aplicações no *cluster Kubernetes*. Através desta ferramenta é possível instalar aplicações previamente configuradas e prontas a utilizar (Sayfan, 2017).

A utilização desta ferramenta apresenta as seguintes vantagens:

- Gestão da complexidade ao implantar aplicações;
- Facilita a partilha de aplicações;
- Facilita quer a atualização quer o retorno à versão anterior da aplicação.

2.14 Cloud computing

Cloud computing, consiste no fornecimento de recursos informáticos através da *internet,* contrariamente ao armazenamento dos serviços em *hardware* físico (Thomas Erl et al., 2013).

Os provedores de *cloud* disponibilizam diversos serviços que apresentam "*diferentes níveis de abstração e esforço a implantação dos serviços*" (Michael J. Kavis, 2014), entre eles podem distingir-se:

- IaaS (*Infrastructure as a Service*): permite adquirir recursos computacionais escaláveis e automatizados. Este modelo visa evitar despesas em *hardware* e também a complexidade de gestão dos servidores físicos e infraestruturas como *datacenters*;
- PaaS (*Platform as a Service*): permite o desenvolvimento da aplicação e a sua implantação através da *internet*. Este serviço disponibiliza a plataforma na qual a aplicação será executada. Assim, reduz às equipas a necessidade de criação e manutenção das plataformas de cada aplicação implantada;
- SaaS (*Software as a Service*): permite disponibilizar e aceder ao *software* através da *internet*. O provedor do serviço é responsável pelo *hardware*, disponibilidade, segurança e os dados da aplicação.

De modo geral algumas das várias vantagens que a utilização da *cloud* apresenta para os utilizadores são:

- Redução de custos, tendo apenas custos proporcionais: através da *cloud* é possível pagar apenas pela quantidade de recursos tecnológicos consumidos (*pay-per-use*). Evitando gastos monetários com equipamentos físicos que porventura poderiam não ser utilizados;

- Aumento da escalabilidade, tornando possível obter elasticidade: possibilidade de aumento ou diminuição instantânea dos recursos de modo a ajustar aos elevados ou baixos fluxos de utilizadores;

- Aumento da disponibilidade e confiabilidade: "foco na minimização e eliminação das interrupções do tempo de execução".

2.14.1 *Google Cloud Platform*

Esta plataforma *cloud,* é disponibilizada pela *Google* tendo surgido em 2011, sendo a mais recente neste domínio. Contudo, ao longo dos últimos anos tem demostrado um elevado crescimento relativamente ao número de clientes no domínio.

Esta plataforma oferece "*conexões rápidas e confiáveis aos seus utilizadores*" (Intellipaat, 2019). As suas vantagens (Intellipaat, 2019) são:

- "Experiência em DevOps";

- "Projetada especificamente para negócios baseados na nuvem";

- "Descontos flexíveis e contratos".

2.14.2 *Amazon Web Services*

Esta plataforma surgiu em 2006, disponibilizada pela Amazon, sendo a primeira, a mais experiente e a que domina o mercado *cloud* (Intellipaat, 2019).

Oferece diversos serviços configuráveis de forma independente, com o intuito de gerir as aplicações, base de dados, segurança, entre outros.

As principais vantagens (Intellipaat, 2019), desta *cloud* são apresentadas abaixo:

- Domínio do mercado *cloud;*

- Integração com ferramentas *open-source*;
- Qualidade dos serviços dado a experiência adquirida.

2.14.3 *Microsoft Azure*

Microsoft Azure é uma plataforma *cloud* disponibilizada pela *Microsoft* em 2010. Está em segundo lugar em termos de domínio no mercado.

O uso desta plataforma é recomendado quando existe integração com ferramentas da *Microsoft*.

2.14.4 DigitalOcean

A *DigitalOcean* é uma plataforma *cloud* que surgiu em 2011 fundada por *Ben Uretsky* e *Moisey Uretsky*.

De acordo com (Saleem, 2019), esta *cloud* visa responder às necessidades dos "*desenvolvedores de software individuais e empresas startup*", com menor dimensão menos necessidades, portanto. Esta em 2018, foi premiada com o título de terceira maior empresa de hospedagem na *cloud*.

2.15 JMeter

JMeter é uma ferramenta *open-source* desenvolvida pela *Apache* e permite a definição e execução de testes através de uma interface gráfica ou através da escrita de *scripts.* Esta ferramenta permite também testar o comportamento e o desempenho de aplicações, através da execução de testes de carga, automatizando os pedidos efetuados à mesma (JMeter, 2020).

É uma ferramenta que exporta os dados dos testes efetuados e auxilia no seu processo de análise através de tabelas e gráficos.

2.16 Servidor de autenticação único

Este servidor de autenticação, permite que o cliente do produto, caso possua mais produtos, possa aceder aos mesmos, efetuando apenas *login* uma única vez. Esta funcionalidade beneficia a experiência de utilização dos produtos e simplifica a

integração entre os mesmos, uma vez que estes não necessitam de implementar os seus próprios processos de autenticação (CoMakeIT, 2018).

Atualmente, na empresa Flowinn, a autenticação, é efetuada através de um servidor único, *Keycloak*. Através deste sistema, é possível partilhar o utilizador com *login* efetuado e todos os utilizadores das aplicações num local único.

3 Migração entre arquiteturas

Neste capítulo apresentam-se e analisam-se, com base em experiências idênticas, algumas soluções da migração de arquiteturas monolíticas para arquiteturas baseadas em microsserviços.

A migração de arquiteturas monolíticas para arquiteturas baseadas em microsserviços, surge como possível solução para a complexidade existente num único sistema. A atual demanda dos clientes por rápidas entregas e novas funcionalidades nos seus produtos, terá de passar, necessariamente, pelo desenvolvimento de soluções baseadas em microsserviços, já que estas apresentam inúmeras vantagens neste aspeto, tais como, facilitação do processo de manutenção do código, permitem o desenvolvimento e implantação independente levando a que a quebra de uma funcionalidade não comprometa a totalidade da solução, por fim, facilitam o processo de CD.

3.1 Estratégias de migração

Segundo (Newman, 2019), estas migrações devem ser decisões conscientes, que visam resolver questões ou problemas que por meio de soluções monolíticas não é possível.

Para o processo de definição da estratégia para efetuar a migração, existem diversas opções. Uma delas, a transformação de toda a aplicação monolítica em várias soluções baseadas em microsserviços, simultaneamente e de uma só vez, apesar de parecer a solução mais fácil, é impraticável, uma vez que, existem problemas consequentes desta mudança. A entrega da totalidade do *software* é mais suscetível a falhas que poderiam ser identificadas e corrigidas ao longo da migração, e não permite obter resultados do processo.

"*If you do a big-bang rewrite, the only thing you're guaranteed of is a big bang*". - *Martin Fowler* (Newman, 2019)

Como se referiu, a realização da alteração for efetuada de uma vez, implica um longo investimento temporal e impossibilita a utilização da solução até que todo o processo de migração seja concluído. Assim, não é possível obter rapidamente *feedback* relativamente ao valor acrescentado com a alteração, bem como, verificar o sucesso da alteração no decorrer do seu processo. Este método apresenta ainda a desvantagem de não permitir desenvolver novas funcionalidades enquanto a migração não estiver totalmente concluída, uma vez que estas necessitam que a lógica aplicacional seja completamente migrada para a nova solução.

De modo a facilitar a migração de aplicações com arquiteturas monolíticas para arquiteturas baseadas em microsserviços existem diversos padrões, um deles é o *Strangler Fig Application*. Foi definido em 2004 por *Martin Fowler* e consiste em transformar de forma incremental o monólito em diversos microsserviços (Newman, 2019). Segundo (Kyle Brown, 2017), esta transformação consiste em três etapas:

- *Transform* (Transformação): Desenvolvimento de uma determinada funcionalidade já existente no monólito num microsserviço externo ao monólito. Este microsserviço faz parte da nova solução;

- *Co-exist* (Coexistir): De modo a testar a funcionalidade desenvolvida em paralelo com a mesma funcionalidade existente, é necessário que ambas executem em paralelo nos dois sistemas;

- *Eliminate* (Eliminar): Elimina o componente da aplicação monolítica, retornando todo o tráfego para o novo serviço desenvolvido.

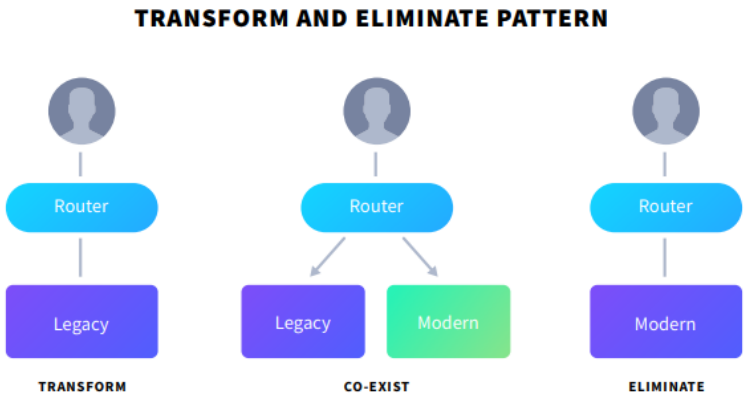

Figura 4 – Etapas do padrão *Strangler Fig Application* (Samir Behara, 2018)

Como apresentado na Figura 4, a aplicação deste padrão requer a utilização de um mecanismo de encaminhamento de tráfego, que permita distribuir os pedidos efetuados à funcionalidade sobre a aplicação monolítica para a nova solução desenvolvida no microsserviço. Após análise pode concluir-se que, a segunda etapa (coexistir) requer um esforço maior por ser necessária a manutenção de duas soluções e respetivas base de dados.

3.2 Exemplos de Migrações

3.2.1 Wix

Neste primeiro caso, a migração foi efetuada pela Wix. Esta "é uma plataforma de desenvolvimento web, que permite aos seus utilizadores criar websites sem qualquer conhecimento em programação" (Yoav Abrahami, 2015).

Numa conferência (Aviran Mordo, 2006), realizada no evento *GOTO Conferences*, em novembro de 2016, *Aviran Mordo*, vice-presidente de engenharia na Wix, apresentou a estrutura arquitetural das suas soluções e a mudança que necessitaram de efetuar para ultrapassar e escapar aos problemas do seu monólito, nomeadamente:

- "Uma falha em áreas não relacionadas causará falhas em todo o sistema".
- "Alterações em áreas não relacionadas implicam a implantação de todo o sistema".
- "Servidor monólito é responsável por todos os processos".

A estratégia adotada para a migração foi a realização de um processo incremental que passou por separar as funções do monólito com base nos dois segmentos de clientes, nomeadamente, edição e visualização de *websites*. O processo iniciou-se pela separação da parte pública, de visualização dos *websites*, do monólito.

Com base na publicação efetuada, por *Yoav Abrahami*, arquiteto na Wix, este processo de divisão durou cerca de quatro a cinco anos. Este foi descrito por ele como "*difícil e longo, uma vez que a plataforma continuou ativa e sobre desenvolvimento de novas funcionalidades, aquando a extração de funcionalidades existentes no monólito para os novos microsserviços*" (Yoav Abrahami, 2015).

Com esta migração e de acordo com *Aviran Mordo* (Aviran Mordo, 2006), a Wix obteve resultados que com ao sistema anterior, não seriam possíveis nomeadamente,

escalabilidade, manutenibilidade e autonomia de desenvolvimento por equipas. Contudo, a nova arquitetura implementada apresenta algumas desvantagens relativamente ao monólito, nomeadamente complexidade do sistema e uma maior dificuldade de testar as funcionalidades.

3.2.2 Netflix

Segundo *Yury Izrailevsky*, a ideia de migrar a aplicação monolítica surgiu em meados de 2008, quando este apresentou uma falha nos *datacenters* que obrigou a indisponibilidade do sistema da Netflix durante três dias, impossibilitando, na altura, o envio de DVD para os seus membros (Yury Izrailevsky, 2016).

O objetivo desta mudança, foi obter uma redução de custos relativamente aos *datacenters* e tirar vantagem da elasticidade e escalabilidade da nuvem, o que permite escalar mais rapidamente que os *datacenters* físicos.

Esta migração, consistiu na transformação de um monólito e *datacenters*, em vários microsserviços implantados na nuvem. A migração efetuada, pretendia melhorar a disponibilidade, escalabilidade e performance da solução. Com o intuito de evitar a migração de problemas presentes no monólito, a nova solução consiste em centenas de microsserviços e base de dados *NoSQL* resultantes da desnormalização do modelo de dados anterior (Netflix Technology Blog, 2012; Yury Izrailevsky, 2016).

A migração foi um processo incremental, que teve a duração de sete anos, contudo, a nova infraestrutura na nuvem, aumentou drasticamente a disponibilidade e escalabilidade do sistema, permitindo à Netflix expandir-se ao longo de 130 novos países, "*sem terem de se preocupar com a falta de espaço ou poder computacional*", afirmou *John Piela*, com base nas declarações de *Adrian Cockcroft*, "*arquiteto da cloud da Netflix*" (John Piela, 2016).

4 Análise de valor

4.1 Novo Modelo de Desenvolvimento de Conceitos

O novo modelo de desenvolvimento de conceitos, defendido por *Peter Koen et al*., providencia uma linguagem comum e a definição dos componentes chave do *Fuzzy Front-End* (FEE). Este modelo, acrescenta clareza e racionalidade ao *front end*, para atingir o objetivo de identificar claramente os requisitos, planos de negócios e definição de mercado para o novo projeto (Koen et al., 2002).

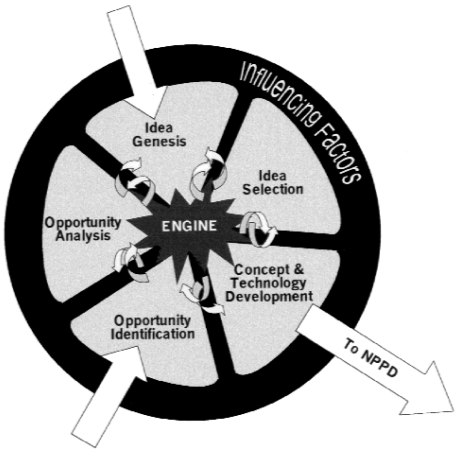

Figura 5 – Representação do Novo Modelo de Desenvolvimento de Conceitos (Koen et al., 2002)

Este modelo, divide o *front end* em três áreas distintas, sendo elas o motor, a parte interna e os fatores de influência. A primeira impulsiona o processo de inovação e consiste nos valores da empresa, liderança e cultura. A parte interna inclui os cinco elementos do *Front End Inovation* (FEI), sendo eles (Geração de Ideias, Seleção de Ideias, Definição do Conceito e Tecnologia, Identificação de Oportunidades e Análise de Oportunidades). Esta parte apresenta dois pontos de entrada; a Identificação da

oportunidade e a geração da ideia. Por fim, os fatores de influência são fatores externos, que influenciam a organização como por exemplo, fatores económicos, culturais ou legais, clientes e influência proporcionada pelos concorrentes (Koen et al., 2002).

4.1.1 Identificação da Oportunidade

Neste elemento, são identificadas as oportunidades que vão de encontro à estratégia da empresa. Estas, geralmente, são orientadas pelos objetivos do negócio e podem ser respostas a ameaças competitivas, produtos inovadores com o intuito de ganhar vantagens ou um meio de reduzir os custos atuais das operações. Existem diversos métodos para avaliar as oportunidades (Koen et al., 2002), nomeadamente:

- *Roadmapping*, é uma forma gráfica que permite planear recursos tecnológicos. Este método permite expor facilmente a complexidade do produto para as pessoas que não fazem parte da equipa do projeto;

- Análise das tendências tecnológicas e dos clientes, visa identificar as características dominantes do mercado e os consumidores a ele associados, com o intuito de obter oportunidades competitivas para o projeto;

- Análise da inteligência competitiva, refere-se à prática de recolha, análise e comunicação das informações disponíveis sobre as tendências competitivas;

- Pesquisa do mercado, consiste na análise do mercado envolvente do projeto a desenvolver;

- Planeamento de cenários, é uma abordagem disciplinada para imaginar e planear o futuro de modo a encontrar decisões que, de outra forma, poderiam ser ignoradas.

A análise das tendências tecnológicas e dos clientes é o método mais adequado para identificar a oportunidade corretamente. Posteriormente serão analisadas as duas arquiteturas, a atual e a que será desenvolvida.

As aplicações monolíticas, como se descreveu anteriormente, são mais simples no que respeita à sua implementação, porém ao longo do seu ciclo de vida e das constantes mudanças ao nível funcional, podem dar origem aos problemas identificados na secção 1.2 deste documento. A aplicação *Logistics*, sobre a qual este trabalho incide, não é exceção, através do uso contínuo do cliente surgiu a ideia de melhorar o produto arquiteturalmente, visando tornar este sistema mais manutenível, escalável e legível para novos membros da equipa. Com estas melhorias, pretende-se obter redução de

custos e esforços em implementações futuras sobre o produto. Através desta ideia, a Flowinn poderá ver o seu produto modernizado e atual, podendo levar a vantagens sobre os seus concorrentes diretos no mercado.

4.1.2 Análise da Oportunidade

Posteriormente à identificação das oportunidades, é necessário proceder à sua avaliação para verificar se estas valem o investimento. Esta avaliação pode ser efetuada formalmente ou iterativamente. Neste elemento, é pressuposto uma análise superficial sobre a oportunidade, podendo resultar em incertezas nas tecnologias e no mercado.

As técnicas usadas para a identificação de oportunidades, descritas no ponto 4.1.1, podem ser utilizadas para este elemento, contudo, deverá ser mais descritivo e apresentar mais detalhes. Podem ainda ser utilizadas quatro técnicas, enquadramento estratégico, avaliação do mercado, análise dos concorrentes e avaliação dos clientes (Koen et al., 2002).

Para a análise da ideia descrita neste documento, o enquadramento estratégico é a técnica mais adequada uma vez que, se pretende acrescentar valor interno à empresa Flowinn, este por sua vez trará valor aos diversos clientes do produto *Logistics*. Atualmente, o processo de implementação de uma nova funcionalidade consiste em quatro principais fases, especificação e detalhe da funcionalidade, desenvolvimento, testes manuais no ambiente de testes e implantação em produção. Este processo, devido à complexidade atual da solução, é moroso, arriscado e por vezes leva a falhas, visto que, dada a inexistência de testes, quando se realizam novas implementações, são, por vezes, quebradas funcionalidades previamente desenvolvidas. Portanto, na fase de testes manuais, esta anomalia pode não ser detetada, uma vez que é testada apenas a nova funcionalidade e não a totalidade da versão do produto a ser colocado no cliente. A ideia em análise, tenciona melhorar as fases de desenvolvimento, testes manuais no ambiente de testes e implantação em produção, dado que com solução em vista é expectável acelerar o processo diretamente e indiretamente. A solução em vista, visa reduzir a complexidade aglomerada em um monólito, facilitando a perceção das funcionalidades existentes e consequente redução do tempo despendido nas novas implementações. Através de uma correta automatização de testes, será possível detetar a quebra indesejada de funcionalidades. Por fim, relativamente à fase de implantação no cliente, é pretendido acelerar e automatizar o processo de entrega, com isto os clientes terão acesso às novas funcionalidades num espaço de tempo menor, podendo dar a sua opinião e melhorar continuamente o seu produto, ao invés de procurar soluções alternativas no mercado.

4.1.3 Geração e Enriquecimento de Ideias

Este elemento tem o propósito de criar, desenvolver e amadurecer as ideias, este é um processo evolutivo, podendo estas sofrer diversas mudanças à medida que são estudadas e examinadas. Contactos com o cliente ou utilizadores finais do produto e comunicação com outras empresas, é comum para enriquecer as ideias.

O processo de geração de ideias pode ser formal, através de sessões de *brainstorming* e ideias que vão sendo armazenadas, ou informais tais como, experiências falhadas ou pedidos incomuns dos clientes (Koen et al., 2002).

Atualmente, a empresa Flowinn apresenta todos os seus produtos desenvolvidos sobre arquiteturas monolíticas, sendo a ideia apresentada neste trabalho o primeiro projeto com base em microsserviços. Assim, é expectável nesta fase, a elaboração de uma pesquisa aprofundada das possíveis alternativas dos componentes a utilizar na concretização desta ideia. Estas pesquisas visam ser iterativas, uma vez que devem ser apresentadas em reuniões, as possíveis soluções de modo a que os elementos possam contribuir com as suas opiniões e experiências, com o intuito de expor situações que não tenham sido pensadas inicialmente e acrescentem valor à ideia.

4.1.4 Seleção da Ideia

Neste ponto, deve ser selecionada a ideia que permite alcançar o maior valor comercial, assim sendo, este processo é fulcral para o sucesso e futuro do projeto a desenvolver. No entanto de acordo com (Koen et al., 2002), não existe um único que garanta a escolha da melhor solução.

No âmbito deste projeto, de modo a proporcionar as maiores vantagens organizacionais, tendo em conta gastos financeiros e temporais necessários, será utilizado o método AHP (*Analytic Hierarchy Process*), posteriormente analisado com detalhe no capítulo 5.

4.1.5 Definição de conceito

A definição de conceito é o elemento final do novo modelo de desenvolvimento de conceito e fornece a única saída deste modelo. Aqui a oportunidade escolhida deve ser defendida, justificando o investimento necessário para o seu desenvolvimento.

Atualmente, o produto utilizado neste trabalho já se encontra comercializado. A migração apresentada pretende melhorar a solução existente, beneficiando a empresa e os seus clientes.

4.2 Valor para o cliente e valor percebido pelo cliente

O valor para o cliente, consiste na diferença entre os benefícios que o produto lhe oferece e os custos que o cliente teve para o obter. Este valor pode ser diferente do valor definido pela empresa.

Valor percebido é o valor interpretado pelo cliente, com base na satisfação das suas necessidades, e não o valor monetário do produto (Woodall, 2003).

Neste projeto, o cliente valoriza o desempenho, a usabilidade e correta funcionalidade do sistema, sendo fulcral minimizar os erros presentes no ambiente de produção.

Este projeto visa responder às necessidades do cliente e facilitar a continuidade da entrega de valor em futuras funcionalidades pedidas pelo mesmo. Estes benefícios, devem ser corretamente transmitidos para os clientes, de modo a manter as suas perceções alinhadas em relação aos benefícios do produto.

Por outro lado, este projeto apresenta custos, tais como, custo monetário para o cliente, custo temporal de implementação da arquitetura e esforço necessário da respetiva implementação.

4.3 Business Model Canvas

O modelo de negócio Canvas, foi proposto por *Alexander Osterwalder* e visa descrever de uma forma gráfica, clara e lógica, como uma organização cria e proporciona valor através do seu negócio. Este modelo, é segmentado por nove blocos sendo eles, parceiros chave, atividades chave, recursos chave, estrutura de custos, proposta de valor, relação com o cliente, canais de comunicação, segmento de clientes e fontes de receita, abrangendo as quatro áreas principiais do negócio que são, clientes, oferta, infraestrutura e viabilidade financeira (Osterwalder et al., 2010).

Tabela 3 – Business Model Canvas

<table>
<tr>
<td rowspan="2">Parceiros chave

• Flowinn</td>
<td>Atividades chave

• Design da solução
• Desenvolvimento da solução
• Testes automáticos
• Configuração dos ambientes de desenvolvimento e produção
• Configuração do processo de implantação</td>
<td rowspan="2">Proposta de valor

• Escalabilidade
• Manutenibilidade
• Flexibilidade
• Sustentabilidade
• Disponibilidade</td>
<td>Relação com o cliente

• Portal Service Desk</td>
<td rowspan="2">Segmento de clientes

• Clientes da aplicação Logistics
• Flowinn</td>
</tr>
<tr>
<td>Recursos chave

• Programadores
• Serviços cloud
• Git
• Servidor de mensagens assíncronas</td>
<td>Canais de Comunicação

• Contato presencial
• Email</td>
</tr>
<tr>
<td colspan="3">Estrutura de custos

• Desenvolvimento da solução
• Manutenção da solução
• Infraestrutura</td>
<td colspan="2">Fontes de receita

• Diminuição de tempo gasto em manutenção de software (manutenibilidade)
• Diminuição do tempo de desenvolvimento
• Entrega mais rápida do produto aos clientes
• Entrega de software menos propício a falhas devido aos testes implementados
• Venda do projeto aos clientes</td>
</tr>
</table>

Com base na Tabela 3, pode observar-se que o modelo canvas é constituído pelos seguintes segmentos:

- Parceiros chave

"*Identificam os fornecedores e os parceiros essenciais para o bom funcionamento do negócio*". (Osterwalder et al., 2010).

O parceiro chave deste projeto é a empresa Flowinn.

- Atividades chave

"*Representam as ações mais importantes que a empresa deve realizar para que o negócio seja bem-sucedido*" (Osterwalder et al., 2010).

As atividades chave deste projeto são, *design* e desenvolvimento da solução, implementação de testes automáticos e configuração do processo automático de implantação para os ambientes de desenvolvimento e produção de cada cliente.

- Recursos chave

"*Descrevem os bens necessários para a empresa, através do seu modelo de negócio, entregar valor aos seus clientes*" (Osterwalder et al., 2010).

Os recursos chave passam por programadores, serviços *cloud*, *git* e servidor de mensagens assíncronas.

- Estrutura de custos

"*Descreve todos os custos envolvidos no modelo de negócio*" (Osterwalder et al., 2010).

Os custos são desenvolvimento da solução, manutenção da solução e infraestrutura.

- Proposta de valor

"*Representa o conjunto de produtos e serviços que acrescentam valor para um segmento de cliente específico*" (Osterwalder et al., 2010).

As propostas de valor são escalabilidade, manutenibilidade, flexibilidade, sustentabilidade e disponibilidade.

- Relação com o cliente

"Descreve o tipo de relação entre a organização e os seus clientes" (Osterwalder et al., 2010).

A relação com o cliente é efetuada através do Portal *Service Desk*, que consiste numa aplicação *web*, integrada no sistema interno da empresa Flowinn, com o intuito de fornecer um suporte eficiente ao cliente.

- Canais de comunicação

"*Descreve os meios que a companhia utiliza para comunicar e transmitir a sua proposta de valor aos segmentos de cliente*" (Osterwalder et al., 2010).

A comunicação com os clientes é efetuada através de *emails* e contatos presenciais.

- Segmentos de clientes

"*Representa os diferentes grupos de pessoas ou organizações que a empresa procura alcançar e servir*" (Osterwalder et al., 2010).

Neste ponto a aplicação destina-se aos clientes do produto *Logistics* e à empresa Flowinn, já que esta, a longo prazo, beneficiará do produto, nomeadamente na manutenção e sustentabilidade.

- Fontes de receitas

"*Fontes de receita descrevem as metodologias que a organização segue para gerar dinheiro através segmento de cliente*" (Osterwalder et al., 2010).

Concluindo, as fontes de receita são, diminuição de tempo gasto em manutenção de *software* e de desenvolvimento, entrega mais rápida do produto aos clientes, entrega de *software* menos propício a falhas devido aos testes implementados e valor gerado da venda do projeto aos clientes.

4.4 Analytic Hierarchy Process

O método de análise hierárquica AHP, foi desenvolvido em 1980 por *Thomas L. Saaty*.

Este método, auxilia a escolha e respetiva justificação dadas as alternativas possíveis. A ideia principal deste método é decompor o problema de decisão em níveis hierárquicos, de modo a facilitar a sua compreensão.

A primeira etapa do mesmo, consiste na construção da árvore hierárquica de decisão, esta é constituída pelo objetivo da decisão no topo, seguido pelos critérios de avaliação associados ao problema e por fim as alternativas disponíveis para solucionar o problema (Saaty, 2008).

Posteriormente, segue-se a segunda etapa, que consiste na atribuição de prioridades aos diferentes pares de critérios escolhidos. De modo a efetuar esta atribuição, é necessária a utilização da escala de *Saaty*, que é definida por valores entre 1 e 9, sendo que 1 representa igualdade de importância dos critérios e 9 representa uma diferença de importância absoluta entre os mesmos (Saaty, 2008).

Por fim, é necessário garantir a consistência das prioridades relativas, através dos cálculos matemáticos que serão apresentados no decorrer da execução do método, posteriormente indicada na secção 5.4.

5 Abordagens tecnológicas

Este capítulo tem como objetivo apresentar algumas das abordagens possíveis para a implementação da solução baseada em microsserviços, bem como, o processo de seleção da abordagem a implementar. Esta seleção será efetuada com base no método de análise hierárquica, AHP.

5.1 Gateway e JHipster Registry

JHipster Registry é uma aplicação *open-source*, "*baseado em Spring Cloud Netflix, Eureka e Spring Cloud Config*", disponibilizada pela equipa do *JHipster* (JHipster, 2017a).

A utilização desta abordagem resulta numa solução que apresenta a arquitetura que se pode observar na Figura 6.

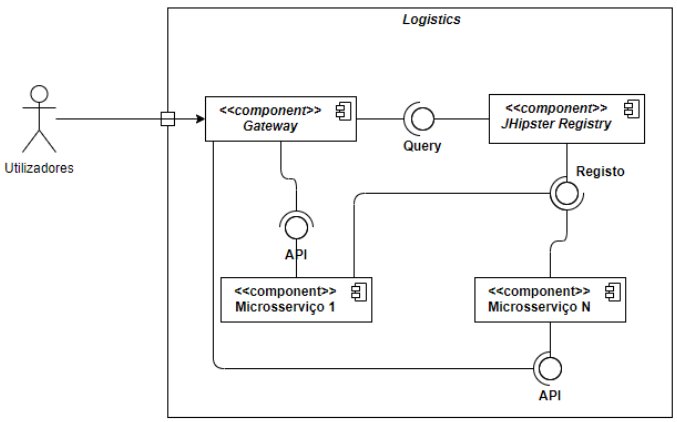

Figura 6 – Diagrama de componentes utilizando *JHipster Registry,* baseado em (JHipster, 2017a)

Os utilizadores, como ilustrado na figura acima, interagem com o microsserviço de *Gateway*, através dos pedidos efetuados no seu *browser*. De modo a concluir os diversos pedidos dos utilizadores, o componente *Gateway*, comunica com o componente *JHipster Registry*. Este indicará qual o microsserviço com que deve comunicar posteriormente e a sua localização, assim o componente *Gateway* efetua o pedido ao respetivo microsserviço, que lhe dará a resposta ao pedido pretendido.

O *Gateway*, é um componente constituído pelo *front-end*, visível pelos utilizadores e pelo encaminhamento dos pedidos para o devido microsserviço. Este encaminhamento é efetuado de forma dinâmica, aplicando o algoritmo *round robin* no caso de existirem várias instâncias de um microsserviço. (JHipster, 2017a; Sendil Kumar N & Deepu K Sasidharan, 2018).

O *JHipster Registry*, é um *service registry*, responsável por armazenar a localização dos microsserviços constituintes da solução, permitindo a comunicação com eles. Este componente disponibilizado pelo *JHipster*, é um servidor constituído por *Eureka* e *Spring Cloud Config,* o comportamento deste componente pode ser alterado, uma vez que, o seu código fonte é disponibilizado para desenvolvimento.

O componente permite as seguintes funcionalidades (Sendil Kumar N & Deepu K Sasidharan, 2018):

- Descoberta de serviços, através do *Eureka*;
- Registo e configuração dos microsserviços em tempo de execução, através do *Spring Cloud Config*;
- *Dashboard* administrativa configurável, que contém informação bastante detalhada de todos os microsserviços, como, instâncias e o seu estado, consumo de recursos por microsserviço, histórico e documentação de cada microsserviço, número de pedidos efetuados ao microsserviço, entre outros.

Pela observação da Figura 6, pode concluir-se que esta abordagem utiliza os seguintes padrões, descoberta de serviços *(service discovery)*, *self registration* e *service registry,* descritos anteriormente no capítulo 2.

5.2 Gateway e Consul

O *Consul* é um produto *open-source* da *Hashicorp*, desenvolvido com recurso à linguagem *Go*.

Este, segundo *Armon Dadgar* (Armon Dadgar, 2018), diretor de tecnologia CTO (*Chief Technology Officer*) e cofundador da *HashiCorp* e (Sendil Kumar N & Deepu K Sasidharan, 2018), apresenta as seguintes funcionalidades:

- Descoberta de serviços: regista os serviços e o seu estado, permitindo a comunicação com os mesmos. A descoberta de um serviço pode ser efetuada por DNS (*Domain Name System*) ou HTTP (*Hypertext Transfer Protocol*);
- Deteção de falhas: informação sobre os serviços registados, indicando se está disponível ou em falha;
- Armazenamento chave valor: permite armazenar objetos, através de uma chave única. Esta poderá ser utilizada pelos serviços registados para obter o objeto armazenado;
- Permite vários *datacenters*: cada *datacenter* contém o seu armazenamento chave valor, permitindo configurações diferentes ao utilizador;
- Monotorização: apresenta a informação do estado de cada microsserviço através de uma *dashboard*;
- Segurança na comunicação: permite definir regras de comunicação entre os microsserviços;
- Documentação: permite visualizar a documentação de cada microsserviço, através de *swagger*.

A utilização desta abordagem resulta numa solução que apresenta a seguinte arquitetura demonstrada.

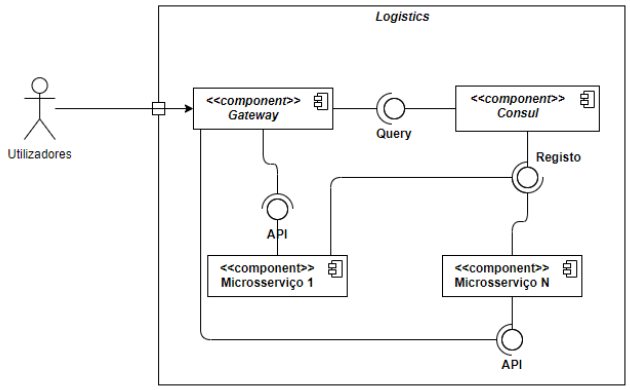

Figura 7 – Diagrama de componentes utilizando *Consul*, baseado em (JHipster, 2017b)

O *Consul*, similarmente ao *JHipster Registry*, disponibiliza uma API (*Application Programming Interface*), que permite o registo dos microsserviços. Estes, quando iniciam podem registar-se através de um pedido *HTTP* à API, ou através de um ficheiro de configuração.

Como observado na Figura 7, o processo de comunicação entre os microsserviços é similar ao *JHipster Registry*, contudo a utilização do *Consul* apresenta "*a vantagem de se focar na consistência em vez de disponibilidade*" (Sendil Kumar N & Deepu K Sasidharan, 2018).

5.3 WSO2 API Manager

O *WSO2 API Manager,* é uma ferramenta *open-source,* que permite implantar e gerir o ciclo de vida das API. Esta solução visa facilitar a "*criação, publicação, segurança e versionamento das API*" através da disponibilização de componentes específicos, tais como, *Developer Portal, API Gateway, Key Manager, Traffic Manager* e *Analytics* (WSO2, consultado em 2019).

A utilização desta abordagem resulta numa solução com a seguinte arquitetura.

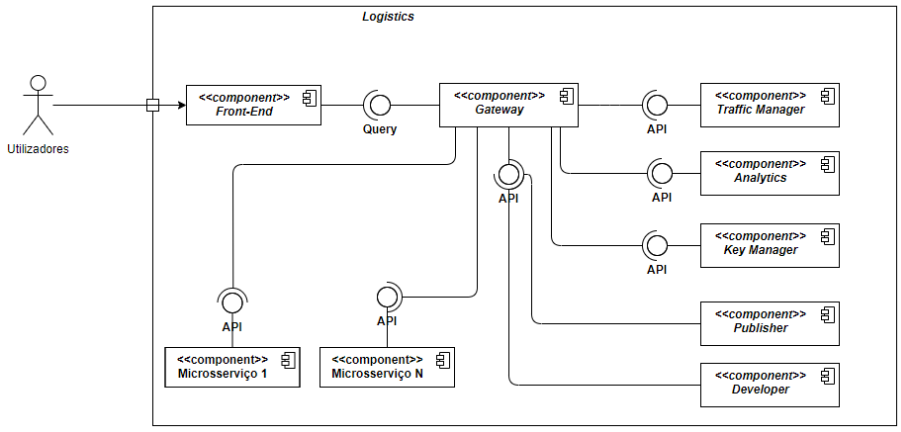

Figura 8 – Diagrama de componentes utilizando *WSO API Manager*, baseado em (Skalena, 2018)

Como é possível observar na Figura 8, a utilização do WSO2 acrescenta os seguintes componentes à solução (WSO2, consultado em 2019):

- *Analytics* – Apresenta, através da recolha de dados efetuada pela *Gateway*, dados estatísticos dos acessos a cada microsserviço existente na solução.

- *Developer* – Permite testar as API e registar os seus clientes.

- *Publisher* – Interface que permite a publicação dos microsserviços, respetivas documentações e controlar o ciclo de vida das API. Contrariamente às soluções anteriores, apresenta a funcionalidade de colocar ficheiros externos e documentação através de *swagger*.

- *Key Manager* – Componente responsável pela gestão dos *tokens* de acesso e segurança. O API *Gateway* efetua comunicação com este componente para obter e validar o *token* de acesso.

- *Traffic Manager* – Permite restringir o acesso às API, através do controlo dos acessos às mesmas.

- G*ateway* – Como apresentado na subsecção 2.8, a API *Gateway* consiste no ponto de entrada para os clientes que pretendem consumir os serviços. Este componente é responsável por direcionar os pedidos recebidos para os respetivos microsserviços, de acordo com as políticas de uso e verificações de segurança. Este *Gateway* deverá utilizar o API *Microgateway* do WSO2. Através deste componente, será possível a utilização do padrão *service discovery*, disponibilizado pelo WSO2, que consiste no armazenamento dos endereços públicos através de um servidor *etcd*. Neste são armazenados em formato chave valor, os endereços e uma chave identificadora, que posteriormente, poderá ser utilizada para modificar o endereço (Viraj Salaka, 2019).

5.4 Avaliação de abordagens

No contexto deste projeto, o método hierárquico AHP (*Analytical Hierarchy Process*) descrito na secção 4.4, será utilizado com o objetivo de avaliar qual a melhor tecnologia a utilizar no desenvolvimento da nova solução para a resolução do problema descrito no capítulo 1.

Para a utilização do método AHP, é necessário definir os critérios a utilizar assim como as respetivas prioridades. Neste projeto, os critérios associados ao problema descrito são:

- Personalização: pretende avaliar as alternativas consoante a possibilidade de criar ou alterar o componente. Estas alterações podem ser visuais ou comportamentais, caso seja necessário para a implementação da solução;

- Monitorização: pretende avaliar a monitorização oferecida pelos componentes das alternativas, focando aspetos como, indicação de tráfego, estado de cada microsserviço, entre outros;

- Complexidade: pretende avaliar a complexidade exigida para a integração do componente na solução a ser desenvolvida.

Após a definição dos critérios, iniciou-se a comparação entre os mesmos, definindo os respetivos pesos indicados na Tabela 4.

Tabela 4 – Comparação dos critérios elegidos

Critérios	Personalização	Monitorização	Complexidade
Personalização	1	1/4	1/3
Monitorização	4	1	2
Complexidade	3	1/2	1

A seguinte fase, inicia-se pela realização de soma de cada coluna da tabela acima representada (Saaty, 2008), resultando os valores presentes na Tabela 5.

Tabela 5 – Soma das colunas da comparação dos critérios

Soma	8	7/4	10/3

Através da soma dos valores de cada coluna, é possível realizar-se a normalização da matriz inicial, dividindo cada elemento pela soma total da coluna (Saaty, 2008), ilustrado na Tabela 6.

Tabela 6 – Matriz normalizada

Critérios	Personalização	Monitorização	Complexidade
Personalização	0.125	0.1429	0.1

Monitorização	0.5	0.5714	0.6
Complexidade	0.375	0.2857	0.3

Os valores presentes na Tabela 6, podem ser apresentados em forma de tabela ou através de matriz.

De modo a terminar esta fase, é necessário obter o vetor de prioridades relativas. Este resulta da média dos valores de cada linha da matriz normalizada ilustrada na Tabela 6.

$$\text{Vetor de prioridades} = \begin{bmatrix} 0.1226 \\ 0.5571 \\ 0.3202 \end{bmatrix}$$

Através deste valor, é possível concluir que o critério **monotorização** é considerado o mais importante, seguido da **complexidade** e **personalização**.

Uma vez terminada esta fase, segue-se a avaliação da consistência das prioridades relativas. Deste modo, é necessário o cálculo da razão de consistência (RC), que é obtida através da divisão do índice de consistência (IC) pelo índice aleatório (IR), "*referente a um grande número de comparações par a par efetuadas*" (Saaty, 2008). Para calcular o IC é necessário efetuar a multiplicação da matriz representada na Tabela 4 pelo vetor de prioridades.

Efetuados os primeiros cálculos obteve-se o seguinte vetor:

$$\begin{bmatrix} 0.3686 \\ 1.6879 \\ 0.9666 \end{bmatrix}$$

Posteriormente, para obter o valor próprio para calcular o IC, é necessário dividir o vetor resultante pelo vetor de prioridades e efetuar a média dos resultados obtidos.

$$\text{Valor próprio} = 3.0183$$

Com este valor é possível obter o IC e posteriormente o RC:

$$\text{IC} = 0.0091$$
$$\text{RC} = 0.0158$$

Uma vez que o valor obtido é menor que 0.1, pode se afirmar que os valores das prioridades relativas estão consistentes (Saaty, 2008).

De seguida, é necessário construir uma matriz que compare as alternativas relativamente ao critério a analisar e repetir o processo de determinar a prioridade relativa, como demostrado acima.

Critério personalização:

Alternativas	Gateway e JHipster Registry	Gateway e Consul	WSO2 API Manager
Gateway e JHipster Registry	1	5	3
Gateway e Consul	1/5	1	1/3
WSO2 API Manager	1/3	3	1
Soma	**23/15**	**9**	**13/3**

$$\text{Matriz normalizada} = \begin{bmatrix} 0.6522 & 0.5556 & 0.6923 \\ 0.1304 & 0.1111 & 0.0769 \\ 0.2174 & 0.3333 & 0.2308 \end{bmatrix}$$

$$\text{Vetor de prioridades} = \begin{bmatrix} 0.6333 \\ 0.1062 \\ 0.2605 \end{bmatrix}$$

Critério monitorização:

Alternativas	Gateway e JHipster Registry	Gateway e Consul	WSO2 API Manager
Gateway e JHipster Registry	1	3	7
Gateway e Consul	1/3	1	5

WSO2 API Manager	1/7	1/5	1
Soma	**31/21**	**21/5**	**13**

$$\text{Matriz normalizada} = \begin{bmatrix} 0.6774 & 0.7143 & 0.5385 \\ 0.2258 & 0.2381 & 0.3846 \\ 0.0968 & 0.0476 & 0.0769 \end{bmatrix}$$

$$\text{Vetor de Prioridades} = \begin{bmatrix} 0.6434 \\ 0.2828 \\ 0.0738 \end{bmatrix}$$

Critério complexidade:

Alternativas	Gateway e JHipster Registry	Gateway e Consul	WSO2 API Manager
Gateway e JHipster Registry	1	1	5
Gateway e Consul	1	1	5
WSO2 API Manager	1/5	1/5	1
Soma	**11/5**	**11/5**	**11**

$$\text{Matriz normalizada} = \begin{bmatrix} 0.4545 & 0.4545 & 0.4545 \\ 0.4545 & 0.4545 & 0.4545 \\ 0.0909 & 0.0909 & 0.0909 \end{bmatrix}$$

$$\text{Vetor de Prioridades} = \begin{bmatrix} 0.4545 \\ 0.4545 \\ 0.0909 \end{bmatrix}$$

De acordo com (Saaty, 2008), após o cálculo do vetor de prioridade de cada critério é necessário obter a prioridade composta para as alternativas em avaliação. Esta, é obtida através da multiplicação dos vetores anteriores agregados numa única matriz, pelos valores dos pesos dos critérios, obtidas no início deste método.

$$\begin{bmatrix} 0.6333 & 0.6434 & 0.4545 \\ 0.1062 & 0.2828 & 0.4545 \\ 0.2605 & 0.0738 & 0.0909 \end{bmatrix} \text{x} \begin{bmatrix} 0.1226 \\ 0.5571 \\ 0.3202 \end{bmatrix} = \begin{bmatrix} 0.5816 \\ 0.3161 \\ 0.1021 \end{bmatrix}$$

Pode concluir-se que a solução mais indicada para resolver o problema proposto é a utilização da alternativa ***Gateway*** **e** ***JHipster Registry****,* sendo esta a opção selecionada para este trabalho.

Em segundo lugar ficou a alternativa *Gateway* e *Consul* e por último, a alternativa do *WSO2 API Manager*.

6 Estudo de caso – Logistics

Como se referiu no capítulo 1 a aplicação Logistics desenvolvida pela empresa Flowinn, maioritariamente dedicada à gestão de armazéns da indústria farmacêutica encontra-se desenvolvida assente numa arquitetura monolítica. No entanto a empresa percebeu que deve melhorar o seu produto para se manter competitiva no mercado. O principal objetivo desta aplicação é auxiliar a gestão dos processos presentes no quotidiano dos armazéns dos seus clientes. Após um processo de análise à ferramenta e aos pedidos dos clientes, a empresa optou por tentar a abordagem de transformar a sua aplicação monolítica numa baseada em microsserviços, partindo das premissas do capítulo 1.

As seções seguintes apresentam a atual arquitetura (AS IS), e aquela que neste trabalho se propõe para dar resposta aos problemas identificados no capítulo 1 (TO BE).

6.1 Aplicação monolítica – "As is"

A aplicação monolítica abordada nesta dissertação, como descrito no capítulo 1, surgiu com o intuito de auxiliar a gestão dos processos presentes no quotidiano dos armazéns dos seus clientes.

Neste capítulo, será analisada de forma detalhada esta aplicação, de modo a contextualizar os seus componentes e respetiva lógica.

6.1.1 Arquitetura

A aplicação monolítica atual, Logistics, desenvolvida pela empresa Flowinn, com recurso à *framework JHipster*, utiliza *Spring* no *back-end* e *Angular* 8 no *front-end*. O

intuito desta aplicação é auxiliar e otimizar os principais processos dos armazéns dos clientes, sendo eles: receção, arrumação e a expedição.

Para a otimização supracitada, a aplicação Logistics partilha a sua base de dados com a aplicação Logistics Intelligence. Esta é responsável pela análise das ações no quotidiano do armazém do cliente e pela aplicação de modelos matemáticos, visando otimizar a eficácia das mesmas.

Os utilizadores da solução Logistics não interagem diretamente com a aplicação Logistics Intelligence, uma vez que esta não apresenta *front-end*, apresenta apenas *back-end* desenvolvido em *Spring*, responsável por consumir diretamente os dados da aplicação Logistics, através do acesso direto à sua base de dados.

A aplicação Logistics comunica também com o ERP (*Enterprise Resource Planning*) de cada cliente. O ERP é um sistema de informação utilizado pelos clientes para gerir todos os processos do armazém. Contudo, este é um sistema legado, de difícil consulta da informação diretamente no mesmo.

A arquitetura das soluções acima referidas é apresentada na Figura 9.

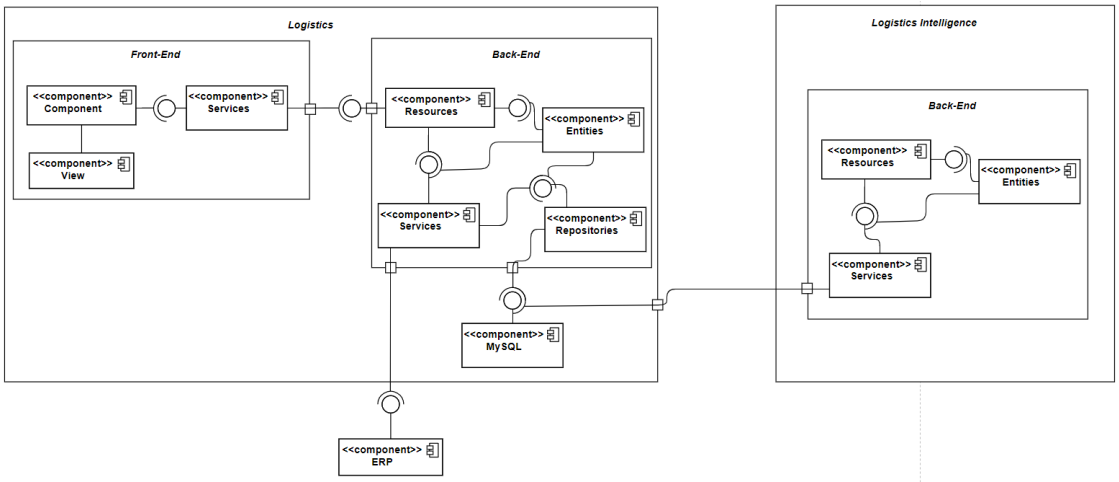

Figura 9 – Diagrama de componentes das aplicações monolíticas

Neste diagrama pode verificar-se que o *front-end* é constituído por três componentes, sendo eles:

- *View*: *Template* HTML (*HyperText Markup Language*) responsável por apresentar os dados aos utilizadores da aplicação;
- *Component*: Permite criar e apresentar as *views* e são responsáveis por gerir a interação do utilizador com a aplicação;

- *Services*: Responsável por comunicar com o *back-end* da aplicação através de pedidos REST (*Representational State Transfer*).

O *back-end* da mesma, por sua vez, é constituído por quatro componentes:

- *Resources*: Declara os métodos HTTP da aplicação e encaminha os pedidos do *front-end* para o respetivo *service;*
- *Services*: Responsável por aplicar as regras e lógica de negócio da aplicação;
- *Repository*: Camada responsável por gerir a comunicação com a base de dados e garantir a persistência da informação;
- *Entities*: Representação dos objetos presentes no domínio.

O *back-end* da aplicação Logistics Intelligence é constituído pelos *resources*, *entities e services* que apresentam a mesma responsabilidade dos anteriores referidos, no entanto estes últimos são também responsáveis por abrir uma conexão à base de dados e guardar as alterações.

No diagrama está representada a base de dados MySQL, na qual é armazenada a informação consumida pelas aplicações Logistics e Logistics Intelligence e o ERP, que representa o sistema de informação dos clientes.

6.1.2 Componentes

Neste subcapítulo serão abordados os componentes do domínio das aplicações monolíticas, bem como as suas dependências, sendo que apenas terão análise os componentes mais relevantes da aplicação Logistics, uma vez que por serem numerosos torna o diagrama de difícil leitura.

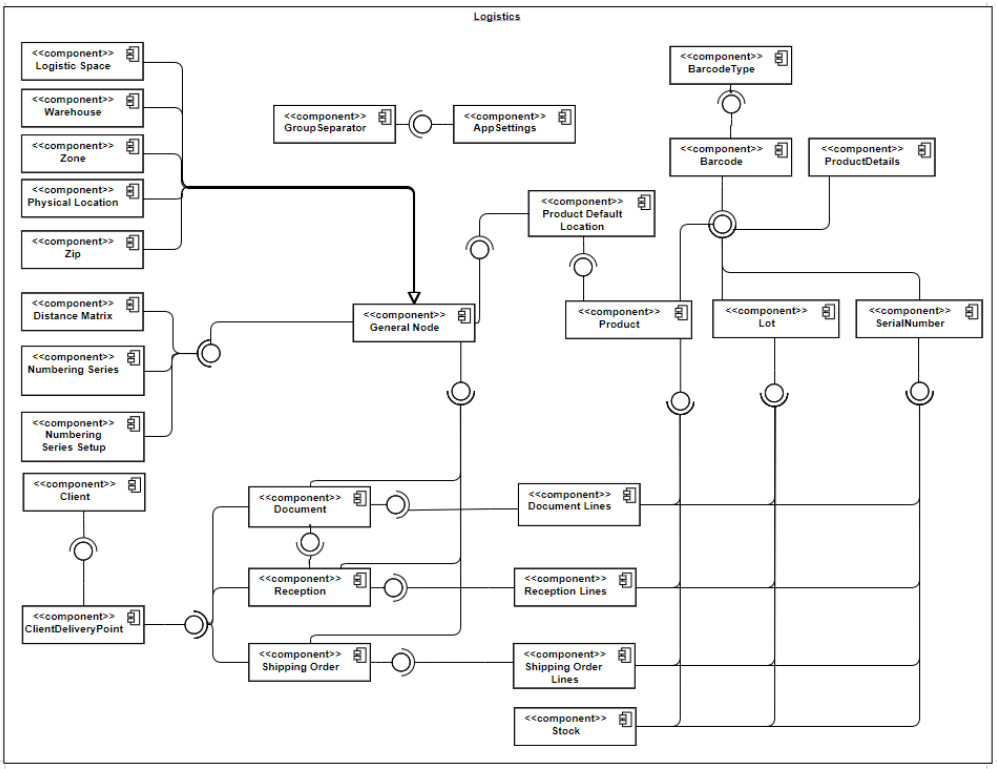

Figura 10 – Diagrama de componentes da aplicação Logistics

Observando a Figura 10, pode concluir-se que existe uma elevada dependência em quatros componentes, o *general node*, *product*, *lot* e *serial number*.

O *general node*, como o próprio nome indica, consiste na generalização das localizações existentes no armazém e tem o intuito de dividir o mesmo em locais adequados à prática dos diversos processos. Este componente permite que os utilizadores identifiquem as diversas localizações do armazém e o local de execução de cada processo.

Por sua vez, os componentes *product*, *lot* e *serial number*, representam respetivamente, os produtos presentes no armazém, os diversos lotes e o número de série de cada produto ou de cada lote.

Um lote, representa um conjunto de produtos com uma determinada característica que lhes é idêntica, enquanto que o número de série identifica a unidade de um produto. Assim, através destes componentes é possível aos utilizadores conseguirem rastrear cada produto desde o momento da sua entrada no armazém até à saída do mesmo, o que facilita a organização e a economia de tempo ao longo dos processos.

Como analisado na Figura 9, a aplicação Logistics partilha a sua base de dados com a aplicação Logistics Intelligence. Esta partilha poderá, posteriormente, condicionar a

migração arquitetural deste projeto, assim sendo, é necessário entender os componentes desta última aplicação que são essenciais e poderão influenciar a migração.

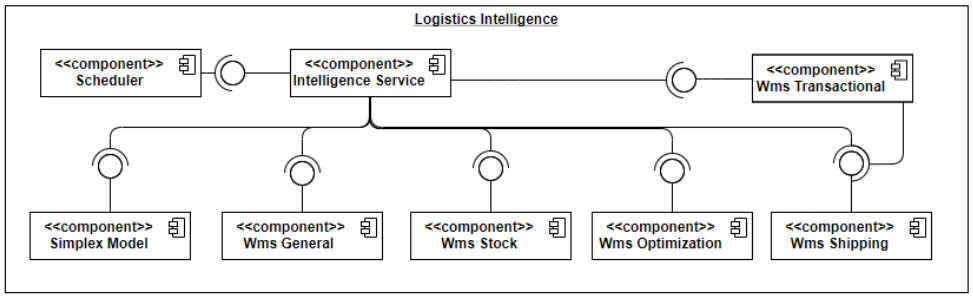

Figura 11 – Diagrama de componentes da aplicação Logistics Intelligence

A aplicação Logistics Intelligence verifica, periodicamente a existência de processos que necessitam de otimização através do componente *Scheduler*. Se for verificada a existência desses processos, esta aplicação necessitará de recorrer à base de dados para obter toda informação necessária a fim de aplicar os modelos matemáticos de otimização desenvolvidos no componente *Simplex Model*. A pesquisa pela informação necessária está dividida em componentes com base nas tabelas existentes na base de dados partilhada, e é efetuada separadamente em cada componente, nomeadamente o *Wms General*, *Wms Stock*, *Wms Shipping* e *Wms Optimization*. Por fim, o *Wms Transactional* é um componente responsável pelo retrocesso das transações efetuadas na base de dados caso ocorra algum erro ao longo dos processos.

6.1.3 Implantação

Este subcapítulo descreve o processo de implantação da aplicação monolítica em utilização pela empresa Flowinn e a que será utilizada posteriormente no processo de avaliação de resultados para a comparação entre esta aplicação e aquela que é desenvolvida no âmbito deste trabalho baseada em microsserviços.

Atualmente, a empresa Flowinn, usufrui do serviço *Bitbucket Pipelines*, contudo devido à extensa duração das *builds* da aplicação, este serviço não se encontra em utilização. Assim, quando é necessária a atualização da aplicação nos diversos ambientes, nomeadamente em testes ou produção, é necessária a execução de diversos processos de forma manual por parte dos programadores da empresa.

Inicialmente, o programador executa a *build* no seu computador através do *Maven*, e transfere o ficheiro resultante da mesma para o ambiente que pretende efetuar a implantação, através de uma ligação SSH (*Secure Shell*).

Posteriormente, efetua uma sequência de comandos, responsáveis pela criação do *backup* da implantação anterior e a execução da nova versão a implantar.

A figura seguinte ilustra o ambiente de produção desta aplicação.

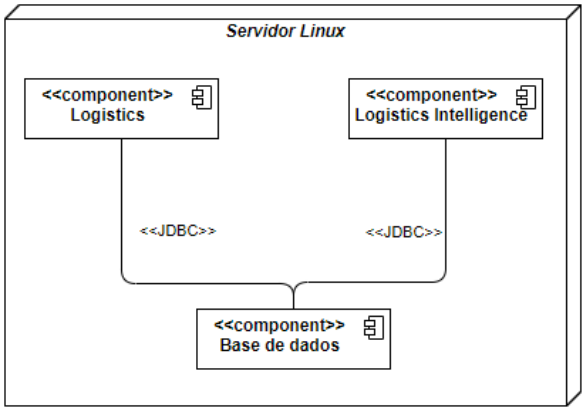

Figura 12 – Diagrama de implantação da aplicação monolítica

Como é possível observar no diagrama, a implantação é constituída pelos monólitos, Logistics e Logistics Intelligence, que se conectam à base de dados única.

O servidor representa uma máquina *Linux* com 8 RAM (memória que permite leitura e escrita de informação) e 2 CPU.

6.2 Arquitetura baseada em microsserviços – "To be"

Nesta secção, é apresentada a arquitetura que se propõe implementar, com o intuito de responder ao problema descrito no capítulo 1, no entanto, deve ressalvar-se que este trabalho é desenvolvido num ambiente dinâmico, no que respeita à estratégia e opções tomadas pela empresa Flowinn, a arquitetura proposta encontra-se representada na Figura 13.

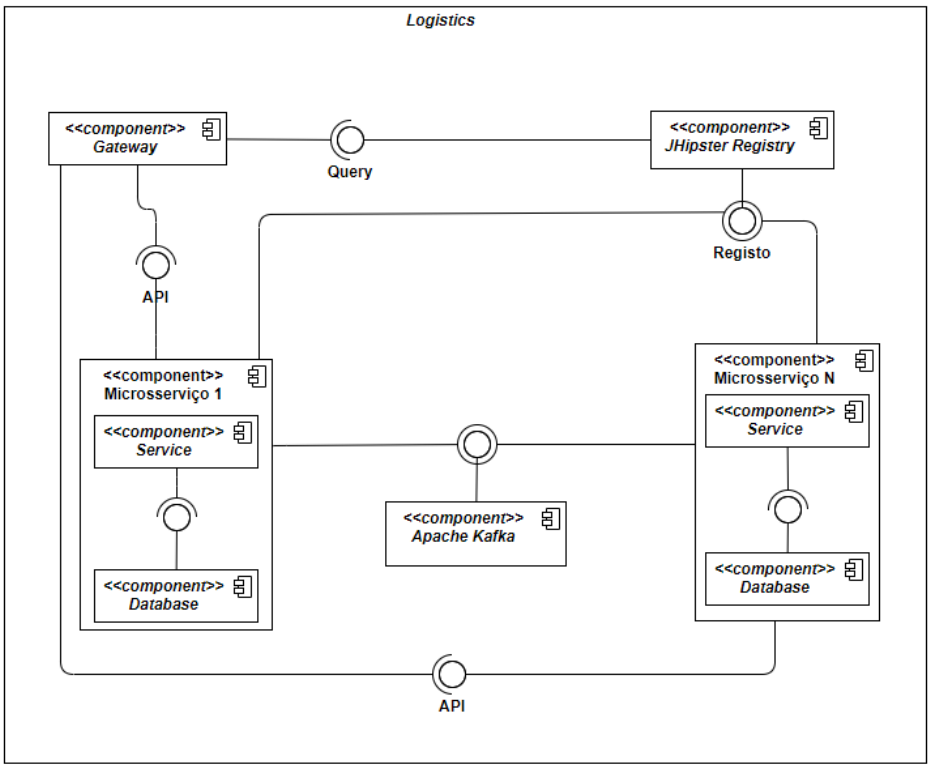

Figura 13 – Diagrama de componentes da solução a implementar

6.2.1 Gateway e JHipster Registry

Como se concluiu no capítulo anterior, a alternativa mais adequada para a implementação desta solução é o *Gateway* e *JHipster Registry*. Esta escolha deve-se à detalhada monitorização de toda a solução e à reduzida complexidade acrescentada.

A utilização desta alternativa permite resolver problemas comuns em arquiteturas baseadas em microsserviços, tais como, descoberta de serviços, monitorização da disponibilidade e gestão dos mesmos.

6.2.2 Base de dados para cada microsserviço

Com o intuito de efetuar a migração da aplicação monolítica para microsserviços, deverá ter-se em conta a divisão da base de dados única, para uma base de dados por microsserviço. Esta abordagem, como apresentado na secção 2.4, permite manter o baixo acoplamento e isolar a informação presente em cada microsserviço, contrariamente à abordagem de uma base de dados única para todos os microsserviços. Esta última, aumenta as dependências no caso dos microsserviços serem desenvolvidos

por diversas equipas, dificulta e atrasa o processo de implantação automático, uma vez que, pode necessitar de coordenação das alterações efetuadas à base de dados.

6.2.3 Apache Kafka

Apache Kafka é a abordagem pretendida para a solução, uma vez que, um dos objetivos é a escalabilidade. Como se referiu na secção 2.12, esta abordagem apresenta melhores resultados em relação ao *RabbitMQ*, prevenindo assim, possíveis problemas neste âmbito, bem como, melhor desempenho através de menos recursos, perante situações de elevado fluxo de dados.

Relativamente ao processo de implantação, este será automatizado com recurso às ferramentas apresentadas na Figura 14.

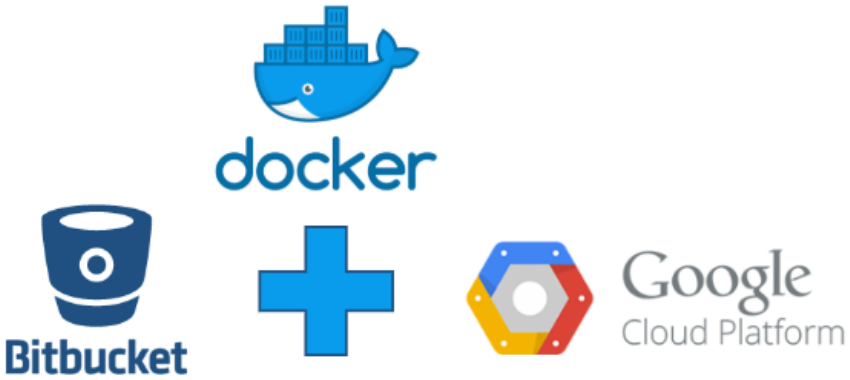

Figura 14 – Ferramentas a utilizar na automatização do processo de implantação

6.2.4 Bitbucket Pipelines

O *Bitbucket Pipelines* é a abordagem escolhida para esta solução.

Apesar do *Jenkins* ser uma abordagem gratuita, necessita mais esforço para colocar em prática. A sua utilização necessita a configuração do servidor e dos *plugins* necessários. Por sua vez, este servidor necessita de ser implantado para que seja acedido, apresentando custos adicionais à empresa.

A abordagem escolhida, *Bitbucket Pipelines* é um serviço incorporado no *Bitbucket,* o que evita a utilização recursos para a sua instalação. Os serviços disponibilizados por esta abordagem, apresentam vantagens para o quotidiano dos programadores da

mesma, uma vez que, será possível visualizar as funcionalidades implantadas em cada *build* através do produto *Jira*.

Por fim, apesar desta abordagem ser um produto pago, atualmente, a empresa Flowinn, usufrui de um plano constituído pelos produtos *Bitbucket* e *Jira*. Este oferece um *plafond* de minutos mensais para *builds* através do serviço *Bitbucket Pipelines*, apesar de atualmente, esta funcionalidade não estar a ser utilizado pela empresa.

6.2.5 Docker

Docker, como descrito na subsecção 2.13.4, é uma ferramenta que permite a implantação em ambientes controlados e configuráveis.

Esta ferramenta será utilizada com o intuito de controlar as dependências e o ambiente onde a aplicação será implantada. Assim, em caso de necessidade de análise de erros, o ambiente onde estes ocorrem é facilmente replicado e analisado.

6.2.6 Google Cloud Platform

O provedor de serviços na *cloud* escolhido para este projeto é a *Google*, optando pelo serviço *Google Cloud Platform*. Os fatores que levam à escolha desta alternativa são o preço dos serviços disponibilizados e a satisfação das necessidades da solução, nomeadamente escalabilidade, disponibilidade e processos de CI e CD.

Atualmente, a empresa Flowinn definiu o objetivo de mover os seus produtos para serviços na *cloud*, usufruindo de um contrato com este provedor. Este por sua vez, oferece os serviços pretendidos por um valor mais rentável à empresa.

Relativamente às outras abordagens estudadas, estas não justificariam uma mudança, considerando os critérios de preço e os serviços necessitados.

7 Análise e conceção

O processo de análise e conceção visa descrever as tarefas de decomposição, análise e implementação da nova solução. Ao longo deste capítulo será descrito o processo de análise do domínio e de levantamento de requisitos, bem como as frações a conceber para a nova solução.

7.1 Requisitos

O processo de desenvolvimento de *software* deve iniciar-se pela análise de requisitos. Este permite, iterativamente, identificar as funcionalidades pretendidas pelas partes interessadas para o sistema, os requisitos funcionais, requisitos não funcionais e ainda as características do mesmo.

O levantamento de requisitos exaustivo, foi efetuado anteriormente pela empresa Flowinn aquando do processo de desenvolvimento da aplicação monolítica descrita na secção 6.1. Dado o elevado número de requisitos envolvidos serão apenas analisados, através do diagrama ilustrado na Figura 15, os mais importantes e que de alguma forma condicionam a migração arquitetural proposta neste projeto.

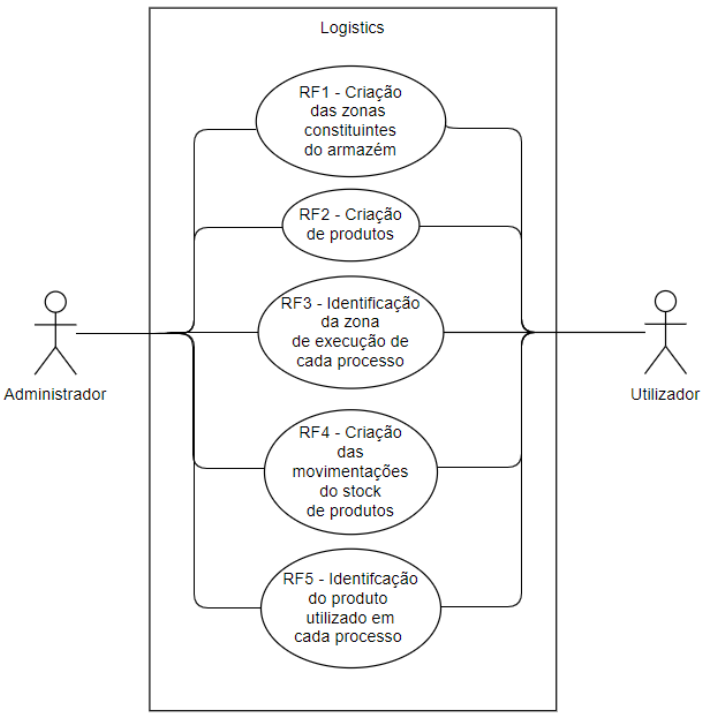

Figura 15 – Diagrama de casos de uso

Como é possível observar no diagrama da Figura 15, a aplicação Logistics apresenta dois tipos de utilizador, o administrador e os utilizadores regulares. O primeiro é exclusivamente utilizado pela Flowinn para gerir a aplicação, o segundo engloba os funcionários do armazém do cliente.

Por fim, foram identificados os requisitos não funcionais da nova solução:

- RNF1: O sistema deverá ser facilmente escalável, suportando o fluxo de pedidos efetuados pelo cliente;
- RNF2: O sistema deve efetuar a importação de dados proveniente do armazém de forma assíncrona, possibilitando que os utilizadores continuem a utilizar a aplicação;
- RNF3: A informação presente em todas as bases de dados da aplicação deverá ser coerente;
- RNF4: A aplicação deverá suportar os erros que poderão ocorrer nos processos quotidianos de gestão do armazém.

7.2 Representação do domínio

A decomposição da aplicação monolítica será iniciada através da aplicação do padrão DDD (*Domain-driven design*), descrito no subcapítulo 2.3. Através deste padrão pretende-se dividir o monólito e torná-lo em pequenas partes isoladas, os microsserviços, que representam um determinado contexto do domínio da aplicação e cumprem os seus requisitos.

Finalizada a análise do domínio completo da aplicação e da aplicação do padrão descrito, podem observar-se os microsserviços projetados para a nova solução, na Figura 16.

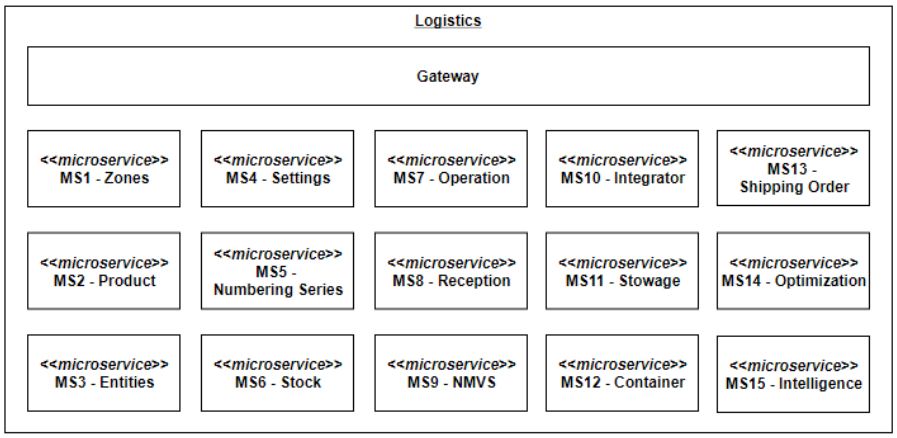

Figura 16 – Microsserviços *Logistics*

A Figura 16 ilustra os microsserviços constituintes da nova arquitetura da aplicação Logistics. A seguir descrevem-se os microsserviços mais relevantes da solução. A descrição dos restantes pode ser encontrada nos anexos, uma vez que a sua inclusão neste ponto do documento torná-lo-ia demasiado maçador.

7.2.1 Gateway

O *Gateway* é o ponto de entrada da nova solução, este é responsável por omitir a implementação dos microsserviços representados, assim o cliente não necessitará de comunicar individualmente com os mesmos, uma vez que comunicará sempre com o *Gateway*.

Este componente é constituído pelo *front-end*, que é composto por toda a informação visível para os utilizadores, e pelo *back-end*, responsável pelo encaminhamento dos pedidos para os respetivos microsserviços com base na descoberta de serviços, proveniente do *JHipster Registry* e pela gestão dos utilizadores da aplicação. Esta gestão

permite detetar transversalmente o utilizador ativo na aplicação e partilhar o seu identificador para que seja possível perceber as suas ações.

7.2.2 MS1 – *Zones*

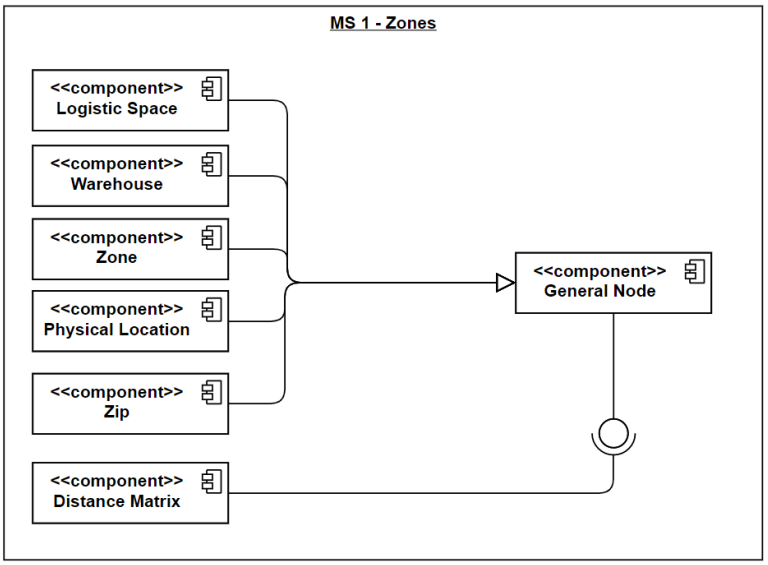

Figura 17 – Representação do microsserviço *Zones*

Este microsserviço representa o domínio das zonas nos armazéns, é responsável pela gestão das mesmas e por aplicar a respetiva lógica de negócio, ilustrada no RF1. Não apresenta qualquer dependência de outros microsserviços, contudo, é um dos mais importantes, uma vez que como descrito no RF3, sempre que exista a necessidade de situar algum processo no armazém, será necessário identificar a respetiva zona. Nos microsserviços que envolvam estes processos será guardada uma identidade única da mesma, denominada de id. Assim, é possível associar a zona ao processo, mantendo o isolamento e baixo acoplamento de cada microsserviço.

7.2.3 MS2 – Product

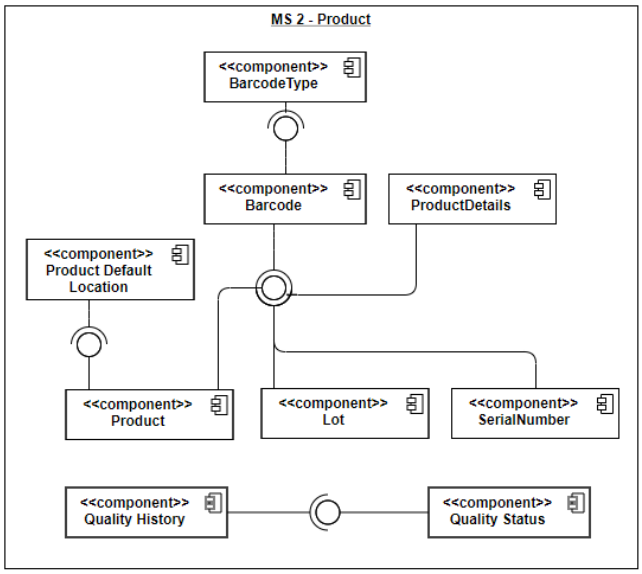

Figura 18 – Representação do microsserviço *Product*

O *Product* representa o domínio dos produtos existentes nos armazéns, identifica todos os produtos presentes no mesmo, RF2, e as suas respetivas características, nomeadamente o lote, código de barras e número de série. Este microsserviço engloba também a qualidade em que os produtos se encontram quando são recebidos no processo da receção. Similar ao MS 1 - *Zones*, este microsserviço é dos mais importantes na solução, uma vez que outros necessitam do domínio de produtos para realizar os seus processos, como enunciado no RF5. De modo a resolver esta dependência de domínio e melhorar a *performance* da solução na apresentação de dados para os utilizadores, será duplicada a informação do produto nos microsserviços, nomeadamente a descrição e o seu identificador. Assim, será possível apresentar a informação através de um pedido ao respetivo microsserviço do processo, sem ser necessário consultar o produto em todos os pedidos efetuados. Apesar desta vantagem, esta duplicação de informação é suscetível à incoerência de dados ao longo dos microsserviços, obrigando à implementação de um controlo da informação, com o intuito de a manter sempre coerente ao longo dos microsserviços. Este controlo é efetuado com recurso às mensagens publicadas no MB, *Apache Kafka*. Sempre que estes atributos sejam modificados, é publicada uma mensagem no respetivo tópico do MB, os microsserviços que guardam esta informação duplicada subscrevem o tópico e atualizam a sua base de dados local, mantendo assim a informação coerente em toda a aplicação.

7.2.4 MS5 – Numbering Series

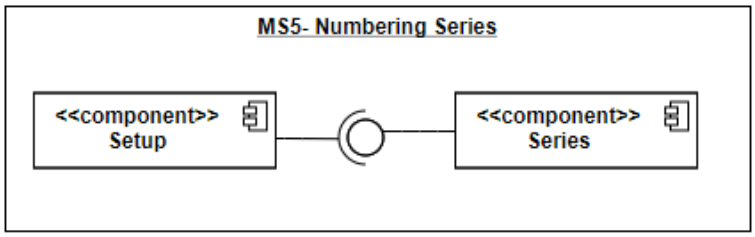

Figura 19 – Representação do microsserviço *Numbering Series*

A necessidade de o cliente identificar cada processo efetuado no armazém, implica uma lógica especifica no momento de criação do mesmo e uma vez que é transversal aos diversos processos, optou-se por isolar a mesma neste microsserviço, *Numbering Series*. Aquando a criação de um processo que necessite seguir a respetiva ordem aplicada pela lógica acima referida, será necessário comunicar com este microsserviço e obter o respetivo número de série do processo.

7.2.5 MS6 – Stock

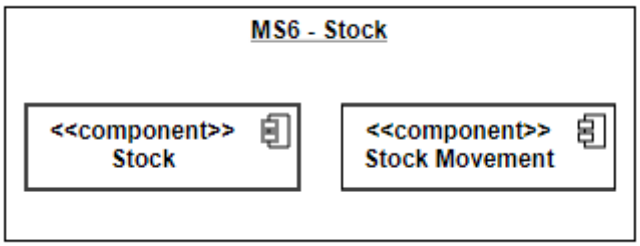

Figura 20 – Representação do microsserviço *Stock*

Este microsserviço tem o intuito de armazenar as entradas e saídas dos produtos nos armazéns, permitindo o controlo do *stock* atual nos mesmos, este processo é referido no RF4. Para tal, subscreverá eventos, relativos aos processos que envolvam a movimentação de produtos no armazém, publicados no MB, *Apache Kafka*, e registará as respetivas movimentações na sua base de dados. De modo a garantir a coerência entre a informação presente nos diversos microsserviços é necessário recorrer ao padrão SAGA, descrito na subsecção 2.5, uma vez que caso ocorram falhas, nomeadamente rutura de stock, é necessário garantir que as operações resultantes do processo denotem as mesmas para que o utilizador as consulte posteriormente.

7.2.6 MS8 – Reception

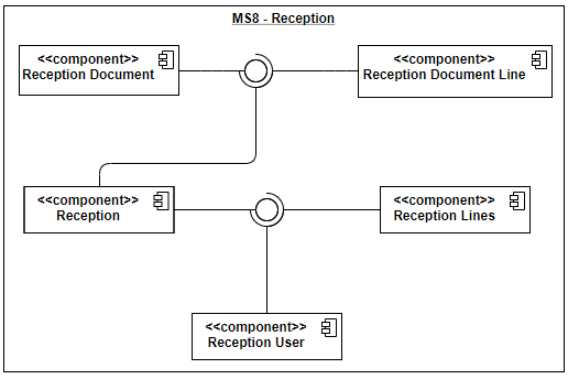

Figura 21 – Representação do microsserviço *Reception*

O *Reception* pretende agrupar o domínio relacionado com o processo de receção no armazém, nomeadamente, o produto rececionado e a sua qualidade. Esta informação será armazenada e associada à respetiva receção. Por sua vez, o *reception document* permite ao cliente identificar os documentos que dão origem às receções.

Por fim, existindo a necessidade de identificar os utilizadores que executam as receções, este microsserviço engloba os *reception users* que duplicam a identificação única do utilizador, associando-o às respetivas receções.

Este microsserviço é um dos mais importantes da solução, visto que engloba o processo de receção, sendo que este é o mais praticado no armazém do cliente.

7.2.7 MS10 – Integrator

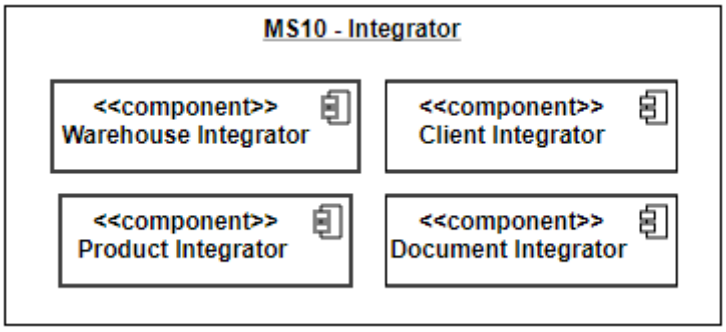

Figura 22 – Representação do microsserviço *Integrator*

Este microsserviço tem o intuito de centralizar as configurações de comunicação com o ERP utilizado por cada cliente e efetuar as integrações necessárias no mesmo, nomeadamente a importação e exportação da informação. Assim, as comunicações com sistemas ERP serão efetuadas exclusivamente através deste microsserviço,

reduzindo a possível duplicação das configurações de cada sistema nos restantes microsserviços.

Posteriormente à comunicação com os sistemas referidos, será publicado um evento no respetivo tópico do MB, permitindo ao microsserviço referente ao domínio da integração subscrever o mesmo, interpretar e transformar a informação recolhida na representação do domínio e armazenar na sua base de dados.

7.2.8 MS14 – Optimization

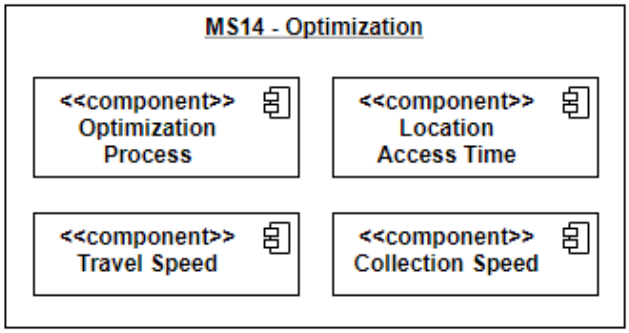

Figura 23 – Representação do microsserviço *Optimization*

O *Optimization* é responsável por gerir a otimização dos processos de expedição no armazém. Este irá armazenar a informação que será utilizada, posteriormente, para auxiliar os modelos matemáticos aplicados pelo M15 - *Intelligence*. Sendo esta informação referente à aplicação *Logistics*, optou-se pela criação deste microsserviço, invés de armazenar a informação no M15 - *Intelligence*. Assim, os modelos matemáticos desenvolvidos no mesmo poderão ser utilizados em diferentes contextos.

7.2.9 MS15 – Intelligence

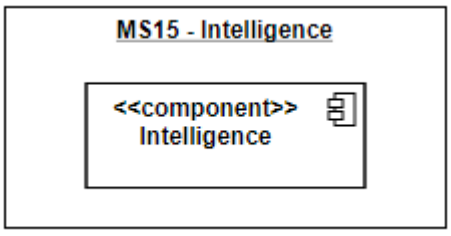

Figura 24 – Representação do microsserviço *Intelligence*

A aplicação monolítica Logistics Intelligence, descrita na subsecção 6.1.2, obtém a informação presente na aplicação Logistics através do acesso direto à base de dados.

Assim, com a migração proposta neste trabalho, o acesso direto obriga também a migração da primeira aplicação. Pois, anteriormente a informação encontrava-se presente na única base dados da aplicação Logistics, porém agora está dispersa ao longo dos microsserviços conforme o seu domínio.

Este acesso direto opõe-se aos princípios e vantagens dos microsserviços apresentados na subsecção 2.2, nomeadamente o desacoplamento e isolamento dos microsserviços.

7.3 Processos do armazém

Neste subcapítulo serão descritos os processos dos armazéns e algumas das etapas frequentemente efetuadas no quotidiano. Como consequente resultado da mudança arquitetural e da utilização do padrão base de dados para cada microsserviço, a informação é armazenada ao longo dos diversos microsserviços, contrariamente à solução monolítica, que se encontra numa base de dados única.

De forma a manter a informação coerente e comunicar as alterações ao longo de todos os microsserviços envolvidos no processo, é necessária a utilização dos padrões SAGA e MB, descritos na secção 2.

Assim sendo, existiu a necessidade de adaptar a implementação dos diversos processos dos armazéns, uma vez que exigem o processamento de informação proveniente de vários microsserviços.

7.3.1 Receção

A receção é um dos processos mais importantes no quotidiano do armazém, inicia-se com a chegada dos produtos adquiridos nos fornecedores, ao armazém do cliente. Este processo é constituído por várias etapas, nomeadamente, receção dos produtos recebidos, verificação da autenticidade e qualidade dos mesmos, separação e reorganização dos produtos nos locais destinados à prática deste processo e por fim o acréscimo dos produtos rececionados ao *stock* presente no armazém.

Estas etapas exigem rigor, pois uma simples falha pode causar elevados prejuízos e influenciar a produtividade nos seguintes processos do armazém.

O diagrama ilustrado na Figura 25 descreve o processo de receção de um produto.

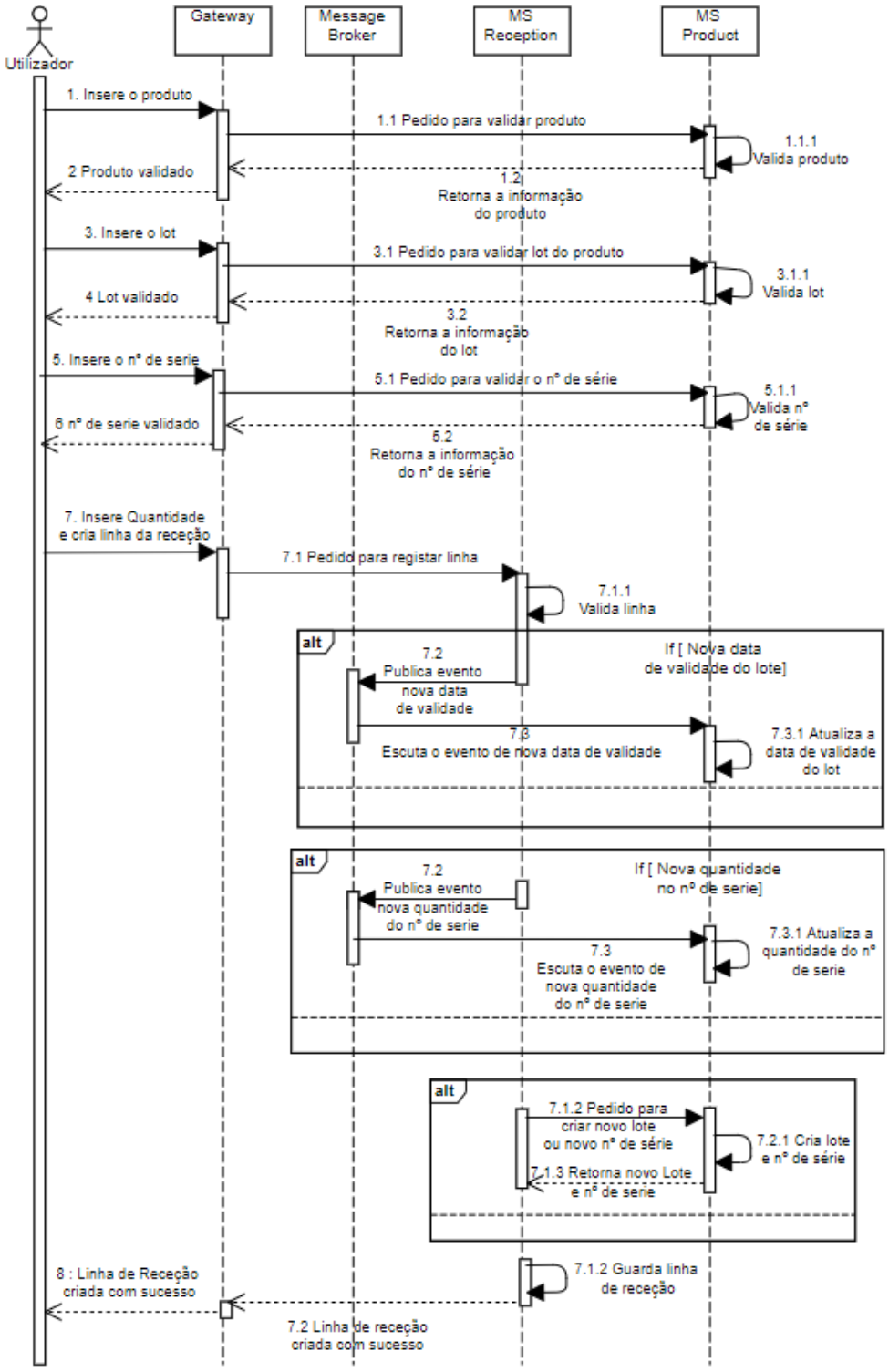

Figura 25 – Diagrama de sequência de rececionamento de produtos

Como se pode observar, o utilizador apenas comunica com o *Gateway* ao longo de todos os pedidos efetuados. Este é responsável por comunicar com os restantes microsserviços dependendo da ação que pretende efetuar. O processo inicia-se após o

utilizador da aplicação indicar o código do produto que pretende rececionar e este ser validado pelo MS *Product*. Neste caso o utilizador preenche também o lote e o número de série do produto em questão e estas informações são validadas no MS *Product*.

Após o utilizador inserir a quantidade que pretende rececionar, o *Gateway* comunica com o MS *Reception*. Este é responsável pela validação e armazenamento da linha de receção efetuada pelo utilizador. Com o intuito de manter a coerência da informação, estes dois microsserviços necessitam de aplicar a lógica de negócio ilustrada no diagrama Figura 25. No caso em que o utilizador comunica uma nova data de validade de um lote para um produto em questão, o MS *Reception* publica o evento no MB utilizado nesta arquitetura, *Apache Kafka*, que será subscrito pelo MS *Product*. Este, posteriormente, atualizará a respetiva data de validade para o lote em questão. No caso em que se verifique a necessidade de atualizar a quantidade de um produto com número de série, será publicada um outro evento no MB, que permitirá ao MS *Product* subscrever e efetuar a lógica de negócio especifica. Contudo, no caso de ser necessário criar um lote ou número de serie, será efetuado um pedido síncrono ao MS *Product*, uma vez que o MS *Reception* necessita do identificador único do lote ou do número de série para continuar a aplicar a sua lógica de negócio e posteriormente armazenar a informação relativa ao produto rececionado. Através deste pedido síncrono é possível assegurar a coerência da informação entre ambos os microsserviços.

Após o armazenamento da informação sobre o produto rececionado, o MS *Reception*, comunica o sucesso do procedimento ao G*ateway*, que por sua vez informa o utilizador.

Posteriormente à conclusão do processo de rececionamento dos produtos anteriormente detalhado, o utilizador irá finalizar a receção, iniciando o processo descrito na Figura 26.

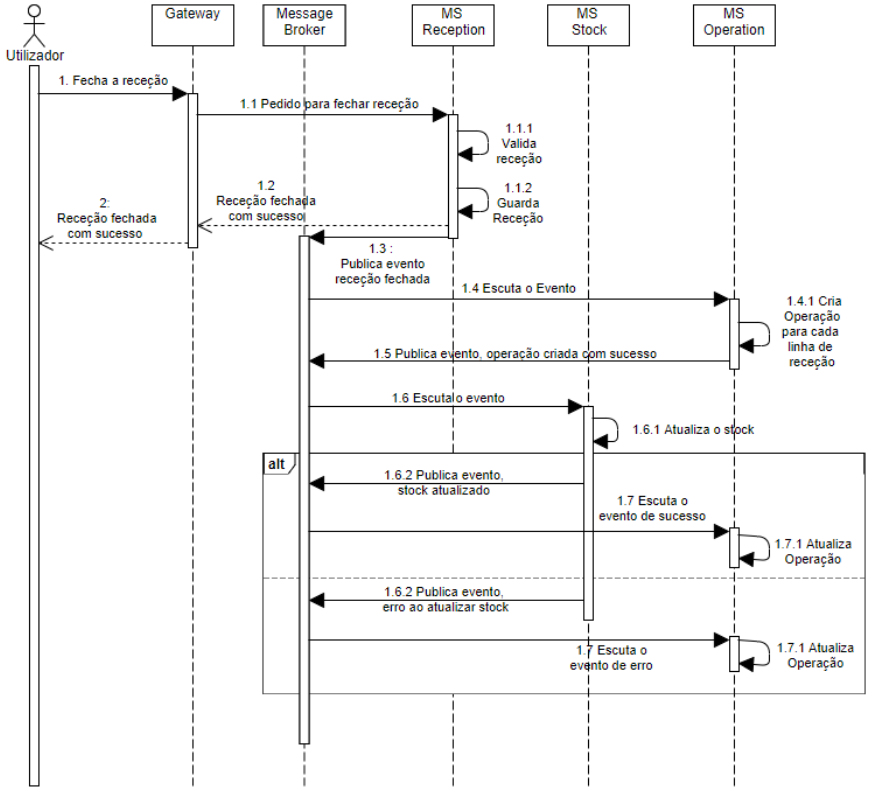

Figura 26 – Diagrama de sequência de finalização da receção

Inicialmente, este processo irá validar a informação submetida pelo utilizador e guardar na base de dados do microsserviço *Reception*, seguidamente comunicará o sucesso com o *Gateway* que por sua vez informará o utilizador. Aquando a finalização da receção, é publicado um evento no MB, este será subscrito pelo MS *Operation* que dará início à criação das operações resultantes da receção em questão.

De forma similar ao rececionamento de produtos, este processo exige a coerência entre a informação presente na base de dados de dois microsserviços, o MS *Stock* e o MS *Operation*, sendo necessário recorrer à utilização do padrão SAGA descrito no subcapítulo 2.5. Este terá início através da publicação do evento acima descrito. Este evento será subscrito pelo MS *Stock*, responsável por validar e atualizar o stock com base nas operações criadas. Dependendo do resultado da movimentação do *stock*, este microsserviço publica um evento que será subscrito pelo MS *Operation*, este, por sua vez, irá atualizar as respetivas operações concluindo o processo de finalização da receção e garantindo a coerência entre a informação presente na base de dados de cada microsserviço.

7.3.2 Arrumação

Seguidamente, será analisado o processo de arrumação, este consiste na movimentação dos produtos de uma zona do armazém para outra. Este processo é geralmente efetuado para mover os produtos das zonas de receção para as respetivas zonas do armazém, mantendo assim o stock de produtos no armazém organizado, o que facilita os processos posteriores de expedição no armazém.

Seguidamente, na Figura 27, será analisado detalhadamente a arrumação de um produto no armazém.

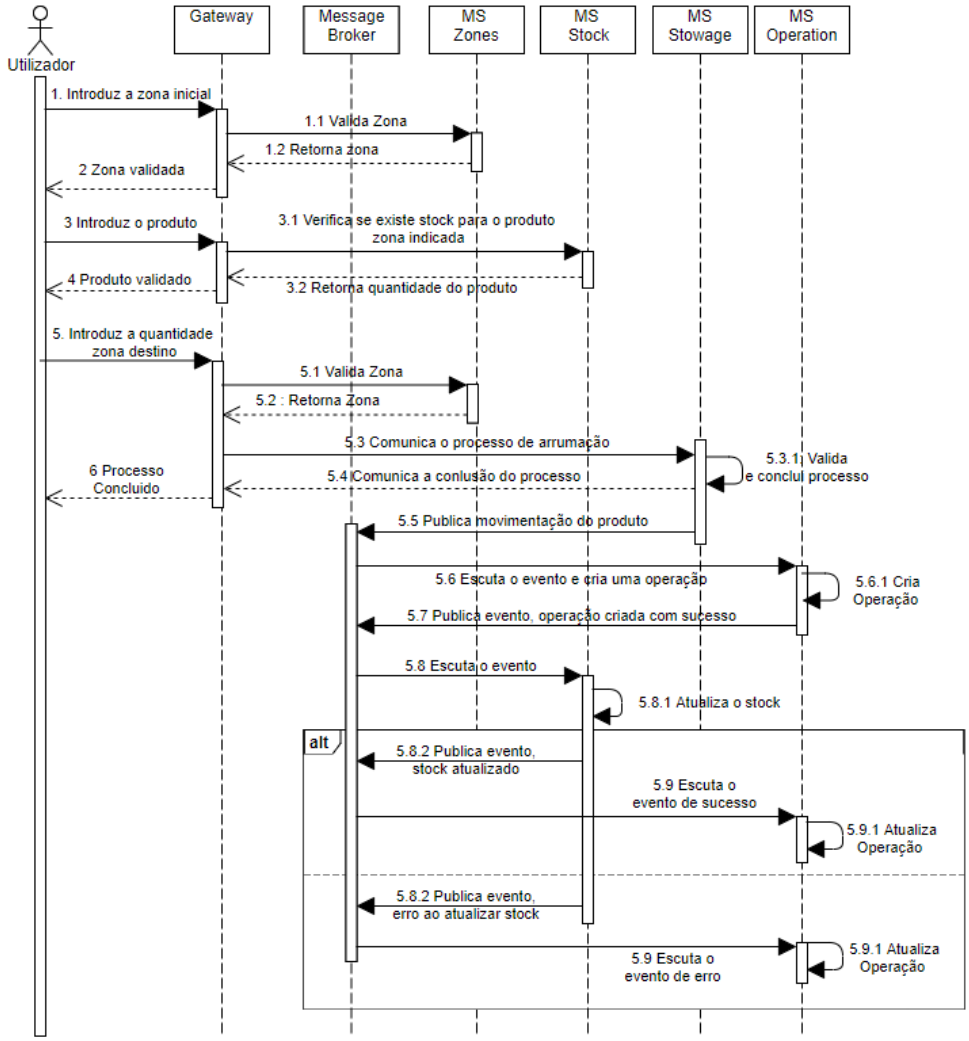

Figura 27 – Diagrama de sequência da arrumação de um produto

Como é possível observar na Figura 27, este processo é iniciado pelo utilizador indicando a zona onde os produtos estão inicialmente localizados. Esta ação é comunicada com o *Gateway,* que por sua vez recorre ao microsserviço *Zones* para validar a zona introduzida. Posteriormente, o utilizador deverá indicar o produto que pretende mover, este será validado através do microsserviço *Stock*, que verificará a

existência do produto em questão e retornará a quantidade existente no local. Posteriormente, o utilizador indicará a quantidade do produto que pretende movimentar e a zona destino que será validada novamente pelo MS *Zones*.

Seguidamente, será comunicado ao MS *Stowage*, o processo de arrumação pretendido. Este microsserviço valida e conclui o processo desencadeando duas ações simultâneas, nomeadamente a comunicação da conclusão do processo ao *Gateway*, que informará o utilizador, e a publicação de um evento no MB. Este evento será subscrito pelo MS *Operation*, que criará a operação resultante do processo e seguidamente publicará um novo evento no MB.

Similarmente ao processo de finalização da receção, este também exige a coerência entre a informação presente na base de dados do MS *Operation* e do MS *Stock*, pelo que será aplicado o mesmo padrão SAGA, ou seja, o MS *Stock* irá subscrever o evento publicado pelo MS *Operation* e realizar a movimentação do *stock* do produto para a zona destino. Conforme o sucesso desta movimentação, será publicado um evento que será subscrito pelo MS *Operation*, que irá atualizar a respetiva operação do processo de arrumação em questão.

7.3.3 Expedição

O processo de expedição engloba a gestão das encomendas que os clientes efetuam ao armazém. Este processo deve ser rigoroso e bem executado de modo a obter o maior lucro possível para o armazém, sendo o seu foco entregar os produtos aos clientes nas melhores condições e dentro dos prazos estabelecidos. Atualmente a aplicação permite efetuar esta gestão através de três processos distintos, a criação de encomendas e das linhas de encomenda e otimização das mesmas.

De modo a iniciar um processo de expedição, o utilizador deverá registar a informação importante relativa à encomenda recebida, nomeadamente, o tipo de processo, o local e data da entrega da encomenda. Posteriormente o utilizador poderá iniciar o segundo processo, criação de linhas da encomenda, que pretende registar os produtos e as respetivas quantidades encomendadas pelo cliente.

Por fim, devido à mudança arquitetural, o processo de otimização foi divido em subprocessos sendo estes, a aprovação da encomenda e estimativa da mesma.

Esta divisão visa diminuir a complexidade do processo em si e obter o máximo proveito da assincronia oferecida pelo MB implementado na nova solução. Os subprocessos executam sem a necessidade de o utilizador intervir e os seus procedimentos serão analisados nos seguintes diagramas.

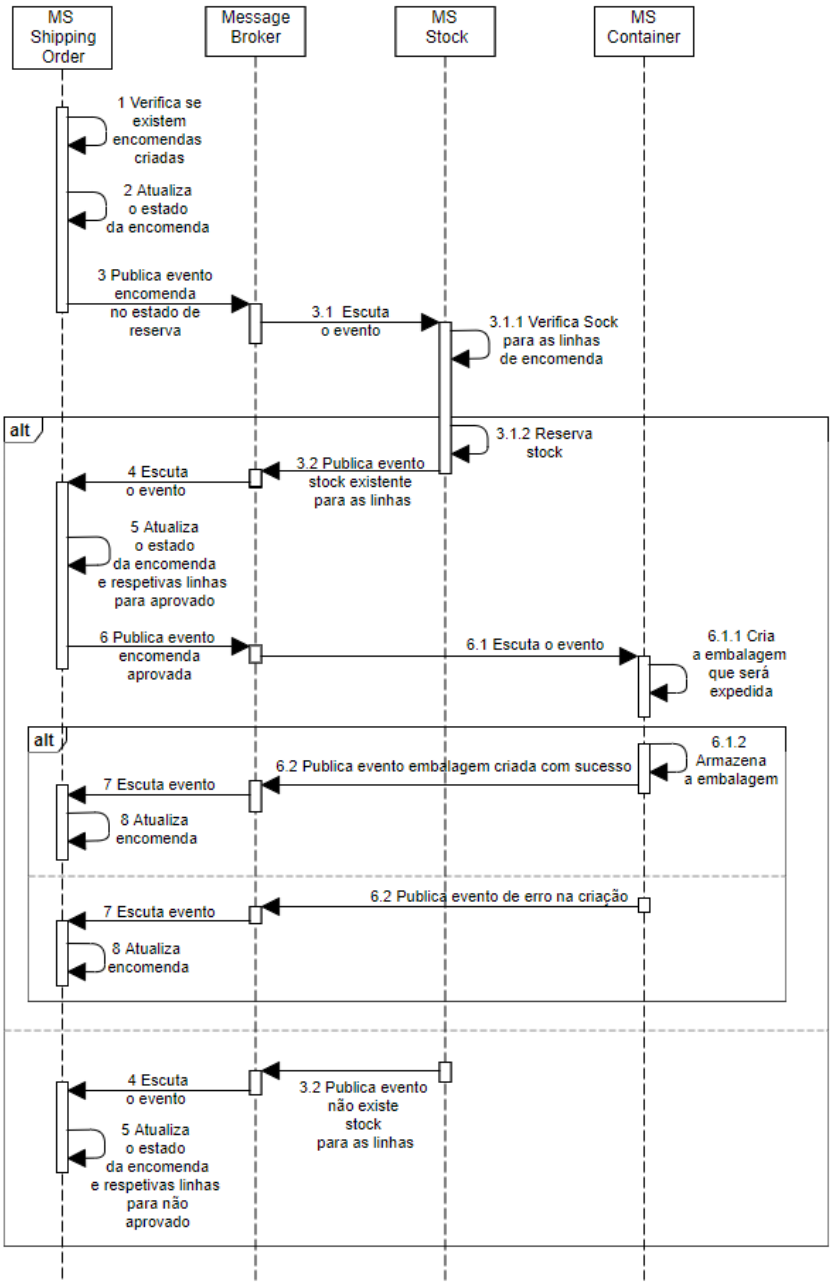

Figura 28 – Diagrama de sequência da aprovação da encomenda

Contrariamente aos processos anteriormente analisados, este inicia-se de forma automática. O MS *Shipping Order* verifica, periodicamente, a existência de novas encomendas e inicia o processo de aprovação, atualizando o estado da encomenda. Aquando esta atualização, é publicado um evento no MB, que será subscrito pelo MS *Stock*. Este irá verificar se os produtos pretendidos existem em *stock*, de modo a satisfazer a encomenda em questão.

No caso de ausência de stock, o processo de aprovação irá terminar através da publicação de um evento no MB. O MS *Shipping Order* irá interpretar o evento de não aprovação e atualizará a encomenda com o respetivo estado. Pelo contrário, na existência de *stock* para a encomenda, o MS *Stock*, reserva as quantidades necessárias publicando o sucesso no MB. Este será escutado pelo MS *Shipping Order*, que por sua vez irá atualizar a encomenda para o estado de aprovada. Esta atualização, permitirá o MS *Container* escutar a aprovação e atribuir a embalagem necessária para enviar os produtos pretendidos ao cliente. Aquando esta atribuição publicará um evento conforme o sucesso da operação, permitindo assim ao MS *Shipping Order*, escutar o mesmo e atualizar a encomenda na sua base de dados, mantendo a informação coerente ao longo dos microsserviços.

A implementação do MB, permitiu reduzir o tempo total deste processo, dado que anteriormente, na solução monolítica, primeiramente era efetuado a reserva de *stock* sem despoletar o início da criação da embalagem, sendo obrigatório a espera pela nova iteração da tarefa que ocorre a um dado intervalo de tempo.

Após a conclusão da aprovação das encomendas, estas necessitam de ser estimadas e otimizadas de modo a reduzir o tempo de entrega ao cliente. Este processo será analisado detalhadamente no diagrama ilustrado na Figura 29.

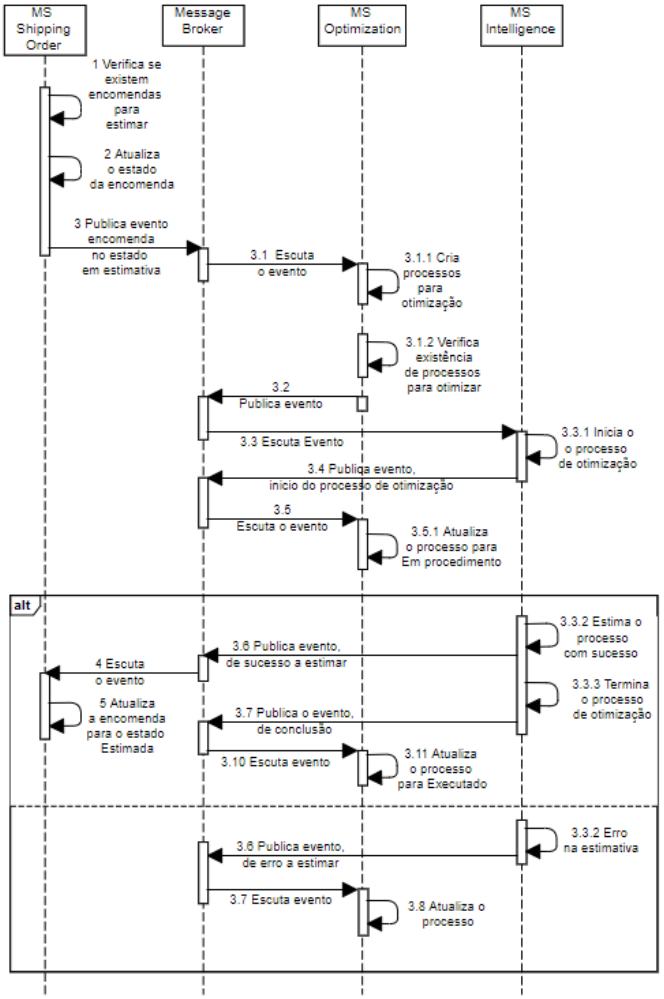

Figura 29 – Diagrama de sequência da estimativa das encomendas

O MS *Shipping Order* inicia, periodicamente, este processo, verificando a existência de encomendas disponíveis para estimar, atualizando o seu estado e publicando um evento no MB. O MS *Optimization* irá interpretar o evento anteriormente subscrito e criar um processo a ser otimizado pelo MS *Intelligence*. Para tal, o primeiro realiza uma tarefa periódica responsável por verificar a existência de novos processos a ser otimizados, posteriormente publicados no MB, em forma de evento. Com isto, o MS *Intelligence*, subscreverá estes eventos e iniciará a estimativa da encomenda através dos modelos matemáticos desenvolvidos. Estes visam otimizar as viagens dos utilizadores no armazém para recolha de produtos. Em caso de insucesso este processo de estimativa termina com a publicação de um evento no MB, que permite ao MS *Optimization* atualizar o seu processo. Contrariamente, em caso de sucesso serão publicados dois eventos, um que permite ao MS *Shipping Order* atualizar a encomenda em questão com a respetiva estimativa calculada, e um segundo que permite sinalizar o processo como executado corretamente no MS *Optimization*.

7.4 Processos de sincronização

Neste subcapítulo serão descritos os processos de importação de dados provenientes do ERP do cliente para a aplicação baseada em microsserviços.

Anteriormente, como descrito na subsecção 6.1.1 e com o intuito de manter a informação coerente na aplicação relativa aos produtos, clientes, armazéns e documentos, a aplicação monolítica comunicava com o ERP do cliente. Atualmente, é necessário manter a importação de dados funcional e para tal foi necessário criar o MS *Integrator*, descrito na subsecção 7.2.7.

É possível efetuar a importação dos dados através da sequência descrita no diagrama Figura 30.

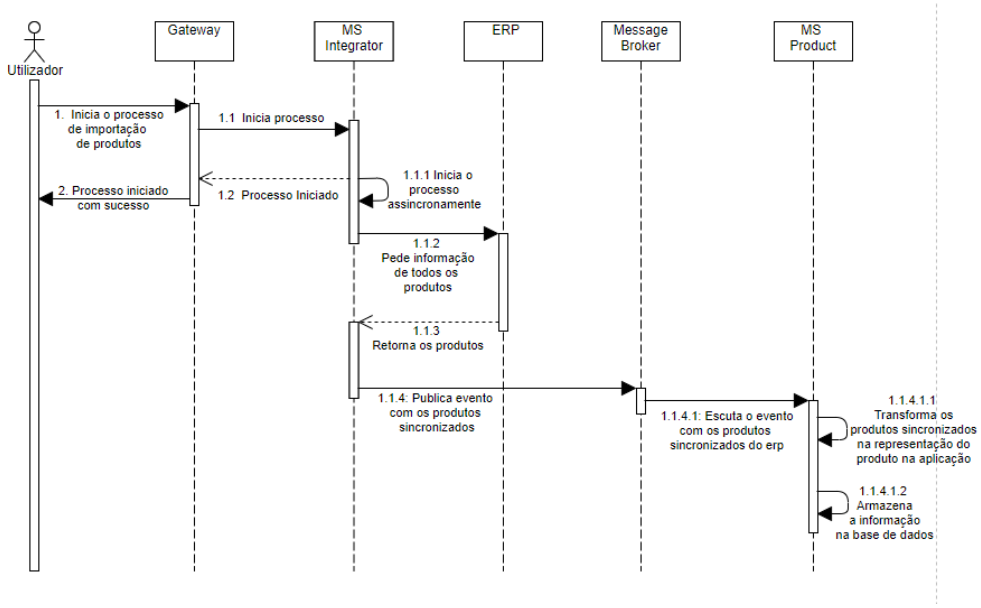

Figura 30 – Diagrama de sequência da importação de produtos

O diagrama acima ilustra o processo manual de importação de dados dos produtos do ERP.

Este processo inicia-se pela ação do utilizador no *Gateway*, que comunica a ação pretendida ao MS *Integrator*. Este microsserviço aquando a receção do pedido, inicia a integração dos dados de forma assíncrona e comunica ao *Gateway* que o pedido foi aceite, este último por sua vez, indica ao utilizador a importação dos produtos. O MS *Integrator* é responsável pela comunicação e receção dos dados proveniente do ERP, publicando os mesmos no MB através de um evento. Posteriormente, o MS *Product* escuta o evento e transforma os dados provenientes do ERP na representação de

domínio dos produtos, armazenando os mesmo na base de dados local. Termina assim o processo de importação dos dados.

Este processo que representa a importação de dados dos produtos é similar às restantes integrações, nomeadamente importação de clientes, armazéns e documentos. Se o domínio a importar difere, o evento publicado e o microsserviço que subscreve o mesmo, também diferem.

7.5 Padrões utilizados

Com o intuito de facilitar e melhorar a qualidade da nova solução, baseada em microsserviços, foram utilizados padrões que visam enfrentar algumas das desvantagens comuns nestas arquiteturas.

Inicialmente no processo de decomposição da aplicação monolítica foi utilizado o padrão *Domain-driven design*. Este auxiliou a definição dos microsserviços implementados através da divisão dos mesmos pelo domínio a representar, isolando assim as regras e lógica de negócio do mesmo.

Relativamente ao *Gateway,* aplica o padrão *API Gateway*, descrito na subsecção 2.9, este será o único ponto da aplicação acessível diretamente pelos utilizadores, ocultando a implementação de microsserviços efetuada. Esta dissimulação permite facilmente efetuar alterações à arquitetura de microsserviços, sem adicionar complexidade no *front-end* visível pelo cliente.

De modo a solucionar a identificação dos microsserviços e possibilitar a comunicação com os mesmos, foi necessário a utilização do padrão *Service Discovery*, através do componente *JHipster Registry*, detalhado na subsecção 5.1. Este é responsável por armazenar a localização dos microsserviços e verificar a sua disponibilidade periodicamente.

Na decomposição da base de dados foi aplicado o padrão *Database per service*, que consiste na utilização de uma base de dados por microsserviço, permitindo isolar a informação armazenada do domínio, diminuir o acoplamento dos microsserviços e possíveis quebras de performance no acesso e persistência da informação. Um dos aspetos mais importantes deste padrão é garantir que as bases de dados são acedidas apenas pelo microsserviço a que pertencem, para tal como descrito na análise do MS *Intelligence*, foi necessário alterar o anterior comportamento da aplicação Logistics Intelligence, migrando a mesma para o microsserviço descrito.

Derivado à divisão efetuada na base de dados, foi necessário implementar mecanismos que garantam a coerência entre a informação presente nos microsserviços, nomeadamente nos casos em que exista a necessidade de modificar a informação presente em diversos microsserviços, foi aplicado o padrão SAGA, descrito na subsecção 2.5. Este padrão foi aplicado baseado em coreografia, ou seja, cada microsserviço após realizar a transação local, publica um evento no MB, que será subscrito por outros microsserviços, acionando assim as suas transações locais. Contudo, se uma transação local falhar, será publicado o evento de erro, permitindo aos microsserviços reverterem as alterações anteriormente efetuadas, garantido assim a coerência da informação nos casos de sucesso e insucesso dos processos.

De modo a efetuar a comunicação entre os microsserviços, foram utilizados os dois padrões de comunicação descritos nas subsecções 2.11 e 2.12, nomeadamente comunicação síncrona e assíncrona. No decorrer da implementação foi priorizada a utilização da comunicação assíncrona, uma vez que esta apresenta vantagens sobre a síncrona, nomeadamente o microsserviço que pretende comunicar não necessita de saber a localização do destinatário, apenas precisa de publicar uma mensagem no MB. Estas permitem que os destinatários possam estar indisponíveis no momento da comunicação, já que permanecerá no MB até ser processada, isto poderá ocorrer quando o microsserviço destino estiver disponível.

Como descrito na subsecção 6.2.3, o MB implementado nesta solução foi o *Apache Kafka*, visto que apresenta desempenho superior à alternativa *RabbitMQ* analisada. Este componente através da publicação e subscrição de eventos pelos microsserviços, permite a comunicação entre estes sem saberem da sua existência, proporcionando a redução do acoplamento entre os microsserviços.

Por fim, a comunicação síncrona foi utilizada apenas em último recurso, nos casos em que a assíncrona não poderia ser aplicada, nomeadamente nos processos que necessitam de informação imediata de outro microsserviço para avançar com sua lógica de negócio.

8 Implantação da nova solução

Para os processos de desenvolvimento e implantação da nova solução foram aplicados os conceitos de CI e CD, através da utilização das ferramentas abordadas na subsecção 6.2, o que facilita o processo.

Como já referido na subsecção 6.2, o sistema de controlo de versões utilizado pela empresa Flowinn é o *Bitbucket*. Este foi utilizado de modo a hospedar os diversos repositórios *Git* e criar versões do respetivo código-fonte desenvolvido neste projeto.

Neste projeto foi utilizado um repositório *Git* para cada microsserviço e *Gateway*, resultando no total de 16 repositórios, assim será possível disponibilizar os mesmo de forma independente.

Ao longo das próximas subsecções serão descritos o processo de CD e o ambiente de implantação da nova solução.

8.1 Bitbucket Pipelines

Foi utilizado o *Bitbucket Pipelines*, serviço descrito na subseção 6.2.4, para promover a automatização das implantações e evitar os processos manuais com maior suscetibilidade a erros. Este, permite aplicar o padrão de CD através do *pipeline* desenvolvido neste projeto e integrar as implantações com o produto *Jira*. Esta integração será analisada com mais detalhe no anexo A.2.

Cada microsserviço terá um ciclo de vida independente do desenvolvimento e implantação. Desta forma, cada um terá um *pipeline* próprio, o que se pode observar com mais detalhe no anexo A.2. Este contém as tarefas ilustradas na Figura 31.

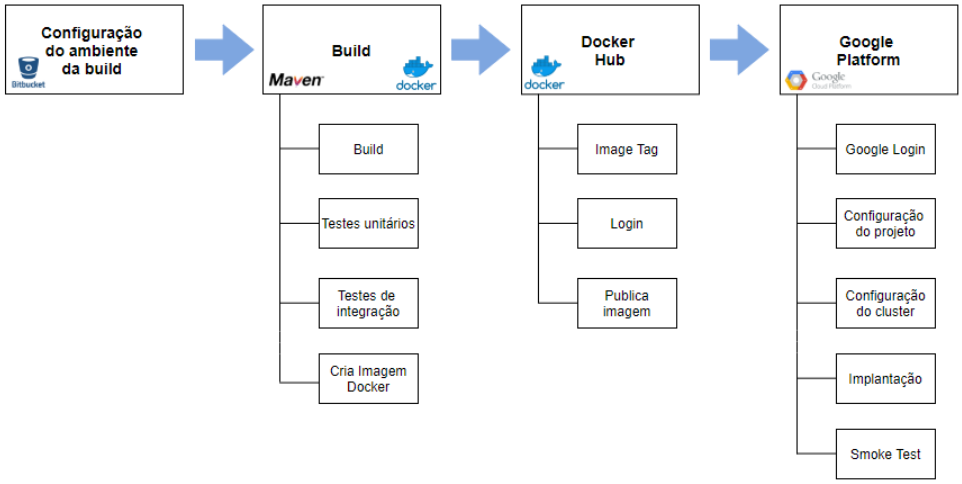

Figura 31 – *Pipeline* desenvolvido no *Bitbucket Pipelines*

O *pipeline* desenvolvido, pretende automatizar o processo de criação da *build* e disponibilização da mesma no ambiente de testes e de produção. Como é possível observar na imagem, este é constituído por quatro *steps*, sendo eles, configuração do ambiente de build, *Docker Hub*, *build, Google Platform*.

O primeiro *step*, consiste na definição do ambiente em que o *pipeline* irá correr, esta configuração poderá assumir dois valores, *dev*, representante do ambiente de testes e *prod*, que representa o ambiente de produção.

O segundo é o *step build*¸ que é constituído por quatro subprocessos, nomeadamente a execução da *build* através do *Maven*, execução dos testes unitários seguidos de testes de integração e por fim é criada a imagem *Docker*. Esta, no *step* seguinte, *Docker Hub,* será publicada no repositório de imagens através de três subprocessos. O primeiro *Image Tag,* atribui uma *Docker Tag* à imagem anteriormente gerada, fornecendo assim mais informação sobre a mesma e facilitando a sua identificação e gestão no futuro. O subprocesso seguinte consiste no *login* no repositório de imagens, *Docker Hub*, sendo que por fim, o último subprocesso publica a imagem no mesmo.

Com o intuito de facilitar a gestão dos repositórios de cada microsserviço foi definido a seguinte nomenclatura.

<<nome da conta>>/<<microsserviço>>-<<ambiente de execução>>

Com esta nomenclatura é possível identificar o microsserviço e o ambiente do repositório. O controlo da versão das imagens é efetuado através do número da *build* no *Bitbucket Pipeline*, resultando na seguinte nomenclatura para cada imagem:

<<nome da conta>>/<<microsserviço>>-<<ambiente de execução>>:<<nº da *build*>>

Por fim, o último *step* é efetuado de forma distinta conforme o ambiente definido no *pipeline*, através do *Bitbucket Deployments*. Este serviço permite a gestão dos vários ambientes de implantação através da atribuição de nomes diferentes aos mesmos. Permite também a utilização de variáveis diferentes conforme a implantação, deste modo, como descrito detalhadamente no anexo A.2, foi possível a reutilização de um *step* genérico de implantação, responsável por implantar conforme o *Deployment* pretendido e previamente configurado.

Este *step*, é constituído por cinco subprocessos. Os primeiros três, são responsáveis por efetuar o *login* na *Google*, a ligação ao projeto e ao *cluster* onde se pretende implantar. O quarto *step* atualiza a versão da imagem *Docker* executada no cluster *Kubernetes* e o último executa um *smoke test*, com o intuído de confirmar a disponibilidade da implantação da solução.

No dia 14 de Agosto, por questões internas, a empresa Flowinn, abdicou da ideia de implantar a solução na *Google*, optando pela *DigitalOcean*. Assim, de modo a cumprir os requisitos da empresa e promover a continuidade do projeto, foi necessário a adaptação do *pipeline* já desenvolvido para a *Google*.

Tendo estudado e conseguido seguir todos os passos da implantação na plataforma *Google*, para a implantação na plataforma *DigitalOcean* apenas foi necessário a criação do *script* de implantação no novo provedor de serviços *cloud*, a configuração do ambiente da nova implantação e a substituição do ambiente a implantar no último *step* do pipeline. Assim, o *pipeline* resultante, apenas difere no quarto *step* anteriormente descrito na Figura 31. Este foi substituído pelo seguinte *step*.

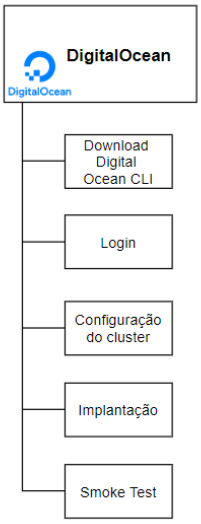

Figura 32 – *Step* de implantação na *DigitalOcean*

Como é possível observar na Figura 32, o *step* responsável por implantar a solução na *DigitalOcean* é constituído por cinco subprocessos. O primeiro consiste no *download* da interface de linha de comandos da *DigitalOcean*.

Contrariamente à implantação na *Google*, à data da realização deste projeto, a *Atlassian* não disponibiliza uma imagem da *DigitalOcean* (Properdesign, 2019), impossibilitando a utilização dos comandos necessários para a configuração sem efetuar o *download* da interface descrita. Posteriormente são efetuados o *login* e a configuração do *cluster* onde se pretende implantar. Finalizando com a implantação da nova imagem criada no ambiente pretendido e com a execução do *smoke test*. Este visa confirmar se a implantação é executada com sucesso e se encontra disponível para os utilizadores.

8.2 Kubernetes

Neste subcapítulo será descrito detalhadamente o processo de configuração efetuado no *Kubernetes* e na *Google Cloud* que possibilitou a implantação da solução desenvolvida em ambientes *cloud*.

Com o intuito de proporcionar um ambiente de testes para a solução, foi necessária a criação do cluster *Kubernetes* e efetuar a gestão dos recursos alocados a cada microsserviço no mesmo. Ao longo do estudo e preparação desta implantação foram seguidas boas práticas apresentadas pela *Google* (Qwiklabs, 2020) e *Kubernetes* (*Configuration Best Practices*, 2020), e apresentadas implantações diferentes à empresa Flowinn, uma vez que esta irá suportar os custos de manutenção da solução desenvolvida. Para atingir um valor económico suportável pela empresa, foi necessária

a redução de nodes a utilizar no cluster e a redução e condicionamento de recursos disponíveis para cada microsserviço. Como consequência desta restrição, posteriormente, quer a escalabilidade quer o desempenho da solução serão condicionados.

8.2.1 Implantação na *Google*

De modo a facilitar a leitura do diagrama, a implantação será divida em dois. O primeiro é de alta granularidade, com o intuito de representar o fluxo de comunicação entre os microsserviços implantados e o segundo pretende complementar a informação com um maior nível de detalhe.

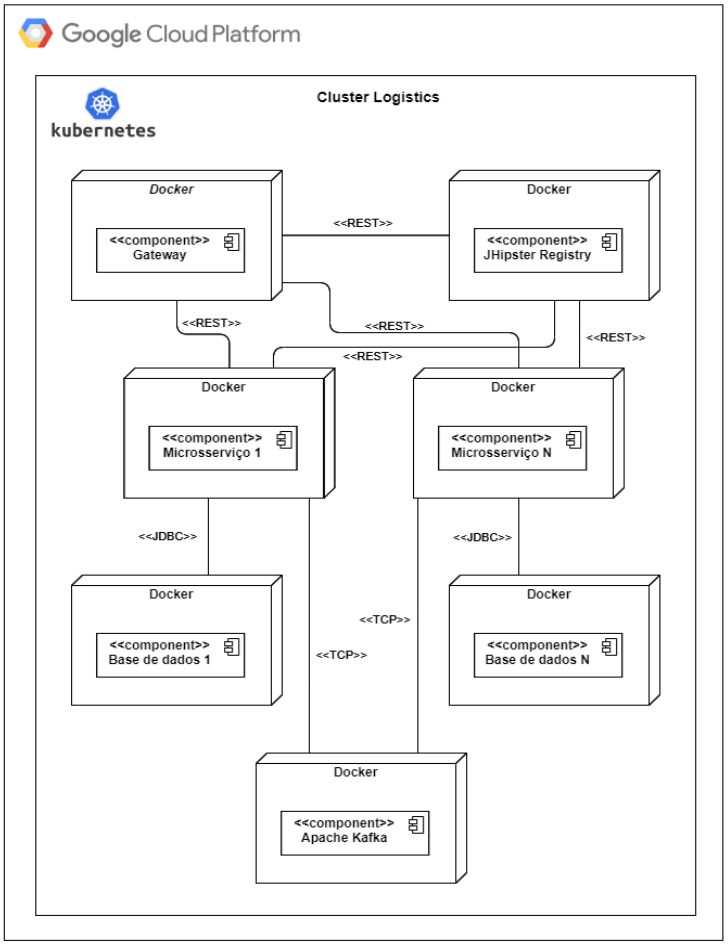

Figura 33 – Diagrama de implantação de alta granularidade

Como se pode observar, cada componente é executado com recurso à sua imagem *Docker* armazenada em repositórios distintos do *Docker Hub*. Nesta solução as comunicações serão efetuadas através dos padrões REST, para comunicações síncronas, e TCP (*Transmission Control Protocol)* para as assíncronas. Por sua vez, as comunicações

entre os microsserviços e as suas respetivas bases de dados são efetuadas através de JDBC (*Java Database Connectivity*).

Por fim, a implantação da solução desenvolvida será ilustrada no Figura 34.

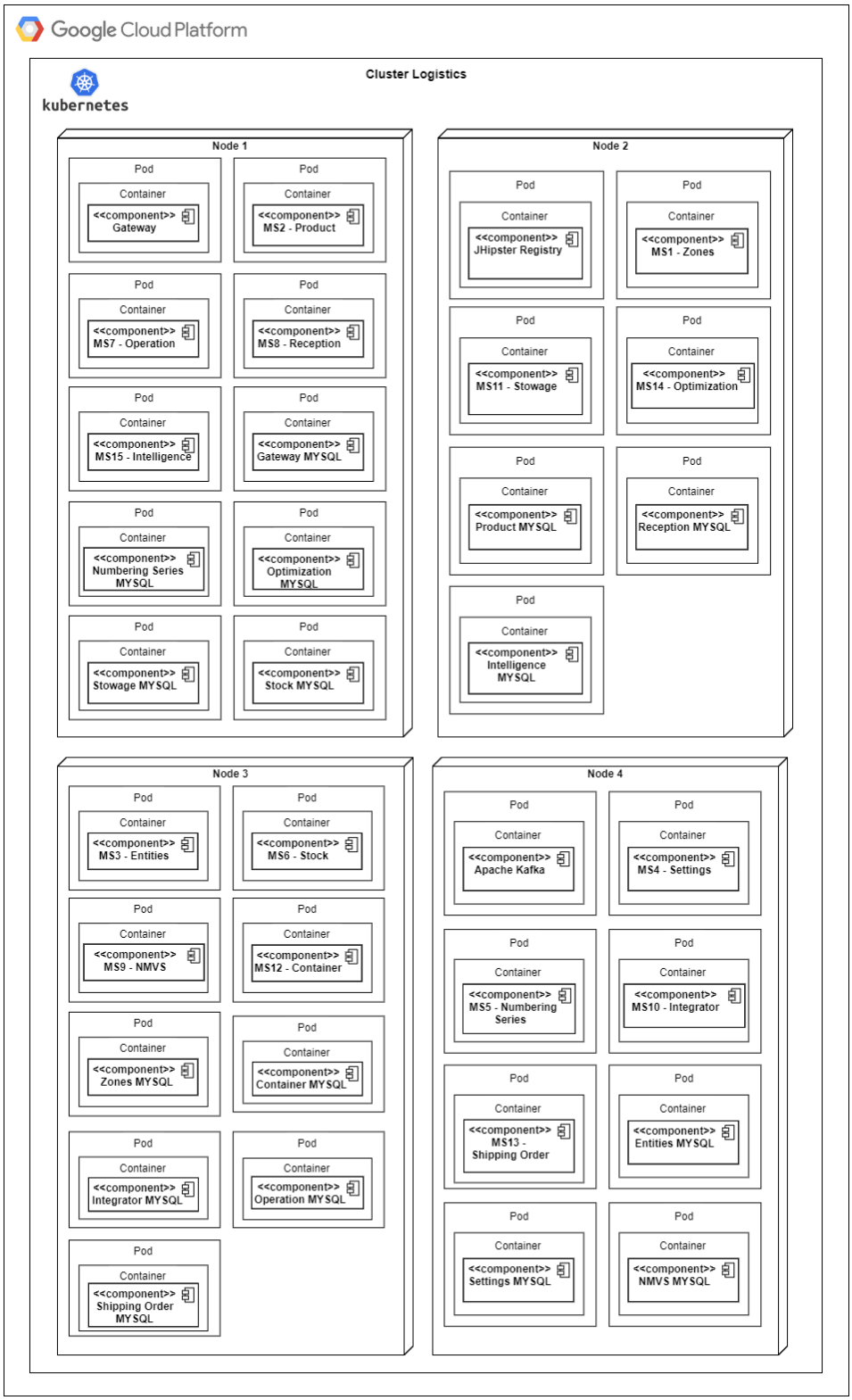

Figura 34 – Diagrama de implantação na *Google Cloud Platform*

Como é possível observar, a implantação resulta no *cluster Kubernetes*, Logistics, implantado na *Google Cloud Platform*. Este *cluster* foi segmentado em diferentes

namespaces, permitindo a implantação de diferentes ambientes, nomeadamente desenvolvimento e produção.

Este *cluster* é constituído por 4 *nodes*. Cada *node* representa uma máquina virtual do tipo N1, constituída por 2 vCPU's (unidade de processamento em máquinas virtuais) e 7.5 GB (*Gigabyte*) de RAM.

Estes são responsáveis por assegurar o funcionamento dos *pods* ilustrados no digrama, desde a disponibilização, acessibilidade, escalabilidade e tolerância às falhas. Após a configuração dos *nodes*, iniciou-se a configuração e implantação dos respetivos *pods*, importante referir que cada *pod* é constituído por um *container Docker.* Esta configuração foi efetuada através de ficheiros na linguagem *yaml*, como é possível observar no exemplo descrito no anexo A.3, com exceção do *pod Apache Kafka*. Este último, foi implantado com o auxílio da ferramenta *Helm*, descrita no subcapítulo 2.13.6.

Na implantação efetuada foram apenas expostos dois componentes, o *Gateway* e o *JHipster Registry*. O primeiro como descrito anteriormente, será o ponto de entrada dos utilizadores na aplicação, o segundo apenas será utilizado pela empresa Flowinn uma vez que apresenta informação detalhada sobre os microsserviços e os seus consumos.

Apesar da implantação estar em completo funcionamento na *Google Cloud Platform*, a avaliação da solução terá de ser efetuada na *DigitalOcean*, pelo motivo descrito no subcapítulo 8.1.

8.2.2 *DigitalOcean*

A implantação foi realizada através dos mesmos ficheiros *Kubernetes*, ilustrados no anexo A.3, uma vez que este é independente do provedor de serviços *cloud*.

A *DigitalOcean* apresenta máquinas virtuais de diferentes especificações, tornando inevitável a mudança dos *nodes* a utilizar. Através da observação do diagrama de baixa granularidade representado na Figura 35 é possível observar a nova implantação efetuada. O diagrama de alta granularidade, mantém-se igual, com exceção de agora se encontrar implantado na *DigitalOcean*.

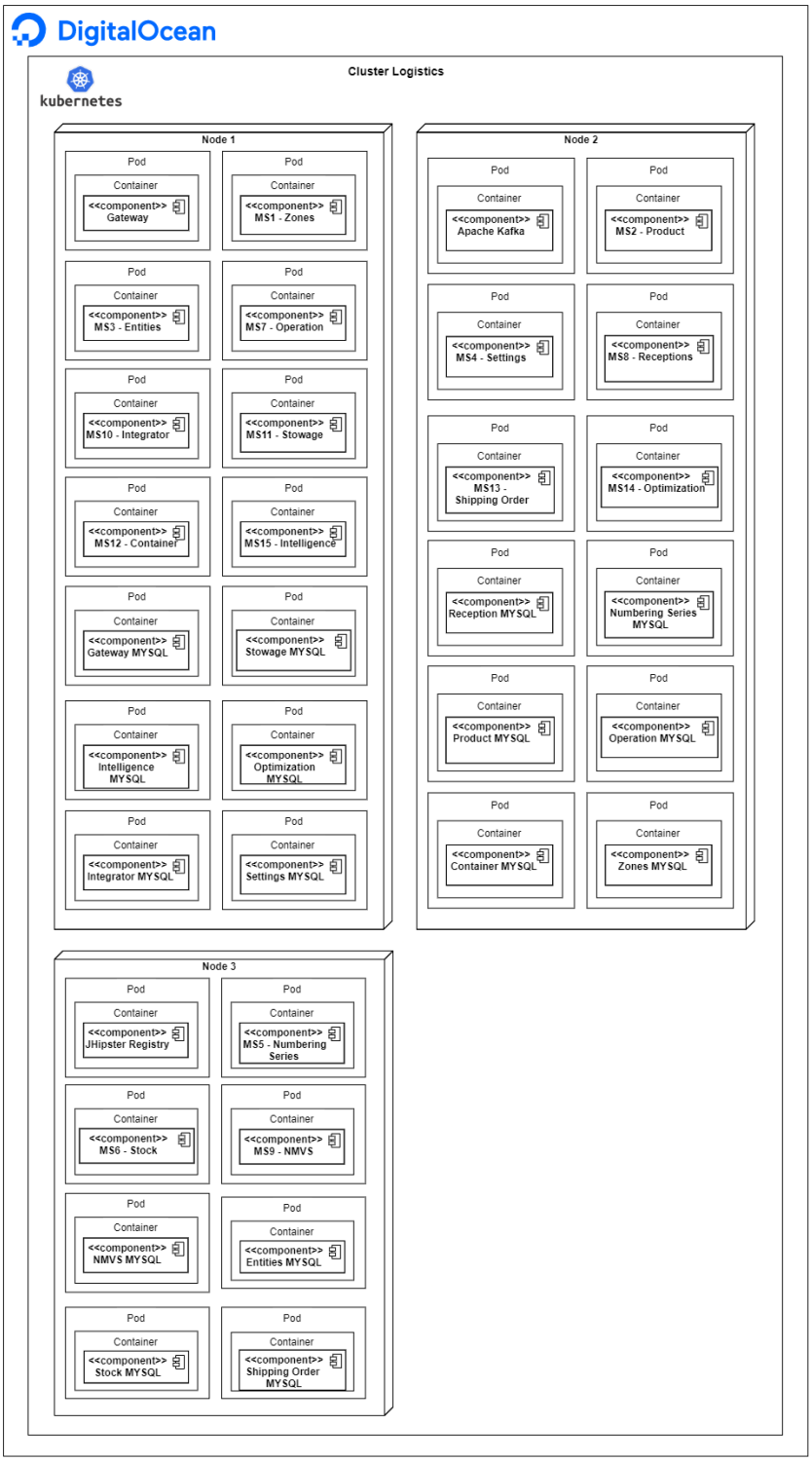

Figura 35 – Diagrama de implantação na *DigitalOcean*

Similarmente à implantação na *Google*, esta resulta no *cluster Kubernetes*, Logistics, segmentado por *namespaces.*

Atualmente, este *cluster* é constituído por três *nodes* de diferentes especificações. Os *nodes* 1 e 2 ilustrados na Figura 35, representam uma máquina virtual constituída por 4 vCPU's e 8 GB de RAM cada. O node 3 representa uma máquina virtual diferente, constituída por 1 vCPU's e 2 GB de RAM.

A escolha de diferentes máquinas surge porque na DigitalOcean, quanto maior for a capacidade do *node*, mais recursos ficam disponíveis para a implantação, ou seja, a utilização de dois nodes idênticos ao node 3, ofereceria menos recursos que a utilização do node 1 ou 2.

Os componentes expostos desta implantação são o *Gateway* e o *JHipster Registry*, pelos motivos previamente descritos na implantação anterior.

Por fim, é de realçar que esta implantação na plataforma *DigitalOcean* foi relativamente simples, uma vez que foram adquiridos os conhecimentos necessários com trabalho efetuado com a *Google Cloud Platform*.

9 Avaliação

A avaliação da solução desenvolvida é imprescindível, pois permite avaliar se os objetivos inicialmente propostos são atingidos e os problemas ultrapassados. De modo a efetuar um correto processo de avaliação, primeiramente, deve-se efetuar o seu plano. Este consiste na definição das métricas e metodologias que permitem a avaliação das mesmas. Assim, este capítulo destina-se à definição e execução do plano de avaliação da solução.

9.1 Métricas

Através da análise do problema e objetivos definidos no capítulo 1, foram identificadas métricas que devem ser utilizadas, de modo a que seja possível aferir as vantagens e desvantagens da implementação da nova solução comparativamente ao produto atual, Logistics.

9.1.1 Desempenho e escalabilidade

O **desempenho** é um dos fatores que deve estar melhorado na nova solução, resultante da migração descrita neste trabalho. Esta métrica é mensurável, uma vez que à data, existem ferramentas gratuitas que permitem avaliar aplicações, nomeadamente o *JMeter* descrito no subcapítulo 2.15. Esta ferramenta permite automatizar o processo de solicitação de funcionalidades das aplicações, permitindo assim, verificar o comportamento das mesmas em diversos cenários, tais como, grande quantidade de pedidos recebidos.

Para avaliar esta métrica foi necessário efetuar os mesmos testes às duas soluções em circunstâncias idênticas, de modo a obter resultados válidos e fidedignos.

Antes de iniciar a avaliação, foi necessário replicar o ambiente de produção da aplicação monolítica, descrito no subcapítulo 6.1.3, numa máquina idêntica de testes da empresa Flowinn. Relativamente a solução baseada em microsserviços foi avaliada a implantação na *DigitalOcean* descrita no subcapítulo 8.2.2.

As métricas definidas para avaliar o desempenho e escalabilidade das soluções têm por base o tempo de resposta ao utilizador e a duração da execução do caso de teste.

Através da interface gráfica do *JMeter* foram configurados os pedidos correspondentes aos casos de teste e as variáveis de execução, tais como, o intervalo de tempo para a realização dos pedidos e a quantidade dos mesmos. Para o primeiro foi definido o valor fixo de dez segundos enquanto que o segundo varia conforme o caso de teste pretendido.

De modo comparar as duas soluções foram testados os diversos processos do armazém e sincronização.

Nos casos de teste onde possa existir impacto de latência da rede, foi repetido o processo cinco vezes e posteriormente calculou-se a média dos resultados.

- Receção

Como descrito no subcapítulo 7.3, o processo de receção é dos mais importantes do armazém. No ambiente de produção, este processo é o mais utilizado e apresenta receções de grande volume de dados. De modo simular o comportamento das aplicações perante um cenário idêntico ao descrito foram realizados vários testes às mesmas, variando o número de linhas constituintes da receção entre 50 e 5000.

Inicialmente, testou-se o processo de criação das linhas de receção obtendo os resultados detalhados no anexo A.4.1. para posteriormente se realizar a média dos mesmos. Os valores obtidos estão apresentados no seguinte gráfico.

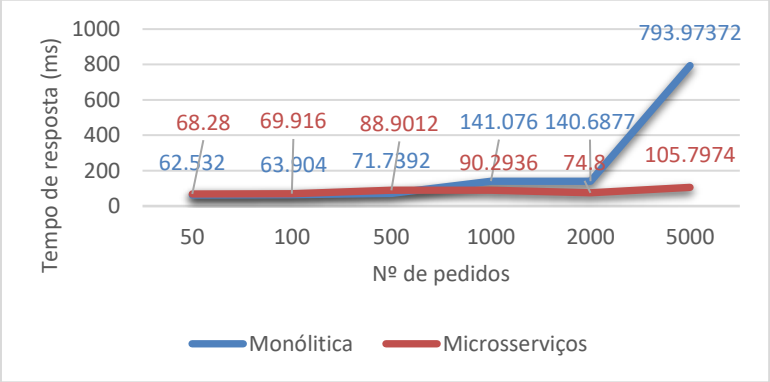

Figura 36 – Tempo médio de resposta na criação de linhas de receção

Através da análise dos resultados pode concluir-se que a aplicação monolítica demonstrou melhor desempenho perante um reduzido número de linhas, nomeadamente 50, 100 e 500 pedidos. Porém para números mais elevados, 1000, 2000 e 5000 pedidos, a aplicação baseada em microsserviços, apresentou melhores tempos de resposta, estes devem-se ao comportamento de escalabilidade automática do *Kubernetes*. Após o aumento significativo do número de pedidos, o *Kubernetes* atribui mais recursos de processamento e memória ao microsserviço responsável pelas receções, MS8 – *Receptions*, resultando na diminuição do tempo de resposta ao utilizador.

Assim conclui-se que, para receções com elevado número de linhas a aplicação baseada em microsserviços é significativamente mais rápida.

Posteriormente à receção de todos os produtos é necessário finalizar as respetivas receções, despoletando o processo ilustrado na Figura 25 do subcapítulo 7.3.

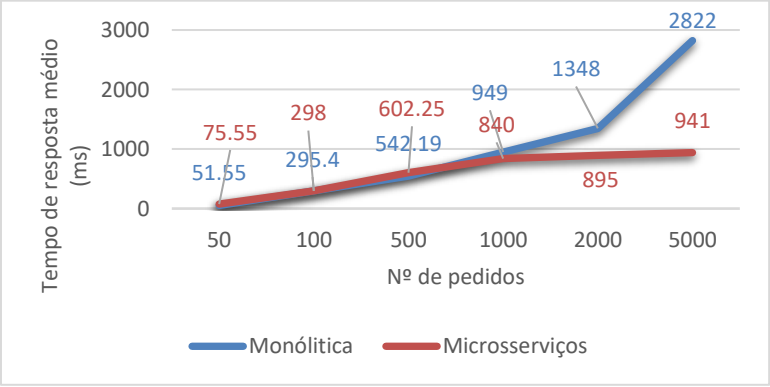

Figura 37 – Tempo de resposta médio na finalização das receções

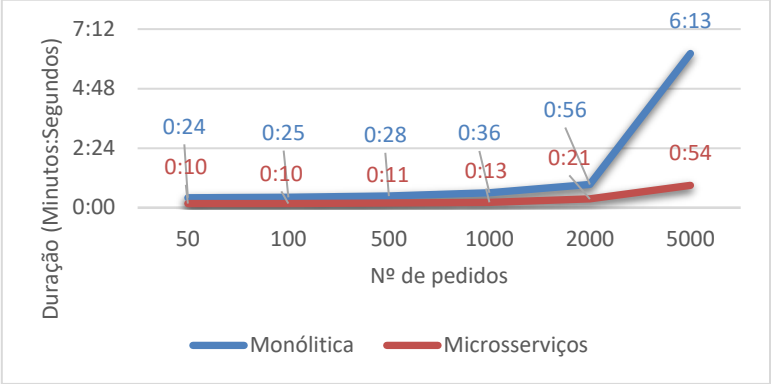

Figura 38 – Duração do processo de finalização das receções

Como é possível observar no gráfico Figura 37, similarmente à criação das linhas de receção, para o número de 50, 100 e 500, a aplicação monolítica respondeu em menor espaço de tempo comparativamente a aplicação baseada em microsserviços, contudo para os testes com maior número de pedidos, a aplicação monolítica demora significativamente mais tempo a responder ao utilizador.

Após receber os pedidos, as aplicações executam o processo de finalização da receção, este processo deve ser realizado no menor tempo possível pois a sua conclusão é fulcral para manter a coerência entre o *stock* de produtos no armazém e para a continuação dos restantes processos do mesmo. Assim, optou-se pela recolha da duração dos processos em ambas as soluções, representada no gráfico Figura 38. A solução baseada em microsserviços apresenta uma melhoria significativa no tempo de duração do processo, demostrando que a utilização da comunicação assíncrona implementada ao longo dos microsserviços através da utilização de eventos com recurso ao MB *Apache Kafka*, beneficia o processo, apresentando uma redução da duração relativamente à comunicação síncrona atualmente em uso na aplicação monolítica.

- Arrumação

Este processo foi avaliado com base no mesmo método utilizado na finalização da receção, uma vez que para a realização deste processo é importante saber o tempo de resposta da aplicação ao utilizador e o intervalo de tempo para o processo ser concluído. Este processo, similarmente às receções, reflete a movimentação do *stock* de produtos no armazém, exigindo assim a coerência da informação presente na aplicação e no armazém do cliente, possibilitando o processo posterior, a expedição dos produtos.

Assim, foram efetuados os testes ao processo, simulando a arrumação de 50, 100, 500 produtos, através da realização dos pedidos às aplicações. Contrariamente ao processo de receção, a arrumação não é solicitada com tanta regularidade pelos funcionários do

armazém, sendo assim aceitável a redução do número máximo de pedidos para 500 invés de 5000.

Os resultados obtidos após a realização dos testes descritos são ilustrados nos seguintes gráficos.

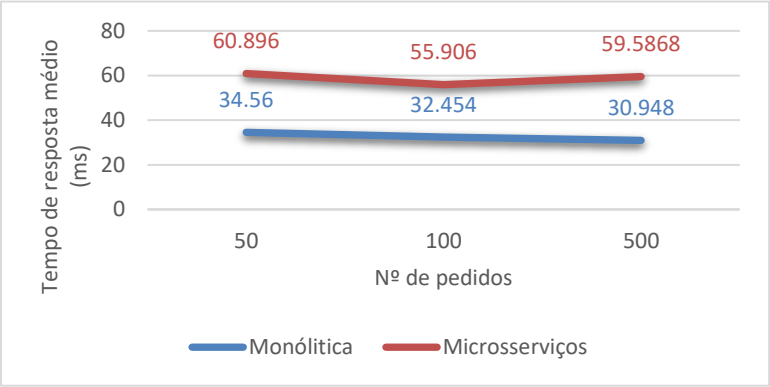

Figura 39 – Tempo de resposta médio do processo de arrumação

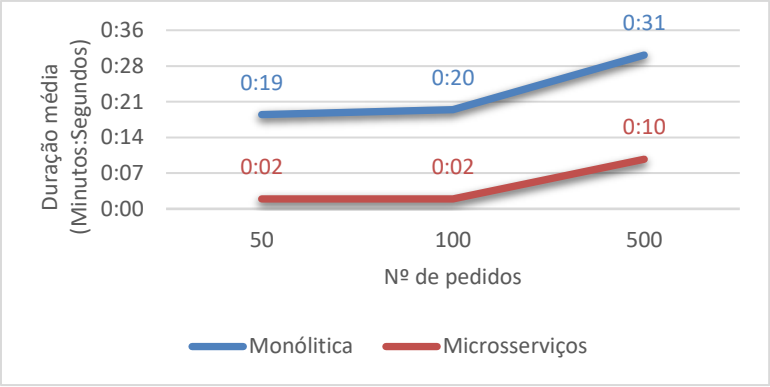

Figura 40 – Duração média do processo de arrumação

Através dos gráficos apresentados pode concluir-se que o tempo de resposta da aplicação monolítica é inferior à aplicação baseada em microsserviços em todos os casos de testes, contudo o mesmo não acontece com a média da duração do processo em cada caso de testes. Esta, na aplicação baseada em microsserviços é significativamente mais reduzida que na aplicação monolítica. Esta redução, deve-se à alteração efetuada ao processo, uma vez que na nova solução são utilizados eventos para dar seguimento ao processo, contrariamente ao processo na aplicação monolítica que recorre ao agendamento da tarefa que verifica se existem novos processos para iniciar.

Por fim, serão descritos os testes realizados ao último processo do armazém, a expedição.

- Expedição

Atualmente, o processo de expedição, similarmente à arrumação, não é frequentemente utilizado pelos funcionários do armazém, sendo assim necessário realizar casos de testes para um processo de expedição com 50, 100 e 500 linhas. Estes valores visam simular o comportamento das aplicações no ambiente de produção, permitindo identificar se a nova solução trará melhorias imediatas à empresa Flowinn.

Como é possível verificar na Figura 28 do subcapítulo 7.3, o processo de expedição inicia-se sem a ação do utilizador, assim foi recolhida a duração do processo para cada caso de teste. Este procedimento foi repetido 5 vezes, tendo-se calculado por fim a média da duração como observar nos seguintes parágrafos e no anexo A.4.1.

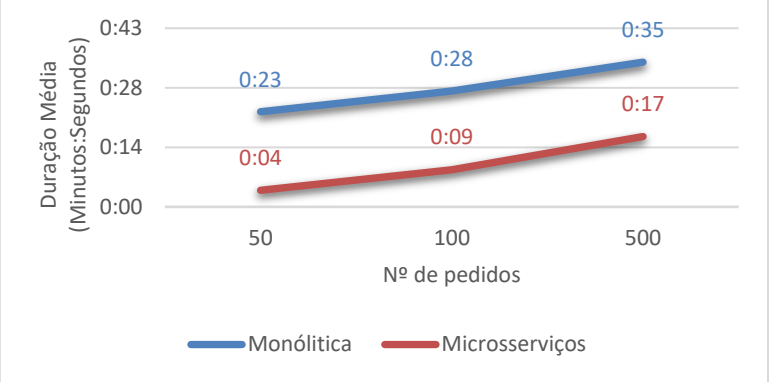

Figura 41 – Duração média do processo de expedição

Similarmente aos resultados obtidos nos processos anteriores, a aplicação baseada em microsserviços demostrou melhor desempenho, reduzindo significativamente o tempo de duração de cada processo de expedição comparativamente aos demostrados pela aplicação monolítica.

Concluídos os testes aos processos do armazém, seguiu-se com os dos processos de sincronização, nomeadamente a integração dos produtos e das entidades.

De modo a aproximar os testes realizados com o cenário presente em produção, foi utilizado o ERP do cliente para sincronizar a informação nas aplicações.

- Entidades

Neste processo de sincronização são realizados quatro pedidos ao ERP da empresa onde a aplicação está instalada, solicitando a informação sobre os clientes, fornecedores e os pontos de entrega de cada um dos anteriores. No total foram retornados 2169 entidades e 17 pontos de entrega.

Os testes efetuados visam obter informação sobre as duas métricas analisadas anteriormente, tempo de resposta ao utilizador e duração do processo de sincronização. Assim, este processo foi executado dez vezes em cada aplicação. Os resultados obtidos são apresentados na Tabela 7.

Tabela 7 – Resultados do processo de sincronização das entidades

Monolítica	**Tempo de resposta médio**	02:47 Minutos
	Duração média	02:47 Minutos
Microsserviços	**Tempo de resposta médio**	72.744 ms
	Duração média	01:49 Minutos

Como é possível observar, a implantação da nova solução nos clientes da empresa trará uma melhoria significativa nas métricas definidas. O tempo de resposta é significativamente reduzido, pois a nova solução responde ao utilizador de imediato, informando-o que os dados pedidos serão sincronizados, ao invés de esperar o término do processo para informar o utilizador. Esta alteração, permite ao utilizador continuar a usufruir das restantes funcionalidades da aplicação contrariamente ao que acontece atualmente na aplicação monolítica. Relativamente à duração do processo, na aplicação baseada em microsserviços, é menor, uma vez que, como descrito na Figura 30 do subcapítulo 7.4, após a conclusão de cada pedido é publicado um evento no MB, permitindo ao MS3 – *Entities* armazenar a informação recebida enquanto os restantes pedidos são executados pelo MS10 – *Integrator*, contrariamente ao processo sequencial executado na aplicação monolítica.

- Produtos

Contrariamente ao processo de entidades, este é constituído apenas por um pedido ao ERP, responsável por devolver a informação dos produtos e código de barras. No total este pedido devolve 2054 produtos e 1372 códigos de barras.

Para efetuar os testes ao processo foram utilizadas as mesmas métricas e forma de execução do processo anterior, repetindo o processo dez vezes e calculando a média dos valores obtidos.

Os resultados obtidos são apresentados na seguinte tabela.

Tabela 8 – Resultados do processo de sincronização dos produtos

Monolítica	**Tempo de resposta médio**	02:27 Minutos
	Duração média	02:27 Minutos
Microsserviços	**Tempo de resposta médio**	75.044 ms

	Duração média	02:21 Minutos

Como é possível observar o tempo de resposta médio é significativamente mais baixo, pelo mesmo motivo descrito no processo de sincronização das entidades. Relativamente à duração média, a aplicação baseada em microsserviços apresenta também melhores resultados, porém não tão expressivos uma vez que este processo é constituído apenas por um pedido ao ERP.

Por fim, é possível concluir que a aplicação baseada em microsserviços apresenta melhor desempenho e escalabilidade em relação à aplicação monolítica, uma vez que apresenta resultados mais favoráveis na maioria dos processos testados. Assim, quando a aplicação for disponibilizada para o cliente, estas melhorias serão imediatamente alcançadas.

Contudo, para atingir os valores apresentados pela aplicação monolítica, nos casos de menor carga de pedidos, poderá ser configurada de forma diferente. Por exemplo, uma das soluções possíveis seria atribuir uma quantidade de recursos maior aquando a sua implantação.

9.1.2 Manutenibilidade, automatização e satisfação

Na avaliação das métricas, manutenibilidade, automatização e satisfação foi utilizado o modelo de avaliação QEF *(Quality evaluation framework)*, ilustrado no anexo A.4.2. Este é constituído por três dimensões, que agrupam os diversos fatores a avaliar. De modo a proceder à avaliação de cada fator, foi necessário definir os requisitos, o seu peso e a forma de avaliação.

- Manutenibilidade

Esta métrica agrupa dois fatores, a **documentação** e a **monitorização**. Os requisitos de cada fator e os resultados obtidos na sua avaliação, serão apresentados nos seguintes parágrafos.

Tabela 9 – Avaliação do fator documentação

Objetivo	Avaliação da métrica	Forma de Avaliação	Escala	Resultados Obtidos
Análise e criação de requisitos	Visualização dos requisitos	Deve ser possível verificar os requisitos da solução	0 ou 100 %	100%
Documentar os componentes	Documentação dos componentes	Deve existir documentação desenvolvida através do *Swagger*, para cada componente	0 a 100 %	100%

desenvolvidos		$\frac{Nº\ de\ componentes\ documentados}{Nº\ total\ de\ componentes} x100$		
Documentar a implantação efetuada	Documento detalhado da implantação	Deve existir um documento explicativo da implantação efetuada	0 ou 100 %	100%
Documentar a nova arquitetura	Documento detalho da nova solução	Deve existir um documento detalhado da solução desenvolvida	0 ou 100 %	100%

Este fator visa avaliar a documentação da solução. Permite obter informação sobre as decisões tomadas ao longo do desenvolvimento da solução e perceber o seu funcionamento. Uma boa documentação permite reduzir a curva de aprendizagem do projeto para novos membros e facilitar o desenvolvimento de implementações futuras.

Como é possível observar na tabela acima, todos os objetivos foram concluídos com sucesso.

A avaliação desta métrica poderia incluir um inquérito, com o intuito de comparar a facilidade de aprendizagem e implementação de novas funcionalidades nas aplicações, monolítica e baseada em microsserviços. Seria expectável obter maior facilidade na nova solução, uma vez que apresenta documentação dos componentes desenvolvidos. Contudo, este não pode ser efetuado, uma vez que não existe um público alvo de programadores para inquerir.

Tabela 10 – Avaliação do fator monitorização

Objetivo	**Avaliação da métrica**	**Forma de Avaliação**	**Escala**	**Resultados Obtidos**
Visualizar os *logs* dos componentes	Cada componente deve conter um ficheiro de *logs*	$\frac{Nº\ de\ componentes\ que\ apresenta\ logs}{Nº\ total\ de\ componentes} x100$	0 a 100 %	100%
Visualizar o *CPU* utilizado	Visualização do *CPU* utilizado	$\frac{Nº\ de\ componentes\ que\ apresenta\ o\ CPU}{Nº\ total\ de\ componentes} x100$	0 a 100 %	100%
Visualizar a Memória utilizada	Visualização da memória utilizada	$\frac{Nº\ de\ componentes\ que\ apresenta\ a\ Memória}{Nº\ total\ de\ componentes} x\ 100$	0 a 100 %	100%
Visualizar a disponibilidade da solução	Visualização do estado e disponibilidade da aplicação	$\frac{Nº\ de\ componentes\ onde\ se\ verifica\ o\ estado}{Nº\ total\ de\ componentes} x\ 100$	0 a 100 %	100%
Controlar o tráfego da solução	Visualização da quantidade de pedidos e	$\frac{Nº\ de\ componentes\ que\ apresente\ o\ tráfego}{Nº\ total\ de\ componentes} x\ 100$	0 a 100 %	100%

	tráfego que o componente recebe			

O fator **monitorização** objetiva avaliar o controlo sobre a aplicação desenvolvida e o acesso ao histórico de execução da mesma, permitindo a identificação e resolução de erros em ambientes de desenvolvimento e produção. Estes, podem ser derivados de quebras de funcionalidade, falta de memória ou processamento, indisponibilidade da solução ou excesso de tráfego na aplicação. Como é possível observar na tabela acima, foi avaliada a existência de mecanismos de controlo dos possíveis erros descritos, tendo-se alcançado todos os objetivos propostos.

- Automatização

Esta métrica pretende avaliar o grau de automatização do processo de implantação desenvolvido ao longo deste trabalho. Visa agilizar as implantações eliminando os erros resultantes do processo manual de implantação atualmente praticado pela empresa Flowinn. Através desta, os programadores da empresa não necessitam de dispensar o seu tempo para realizar as implantações da aplicação.

A dimensão automatização é constituída apenas pelo fator **processo de implantação**, descrito posteriormente.

Tabela 11 – Avaliação do fator processo de implantação

Objetivo	**Avaliação da métrica**	**Forma de Avaliação**	**Escala**	**Resultados Obtidos**
Automatizar a implantação da solução	Componentes devem ser implantados através do *pipeline*	$\frac{Nº\ de\ microserviços\ implantados}{Nº\ total\ de\ componentes} x100$	0 a 100 %	100%
Automatizar o processo de execução de build	Execução da *build* do código-fonte da aplicação	Verificar a execução da *build* do repositório	0 ou 100 %	100%
Testar as funcionalidades desenvolvidas	Execução dos testes unitários	Verificar a execução dos testes unitários	0 ou 100 %	100 %
Testar a integração das funcionalidades desenvolvidas	Execução dos testes de integração	Verificar a execução dos executa testes integração	0 ou 100 %	100%
Assegurar a qualidade do	Análise da qualidade do código	Verificar se existe uma análise do código	0 ou 100 %	0%

código desenvolvido				
Publicar a imagem *Docker*	Publicação na imagem no respetivo repositório	Verificar a publicação da imagem no respetivo repositório do *Docker Hub*	0 ou 100 %	100%
Implantar a solução no ambiente	Implantação da solução efetuada através do pipeline	Verificar a implantação da solução	0 ou 100 %	100%
A implantação efetuada deverá ficar disponível para o utilizador	Aplicação disponível após a implantação	Verificar a execução do s*moke test* à implantação efetuada	0 ou 100 %	100%
Suportar os diversos ambientes	Implantação nos diversos ambientes	Verificar a implantação no ambiente de desenvolvimento e produção	0, 50 ou 100 %	100%

Como é possível observar na tabela, os objetivos foram atingidos todos na sua totalidade com exceção do objetivo "Assegurar a qualidade do código desenvolvido". Este, deverá ser considerado no futuro da aplicação, uma vez que permite garantir a qualidade do produto entregue ao cliente. Assim, a avaliação deste fator, através da aplicação do método QEF, resulta numa taxa de qualidade de 96.55%.

- Satisfação

Esta métrica visa avaliar a **satisfação da empresa** com a solução desenvolvida. Contrariamente às métricas anteriores, esta não é mensurável. Assim para efetuar a avaliação dos objetivos ilustrados no QEF, foi efetuado o questionário ilustrado no anexo A.4.2 e entregue o mesmo aos programadores da empresa integrantes do projeto relativo à aplicação monolítica, no entanto o seu número é irrisório para que possam ser obtidas conclusões, pois só existem dois. De qualquer forma o questionário foi realizado. Para uma dimensão de programadores mais elevada o questionário seria o mesmo, no entanto provavelmente os resultados obtidos teriam um maior nível de credibilidade.

O questionário foi desenvolvido através do *Google Forms* e é constituído por seis questões de escolha múltipla e resposta única.

Tabela 12 – Objetivos e questões

Objetivo	Questão

Melhor desempenho e escalabilidade	"A aplicação baseada em microsserviços apresenta um melhor desempenho e escalabilidade do que a aplicação monolítica."
Simplificar a implementação de novas funcionalidades	A implementação de novas funcionalidades é simplificada na aplicação baseada em microsserviços.
Otimização dos processos do armazém	"A utilização do *message broker*, *Apache Kafka*, permite simplificar e otimizar os processos anteriormente implementados através de ações periódicas, nomeadamente a movimentação de stock."
Agilizar a entrega das funcionalidades desenvolvidas	"A automatização do processo de implantação permite agilizar a entrega das funcionalidades desenvolvidas."
Redução de erros na implantação da solução	"A automatização do processo de implantação permite reduzir possíveis erros dos processos manuais, aumentado a fiabilidade do mesmo."
Evolução arquitetural apresenta benefícios para a empresa	"A evolução para uma arquitetura baseada em microsserviços tornou-se um benefício para os desenvolvedores bem como para a empresa Flowinn."

Posteriormente à entrega do questionário, não foi possível obter um público alvo extenso, pois o projeto na empresa Flowinn é desenvolvido pelo CTO da mesma e pelo autor deste trabalho (os dois programadores anteriormente referidos). Assim, apenas a resposta do CTO ao questionário foi considerada válida.

As questões ilustradas na Tabela 12 são constituídas por cinco possíveis respostas. Estas foram classificadas com percentagem de cumprimento do objetivo, como ilustrado na seguinte tabela, permitindo efetuar a avaliação dos objetivos no QEF.

Tabela 13 – Classificação das respostas

Possível resposta	Percentagem de cumprimento do objetivo
Discordo totalmente	0%
Discordo parcialmente	25%
Indiferente	50%
Concordo parcialmente	75%
Concordo totalmente	100%

Seguidamente à classificação das perguntas foi possível avaliar os objetivos propostos, através do cálculo da média das respostas obtidas no questionário, ilustradas em forma de gráfico no anexo A.4.2, obtendo-se os seguintes resultados.

Tabela 14 – Avaliação do fator satisfação da empresa

Objetivo	Resultados Obtidos
Melhor desempenho e escalabilidade	100%
Simplificar a implementação de novas funcionalidades	75%
Otimização dos processos do armazém	100%
Agilizar a entrega das funcionalidades desenvolvidas	100%
Redução de erros na implantação da solução	100%
Evolução arquitetural apresenta benefícios para a empresa	100%

Como é possível verificar, os objetivos foram totalmente atingidos com exceção da simplificação da implementação de novas funcionalidades, obtendo-se uma taxa de qualidade de 96.30% no fator de satisfação da empresa.

10 Conclusão

O avanço tecnológico e a elevada exigência do mercado, provocam uma forte necessidade de obtenção de soluções com elevado desempenho e eficácia, o que tornou as aplicações baseadas em microsserviços cada vez mais comuns no quotidiano das empresas.

Com este trabalho pretendeu efetuar-se uma migração arquitetural na aplicação Logistics da empresa Flowinn, decompondo o monólito em diversos microsserviços e utilizando padrões comuns em arquiteturas baseadas em microsserviços.

Como descrito na secção 1.3, o objetivo principal deste projeto passava por implementar uma arquitetura baseada em microsserviços que permitisse obter melhorias no desempenho dos processos do quotidiano dos armazéns do cliente. Como demostrado na avaliação das soluções, a nova arquitetura, através da utilização dos microsserviços e o MB, permitiu atingiu este objetivo, uma vez que apresentou melhores resultados nos testes efetuados.

Os objetivos, implementação de testes automatizados e automatização do processo de implementação, foram também atingidos. Os primeiros permitem assegurar que a aplicação e as suas funcionalidades se encontram operacionais e o segundo permitiu agilizar a implantação da solução, tornado o processo repetível e confiável. Assim, foi possível eliminar os possíveis erros manuais do processo anteriormente praticado na empresa.

Porém, o último objetivo proposto, comunicação com o sistema de autenticação da empresa, não foi atingido. Este facto, deveu-se à alteração de requisitos por parte da empresa, após o início do projeto. A aplicação monolítica, Logistics, é um produto isolado comercializado aos clientes, contudo pretendia iniciar-se a integração da

mesma com os diversos produtos da empresa. Após idealizar a nova solução, aplicação baseada em microsserviços, a empresa optou por manter o produto isolado, excluindo a comunicação com o serviço de autenticação. Assim, esta apresenta uma autenticação JWT (*Json Web Token*) que consiste na validação do *token* de autenticação do utilizador.

Posteriormente, a única limitação apresentada pela empresa foi a redução dos custos da implantação na *Google Cloud Platform*. Após a redução dos recursos de cada microsserviço, foi possível reduzir os mesmos, ainda assim, a empresa optou por trocar de provedor de serviços. Portanto, a aplicação baseada em microsserviços foi implantada na *DigitalOcean*, com uma redução de recursos.

Finalizando, pode concluir-se, observando o capítulo 9, que apesar da complexidade inerente à migração de aplicações monolíticas, o projeto desenvolvido atingiu os principais objetivos pretendidos e a satisfação da empresa.

Regressando à hipótese de investigação, pode observar-se que, de facto há vantagens na transformação da aplicação Logistics de uma arquitetura monolítica, para uma arquitetura baseada em microsserviços, sendo que os objetivos da secção 1.3 foram atingidos e os resultados esperados da secção 1.4 obtidos.

10.1 Trabalho futuro

Com o possível crescimento da solução devido à constante demanda por novas funcionalidades, futuramente seria benéfico mensurar a qualidade e segurança do código entregue em cada implementação da solução. Esta medição pode ser efetuada de forma automática, através da utilização de ferramentas como o *SonarQube* e *FindBugs*.

Em conjunto, futuramente deverá ser aprimorado o desenvolvimento de testes automáticos nomeadamente os testes unitários, testes de integração e testes de usabilidade. Estes, poderão ser desenvolvidos com recurso a ferramentas distintas. O *front-end* poderá ser testado através de *framework*s como *Protractor*, *Jasmine* e *Karma* enquanto que o *back-end* poderá ser testado através do *JUnit*.

A utilização destas ferramentas deverá ter como objetivo atingir um valor próximo da cobertura total do código desenvolvido.

Referências

Armon Dadgar. (2018, Junho 26). *Introduction to HashiCorp Consul*. https://www.youtube.com/watch?v=mxeMdl0KvBI&feature=emb_title

Atlassian. (2019, Julho 22). *Build, test, and deploy with Pipelines—Atlassian Documentation*. https://confluence.atlassian.com/bitbucket/build-test-and-deploy-with-pipelines-792496469.html

Aviran Mordo. (2006). *Journey from Monolith to Microservices and DevOps*. https://gotocon.com/amsterdam-2016/presentation/Journey%20from%20Monolith%20to%20Microservices%20and%20DevOps

Brajesh De. (2017). *API Management: An Architect's Guide to Developing and Managing*. Apress.

Cesar de la Torre, Bill Wagner, & Mike Rousos. (2017). *.NET Microservices: Architecture for Containerized .NET Applications*.

Chris Richardson. (2016a). *Microservices Pattern: Self Registration pattern*. microservices.io. http://microservices.io/patterns/self-registration.html

Chris Richardson. (2016b). *Microservices Pattern: Service registry pattern*. microservices.io. http://microservices.io/patterns/service-registry.html

Chris Richardson. (2018a). *Microservices Pattern: Database per service*. microservices.io. http://microservices.io/patterns/data/database-per-service.html

Chris Richardson. (2018b). *Microservices Pattern: Shared database*. microservices.io. http://microservices.io/patterns/data/shared-database.html

Chris Richardson. (2018c). *Microservices Patterns: With examples in Java*. Manning Publications.

CoMakeIT. (2018, Agosto). *A Quick Guide To Using Keycloak For Identity And Access Management*. https://www.comakeit.com/blog/quick-guide-using-keycloak-identity-access-management/

Configuration Best Practices. (2020, Julho 17). Kubernetes. https://kubernetes.io/docs/concepts/configuration/overview/

Daniel Viana. (2017, Janeiro 30). *O que é front-end e back-end?* Blog da TreinaWeb. https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end/

Dominic Betts, Julian Dominguez, Grigori Melnik, Mani Subramanian, Fernando Simonazzi, & Greg Young. (2013). *Exploring CQRS and Event Sourcing*. Microsoft patterns & practices.

Edson Yanaga. (2017). *Migrating to Microservice Databases*. O'Reilly Media, Inc. https://www.oreilly.com/library/view/migrating-to-microservice/9781492048824/ch04.html

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 359.

Fanis Despoudis. (2017, Agosto). *Understanding SOLID Principles: Single Responsibility*. Medium. https://codeburst.io/understanding-solid-principles-single-responsibility-b7c7ec0bf80

Intellipaat. (2019, Junho 25). AWS vs Azure vs Google—Detailed Cloud Comparison. *Intellipaat Blog*. https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/

Jakub Korab. (2017). *Understanding Message Brokers [Book]*. O'Reilly Media, Inc. https://www.oreilly.com/library/view/understanding-message-brokers/9781492049296/

Jez Humble, & David Farley. (2010). *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison Wesley.

JHipster. (2017a). *API Gateway*. https://www.jhipster.tech/api-gateway/

JHipster. (2017b). *Consul*. https://www.jhipster.tech/consul/

JMeter. (2020). *Apache JMeter—Apache JMeter™*. https://jmeter.apache.org/

John Ferguson Smart. (2011). *Jenkins: The Definitive Guide: Continuous Integration for the Masses.* O'Reilly Media.

John Piela. (2016, Abril 2). *Why Netflix Moved to a Microservices Architecture*. ProgrammableWeb. https://www.programmableweb.com/news/why-netflix-moved-to-microservices-architecture/elsewhere-web/2016/04/02

Kafka Architecture. (2017, Maio). http://cloudurable.com/blog/kafka-architecture/index.html
Koen, P. A., Ajamian, G. M., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S., Johnson, A., Puri, P., & Seibert, R. (2002). Effective Methods, Tools, and Techniques. *The PDMA ToolBook for New Product Development*, 32.
Kyle Brown. (2017, Fevereiro 13). Apply the Strangler Application pattern to microservices applications. *IBM Developer*. https://developer.ibm.com/technologies/microservices/articles/cl-strangler-application-pattern-microservices-apps-trs/
Martin Fowler. (2005, Dezembro). *Event Sourcing*. martinfowler.com. https://martinfowler.com/eaaDev/EventSourcing.html
Martin Fowler. (2006, Maio). *Continuous Integration*. martinfowler.com. https://martinfowler.com/articles/continuousIntegration.html
Michael J. Kavis. (2014). *Architecting the Cloud*. John Wiley & Sons Inc.
Netflix Technology Blog. (2012, Julho 9). *Embracing the Differences: Inside the Netflix API Redesign*. https://netflixtechblog.com/embracing-the-differences-inside-the-netflix-api-redesign-15fd8b3dc49d
Newman, S. (2019). *Monolith to Microservices*. O'Reilly Media.
O que é Docker? | Red Hat. (2018). https://www.redhat.com/pt-br/topics/containers/what-is-docker
Osterwalder, A., Pigneur, Y., & Clark, T. (2010). *Business model generation: A handbook for visionaries, game changers, and challengers*. Wiley.
Pieter Humphrey. (2017, Abril). *Understanding When to use RabbitMQ or Apache Kafka*. https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka
Properdesign. (2019, Agosto 27). Redeploying a Kubernetes service on DigitalOcean managed cluster in response to a build on Bitbucket Pipelines. *Proper Design*. https://properdesign.co.uk/redeploying-a-kubernetes-service-on-digitalocean-managed-cluster-in-response-to-a-build-on-bitbucket-pipelines/
Qwiklabs. (2020). *Qwiklabs—Hands-On Cloud Training*. Qwiklabs. https://run.qwiklabs.com/
Roy, G. M. (2017). *RabbitMQ in depth*. Manning Publications Co.
Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, *1*(1), 83. https://doi.org/10.1504/IJSSCI.2008.017590
Saleem, S. (2019, Março 27). *What Is DigitalOcean and Why Should You Select It for Your Web Hosting?* The Official Cloudways Blog. https://www.cloudways.com/blog/what-is-digital-ocean/
Samir Behara. (2018, Dezembro). *Monolith to Microservices With the Strangler Pattern—DZone Microservices*. Dzone.Com. https://dzone.com/articles/monolith-to-microservices-with-the-strangler-patte
Sayfan, G. (2017). *Mastering Kubernetes: Large scale container deployment and management*. Packt Publishing.
Sendil Kumar N, & Deepu K Sasidharan. (2018). *Full Stack Development with JHipster [Book]*. https://www.oreilly.com/library/view/full-stack-development/9781788476317/
Skalena. (2018). *WSO2 API Manager*. Skalena. https://www.skalena.com/wso2-apimanager?lang=en
Stephane Maarek. (2019, Março 26). *What is Zookeeper?* https://www.youtube.com/watch?v=AS5a91DOmks
Thomas Erl, Richardo Puttini, & Zaigham Mahmood. (2013). *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall.
Umesh Ram Sharma. (2017). *Practical Microservices*. Packt Publishing.
Vinicius Feitosa Pacheco. (2018). *Microservice Patterns and Best Practices*. Packt Publishing. https://www.oreilly.com/library/view/microservice-patterns-and/9781788474030/
Viraj Salaka. (2019, Agosto). *Service Discovery with WSO2 API Microgateway*. https://wso2.com/blogs/thesource/2019/08/service-discovery-with-wso2-api-microgateway/
What is Apache Kafka? | AWS. (2019). Amazon Web Services, Inc. https://aws.amazon.com/msk/what-is-kafka/
Woodall, T. (2003). *Conceptualising «Value for the Customer»: An Attributional, Structural and Dispositional Analysis*. 44.
WSO2. (2019). *Key Concepts—API Manager 2.6.0—WSO2 Documentation*. https://docs.wso2.com/display/AM260/Key+Concepts
Yoav Abrahami. (2015). *Scaling Wix to 60M Users—From Monolith to Microservices—Wix Tech Stack*. https://stackshare.io/wix/scaling-wix-to-60m-users-from-monolith-to-microservices

Yury Izrailevsky. (2016, Fevereiro 11). *Completing the Netflix Cloud Migration*. Netflix Media Center. https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration

A Anexos

A.1 Microsserviços

Os restantes microsserviços presentes na arquitetura baseada em microsserviços, serão descritos ao longo dos próximos parágrafos. Englobam o restante domínio da aplicação Logistics e possibilitam a realização dos processos presentes nos armazéns, nomeadamente receção, arrumação e expedição.

- MS3 – Entities

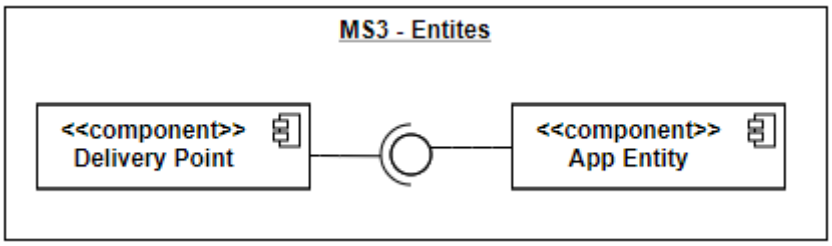

Figura 42 – Representação do microsserviço *Entities*

Este microsserviço representa o domínio dos clientes, os fornecedores do armazém e os respetivos pontos de entrega.

- MS4 – Settings

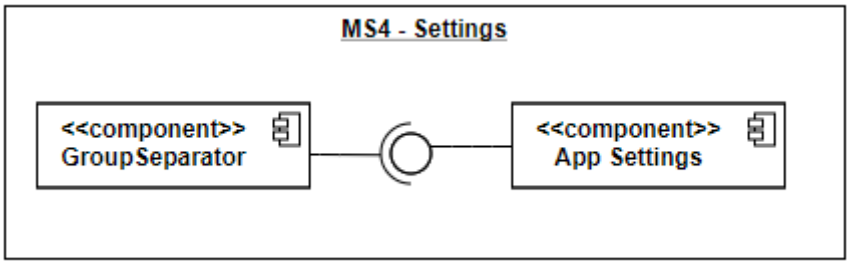

Figura 43 – Representação do microsserviço *Settings*

S*ettings*, microsserviço que engloba as configurações transversais à aplicação, necessárias para definir a forma de execução e as várias validações a ter em conta ao longo da aplicação. Este microsserviço será também utilizado nos diversos processos de armazém, uma vez que cada um destes poderá ser efetuado de forma distinta com base em alguma configuração.

- MS7 – Operation

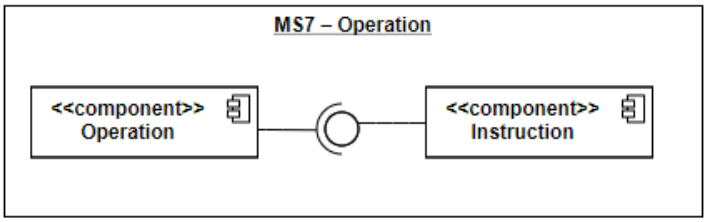

Figura 44 – Representação do microsserviço *Operation*

O microsserviço *Operation*, engloba o domínio das operações resultantes dos seguintes processos presentes no armazém, receção e arrumação. Cada um destes processos origina várias operações. Assim sendo, o microsserviço responsável pelo processo publicará um evento no MB, e este subscreverá o mesmo, criando as respetivas operações. Posteriormente, no processo de expedição, serão criadas e utilizadas as instruções responsáveis por auxiliar os funcionários dos armazéns.

- MS9 – *NMVS*

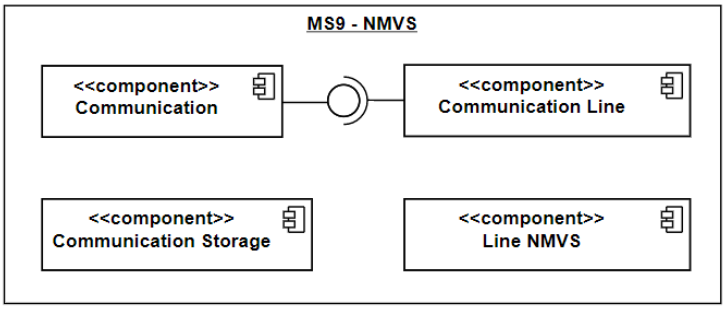

Figura 45 – Representação do microsserviço *NMVS*

Os clientes desta aplicação são do ramo farmacêutico e como tal, necessitam de verificar a autenticidade de certos produtos, que contêm dispositivos de segurança com o intuito de evitar a comercialização de produtos falsificados. Este microsserviço, *NMVS*, é responsável por comunicar os produtos descritos às entidades certificadoras e pela gestão da resposta das mesmas.

- MS11 – *Stowage*

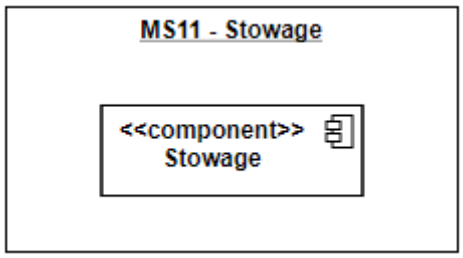

Figura 46 – Representação do microsserviço *Stowage*

Este microsserviço é responsável pela gestão e armazenamento dos processos de arrumação.

- MS12 – Container

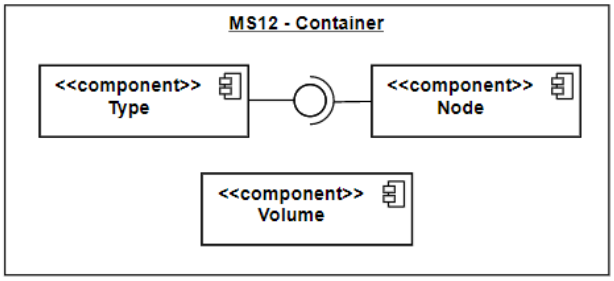

Figura 47 – Representação do microsserviço *Container*

O *Container* engloba os volumes e contentores utilizados no processo de expedição com o objetivo de gerir os produtos e embalagens a ser entregues nas encomendas efetuadas ao armazém.

- MS13 – Shipping Order

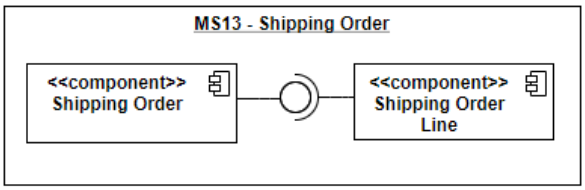

Figura 48 – Representação do microsserviço *Shipping Order*

Este microsserviço, *Shipping Order*, engloba o domínio das encomendas efetuadas ao armazém. Contém a gestão das ordens de expedição, que consistem nas encomendas efetuadas pelos clientes e os artigos com as respetivas quantidades requeridas.

Posteriormente à criação da encomenda e à estimativa de tempo até estar concluída, este dará origem aos processos de *picking* e posteriormente de *packing*.

Estes processos consistem na recolha do produto das diversas zonas no armazém e embalamento do mesmo para posteriormente ser entregue ao respetivo cliente. Estes processos num trabalho futuro poderão ser novos microsserviços que subscreverão o evento publicado aquando a conclusão da estimativa da encomenda, iniciando assim o seu processo e a respetiva logica de negócio do mesmo.

A.2 Bitbucket Pipelines e Jira

Aqui ilustra-se o código do *pipeline* descrito no subcapítulo 8.1 e detalha-se a integração efetuada com o produto *Jira*.

A.2.1 *Pipeline* definido

Figura 49 – *Steps* executados conforme o *branch*

Através da Figura 49, é possível observar que o *pipeline* executa de forma diferenciada conforme o *branch* em que é efetuado um *commit*. A implementação de novas funcionalidades deve ser efetuada em novos *branchs,* do tipo *feature* e aquando o seu termino, deverá ser criada uma *pull-request* para o *branch development*, desta forma, o *pipeline* apenas executará a *build* do repositório e executar os respetivos testes.

Seguidamente, a *build* do *pipeline,* nos *branches development* e *master,* diferem apenas na configuração do ambiente e na execução da implantação, deste modo os *steps* foram criados de forma genérica permitindo a reutilização dos mesmos.

- Configuração do ambiente da *build*

Figura 50 – Definição dos steps de configuração do ambiente

Como é possível observar, nos *steps* referidos é configurada a variável *DEPLOYMENT_ENVIROMENT* conforme o ambiente de execução. De modo a possibilitar a sua utilização nos *steps* futuros, esta é exportada para o artefacto *build.env*.

- Build

```yaml
definitions:
  buildSaveImage: &buildSaveImage
    - step:
        name: Build and Save image build
        script:
          - echo "Start build"
          - mvn -Pprod -B verify jib:dockerBuild
          - >-
            docker save --output build-image-logistics-zones-ms.docker
            $DEFAULT_IMAGE_TAG_NAME
          - echo "Build done and image saved"
        artifacts:
          - build-image-logistics-zones-ms.docker
        caches:
          - maven
          - docker
```

Figura 51 – Definição do *step build*

Como ilustrado na Figura 51, este *step* executa a *build* através do *Maven* e cria a imagem *Docker* com recurso ao *plugin jib*. Posteriormente, esta imagem é colocada como artefacto, possibilitando a sua reutilização nos *steps* seguintes. Cada microsserviço cria uma imagem de nome diferente, sendo este o nome do microsserviço em questão. Deste modo foi configurada uma variável em cada repositório que permite identificar o nome da imagem criada como é apresentado na seguinte tabela.

Tabela 15 – Variáveis utilizadas no *step build*

Variável	Utilização	Tipo de Variável
$DEFAULT_IMAGE_TAG_NAME	Nome da imagem criada	Configurada em cada repositório

- Docker Hub

```
saveDockerHubRepository: &saveDockerHubRepository
  script:
    - echo "Starting Docker hub"
    - source build.env
    - docker load --input ./build-image-logistics-zones-ms.docker
    - >-
      docker image tag $DEFAULT_IMAGE_TAG_NAME
      $DOCKER_HUB_USER/$BITBUCKET_REPO_SLUG-$DEPLOYMENT_ENVIRONMENT:$BITBUCKET_BUILD_NUMBER
    - docker login -u $DOCKER_HUB_USER -p $DOCKER_HUB_PASSWORD
    - >-
      docker push
      $DOCKER_HUB_USER/$BITBUCKET_REPO_SLUG-$DEPLOYMENT_ENVIRONMENT:$BITBUCKET_BUILD_NUMBER
    - echo "Finish push to docker hub"
```

Figura 52 – Definição do *step Docker Hub*

Este *step*, como se pode observar na figura, inicia-se pela importação da variável do ambiente da *build* e pela imagem criada pela mesma. Posteriormente, é atribuído uma *tag* à imagem, seguindo a nomenclatura descrita no subcapítulo 8.1. Finalmente são efetuados o *login* e a publicação da imagem. Este *step* apresenta as seguintes variáveis, ilustradas na tabela seguinte, com intuito de reutilizar o mesmo ao longo de cada microsserviço.

Tabela 16 – Variáveis utilizadas no *step Docker Hub*

Variável	Utilização	Tipo de Variável
$DOCKER_HUB_USER	Nome de utilizador da conta do *Docker Hub*	Configurada em cada repositório
$DOCKER_HUB_PASSWORD	*Token* de acesso à conta do *Docker Hub*	Configurada em cada repositório
$BITBUCKET_REPO_SLUG	Nome do repositório no *Bitbucket*	Existente no *Bitbucket Pipelines*
$DEPLOYMENT_ENVIRONMENT	Ambiente onde executa a *build*	Configurada no *step* configuração do ambiente da *build*
$BITBUCKET_BUILD_NUMBER	Identificador único com o número da *build* do *Bitbucket Pipelines*	Existente no *Bitbucket Pipelines*

- Google Cloud Platform e DigitalOcean

Como descrito no subcapítulo 8.2 a solução foi implantada na *Google Cloud Platform* e posteriormente na *DigitalOcean*, substituindo assim a utilização do *step deployTestesGoogle* pelo *deployTestesDigitalOcean* ilustrados na figura seguinte.

```yaml
deployTestesGoogle: &deployTestesGoogle
  - step:
      name: Deploy on TESTES
      deployment: Testes
      image: 'google/cloud-sdk:latest'
      <<: *deployKubernetesGKE

deployTestesDigitalOcean: &deployTestesDigitalOcean
  - step:
      name: Deploy on TESTES
      deployment: DigitalOcean
      image: 'atlassian/pipelines-kubectl'
      <<: *deployKubernetesDigitalOcean

deployProd: &deployProd
  - step:
      name: Deploy on PROD
      deployment: Production
      trigger: manual
      image: 'atlassian/pipelines-kubectl'
      <<: *deployKubernetesDigitalOcean
```

Figura 53 – *Step* de implantação da imagem no *Kubernetes*

A implantação da solução foi realizada com recurso ao *Bitbucket Deployments*. Este através da palavra-chave *deployment,* identifica o ambiente onde será efetuada a implantação. Assim permite configurar as variáveis conforme o ambiente, reutilizando o *script* de implantação, nomeadamente o *script deployKubernetesDigitalOcean*. Deste modo é também possível configurar o modo de desencadear o *deployment*, sendo automático para o ambiente de testes e manual para o ambiente de produção.

A implantação ocorre através do seguinte *script*.

```
deployKubernetesGKE: &deployKubernetesGKE
  script:
    - source build.env
    - echo deploy on $BITBUCKET_DEPLOYMENT_ENVIRONMENT
    - echo "Configure K8s Deployment"
    - echo $GCLOUD_API_KEYFILE > ~/.gcloud-api-key.json
    - gcloud auth activate-service-account --key-file ~/.gcloud-api-key.json
    - gcloud config set project $GCLOUD_PROJECT_ID
    - gcloud container clusters get-credentials $K8s_CLUSTER_NAME --zone=$K8s_CLUSTER_ZONE --project $GCLOUD_PROJECT_ID
    - gcloud auth configure-docker --quiet
    - echo "Finished Configuration"
    - echo "Starting K8s Deployment"
    - >-
      kubectl set image deployment $DEFAULT_IMAGE_TAG_NAME
      $DEFAULT_IMAGE_TAG_NAME-app=$DOCKER_HUB_USER/$BITBUCKET_REPO_SLUG-$DEPLOYMENT_ENVIRONMENT:$BITBUCKET_BUILD_NUMBER
      --record --namespace=$K8s_NAMESPACE
    - curl -vk $K8S_SERVICE_URL
    - echo "Finished K8s Deployment"

deployKubernetesDigitalOcean: &deployKubernetesDigitalOcean
  script:
    - source build.env
    - echo deploy on $BITBUCKET_DEPLOYMENT_ENVIRONMENT
    - echo "Download doctl"
    - apk --no-cache add curl
    - curl -sL https://github.com/digitalocean/doctl/releases
    /download/v$DOCTL_VERSION/doctl-$DOCTL_VERSION-linux-amd64.tar.gz | tar -xzv
    - mv ./doctl /usr/local/bin
    - echo "Download finished"
    - doctl -t $DOCTL_TOKEN k8s cluster kubeconfig show $K8s_CLUSTER_NAME > kubeconfig.yml
    - echo "Starting K8s Deployment"
    - >-
      kubectl --insecure-skip-tls-verify --kubeconfig=kubeconfig.yml
      set image deployment $DEFAULT_IMAGE_TAG_NAME
      $DEFAULT_IMAGE_TAG_NAME-app=$DOCKER_HUB_USER/$BITBUCKET_REPO_SLUG-$DEPLOYMENT_ENVIRONMENT:$BITBUCKET_BUILD_NUMBER
      --record --namespace=$K8s_NAMESPACE
    - curl -vk $K8S_SERVICE_URL
    - echo "Finished K8s Deployment"
```

Figura 54 – *Script* de implantação

Na figura é apresentado o *script* de implantação na *Google* e na *DigitalOcean.* Será apenas analisado o segundo, sendo este o *script* em utilização na empresa. Inicialmente é importada a variável do ambiente, seguida do *download* da interface de linha de comandos da *DigitalOcean*. Posteriormente é efetuado a configuração de acesso ao *cluster* e é implantada a imagem através da atualização da mesma no *cluster Kubernetes.* Por fim é executado o *smoke test* à aplicação, confirmando se a mesma se encontra disponível após a instalação. De modo efetuar o *script* ilustrado foram utilizadas as seguintes variáveis.

Tabela 17 – Variáveis utilizadas no *step* de implantação

Variável	Utilização	Tipo de Variável
$BITBUCKET_DEPLOYMENT_ENVIRONMENT	Nome do ambiente configurado no *Bitbucket Deployments*	Existente no *Bitbucket Pipelines*
$DOCTL_VERSION	Versão da interface de linha de comandos	Configurada em cada ambiente de implantação
$K8s_CLUSTER_NAME	Nome do *cluster* onde será efetuada a implantação	Configurada em cada ambiente de implantação

$K8S_SERVICE_URL	URL de acesso à aplicação implantada	Configurada em cada ambiente de implantação

A.2.2 Integração *Jira*

Com o intuito de facilitar o trabalho diário e permitir visualizar quais os problemas reportados no produto *Jira* que se encontram resolvidos e instalados nos diversos ambientes, foi integrado o *Bitbucket Pipelines* com o *Jira*.

Esta integração foi efetuada com recursos a dois serviços disponibilizados pela *Atlassian*, nomeadamente *workflow triggers* e *Bitbucket Deployments*.

Anteriormente na empresa Flowinn, já existia um fluxo no ciclo de vida dos *issues*, apresentado na seguinte figura.

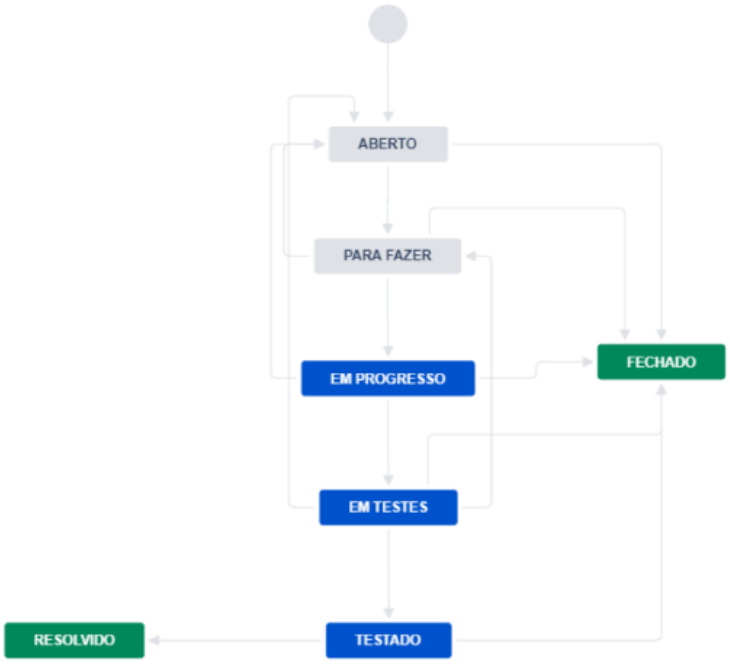

Figura 55 – Fluxo do ciclo de vida de um *issue* existente na empresa (imagem cedida pela *Flowinn)*

De modo a complementar a ciclo existente, foi sugerido a adição de dois *triggers* automáticos nas seguintes transições:

- Para fazer -> Em Progresso: Esta transição do *issue* acontecerá de forma automática, despoletada pela criação de um *branch* relativo ao issue em questão.

- Em progresso -> Em Testes: Com a introdução das *pull-requests* será possível automatizar esta transição, aquando uma *pull-request* relativa ao *issue* em questão for aceite e incluída no *branch development*.

Após validação com a empresa apenas o primeiro foi adicionado, uma vez que o segundo não foi possível aplicar, isto porque esta transição exige o preenchimento obrigatório do tempo despendido no *issue*. De modo a contornar este processo manual seria necessário a criação de um novo estado, contudo não seria obtida nenhuma vantagem em relação ao processo atual.

Relativamente à utilização do *Bitbucket Deployments*, este integra a *build* efetuada e os respetivos *issues* implantados na mesma. É possível visualizar esta integração no menu do *Bitbucket* e nos *issues* do *Jira*, facilitando assim o controlo das versões instaladas nos ambientes.

De modo a configurar esta integração, é necessária a configuração de cada ambiente de execução do *pipeline*, descrito no subcapítulo anterior, configurar o acesso do Bitbucket ao produto *Jira* e por fim relacionar os repositórios pretendidos com o projeto presente no *Jira*.

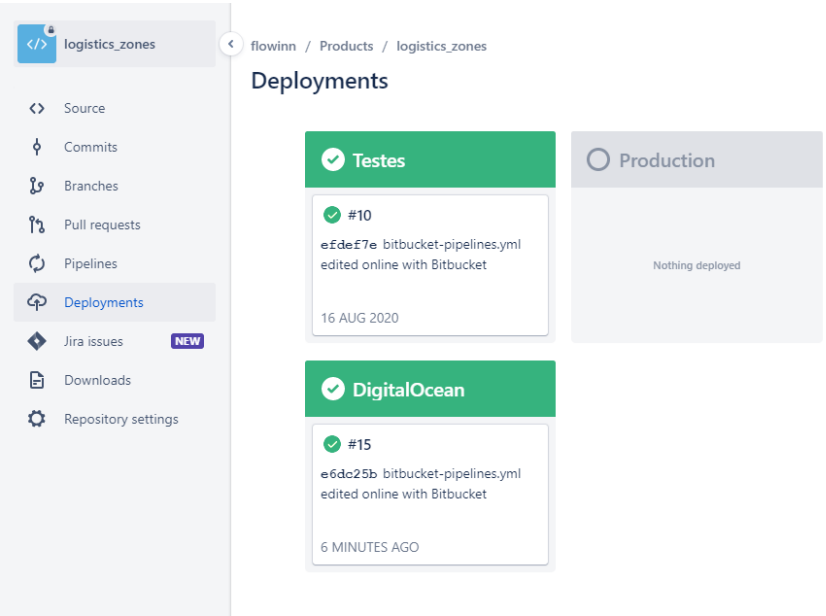

Figura 56 – Ambientes de implantação definidos no *Bitbucket*

A figura anterior, representa os ambientes configurados no *Bitbucket Deployments*, nomeadamente dois ambientes de testes, *Testes* e *DigitalOcean* e o de produção, *Production*. Como é possível observar, cada ambiente identifica claramente qual a *build* instalada e os respetivos *commits* e *issues* incluídos na mesma. É possível verificar com mais detalhe a informação da *build* e do ambiente, acedendo ao mesmo, como apresentado na imagem seguinte.

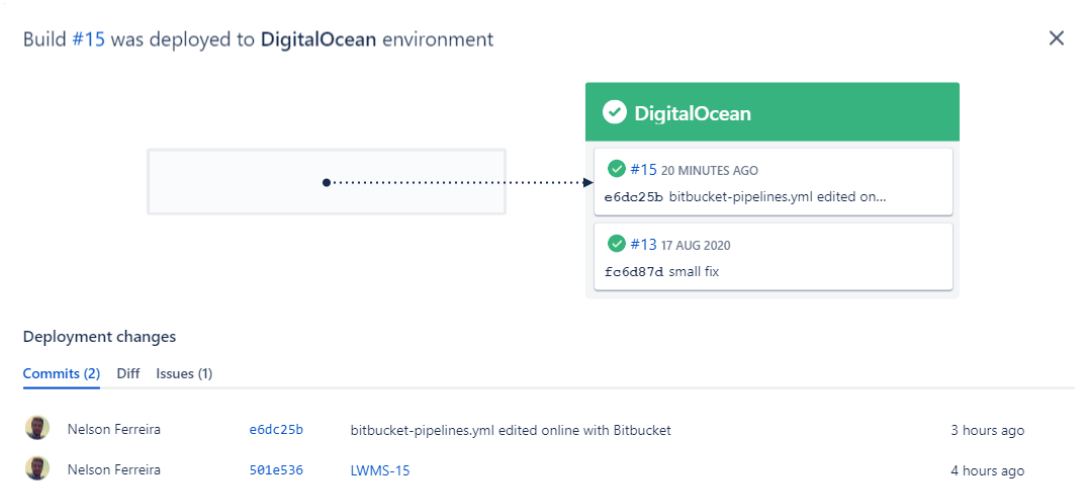

Figura 57 – Informação detalhada do ambiente de testes *DigitalOcean*

Como é possível observar na Figura 57, foi efetuado um *commit* associado ao *issue* LWMS-15, sendo que este já se encontra instalado no ambiente *DigitalOcean*, assim o *issue* presente no projeto do *Jira* contém também a informação relativa à implantação. Será assim possível a identificação dos *issues* prontos para testes.

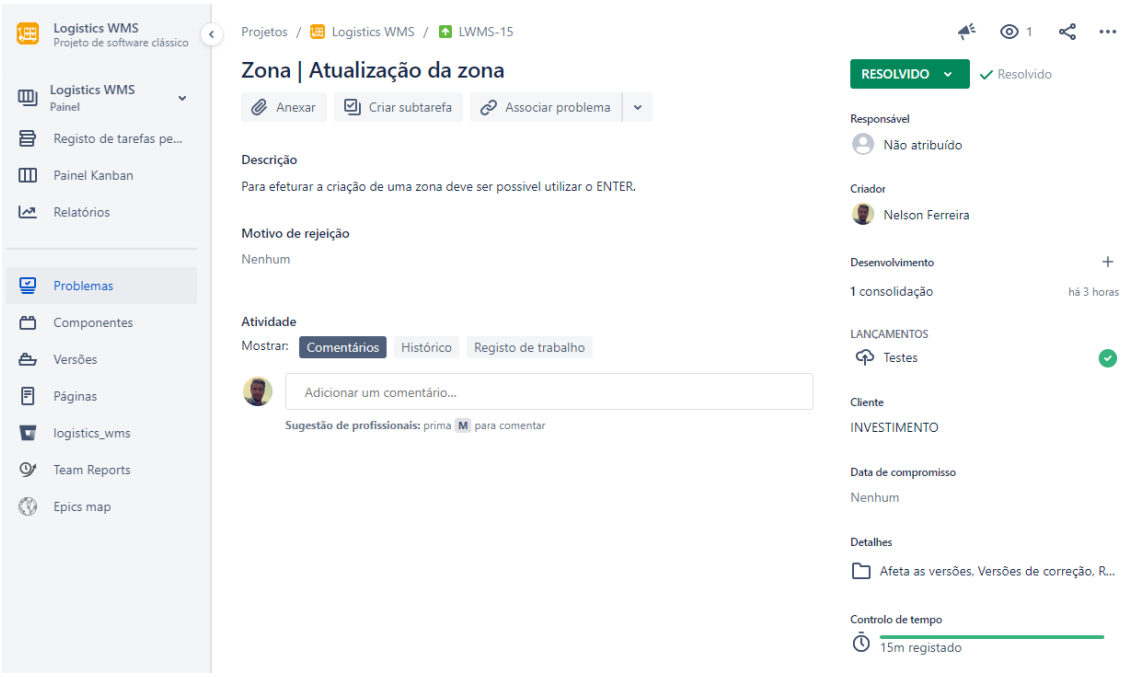

Figura 58 – *Issue* LWMS-15 representado Jira

Como é possível observar, no menu lateral direito é apresentada a informação do *Bitbucket,* nomeadamente o desenvolvimento efetuado e a implantação, denominada de lançamentos. Neste caso em concreto, o desenvolvimento apresenta o *commit* ilustrado na Figura 57 e o lançamento representa a implantação no ambiente *DigitalOcean*, como é possível visualizar com mais detalhe na figura seguinte.

Figura 59 – Detalhe das implantações do *issue* LWMS-15

A.3 Ficheiro de configuração do Kubernetes

Aqui é detalhada a configuração efetuada nos ficheiros *Kubernetes* do microsserviço zones que permitiram a implantação do componente no *cluster*, descrito no capítulo 8. Cada microsserviço contém ficheiros similares diferenciando apenas o repositório da imagem *Docker* e configurações da base de dados.

Figura 60 – Configuração do *deployment* do microsserviço *Zones*

Como é possível observar na Figura 60 para efetuar a implantação de cada microsserviço e *Gateway* foi utilizado o controlador *Deployment* do *Kubernetes*. Este, neste caso concreto, permite identificar os *pods* do microsserviço *Zones* de modo a escalar e assegurar a disponibilidade do mesmo. Quando um *pod* falha, o *Kubernetes* irá tentar substituí-lo imediatamente, visando reduzir o tempo de indisponibilidade do microsserviço em questão. Atualmente, a solução apresenta apenas uma réplica de cada componente, contudo o *cluster* implantado foi configurado de modo a suportar escalonamento vertical e horizontal de forma automática, baseando-se no tráfego do microsserviço.

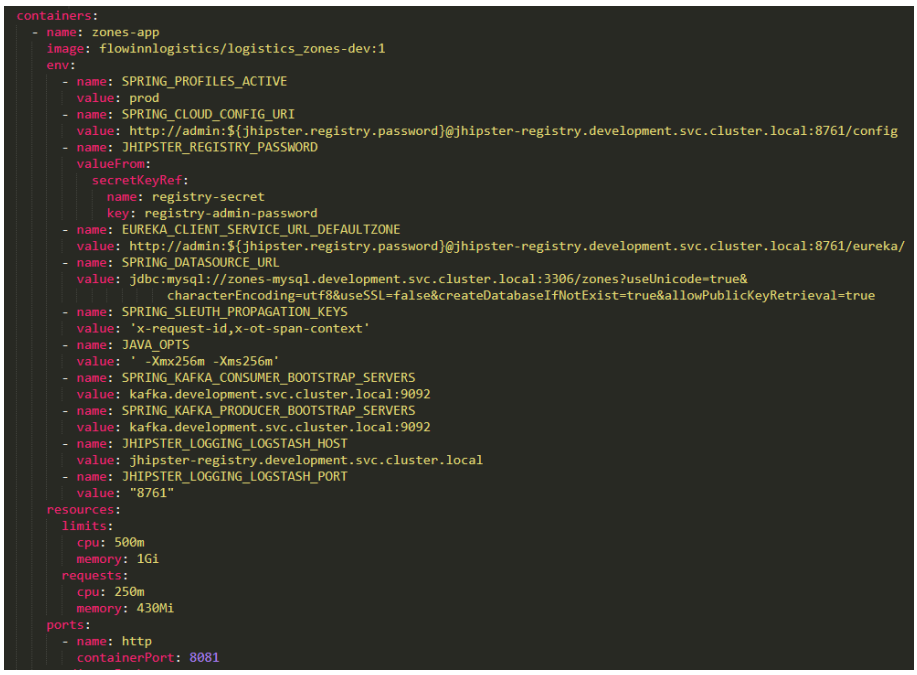

Figura 61 – Configuração do container presente no *pod*

Na Figura 61 são ilustradas as configurações que permitem ao *Kubernetes* identificar a respetiva imagem *Docker* e as configurações que o microsserviço necessita para executar, nomeadamente a configuração das comunicações do microsserviço com o MB *Apache Kafka*, o *JHipster Registry* e respetiva base de dados.

Como é possível verificar é identificado neste ficheiro, o repositório da imagem *Docker*, flowinnlogistics/logistics_zones-dev, e a sua respetiva versão, 1, esta é atualizada através das *builds* do *Bitbucket pipelines.* Relativamente à comunicação com os diversos componentes, esta é efetuada através do serviço *DNS* do *Kubernetes*. Assim de modo a efetuar a comunicação são utilizadas as seguintes variáveis, neste caso em concreto:

Tabela 18 – DNS utilizado para cada comunicação

Comunicação	DNS utilizado
Base de dados	zones-mysql.development.svc.cluster.local
JHipster Registry	jhipster-registry.development.svc.cluster.local
Apache Kafka	kafka.development.svc.cluster.local

Por fim, o ficheiro apresenta a configuração dos recursos que o microsserviço irá consumir e da porta onde estará acessível. Como foi descrito no subcapítulo 8.2, estes recursos foram reduzidos por causa do orçamento disponível para a implantação, assim para os microsserviços mais importantes e mais utilizados nesta solução, nomeadamente *zones*, *product*, *stock* e *receptions*, foi atribuído o inicial de 0.25 vCPU do *node* em que o mesmo será implantado, aos restantes foi atribuído o valor de 0.1 vCPU, pois é expectável que a sua utilização seja menor.

O ficheiro de configuração da implantação da base de dados é apresentado na seguinte figura.

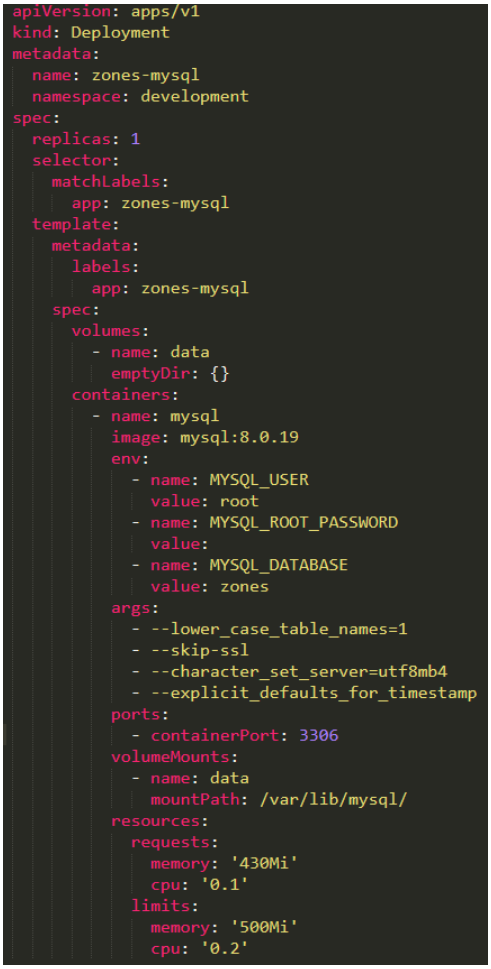

Figura 62 – Ficheiro da implantação da base de dados do microsserviço zones

Similarmente ao ficheiro de configuração de cada microsserviço, este utiliza o controlador *Deployment* do *Kubernetes*. Neste ficheiro são também configurados as credenciais de acesso à base de dados e a imagem *MySQL* utilizada. Estas são coerentes ao longos dos microsserviços. Por fim, é também configurada a porta onde a base de dados estará acessível, 3306, e os recursos disponibilizados para a base de dados em questão. Similarmente aos recursos dos microsserviços, optou-se por atribuir mais recursos à base de dados dos microsserviços mais importantes, nomeadamente 0.1 vCPU e 0.05 vCPU para os restantes.

A.4 Avaliação

Nesta secção serão detalhados os resultados obtidos na avaliação das aplicações e o método de avaliação utilizado no subcapítulo 9.1.2, QEF.

A.4.1 Desempenho e escalabilidade

Apresentando-se os resultados obtidos na execução dos testes do processo de receção e expedição. Através destes valores foi executada a média dos mesmos com o intuído de avaliar as soluções.

- Receção

Tabela 19 – Criação das linhas de receção na aplicação monolítica

Número de pedidos	Tempo de resposta (ms)	Tempo médio de resposta (ms)
50	60.68 66.2 57.7 62.28 65.8	62.532
100	66.48 65.5 56.38 65.33 65.83	63.904
500	70.41 73.09 71.398 70.202 73.596	71.7392
1000	139.892 161.795 105.995 142.018 155.68	141.076
2000	140.6755 140.5805 136.4925 130.69 155	140.6877
5000	731.3068 724.9088 747.301 766.064 1000.288	793.97372

Tabela 20 – Criação das linhas de receção na aplicação baseada em microsserviços

Número de pedidos	**Tempo de resposta (ms)**	**Tempo médio de resposta (ms)**
50	67.54 69.6 66.66 66.1 71.5	68.28

100	69.76 69.59 69.56 70.59 70.08	69.916
500	85.164 80.86 91.122 96 91.36	88.9012
1000	94.918 99.245 106.482 100.464 50.359	90.2936
2000	68.148 98.559 49.976 50.512 106.805	74.8
5000	167.1412 116.9846 97.8608 80.0808 66.9196	105.7974

- Expedição

Tabela 21 – Processo de expedição na aplicação monolítica

Número de linhas na expedição	**Duração (Minutos: Segundos)**	**Duração Média (Minutos: Segundos)**
50	00:22 00:25 00:25 00:19 00:22	00:23
100	00:28 00:27 00:29 00:28 00:28	00:28
500	00:34 00:34 00:34 00:35 00:36	00:35

Tabela 22 – Processo de expedição na aplicação baseada em microsserviços

Número de linhas na expedição	Duração (Minutos: Segundos)	Duração Média (Minutos: Segundos)
50	00:06 00:04 00:04 00:03 00:03	00:04
100	00:07 00:09 00:12 00:09 00:08	00:09
500	00:15 00:21 00:16 00:16 00:15	00:17

A.4.2 Manutenibilidade, automatização e satisfação

Neste subseção apresenta-se o QEF efetuado e o questionário apresentado à empresa.

- QEF

q	D	q_i	Dimension	Q_j	W_{ij} (Factor Weight *j* in Dim *i*) [0,1]	Factor	rw_{jk} (requirement weight *k* in Factor *j*) {2, 4, 6, 8, 10}	Requirement	wf_k % requirement fulfillment *k*) [0,100]
98%	0.05	100	Manutenibilidade	100	0.44	Documentação	10	Análise e criação de requisitos	100
							10	Documentar os componentes desenvolvidos	100
							10	Documentar a implantação efetuada	100
							8	Documentar a nova arquitetetura	100
				100	0.56	Monotorização	10	Visualizar os logs dos componentes	100
							8	Visualizar o CPU utilizado	100
							8	Visualizar a Memória utilizada	100
							10	Visualizar a disponibilidade da solução	100
							8	Controlar o tráfego da solução	100
		96.5517	Automatização	96.55	1.00	Processo de implantação	10	Automatizar a implantação da solução	100
							8	Automatizar o processo de execução de build	100
							4	Execução de testes unitários	100
							4	Execução de testes de integração	100
							2	Assegurar a qualidade do código desenvolvido	0
							6	Publicar a imagem Docker	100
							8	Implantar a solução no ambiente	100
							8	A implantação efetuada deverá ficar disponível para o utilizador	100
							8	Suportar os diversos ambientes	100
		96.2963	Satisfação	96.3	1.00	Satisfação da empresa	10	Melhor desempenho e escalabilidade	100
							8	Simplificar a implementação de novas funcionalidades	75
							10	Otimização dos processos do armazém	100
							8	Agilizar a entrega das funcionalidades desenvolvidas	100
							8	Redução de erros na implantação da solução	100
							10	Evolução arquitetural apresenta benefícios para a empresa	100

Figura 63 – QEF utilizado para avaliar a solução desenvolvido

- Questionário apresentado à empresa Flowinn

Figura 64 – Questionário apresentado e 1ª pergunta

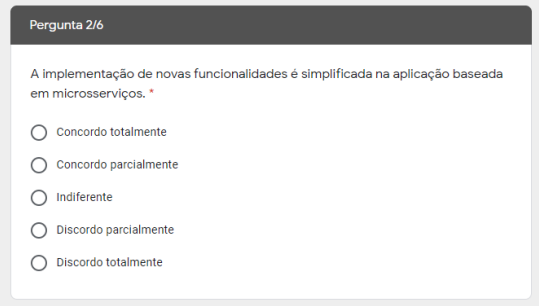

Figura 65 – 2ª pergunta do questionário

Figura 66 – 3ª pergunta do questionário

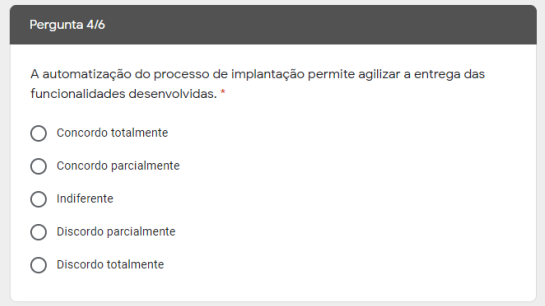

Figura 67 – 4ª pergunta do questionário

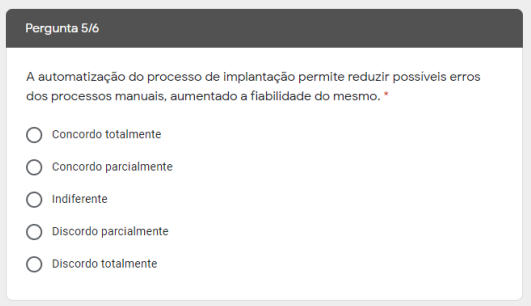

Figura 68 – 5ª pergunta do questionário

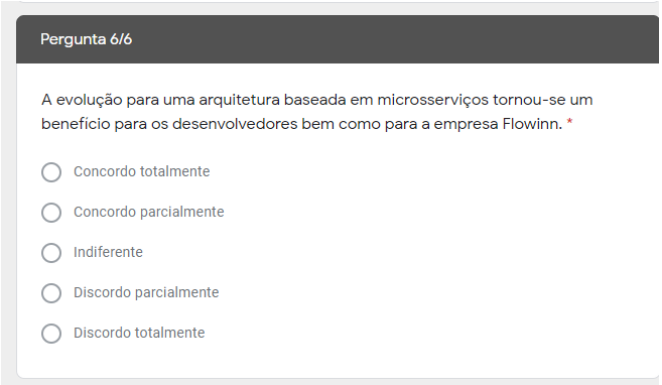

Figura 69 – 6ª pergunta do questionário

- Respostas obtidas no questionário

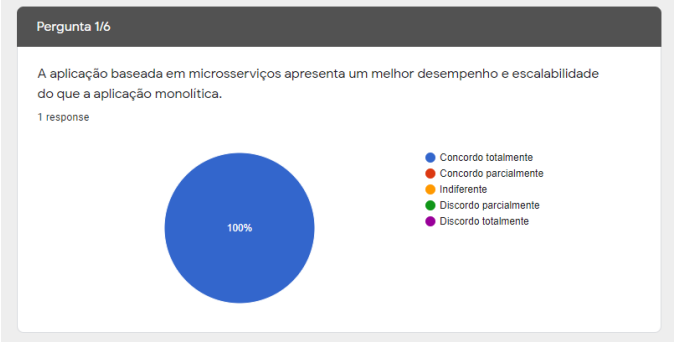

Figura 70 – Respostas obtidas na 1º pergunta

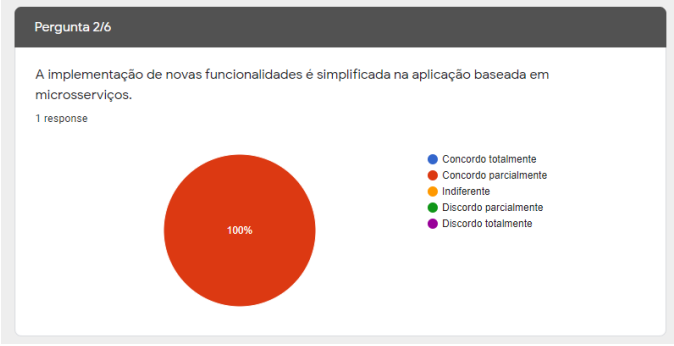

Figura 71 – Respostas obtidas na 2º pergunta

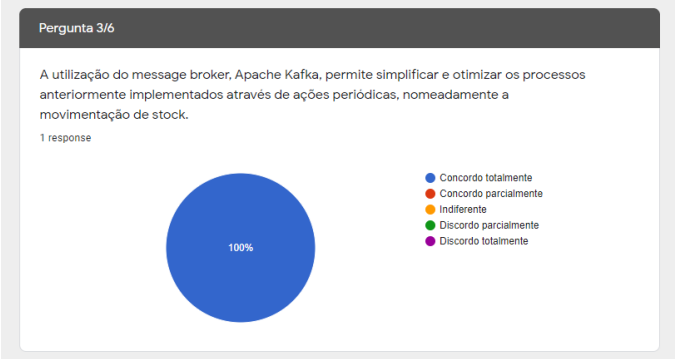

Figura 72 – Respostas obtidas na 3º pergunta

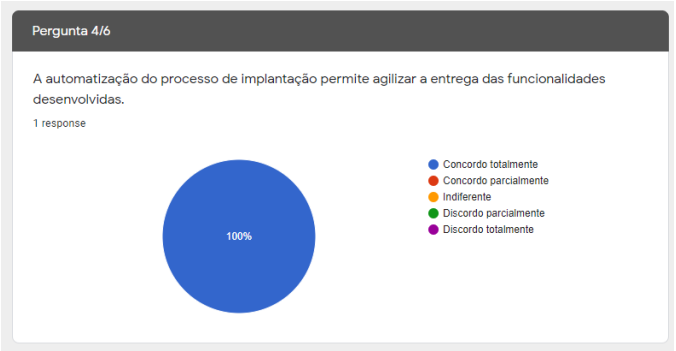

Figura 73 – Respostas obtidas na 4º pergunta

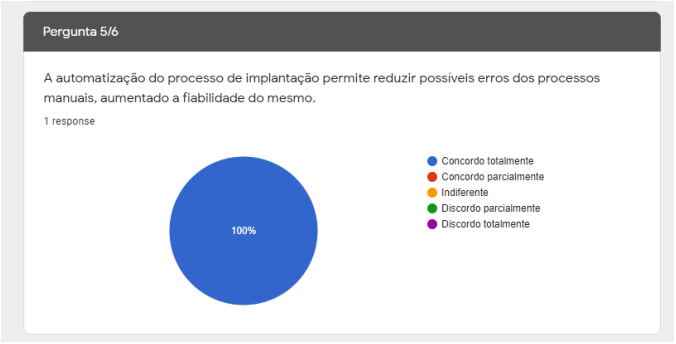

Figura 74 – Respostas obtidas na 5º pergunta

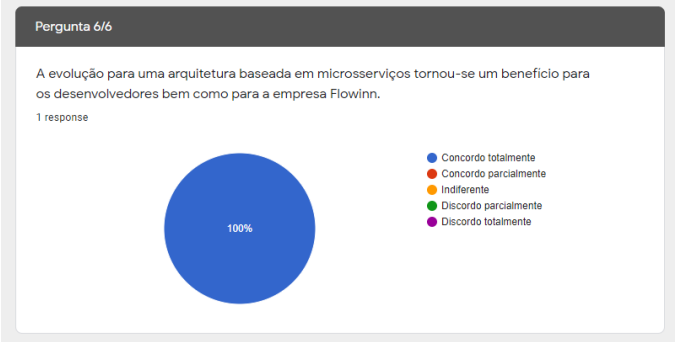

Figura 75 – Respostas obtidas na 6º pergunta