INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




eDocuments as Microservices

HÉLIO JOSÉ ALMEIDA CARDOSO
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

eDocuments as Microservices

Hélio José Almeida Cardoso

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

Orientador: Doutor Alberto Sampaio

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, outubro 2020

Resumo

A plataforma eDocuments é uma aplicação *web* de faturação eletrónica e EDI assente numa arquitetura monolítica.

Recentemente, esta aplicação tem apresentado um aumento de clientes e dados a processar e, consequentemente, têm surgido problemas de desempenho, escalabilidade e de manutenibilidade que se revelam difíceis de resolver com a arquitetura atual. Adicionalmente, o processo de desenvolvimento de software apresentada limitações que dificultam a manutenção e evolução da aplicação.

Assim sendo, a principal finalidade do projeto proposto consiste na reestruturação da arquitetura da aplicação existente de forma a colmatar os problemas resultantes da arquitetura existente. Desta forma, pretende-se mostrar que, com a reestruturação do sistema para uma arquitetura baseada em microsserviços, se poderá usufruir de um sistema escalável, com melhor desempenho e com melhores níveis de manutenibilidade. Simultaneamente, também se pretende mostrar a melhoria da qualidade do processo de desenvolvimento de software com o desenvolvimento e implementação de uma estratégia de *Continuous Delivery* e *Continuous Integration*.

Deste modo, foi desenvolvida uma aplicação com uma arquitetura baseada em microsserviços (para dar resposta aos problemas de escalabilidade, desempenho e manutenibilidade) e implementado um *pipeline* de *Continuous Delivery* e *Continuous Integration* de forma a simplificar e automatizar o processo de entrega de modificações efetuadas ao código fonte face à automação de tarefas que atualmente são manuais.

Palavras-chave: Microsserviços, Escalabilidade, Manutenibilidade, *DevOps, Continuous Delivery, Continuous Integration*

Abstract

The eDocuments platform is an electronic invoicing and EDI web application based on a monolithic architecture.

Recently, this application presents an increase in customers and data to be processed and, consequently, performance, scalability and maintenance problems have arisen that are difficult to solve with the current architecture. The software development process also presented limitations that hinder the maintenance and evolution of the application.

Therefore, the main purpose of the proposed project is to restructure the architecture of the existing application to overcome the problems resulting from the existing architecture. In this way, it is intended to show that, with the restructuring of the system for an architecture based on microservices, we can obtain a scalable system, with better performance and with better levels of maintainability. At the same time, it is also intended to show the improvement in the quality of the software development process with the development and implementation of a Continuous Delivery and Continuous Integration strategy.

In this way, an application was developed with an architecture based on microservices (to respond to scalability, performance and maintainability problems) and a Continuous Delivery and Continuous Integration pipeline was implemented to simplify and automate the process of delivering changes made to source code with the automation of tasks that are currently manual.

Keywords: Microservices, Scalability, Maintainability, *DevOps, Continuous Delivery, Continuous Integration*

Dedicatória

Agradeço e dedico este trabalho aos meus pais por todo o apoio e dedicação que sempre me deram. Em especial dedico este trabalho ao meu pai. Não estando presente na parte final deste percurso, de certeza que onde estiver assistiu ao mesmo e sem dúvida que é o grande impulsionador e, por isso, todo este percurso é por ele e dedicado a ele.

Agradecimentos

Primeiramente quero fazer um agradecimento ao meu orientador Doutor Alberto Sampaio que teve um papel fundamental no acompanhamento deste projeto sempre com disponibilidade, dedicação, interesse e preocupação para que os objetivos fossem atingidos.

À empresa Flowinn, fica o agradecimento pela oportunidade de desenvolver este projeto. Um agradecimento especial ao Ricardo Fernandes, colaborador da Flowinn, que sempre se apresentou disponível para ajudar no desenvolvimento do projeto.

Por fim, deixo um agradecimento especial à minha família e amigos pelo incansável apoio apresentado ao longo de todo este percurso.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

EDI	*Electronic Data Interchange*
ERP	*Enterprise Resource Planning*
AHP	*Analytic Hierarchy Process*
CQRS	*Command Query Responsibility Segregation*
RC	Razão de Consistência
GB	*Gigabyte*
RAM	*Random Access Memory*
CPU	*Central Processing Unit*
QEF	*Quantitative Evaluation Framework*

Lista de Símbolos

$\overline{\mathbf{x}}$	Média aritmética

1 Introdução

1.1 Contexto

1.1.1 Entidade

A proposta deste projeto foi efetuada pela empresa Flowinn. Com escritórios localizados no Porto, Lisboa e Braga, o grupo Flowinn surgiu no ano de 2013 com a missão de desenvolver soluções na área das Tecnologias da Informação acrescentando valor às empresas-clientes "de modo a potenciar o crescimento, desenvolvimento e eficiência dos seus negócios" (*Flowinn – Business Inovation*).

Figura 1 - Logotipo da empresa Flowinn

1.1.2 Plataforma eDocuments

A plataforma eDocuments é uma aplicação *web* de faturação eletrónica e EDI (*Electronic Data Interchange*) (EDI, 2020) que tem como principal funcionalidade a troca de ficheiros de dados entre várias entidades/empresas que se encontram registadas na plataforma.

De facto, estas empresas têm vindo a integrar os seus ERP (*Enterprise Resource Planning*) com esta plataforma com o intuito de verem a troca de dados com os seus clientes e fornecedores facilitada já que a própria plataforma efetua essas operações, ficando a empresa dispensada das mesmas. Nos últimos tempos, esta aplicação tem apresentado um crescimento de entidades/empresas o que resultou num aumento de sistemas a integrar e dados/documentos a processar. Com este aumento tem-se verificado uma cada vez maior necessidade de melhoria, manutenção e incremento de novas funcionalidades.

Ao nível de arquitetura, esta plataforma é uma aplicação monolítica com uma interface com o utilizador desenvolvida em AngularJS, uma API do lado do servidor implementada com recurso a Spring Boot e um sistema de gestão de base de dados utilizando MySQL. Desta forma, toda a lógica de negócio (como, por exemplo, serviços de processamento e integração de documentos) encontra-se na API em execução do lado do servidor.

Relativamente à implantação, a plataforma eDocuments encontra-se em execução num servidor alojado localmente e que conta com um processador de 4 núcleos juntamente com 8GB de memória RAM.

1.2 Problema

Como referido anteriormente, a plataforma eDocuments é uma aplicação *web* que se encontra em crescimento e evolução.

De facto, nos últimos tempos, esta aplicação tem tido picos de crescimento de clientes e de dados a processar. Como consequência desse aumento têm surgido problemas de desempenho e de memória da aplicação, bem como dificuldades na sua manutenção. Devido ao facto da solução atual ser baseada numa arquitetura monolítica e se encontrar implantada num servidor local, não é possível obter escalabilidade nem elasticidade para serviços cruciais como, por exemplo, serviços de processamento e integração de documentos.

Simultaneamente, são reconhecidas limitações no processo de desenvolvimento de software que, entre outros aspetos, têm dificultado a manutenção e evolução da aplicação. Dada a importância que os processos de software têm (Sharma, 2017) existe a necessidade de melhorar todo o processo de desenvolvimento, tornando-o mais ágil.

1.3 Objetivos

Dado o problema existente, o principal objetivo do trabalho proposto consiste na reestruturação da aplicação existente de forma a torná-la escalável e elástica no sentido de dar resposta aos problemas de desempenho e memória. Tendo em consideração a importância do processamento de documentos inerente a esta aplicação, é importante possibilitar que os serviços responsáveis pelos diversos tipos de processamentos possam ser escaláveis e elásticos. Outro aspeto a reter para este objetivo é o facto de existirem períodos em que se verifica um maior fluxo de documentos a processar e integrar. De facto, a maior parte das entidades/empresas têm os seus documentos a processar/integrar na fase inicial de cada mês e existem entidades que chegam a processar/integrar, em média, cerca de 7000 documentos. Assim sendo, seria importante que a aplicação possibilitasse um aumento de recursos alocados aos diferentes serviços responsáveis por processamento e integração de documentos nas fases identificadas como fases de maior afluência a estes mesmos serviços.

Pretende-se, também, aumentar a manutenibilidade do software de modo a facilitar o seu crescimento e a permitir reduzir a complexidade de integração dos sistemas dos clientes assim como a adição e manutenção dos processamentos a efetuar aos documentos/dados da aplicação.

Com o intuito de melhorar a qualidade do processo de desenvolvimento de forma a facilitar a evolução do produto ao longo do seu ciclo de vida, pretende-se, também, que seja desenvolvida uma estratégia de *Continuous Delivery* (secção 2.2.2) e *Continuous Integration* (secção 2.2.1) com automação de diversas tarefas como, por exemplo, *build*, testes e *deploy*.

Inicialmente, para tentar cumprir com os objetivos mencionados, serão identificados, especificados e investigados os pontos/aspetos que, atualmente, não estão a permitir que a solução existente dê a resposta desejada.

Posteriormente, será efetuada uma análise baseada nestes pontos/aspetos e, partindo dos mesmos, será efetuada uma análise do estado da arte com a investigação de arquiteturas e tecnologias que foram aplicadas em problemas semelhantes de forma a efetuar um levantamento de possíveis soluções para o problema em questão. Analisadas as possíveis soluções, as mesmas serão comparadas com vista a perceber quais as vantagens que as mesmas oferecem em relação à atual solução e em relação a outros critérios que serão avaliados como, por exemplo, custos associados, alinhamento com a *stack* tecnológica da empresa, alinhamento com processos aplicados na empresa, entre outros (Análise de valor).

Subsequentemente, a solução implementada terá de passar por um processo de avaliação criterioso (capítulo 6).

1.4 Estrutura do documento

Para efetuar uma clara separação de conteúdos e organização do documento, o mesmo encontra-se dividido em capítulos e respetivos subcapítulos.

Posto isto, o documento encontra-se estruturado da seguinte forma:

- Capítulo 2 – Estado da Arte: capítulo onde são apresentados conceitos importantes relacionados com o problema e os objetivos assim como casos de estudo de organizações que passaram por problemas e/ou objetivos semelhantes. São, também, apresentadas, analisadas e comparadas diferentes abordagens quer arquiteturais quer tecnológicas a considerar para este projeto;
- Capítulo 3 – Análise de Valor: capítulo que aborda a proposta de valor que se pretende atingir e onde são selecionadas alternativas a utilizar para o desenvolvimento da solução final;
- Capítulo 4 – Design Selecionado: apresenta o *Design* da solução final, ou seja, a sua arquitetura, os seus componentes e respetivas relações e, ainda, o modo como será implantada a solução.

- Capítulo 5 – Desenvolvimento da Solução: aborda o desenvolvimento da solução final retratando a forma como foi elaborada a solução no âmbito deste projeto;
- Capítulo 6 – Experimentação e Avaliação: apresenta a hipótese de investigação (o que se pretende demonstrar com este projeto), os indicadores a avaliar (e a forma como serão avaliados) e, ainda, a respetiva avaliação no âmbito deste projeto.
- Capítulo 7 – Conclusões: trata as conclusões do trabalho desenvolvido apontando, ainda, as limitações existentes e o trabalho futuro que dará continuidade a este projeto.

2 Estado da arte

Este capítulo visa apresentar conceitos e o conhecimento atual de casos semelhantes ao problema apresentado com o intuito de extrair conhecimentos benéficos para o posterior desenvolvimento de uma solução para a resolução do problema.

Posto isto, as próximas secções irão expor conceitos importantes para o problema em questão seguindo-se a apresentação de diferentes casos de estudo de organizações como a Wix.com, SmartThings, Danske Bank e Netflix. Por fim, o capítulo é finalizado com a apresentação, análise e comparação de diferentes abordagens arquiteturais e tecnológicas a considerar para o desenvolvimento deste projeto.

2.1 Microsserviços

Nos últimos anos, o termo Microsserviços tem sido muito discutido na área de engenharia de software e utilizado em vários projetos com resultados positivos (Fowler & Lewis, 2014).

Apesar de não existir uma definição precisa, o termo Microsserviços é apresentado como um estilo arquitetural que assenta na implementação de uma aplicação sob um conjunto de pequenos serviços independentes (Fowler & Lewis, 2014). Os serviços podem inclusive ser implementadas com recurso a diferentes linguagens de programação e com utilização de diferentes tipos de armazenamento de dados. Estes serviços "comunicam entre si utilizando mecanismos leves, geralmente HTTP com recurso a API" (Fowler & Lewis, 2014). Carateriza-se, também, por ser "um processo coeso e independente que interage via mensagens" (Dragoni et al., 2017).

A arquitetura de microsserviços é frequentemente comparada com o estilo monolítico a fim de identificar as vantagens de uma sobre a outra. Ao contrário de uma arquitetura de microsserviços, uma aplicação monolítica é geralmente construída sob apenas três componentes, sendo eles a interface com o utilizador, uma base de dados e uma aplicação

servidor (que recebe as solicitação do lado do cliente, aplica lógica de negócio, atualiza e recolhe dados da base de dados) (Fowler & Lewis, 2014). Deste modo, a aplicação do lado do servidor é uma aplicação monolítica que contém todo o domínio e respetiva lógica de negócio, sendo que, toda esta lógica de negócio executa sob o mesmo processo fazendo uso de apenas uma linguagem de programação. Desta forma, aquando de uma alteração numa parte do sistema, a aplicação monolítica necessita de ser reconstruída e implantada como um todo (Fowler & Lewis, 2014). As aplicações monolíticas caraterizam-se, também, por serem pouco coesas e por serem fortemente acopladas com os diferentes módulos/componentes a possuir diversas dependências entre si o que dificulta a manutenção do código, devido ao facto de que uma alteração num módulo pode ter impacto noutros módulos.

Em contrapartida, numa arquitetura de microsserviços, ao invés de uma única aplicação, o lado do servidor é construído sob um conjunto de serviços independentes com o intuito de aumentar a coesão e reduzir o acoplamento. De seguida, serão apresentadas as principais caraterísticas de uma arquitetura baseada em microsserviços assim como as principais práticas e padrões associados a esta arquitetura.

2.1.1 Caraterísticas

2.1.1.1 Autónomos

Num sistema monolítico, o aumento da dificuldade de correção de erros/adição de novas funcionalidades é proporcional ao crescimento do sistema devido à baixa coesão e ao elevado acoplamento dos serviços. Em contrapartida, uma arquitetura de microsserviços adota a abordagem de serviços independentes, com funções bem delineadas e que comunicam por meio das suas interfaces publicadas (Dragoni et al., 2017). “O objetivo principal de um microsserviço é poder alterar um serviço e implantá-lo, sem precisar alterar nenhuma outra parte do sistema.” (Newman, 2015, p. 65). Desta forma, um microsserviço pode ser substituído, atualizado e implantando de forma independente (Fowler & Lewis, 2014).

2.1.1.2 Heterogeneidade

Como referido anteriormente, os microsserviços são independentes e, posto isto, podem ser implementados com recurso a diferentes linguagens de programação. Assim sendo, “permite-nos escolher a ferramenta certa para cada trabalho, em lugar de ter de se selecionar uma abordagem mais padronizada” (Newman, 2015, p. 20). Esta heterogeneidade não se aplica só ao nível de linguagens de programação. Os microsserviços podem, também, utilizar diferentes formas de armazenar os dados consoante as suas necessidades o que é chamada de Persistência Poliglota (Fowler & Lewis, 2014).

A Figura 2 representa um exemplo da heterogeneidade que uma arquitetura baseada em microsserviços poderá apresentar.

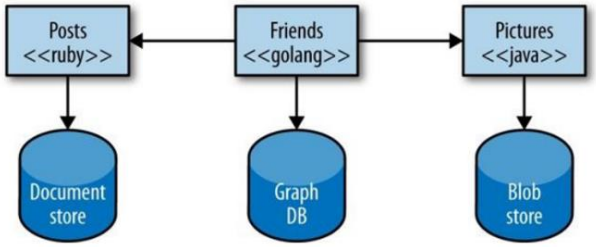

Figura 2 - Microsserviços permitem utilização de diferentes tecnologias (Newman, 2015, fig. 1)

2.1.1.3 Resiliência

Devido ao elevado acoplamento existente num sistema monolítico (elevada dependência entre serviços), se um serviço falhar poderá fazer com que todo o sistema falhe. Em contrapartida, com o baixo acoplamento visado por uma arquitetura baseada em microsserviços, "se um componente de um sistema falha, mas essa falha não ocorre em cascata, pode-se isolar o problema e o sistema pode continuar a funcionar" (Newman, 2015, p. 22). Deste modo, corrigir um problema ou adicionar uma funcionalidade não tem impacto nos restantes microsserviços (Dragoni et al., 2017) permitindo "reescrever um componente sem afetar seus colaboradores" (Fowler & Lewis, 2014).

2.1.1.4 Manutenibilidade

Numa arquitetura de microsserviços, os microsserviços devem ser fracamente acoplados devido à sua autonomia/independência. Assim sendo, "contribui muito para a manutenção de um sistema, minimizando os custos de modificação de serviços, correção de erros ou inclusão de novas funcionalidades" (Dragoni et al., 2017). Desta forma, permite que sejam efetuadas alterações "… frequentes, rápidas e bem controladas no software." (Fowler & Lewis, 2014). Outro aspeto importante é que, devido ao facto de os microsserviços serem de menor dimensão, o custo de modificá-los ou até mesmo substituí-los por completo será menor (Newman, 2015, p. 27).

2.1.1.5 Escalabilidade

Como referido por (Newman, 2015, p. 23), "Com serviços menores, podemos apenas dimensionar os serviços que precisam de escalar, permitindo executar outras partes do sistema em *hardware* menor e menos potente". Num sistema monolítico isto não é possível, sendo necessário escalar toda a aplicação ao invés de escalar apenas algumas partes.

2.1.1.6 Implantação

Com um sistema monolítico, uma alteração numa parte do sistema obriga a que toda a aplicação seja implantada provocando um impacto no tempo de implantação. No entanto, seguindo uma arquitetura baseada em microsserviços, isto pode ser ultrapassado devido à independência dos microsserviços que permite "alterar um único serviço e implantá-lo independentemente do resto do sistema" (Newman, 2015, p. 24). Isto permite reduzir o tempo de implantação de uma alteração ao código fonte disponibilizando-a mais rapidamente aos clientes.

2.1.1.7 Organização e gestão do projeto

Com um menor dimensionamento de código associado a cada microsserviço, surge a possibilidade de organizar as pessoas afetas ao projeto distribuindo-as por equipas mais pequenas associadas a cada microsserviço com o intuito de aumentar a produtividade. Como referido por (Newman, 2015, p. 25), "… equipas menores que trabalham em bases de código menores tendem a ser mais produtivas.". Desta forma, as equipas deverão ser multifuncionais, ou seja, com capacidades para assumir o produto ao longo do seu ciclo de vida. Posto isto, as equipas devem assumir a responsabilidade de desenvolvimento (gestão do projeto, interfaces com utilizador, armazenamentos de dados, implantação, entre outros) e, também, monitorizar e efetuar a manutenção dos sistemas em produção (Fowler & Lewis, 2014).

2.1.1.8 Reutilização

Um aspeto a considerar no desenvolvimento de software é evitar que um determinado comportamento esteja duplicado sendo que, desta forma, uma alteração ao comportamento apenas necessita de ser realizada num único local. Assim sendo, este aspeto introduz ênfase na criação de código passível de ser reutilizado (Newman, 2015, p. 114). Com um microsserviço a possuir responsabilidades menores, permite que as suas funcionalidades sejam consumidas mais facilmente por serviços externos sendo, assim, facilmente reutilizáveis. No entanto, com um microsserviço a ser utilizado por vários microsserviços, uma alteração no código poderá provocar impacto nos microsserviços e implicar que os mesmos tenham de ser atualizados. Assim, a reutilização de código através de vários microsserviços deverá ser utilizada de forma cuidada evitando que se aumente o acoplamento entre eles (Newman, 2015, p. 114).

2.1.2 Práticas

2.1.2.1 Baixo acoplamento e elevada coesão

Existem várias práticas e vários princípios para as arquiteturas baseadas em microsserviços, porém, baixo acoplamento e elevada coesão são os conceitos chave desta arquitetura e se estes dois conceitos não forem aplicados tudo o resto será inútil (Newman, 2015, p. 64).

Um dos principais objetivos desta arquitetura é poder efetuar alterações a um serviço sem provocar impacto noutros serviços. Isto é conseguido se os serviços forem fracamente acoplados, ou seja, sem dependências que façam com que uma alteração num microsserviço implique que seja necessário alterar outro microsserviço. Deste modo, um serviço fracamente acoplado tem um conhecimento reduzido acerca dos serviços com os quais interage sendo possível modificá-lo sem causar impacto nos outros serviços (Newman, 2015, p. 65).

Outro objetivo dos microsserviços é que estes sejam de manutenção e de implantação simples e rápida. Assim, quando é necessário efetuar uma alteração a um determinado comportamento/processo do sistema, quer-se apenas alterar num único local devido ao facto de alterações em diversos locais serem mais demoradas e, também, mais propícias a erros. Assim sendo, um microsserviço é altamente coeso quando possui toda a lógica para a

realização de um comportamento/processo de negócio garantindo que uma alteração nesse processo apenas implique a alteração nesse mesmo microsserviço (Newman, 2015, p. 66).

2.1.2.2 Contexto limitado

Tendo em conta a elevada coesão, os microsserviços devem, também, possuir um contexto limitado. Ou seja, devem possuir um domínio que permita que o microsserviço tenha uma responsabilidade específica. Posto isto, é importante que os microsserviços tenham uma responsabilidade específica e bem definida com um contexto limitado que permita a implementação das funcionalidades necessárias para a realização da sua responsabilidade (Newman, 2015).

2.1.2.3 Partilha de modelos

Eventualmente, podemos encontrar entidades com o mesmo nome em diferentes microsserviços, porém, como são representativos de contextos diferentes, a sua representação e/ou os seus significados serão diferentes (Newman, 2015, p. 69).

Apesar dos microsserviços serem autónomos e possuírem contextos limitados e separados, por vezes, poderá ser necessário um microsserviço obter algumas informações de outros microsserviços.

Para isso, os microsserviços devem ter uma interface para acessos externos, porém, quando um microsserviço precisa de informações de outros microsserviços, não precisa de saber o detalhe do funcionamento interno, apenas necessita de algumas informações. Assim sendo, os microsserviços não devem expor todo o detalhe acerca de um determinado objeto de domínio e, assim, deverão existir duas representações para essa entidade (Newman, 2015, p. 69):

- Representação interna: contém todos os detalhes de uma determinada entidade necessária para o contexto representado no microsserviço;
- Representação externa: contém apenas a informação da representação interna que se pretende expor para o exterior.

A representação externa é crucial para a redução do acoplamento. Se os consumidores estivessem conectados à implementação interna o acoplamento seria aumentado devido ao facto de que uma alteração na lógica de negócio teria repercussões nos microsserviços consumidores. Assim sendo, deve ser evitado que um microsserviço exponha detalhes acerca da sua representação interna (Newman, 2015, p. 85).

Desta forma, deve, também, ser seguida a prática de cada microsserviço aceder apenas à sua base de dados. Se dois ou mais microsserviços partilharem a mesma base de dados eles estão a partilhar a mesma representação interna e, no caso de um microsserviço necessitar de efetuar uma alteração à mesma, isto terá um impacto direto nos outros microsserviços. Assim, a partilha de bases de dados entre microsserviços deverá ser evitada com o intuito de reduzir o acoplamento entre os microsserviço (Newman, 2015, p. 87).

Outro aspeto importante para evitar falhas no sistema é garantir que, quando são adicionados novos campos na representação externa, os microsserviços consumidores desses objetos não devem ser afetados (Newman, 2015, p. 82).

2.1.2.4 API Gateway

Um API Gateway é um serviço que serve como ponto de entrada na aplicação a partir do qual as aplicações externas devem endereçar os pedidos (Richardson, 2018).

Entre outras funcionalidades, o API Gateway é responsável por efetuar o roteamento dos pedidos para o serviço responsável e poderá, também, ser responsável pela autenticação (Richardson, 2018). Posto isto, as aplicações externas deverão efetuar solicitações ao API Gateway que fará o reencaminhamento das solicitação para o microsserviço responsável encapsulando, assim, a estrutura interna da aplicação (Richardson, 2018).

Para além de roteamento, o API Gateway poderá ser responsável por autenticação, monitorização (secção 2.1.2.10), suprimir pedidos externos (secção 2.1.2.9), balanceamento de carga, entre outros. Também poderá efetuar tradução de protocolos, ou seja, utilizar um protocolo de comunicação com aplicações externas e, internamente, utilizar outro protocolo para comunicar com os microsserviços (Richardson, 2018).

Uma prática comum numa arquitetura de microsserviços é ter uma agregação dos serviços num API Gateway.

2.1.2.5 Descoberta de Serviços

Com a existência de vários microsserviços é importante obter informação relativa à localização de cada um (Newman, 2015, p. 397).

De facto, quer para efeitos de monitorização quer por questões de organização para auxiliar os programadores a saberem que API existem e onde se encontram, é importante que esta informação sejam agregada num único local (Newman, 2015, p. 397). Existem várias soluções de software que visam simplificar esta tarefa que, assim sendo, auxiliam na gestão deste processo. Estas soluções permitem que as instâncias dos microsserviços se registem aquando do início da sua execução e fornecem formas de obter informação acerca dos serviços registados (Newman, 2015, p. 397).

2.1.2.6 Comunicação entre microsserviços

A comunicação entre microsserviços é um dos aspetos "… mais críticos para o sucesso do projeto." (Pacheco, 2018, p. 46). Numa aplicação monolítica os serviços executam no mesmo processo sendo que a comunicação entre eles é feita por invocações de métodos ou funções. Porém, numa arquitetura de microsserviços, devido à separação dos serviços em processos diferentes, é necessário alterar o padrão de comunicação entre eles (Fowler & Lewis, 2014).

Existem duas formas de comunicação entre microsserviços, sendo elas a comunicação síncrona e a comunicação assíncrona. A forma mais comum é a comunicação assíncrona por ser mais fácil para escalar, porém, é uma forma mais difícil de aplicar e de compreender já que

não se obtêm retorno da conclusão da operação (Pacheco, 2018, p. 46) (Newman, 2015, p. 89). Em contrapartida, a comunicação síncrona é mais simples de aplicar, de entender e de detetar erros devido ao facto de indicar o retorno de sucesso, porém, é mais difícil de escalar (Pacheco, 2018, p. 46) (Newman, 2015, p. 89).

- Comunicação síncrona:

Com os microsserviços a disponibilizarem interfaces de comunicação para o exterior, a forma mais comum de comunicação síncrona é a utilização do protocolo HTTP com REST (*REpresentational State Transfer*) e com passagem de dados com recurso a JSON (*JavaScript Object Notation*) (Pacheco, 2018, p. 47). Na comunicação síncrona, a operação fica bloqueada até que o servidor remoto (ao qual o pedido foi endereçado) responda. A comunicação síncrona, apesar de simplificar na deteção de erros e de ser simples de aplicar, apresenta impacto no desempenho do sistema devido ao facto de esperar pela resposta acrescentando ao tempo de execução o tempo de latência da rede (Newman, 2015).

- Comunicação assíncrona:

Na comunicação assíncrona a comunicação entre microsserviços é efetuada com recurso a troca de mensagens entre eles por meio de um barramento leve de mensagens. Para isso, a forma mais comum de ser implementada é com utilização de um *Message Broker* que simplesmente garante uma entrega assíncrona e confiável (Fowler & Lewis, 2014). Os serviços que pretendem receber um determinado evento (Consumidores) indicam ao *Message Broker* o evento ao qual pretendem subscrever. Deste modo, quando os serviços que pretendem emitir um evento (Produtores) publicam o evento no *Message Broker*, este irá notificar os serviços que subscreveram este evento. Este tipo de comunicação apresenta um melhor desempenho devido ao facto de, ao contrário da comunicação síncrona, apresentar uma latência baixa pelo facto do serviço não ficar bloqueado à espera da resposta (Newman, 2015, p. 89).

Arquiteturas baseadas em comunicação assíncrona são mais dissociadas o que permite um sistema mais escalável e desacoplado, porém, são sistemas que envolvem maior complexidade e necessitam de uma maior monotorização (Newman, 2015, p. 110).

A Figura 3 ilustra a utilização de um *Message Broker* na comunicação entre microsserviços.

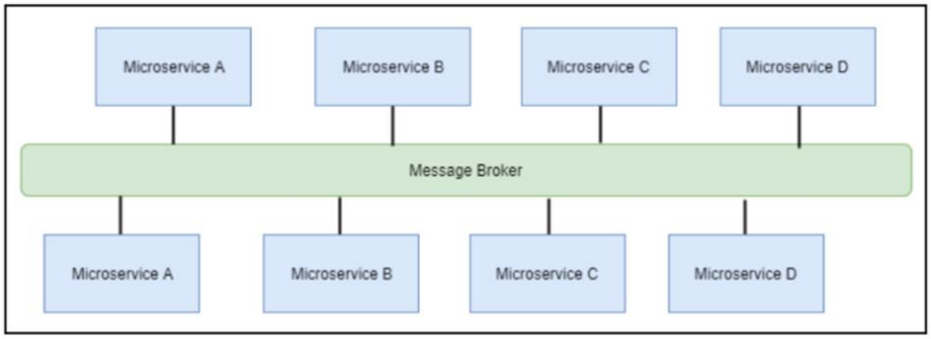

Figura 3 - Comunicação com recurso a um Message Broker (Pacheco, 2018, p. 54)

Também se poderá utilizar o *Message Broker* para comunicação síncrona, porém, a sua utilização possui mais relevo em comunicações assíncronas (Pacheco, 2018, p. 54).

2.1.2.7 Orquestração versus Coreografia

Para a execução de um fluxo de ações com vista a realização de um determinado processo de negócio existem dois estilos de arquitetura a seguir, sendo eles a Orquestração e a Coreografia.

Com a orquestração, necessitamos de um maestro, ou seja, de um serviço central capaz de orientar e conduzir o processo pelo seu fluxo de ações necessárias à sua realização. Este serviço deverá verificar as ações necessárias e, de forma sequencial, irá chamar os serviços responsáveis por cada uma das ações. Só após o retorno de sucesso do serviço executado é que este serviço central irá chamar o seguinte para executar, repetindo o processo até todas as ações serem realizadas (Newman, 2015, p. 91).

Por outro lado, com coreografia, não existe nenhum serviço central para orientar o processo. Cada serviço sabe as suas funções e executa-as reagindo às ações dos outros serviços. De forma assíncrona, cada serviço emite uma mensagem sobre o evento que acabou de executar e os serviços que estavam à escuta desse evento reagem ao evento e executam as ações que são necessárias de acordo com o evento recebido. Assim, basta que os serviços se coloquem à escuta de determinados eventos e, quando estes ocorrerem, executem as ações necessárias conforme esse mesmo evento (Newman, 2015, p. 91).

Em suma, os sistemas que apresentam uma abordagem através de coreografia tendem a ser mais dissociados o que os torna menos acoplados e são mais facilmente modificáveis (Newman, 2015, p. 92).

2.1.2.8 Documentação

Com os microsserviços a disponibilizarem interfaces de acesso para entidades externas é importante que esses métodos de acesso disponibilizados pelas API sejam registados (Newman, 2015, p. 406). Assim sendo, é importante efetuar uma boa documentação de como usar estas interfaces sendo que o Swagger (Swagger (Software), 2020) pode ser uma boa solução para este problema tendo em conta que garantem que a documentação está atualizada de acordo com a API do microsserviço. (Pacheco, 2018, p. 24) (Newman, 2015, p. 406).

2.1.2.9 Suprimir os pedidos externos

Os microsserviços, tal como qualquer outra aplicação, apresentam recursos computacionais finitos. Assim sendo, é importante restringir um consumo excessivo dos mesmos tendo em conta que não é exequível que um determinado microsserviço seja requisitado um elevado número de vezes num curto período. Posto isto, existem várias regras que podem ser usadas para restringir um consumo excessivo dos microsserviços como, por exemplo, limitar o número de pedidos por minuto de um determinado cliente (Pacheco, 2018, p. 27).

2.1.2.10 Monitorização

Em qualquer aplicação é importante efetuar a monotorização da aplicação em tempo real para detetar falhas no sistema e restaurá-las o mais breve possível. É relevante ter acesso à visualização do estado da aplicação bem como de métricas que indicam, por exemplo, o número de pedidos recebidos, tempos de resposta, CPU, memória, entre outros (Fowler & Lewis, 2014).

De facto, a monotorização é um aspeto relevante aquando da utilização de uma arquitetura de microsserviços. Com a existência de vários microsserviços, informações sobre cada um eles como o estado (se está ativo ou não) e métricas representativas de detalhes sobre número de pedidos, tempos médios de resposta, latência, CPU, memória, entre outros, são essenciais (Fowler & Lewis, 2014). Deste modo, o fornecimento de informações acerca dos serviços de forma individual auxilia a equipa de desenvolvimento numa deteção mais rápida de falhas nos serviços (Newman, 2015, p. 49).

Porém, ao contrário de uma aplicação monolítica, numa arquitetura de microsserviços a complexidade de implementação de monotorização é maior devido à necessidade de monitorizar vários serviços (Newman, 2015, Capítulo 8).

2.1.3 Padrões relacionados

2.1.3.1 SAGA

O padrão SAGA é uma padrão que visa a utilização de mecanismos com o intuito de manter a consistência de dados numa arquitetura de microsserviços (Richardson, 2018).

Assim sendo, uma saga é uma sequência de transações coordenadas onde a conclusão de uma transação despoleta o início de outra transação. Desta forma, um serviço, após concluir uma determinada ação, publica uma mensagem que irá despoletar o próximo passo da saga, ou seja, outra ação a realizar por um determinado serviço (Richardson, 2018).

Com este padrão, a utilização de mensagens para despoletar ações permite não só que os participantes da saga (microsserviços no caso de uma arquitetura de microsserviços) sejam "(…) fracamente acoplados, como também garante que a saga é concluída." (Richardson, 2018). Esta conclusão é garantida devido ao facto de, apesar dos destinatários poderem estar temporariamente indisponíveis, o *Message Broker* armazena as mensagens até que o mesmo volte a estar disponível e pronto para receber a mensagem (Richardson, 2018).

No entanto, no caso de deteção de alguma falha de acordo com as regras de negócios, é necessário que o sistema esteja preparado para reverter as transações. Devido ao facto de que uma ação efetuada num determinado serviço efetua alterações à sua camada de dados, um erro numa ação posterior não reverte esta transação de forma automática (Richardson, 2018). Assim sendo, após uma falha numa ação/etapa de uma saga, é necessário executar uma saga na ordem inversa para, desta forma, desfazer as transações (nominadas de

transações compensatórias) já efetuadas pelas ações/etapas anteriores da saga (Richardson, 2018).

A coordenação de uma saga pode ser estruturada seguindo uma lógica de orquestração ou de coreografia (secção 2.1.2.7). Na saga baseada em coreografia a sequência da saga é definida pela interação entre os participantes recorrendo a troca de eventos ao invés da orquestração onde existe um serviço central que indica qual o próximo serviço a executar (Richardson, 2018).

2.1.3.2 Event Sourcing

Como referido em (Fowler, 2005) "Podemos consultar o estado de uma aplicação para descobrir o estado atual do mundo, e isso responde a muitas perguntas. No entanto, há momentos em que não queremos apenas ver onde estamos, também queremos saber como chegamos lá.". Para colmatar esta necessidade, surge o padrão chamado *Event Sourcing*. "A ideia fundamental do *Event Sourcing* é garantir que todas as alterações no estado de uma aplicação sejam capturadas num objeto de evento e que esses objetos de evento sejam armazenados na sequência em que foram aplicados (…)" (Fowler, 2005).

Posto isto, aquando da modificação de uma determinada entidade do sistema, após se armazenar o seu novo estado, deve ser armazenado um objeto relativo a esse evento. Deste modo, para além de se poder visualizar o estado atual que foi armazenado, o facto de os eventos serem armazenados permite que se possa consultar esses eventos com o intuito de perceber como determinado objeto chegou ao estado em que se encontra. O armazenamento dos eventos possibilita, também, que se possa efetuar a reconstrução e/ou reversão dos estados reaplicando a sequência de eventos (Fowler, 2005) (Abdullin, 2010).

Em suma, o armazenamento de eventos poderá ser muito útil em casos onde é importante fornecer dados de auditoria para permitir que sejam visualizadas todas as ações que despoletaram a modificação dos dados do sistema. Para além disso, poderá ser útil em arquiteturas de microsserviços onde os serviços fracamente acoplados comunicam através de mensagens de eventos permitindo o processamento/reprocessamento de fluxos de eventos armazenados.

2.1.3.3 Command Query Responsibility Segregation

Command Query Responsibility Segregation (CQRS) é um padrão onde a ideia principal é a separação das responsabilidades de leitura e escrita de informação. A ideia é que, em sistemas mais complexos, se separe as responsabilidades de leitura e escrita em diferentes processos e, por vezes, em diferente *hardware* (Fowler, 2011) (Newman, 2015, p. 377). Esta separação permite um aumento de desempenho bem como a possibilidade de escalar cada serviço de forma independente (Fowler, 2011) (Newman, 2015, p. 377). Na maioria dos sistemas, a escrita de informação processa uma quantidade de transações inferior à leitura de informação, logo, o serviço de leitura necessita de escalar com mais frequência do que o serviço de escrita (Abdullin, 2010, p. 19).

Posto isto, o serviço de leitura terá apenas métodos de leitura, ou seja, de obtenção de dados enquanto que o serviço de escrita dedica-se ao processamento e armazenamento de dados (Abdullin, 2010). A separação das responsabilidades permite, também, que cada serviço tenha o seu próprio armazenamento de dados.

O CQRS encaixa-se bem em sistemas baseados em eventos com os serviços de leitura e escrita a poderem, também eles, comunicarem por eventos permitindo que estes serviços tirem partido da utilização de *Event Sourcing* (Fowler, 2011). Utilizando a comunicação por eventos, o serviço de leitura deverá capturar os eventos gerados pelo serviço de escrita de forma a sincronizar e a manter uma eventual consistência dos dados (Abdullin, 2010).

2.2 DevOps

DevOps (palavra composta pelos termos Desenvolvimento e Operações) é uma prática que evidencia a colaboração entre as equipas de desenvolvimento e de operações estabelecendo, ainda, a automatização do processo de entrega de software (Sharma, 2017).

De facto, esta prática enfatiza as tarefas de construir, testar e entregar software de forma "(…) rápida, frequente e mais confiável." (Sharma, 2017). Na base desta prática estão os recursos *Continuous Delivery* e *Continuous Integration* que, se não existirem, impossibilitam a existência da prática DevOps (Sharma, 2017).

2.2.1 Continuous Integration

A integração de alterações efetuadas num determinado sistema é uma tarefa essencial para o ciclo de vida de desenvolvimento da aplicação (Sharma, 2017).

Deste modo, as novas implementações devem ser integradas de forma regular com execução de testes sob a integração efetuada. Desta forma, é possível identificar previamente os riscos associados à integração de componentes como, por exemplo, falha de dependências, evitando que falhas possam ocorrer em fases posteriores do ciclo de vida de desenvolvimento do software (Sharma, 2017).

2.2.2 Continuous Delivery

Continuous Delivery é uma área do desenvolvimento de software onde o software é contruído de forma a ser implantado em produção a qualquer momento (Fowler, 2013).

De facto, o objetivo do *Continuous Delivery* é que os novos recursos desenvolvidos sejam fornecidos aos clientes o mais rápido possível. Deste modo, *Continuous Delivery* é aplicado sob o conceito de *Continuous Integration* e, após se efetuar a integração contínua do software à medida que este é desenvolvido, são construídos executáveis que devem ser testados em ambientes semelhantes ao ambiente de produção (Sharma, 2017) (Fowler, 2013). Assim

sendo, a prática de entrega frequente do software num ambiente de testes semelhante ao de produção, permite validar a qualidade do software antes de ser colocado em produção (Fowler, 2013) (Sharma, 2017). Este fluxo encontra-se representado pela Figura 4.

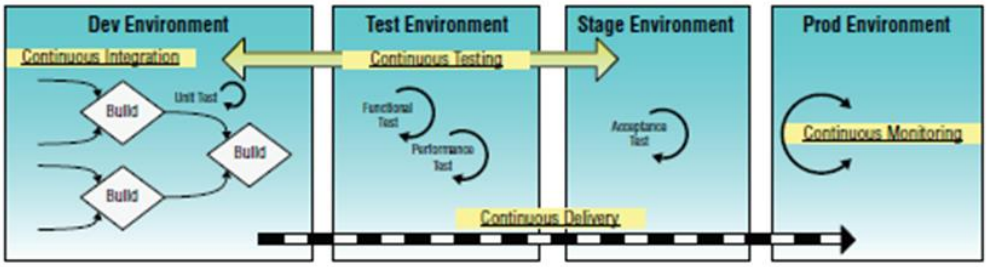

Figura 4 - *Continuous Delivery* (Sharma, 2017, fig. 1–4)

A entrega contínua de software e respetivos testes garante que, a qualquer momento, a versão mais recente da aplicação desenvolvida possa ser implantada em produção permitindo que se efetuem implantações de software de forma frequente (Fowler, 2013). No entanto, a entrega contínua requer a implementação de um *pipeline* que visa automatizar o processo de entrega de novas versões do software. Desta forma, à medida que se vai efetuando uma integração contínua, o software vai progredindo para outras etapas no processo de *Continuous Delivery* (como, por exemplo, implantação em ambiente de testes, execução de testes, entre outros) até ser implantado em produção que, geralmente, é a etapa final do *pipeline* (Sharma, 2017) (Fowler, 2013). As etapas retratadas são apresentadas na Figura 5.

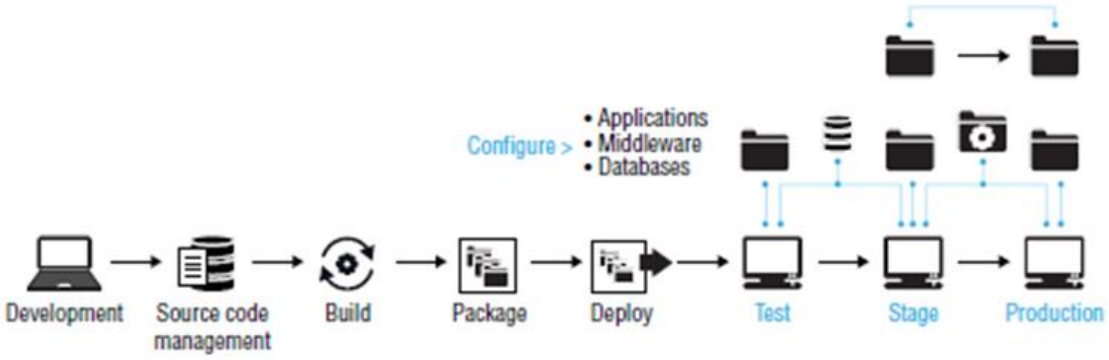

Figura 5 - *Pipeline* de entrega contínua (Sharma, 2017, fig. 1–3)

Em suma, a entrega contínua permite reduzir o risco de implantação devido ao facto de se implementarem alterações menores que são mais fáceis de corrigir no caso de deteção de problemas. A implantação de alterações de menores dimensões de forma frequente permite, também, obter *feedback* de forma mais rápida o que, por vezes, pode evitar que se desperdice tempo com alterações inúteis (Fowler, 2013).

2.3 *Strangler Application Pattern*

Ao contrário dos padrões apresentados anteriormente (secção 2.1.3), *Strangler Application Pattern* não é um padrão utilizado numa arquitetura de microsserviços, mas sim um padrão utilizado na migração para uma arquitetura baseada em microsserviços. O padrão *Strangler Application Pattern* é utilizado para, de forma incremental, migrar uma aplicação monolítica para uma aplicação baseada numa arquitetura de microsserviços (Behara, 2018).

De facto, este padrão consiste em desenvolver novas funcionalidades em microsserviços e, de forma gradual, extrair as funcionalidades existente do monólito para microsserviços. Desta forma, os microsserviços executarão em simultâneo com o monólito até que todas as funcionalidades sejam migradas e, consequentemente, o monólito desapareça ou se torne, também ele, um microsserviço (Richardson, 2018). Posto isto, para extrair as funcionalidades existentes, a ideia principal deste padrão é a utilização de um serviço (API Gateway, secção 2.1.2.4) que interceta as solicitações para o sistema antigo e as redireciona para os novos serviços (ou para o monólito caso a funcionalidade ainda não tenha sido migrada) permitindo, desta forma, que os serviços sejam substituídos de forma progressiva (Newman, 2015) (Richardson, 2018).

Para além disso, para efetuar a extração de serviços, é preciso um processo cuidadoso dada a necessidade de determinar como dividir o monólito eliminando dependências, dividir o modelo de domínio e, ainda, reformular a base de dados (Richardson, 2018). Outro aspeto importante é o facto de que geralmente os serviços extraídos necessitam de aceder a dados ou operações que se encontram no monólito e vice-versa. Desta forma, existe a necessidade de estabelecer uma forma de comunicação entre eles que pode ser implementado, por exemplo, com solicitações REST ou por troca de mensagens utilizando uma saga (secção 2.1.3.1) para manter a consistência dos dados (Richardson, 2018).

Em suma, *Strangler Application Pattern* é um padrão que visa efetuar a migração de forma gradual ao invés de reescrever toda a aplicação do zero com o intuito de obter benefícios imediatos (ainda que de forma incremental), reduzir o risco e esforço de migração e, ainda, evitar a reescrita de recursos que já não são necessários (Richardson, 2018).

2.4 Casos de estudo

Como referido anteriormente, e tendo em conta os problemas apresentados, o objetivo do projeto proposto passa pela migração da aplicação atual e pela melhoria da qualidade do processo de desenvolvimento de software. Assim sendo, são apresentados, de seguida, alguns casos de estudo de organizações que passaram por problemas e/ou objetivos semelhantes aos apresentados neste projeto.

2.4.1 Wix.com

A Wix.com é uma plataforma online que foi fundada em 2006 por Avishai Abrahami, Nadav Abrahami e Giora Kaplan e que permite aos seus utilizadores efetuar a criação de um site de forma simples, rápida e sem necessidade de conhecimentos de programação. Em 2019, de acordo com informações do próprio site, a plataforma é suportada em 190 países e fornecida a 150 milhões de utilizadores.

Segundo Aviran Mordo, vice-presidente de engenharia do Wix.com, "a arquitetura original da empresa era um enorme monólito Java que lidava com tudo" (Doerrfeld). A arquitetura inicial possuía as seguintes características: um servidor monolítico implementado em JAVA que lidava com tudo; a aplicação corria num Tomcat com armazenamento de dados numa base de dados MySQL; inexistência de considerações de desempenho, escalabilidade, testes e existência de dependências entre funcionalidades (Scaling wix with microservices architecture devoxx London 2015).

Com o aumento do número de serviços/funcionalidades (todos interligados na mesma aplicação) começaram a surgir problemas de estabilidade e escalabilidade. Aviran Mordo constatou que "os bugs em uma parte do sistema tinham o potencial de derrubar todo o sistema" (Doerrfeld).

Posto isto, em 2010, para reverter esta situação, a Wix.com começou a dividir a aplicação em serviços menores com o intuito de escalar, melhorar a qualidade dos mesmos e aumentar o desempenho (Doerrfeld). Foi aplicada uma arquitetura baseada em microsserviços e no seguimento da mesma foram seguidas as seguintes linhas de orientação (Scaling wix with microservices architecture devoxx London 2015):

- Cada microsserviço tem a sua base de dados (se a sua utilização for necessária);
- Apenas um serviço deverá escrever numa determinada tabela de base de dados;
- Existência de serviços que apenas leem diretamente da base de dados (por questões de performance).

A Wix.com apresenta, atualmente, uma *stack* tecnológica diversificada desde a implementação de microsserviços em JAVA, Spring, Scala e Jetty, a utilização de distintas formas de armazenamento de dados como MySQL e MongoDB e uso de diferentes serviços *Cloud* como, por exemplo, Google Cloud e Amazon Cloud (Scaling Wix to 60M Users—From Monolith to Microservices—Wix Tech Stack).

Segundo Yoav Abrahami, arquiteto chefe da Wix.com, hoje, a Wix.com, possui “mais de 100 microsserviços. A maioria é baseada na linguagem de programação Scala, com Jetty, Spring e a nossa *framework* interna” (*Scaling Wix to 60M Users—From Monolith to Microservices—Wix Tech Stack*).

Depois de entrar na abordagem de microsserviços, a Wix.com relata como principais dificuldades a comunicação entre os microsserviços, a resolução de falhas e problemas de *debugging* (Doerrfeld).

Efetuada a migração, a Wix.com afirma que aumentou a velocidade de implementação e melhorou a qualidades dos seus serviços fornecidos, o que levou Aviran Mordo a afirmar que "a migração é uma das melhores decisões que já tomamos" (Doerrfeld).

2.4.2 SmartThings

A SmartThings é uma empresa subsidiária da Samsung, com sede na Califórnia, que fornece experiências domésticas de *Internet of Things* aos seus clientes. A SmartThings tem como objetivo atingir velocidades elevadas com tempos de resposta inferiores a 400 milissegundos e acelerar o ciclo de desenvolvimento de forma a entregar rapidamente novas versões de software (Case Study).

A arquitetura inicial da SmartThings era constituída por um servidor monolítico com aproximadamente 550 serviços o que resultou em tempos de execução inaceitáveis durante picos de utilização. Para além disso, eram necessários imensos testes manuais após alterações ao software e os testes automatizados demoravam imenso tempo a executar o que dificultava a entrega contínua de novos desenvolvimentos. Deste modo, com o crescimento da empresa, foi impossível escalar e o desempenho do software era cada vez menor (Case Study).

Assim sendo, a SmartThings constatou que era preciso um sistema mais flexível e, para isso, seguiu uma arquitetura de microsserviços. Esta solução permitiu maior eficiência do sistema bem como o desenvolvimento, execução de testes e entregas mais ágeis de novas funcionalidades já que os serviços funcionavam de forma independente (Case Study).

2.4.3 Danske Bank – FX Core

Danske Bank é o maior banco Dinamarquês e uma das principais instituições financeiras do norte da europa. O Danske Bank possui um sistema de câmbio de moedas chamado FX Core que "atua como um *gateway* entre os mercados internacionais e os clientes da Danske Bank" (Bucchiarone, Dragoni, Dustdar, Larsen, & Mazzara, 2017). Este sistema teve um grande crescimento resultante do aumento das transações efetuadas (Bucchiarone et al., 2017).

Inicialmente, o FX Core era um sistema monolítico aglomerando todos os serviços e fornecendo API como interfaces para clientes e fornecedores externos e um sistema de mensagens (utilizando RabbitMQ) para "(…) delegar as solicitações recebidas (…)" por fornecedores externos (Bucchiarone et al., 2017). A API de fornecedores externos consistia em múltiplos serviços, ou seja, um serviço por cada fornecedor (Bucchiarone et al., 2017).

Atualmente, com a reestruturação do FX Core, este passou a ser baseado em microsserviços e substituiu na totalidade o sistema monolítico. O sistema interno foi dividido em

microsserviços independentes (e com diferentes linguagens de programação) e com a sua própria base de dados. Continua a ser disponibilizada uma API para fornecedores externos e a comunicação entre a API e os microsserviços é efetuada pelo RabbitMQ (Bucchiarone et al., 2017).

Posto isto, com a reengenharia do sistema foi possível diminuir a complexidade do mesmo bem como uma redução do acoplamento, aumento da coesão e integração de sistemas mais simplificada. Ao contrário do que acontecia com o sistema monolítico, o tamanho dos serviços é menor o que facilita a sua manutenção (Bucchiarone et al., 2017).

2.4.4 Netflix

A Netflix é uma empresa Americana, fundada em 1997, que fornece filmes e séries via *streaming* e que conta atualmente com milhões de subscritores.

De facto, a Netflix começou com uma arquitetura monolítica, porém, aquando do surgimento de várias quebras do sistema, decidiram mudar o sistema inteiro para uma arquitetura de microsserviços. O objetivo era que, com esta nova arquitetura, fosse possível ter um sistema escalável com maior disponibilidade e velocidade de serviço. Assim, o sistema foi dividido em mais de 700 microsserviços independentes (reduzindo o acoplamento) o que permite que se efetuem alterações numa parte do sistema sem afetar o resto do sistema. Para melhor aproveitamento de recursos, os microsserviços encontram-se implantados na *Cloud* da Amazon (AWS) de forma a efetuar uma melhor gestão dos recursos necessários (Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience – HYS Enterprise).

A arquitetura de microsserviços da Netflix vai muito além dos microsserviços. Esta arquitetura possui módulos para descoberta dos serviços existentes, comunicação entre serviços, roteamento, monitoramento, entre outros. A Figura 6 mostra um pequeno esboço da arquitetura de microsserviços da Netflix.

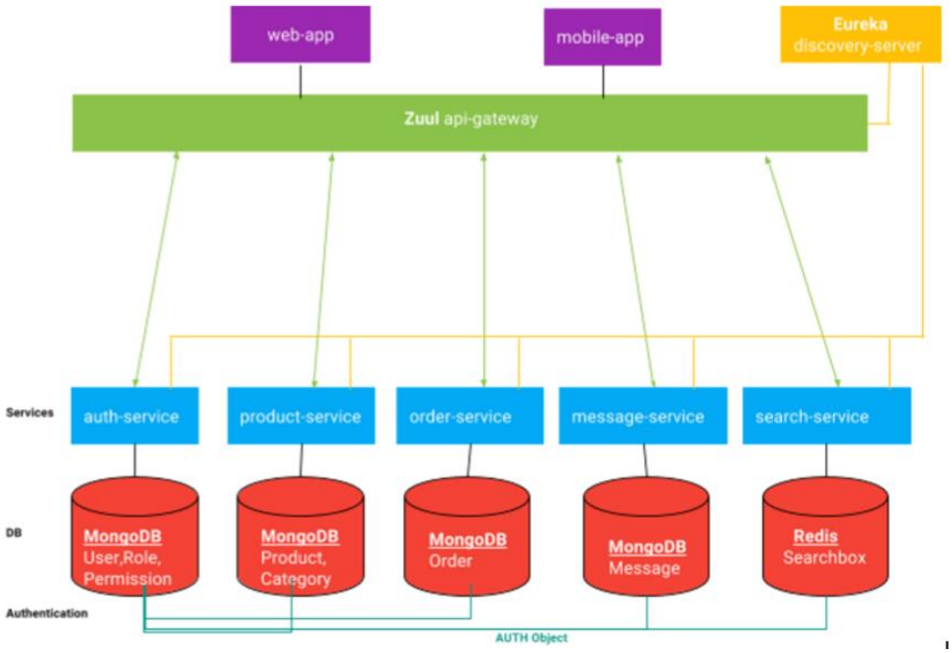

Figura 6 - Microservices implementation—Netflix stack (Thennakoon, 2019, fig. 1)

Como podemos destacar na Figura 6, o sistema da Netflix é separado em diferentes camadas. Existem vários pequenos serviços independentes com responsabilidades bem definidas e cada um com o seu próprio armazenamento de dados. Existe, também, um servidor dedicado à descoberta de serviços (Eureka discovery-server). Basicamente, todos os serviços registam-se neste servidor que, desta forma, armazena a informação detalhada da localização desses serviços para que, quando um serviço precisar de comunicar com outro, seja este servidor a fornecer esta informação para que posteriormente seja efetuada a conexão. Outro servidor importante nesta arquitetura é o Zuul API-Gateway. Este serviço é o responsável por fornecer às aplicações externas (aplicação *web* e aplicação móvel) o acesso aos microsserviços. Também ele necessita de pedir informação ao servidor de descoberta de serviços para saber para que serviço deverá reencaminhar o pedido. Este serviço é, também, o responsável por fazer o balanceamento de cargas para evitar a sobrecarga dos serviços (Thennakoon, 2019).

A Netflix foi das primeiras empresas a adotar sistemas baseados em arquiteturas de microsserviços e, posto isto, estabeleceram várias práticas recomendadas para implementar uma arquitetura de microsserviços (Microservices at Netflix, 2015), tais como:

- Cada microsserviço deverá ter um armazenamento de dados próprios (e cada microsserviço utilizar o tipo de base de dados que melhor se adequar);
- Microsserviços não devem partilhar as estruturas de dados;
- Se queremos adicionar novas funcionalidades a um microsserviço que está a funcionar, devemos criar outro microsserviço de forma a testá-lo

iterativamente e, só quando estiver estável como o original, é que devemos fazer a junção dos dois.

- Implantar os microsserviços em contentores porque, desta forma, só precisamos de uma ferramenta para implantar todo o sistema;
- Implantar os microsserviços na *Cloud* (assim é possível comprarmos os recursos necessários e só pagamos o que realmente usarmos).

2.5 Alternativas

Tendo como base as principais características, práticas, padrões e os casos de estudo analisados, nesta secção são apresentadas diferentes abordagens quer arquiteturais quer tecnológicas a considerar para este projeto. Assim sendo, a análise desta secção servirá de base para a comparação e seleção da arquitetura e das tecnologias a utilizar no âmbito deste projeto (secção 3.5).

2.5.1 Arquitetura

2.5.1.1 Bibliotecas

Uma abordagem que pode ser utilizada para o problema mencionado é a decomposição da aplicação em bibliotecas compartilhadas. Deste modo, a aplicação poderia ser decomposta em diversas bibliotecas distintas que continham toda a lógica de negócio associada à respetiva funcionalidade.

Assim sendo, cada biblioteca seria responsável por determinada funcionalidade o que permitiria uma simplificação da manutenção do mesmo sendo, também, facilmente reutilizável por outras bibliotecas (Newman, 2015, p. 30).

Porém, apesar de poder ser incorporada em qualquer linguagem de programação, esta técnica não permite a existência de heterogeneidade tecnológica. Normalmente, para reutilizar uma biblioteca é preciso que a mesma se encontre na mesma linguagem de programação e, ainda, é necessário que a mesma seja executada na mesma plataforma (Newman, 2015, p. 30). Outra desvantagem é que, apesar de se decompor a aplicação em diferentes bibliotecas, como as mesmas têm de executar sob a mesma plataforma, não é possível escalar as bibliotecas de forma independente (Newman, 2015, p. 30). Deste modo, também não é possível a implantação de bibliotecas de forma independente o que implicaria que uma modificação de uma biblioteca levasse a que toda a aplicação fosse reimplantada (Newman, 2015, p. 30).

2.5.1.2 Microsserviços

Outra abordagem possível para a reestruturação da arquitetura é a decomposição do monólito existente em microsserviços. Assim sendo, é necessário identificar contextos limitados dentro do monólito que possibilitarão a divisão em partes mais pequenas, ou seja, em microsserviços (Newman, 2015).

Deste modo, cada microsserviços possuiria o seu próprio contexto com as respetivas funcionalidades. Ao contrário das bibliotecas, uma arquitetura baseada em microsserviços poderá possuir heterogeneidade tecnológica com os microsserviços a poderem ser implementados com diferentes linguagem de programação, tal como referido na secção 2.1.1.2. Outras caraterísticas dos microsserviços são serem autónomos (secção 2.1.1.1) e resilientes (secção 2.1.1.3) o que facilita o processo de manutenção e adição de novas funcionalidades. Sendo os microsserviços autónomos, uma alteração num microsserviço apenas implica que o mesmo tenha de ser implantado (secção 2.1.1.6) o que é uma vantagem para a implantação de uma estratégia de *Continuous Delivery* e *Continuous Integration.* Assim, cada microsserviço dispõe do seu próprio ciclo de vida permitindo que alterações ao seu código sejam mais rapidamente implantadas. O facto de serem autónomos permite, também, tornar o sistema escalável (secção 2.1.1.5) podendo tirar partido de mais *hardware* em microsserviços que precisem de maior desempenho.

2.5.2 API Gateway, API Management e Service Discovery

De forma a que os microsserviços não sejam acedidos diretamente pelas aplicações clientes, será necessário adicionar uma camada intermédia que faça a ligação entre as aplicações clientes e os microsserviços.

Desta forma, as aplicações clientes deverão efetuar os pedidos a esta camada que ficará responsável por efetuar o reencaminhamento do pedido ao respetivo microsserviço. Para além disso, esta camada poderá apresentar outras funcionalidades para além do roteamento como, por exemplo, balanceamento de carga, monitorização, entre outros.

De seguida serão descritas tecnologias que poderão ser utilizadas para solucionar este problema.

2.5.2.1 JHipster API Gateway + JHipster Registry

O JHipster é um gerador de aplicações gratuito que permite gerar aplicações *web* com API desenvolvidas com Spring Boot. Este gerador de aplicações permite, entre outras opções, gerar um API Gateway desenvolvido com Spring Boot que efetue o roteamento HTTP entre as aplicações externas e os microsserviços (*API Gateway*).

Também fornecido pelo JHipster, o JHipster Registry é um servidor Eureka que é responsável por fazer a descoberta de serviços. O código fonte é todo ele fornecido de forma gratuita sendo que a sua API é desenvolvida em Spring Boot e a interface com o utilizador com recurso a Angular (*JHipster Registry*).

Posto isto, a arquitetura da solução utilizando o JHipster API Gateway e o JHipster Registry é apresentada na Figura 7.

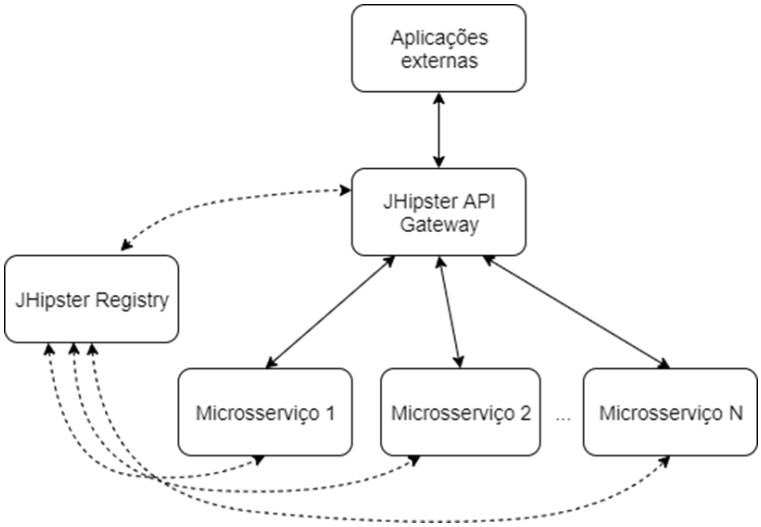

Figura 7 - Arquitetura JHipster API Gateway e JHipster Registry (API Gateway)

Os microsserviços e o JHipster API Gateway deverão, ao serem inicializados, registar-se, por meio de configurações, no JHipster Registry que, desta forma, irá ter conhecimento da localização dos serviços disponíveis. Assim sendo, o JHipster API Gateway poderá, através de informações fornecidas pelo JHipster Registry, reencaminhar o pedido efetuado por aplicações externas para o respetivo microsserviço. Para isso, as solicitações HTTP têm de seguir um determinado padrão que consiste em indicar o nome do microsserviço de onde pretendem obter o recurso. Por exemplo, se existir um microsserviço com o nome "warehouse" que fornece um recurso "products" que retorna a lista dos produtos existentes no armazém, a solicitação HTTP feita pelas aplicações externas ao JHipster API Gateway deverá ser a seguinte: "warehouse/products". Deste modo, o JHipster API Gateway obterá a localização do microsserviço cujo nome é "warehouse" junto do JHipster Registry e reencaminhará a solicitação "products" (*API Gateway*).

No caso de existirem várias instâncias do mesmo microsserviço registadas no JHipster Registry, o JHipster API Gateway efetua balanceamento de carga de maneira a distribuir os pedidos pelas diferentes instâncias em execução. O JHipster API Gateway irá, também, detetar e remover instâncias que não se encontrem disponíveis para que não sejam encaminhadas mais solicitações para as mesmas (*API Gateway*).

Esta arquitetura permite, também, que sejam utilizados diferentes tipos de autenticação. No momento de geração do JHipster API Gateway existe a possibilidade de usar a autenticação com a utilização do padrão OAuth 2.0, do padrão JWT ou com a utilização de um servidor de autenticação externo. No caso da opção OAuth 2.0 ou utilização de um servidor de autenticação externo será gerado todo o código necessário para a utilização da respetiva opção sendo apenas necessário efetuar pequenas configurações. Se o pretendido for utilizar o padrão JWT, será gerado todo o código de autenticação no JHipster API Gateway que ficará

responsável pela autenticação e pela geração de *tokens* de acesso que são enviados para os microsserviços juntamente com as solicitações efetuadas. Estes *tokens* contêm informações relativamente à autenticação e à autorização que podem ser utilizadas pelos microsserviços não sendo necessário obter esta informação através de solicitações a sistemas externos. O facto de se evitar estas solicitações externas constitui uma vantagem para garantir escalabilidade da solução (*API Gateway*).

O JHipster API Gateway dispõe de limites de acessos aos microsserviços facilmente configuráveis através de ficheiros de configuração. Assim, é possível limitar o número de solicitações num determinado período realizadas por um utilizador ou por endereço IP. Estas validações podem ser facilmente adaptadas às necessidades que possam surgir através de implementações customizadas de regras pretendidas (*API Gateway*).

Como referido anteriormente, o JHipster Registry é um servidor Eureka responsável por fazer a descoberta de serviços. Com uma API implementada com Spring Boot e a interface com o utilizador com recurso a Angular, o código fonte é disponibilizado de forma gratuita no GitHub. Este servidor também poderá ser executado a partir da imagem Docker que se encontra disponível no Docker Hub.

Este servidor permite, também, monitorizar e gerir os microsserviços registados. Para isso, disponibiliza diversas métricas acerca dos microsserviços com a apresentação de informações relativamente ao estado do microsserviço (indicação de se está ou não ativo), tempos médios de resposta dos microsserviços por recurso, detalhes sobre ambiente de execução dos microsserviços (como, por exemplo, memória e CPU em utilização), entre outros. Estas informações são apresentadas sob diversas formas como, por exemplo, através de gráficos. Entre estas informações, o JHipster Registry permite, através de simples configurações, que os *logs* dos microsserviços sejam apresentados numa página *web* o que auxilia o processo de monitorização dos microsserviços. Devido ao facto de ser possível aceder diretamente ao código fonte e estando este enquadrado com a *stack* tecnológica da Flowinn existe a possibilidade de customizar de acordo com as necessidades existentes. Assim, será possível, por exemplo, apresentar a informação de um forma mais objetiva ao problema em questão para, desta forma, otimizar a leitura das diferentes métricas e a respetiva interpretação que poderá ser útil na deteção de problemas do sistema (*JHipster Registry*).

A nível de documentação, tanto o JHipster API Gateway como o JHipster Registry disponibilizam um *Swagger* e automaticamente as informações relativamente às interfaces expostas pelos microsserviços são apresentadas. Assim sendo, a documentação pode facilmente ser detalhada (*API Gateway*).

Tanto no JHipster API Gateway como o JHipster Registry existe a possibilidade de aceder diretamente ao código fonte que se encontra desenvolvido em Spring Boot e em Angular no caso da interface disponibilizada pelo JHipster Registry. Assim sendo, a sua *stack* tecnológica é uma vantagem devido ao facto de estar enquadrada com a *stack* tecnológica da Flowinn o que poderá permitir que facilmente se façam adaptações que poderão ser necessárias. Desta

forma, a flexibilidade e customização presente nestas aplicações surgem como pontos importantes para a adaptação das mesmas ao problema em questão.

2.5.2.2 JHipster API Gateway + HashiCorp Consul

Aquando da geração do JHipster API Gateway, além da opção de gerar as configurações para a utilização do JHipster Registry, existe a possibilidade de gerar as configurações para utilizar o HashiCorp Consul como servidor de descoberta de serviços.

Desta forma, a arquitetura da solução seria semelhante à descrita anteriormente (secção 2.5.2.1) sendo que a única diferença passa pela substituição do JHipster Registry pelo HashiCorp Consul como servidor de descoberta de serviços, tal como é ilustrado pela Figura 8.

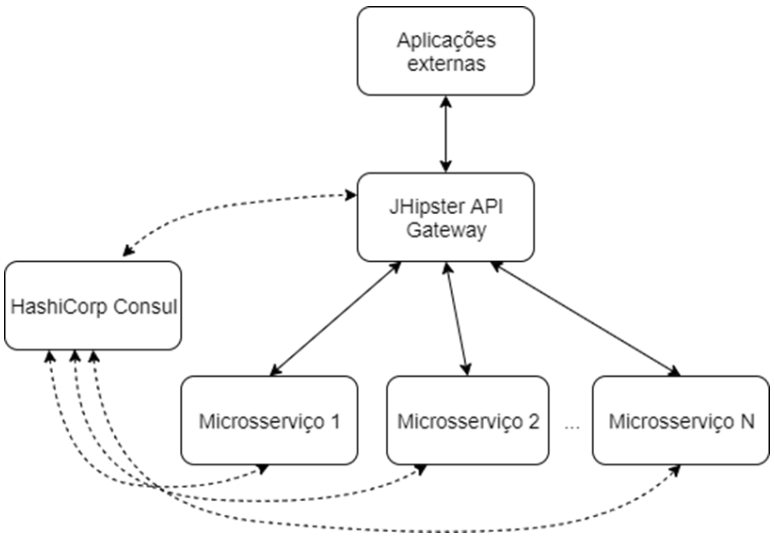

Figura 8 - Arquitetura JHipster API Gateway e HashiCorp Consul

Desenvolvido pela HashiCorp, o Consul é um servidor que permite efetuar a descoberta de serviços. Tal como o JHipster Registry, este é um servidor gratuito, porém não é possível aceder ao seu código fonte (*Consul*).

Para além de fornecer as localizações dos serviços, o Consul efetua verificações de integridade de serviços, ou seja, fornece informação se um determinado microsserviço está a responder ou não (*Introduction—Consul by HashiCorp*). Para esta funcionalidade, o Consul apresenta mais consistência do que a que é apresentada pelo JHipster Registry (*Consul vs Eureka | What are the differences?*).Também fornece informações que permitem monitorizar os microsserviços, porém, com menos quantidade e detalhe do que a apresentada pelo JHipster Registry.

Com a utilização do JHipster API Gateway e do Consul dispomos, também, da possibilidade de construir de forma simples a documentação das interfaces expostas pelos microsserviços através do Swagger, porém, a mesma só ficará disponível para visualização no JHipster API Gateway.

2.5.2.3 WSO2 API Manager

WSO2 API Manager é uma ferramenta gratuita que permite fazer a gestão das API existentes nos diversos microsserviços (*API Management—On-Premise and in the Cloud*). Apesar de gratuito, tal como acontece com HashiCorp Consul, não existe acesso ao código fonte de forma a customizar com base em necessidades que possam surgir.

De facto, o WSO2 API Manager permite que sejam registadas as API dos microsserviços que se pretende que sejam expostas para utilização por parte de aplicações externas. A sua função é semelhante à apresentada pelo JHipster API Gateway fazendo o roteamento HTTP entre as aplicações externas e os microsserviços. A arquitetura de uma possível solução utilizando o WSO2 API Manager poderá ser representada pela Figura 9.

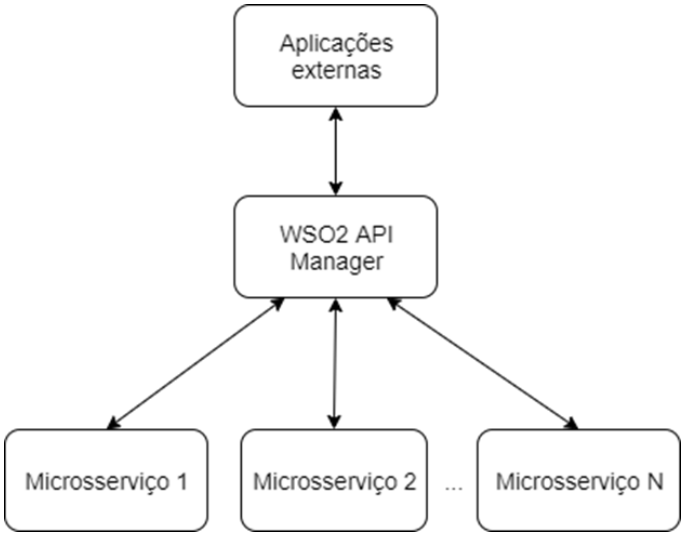

Figura 9 - Arquitetura WSO2 API Manager

Como é possível verificar na Figura 9, ao contrário das soluções anteriores, o WSO2 API Manager não integra na sua versão base um servidor de descoberta de serviços sendo necessário registar os serviços existentes de forma manual. Para usufruirmos de um servidor de descoberta de serviços é necessário integrar com o componente WSO2 API Microgateway que poderá fazer essa função (*Service Discovery with WSO2 API Microgateway*). Porém, a configuração de um servidor de descoberta de serviços não é tão simples como as alternativas anteriormente descritas. Para além disso, ao contrário do JHipster Registry e do HashiCorp Consul, o WSO2 API Manager não dispõe de verificação de integridade dos microsserviços.

O WSO2 API Manager concede as funcionalidades de autenticação e autorização. Para a autenticação fornece diferentes opções como, por exemplo, Basic Authentication e OAuth2. Para proteção da aplicação, o WSO2 API Manager permite limitar o número de solicitações num determinado período. Permite que estes limites sejam aplicados por recurso, por utilizador, por endereço IP, entre outros (*Throttling Use-Cases—WSO2 API Manager Documentation 3.0.0*).

A nível de monitorização, o WSO2 API Manager integra na sua solução o componente WSO2 Analytics (*Working with Analytics—WSO2 Enterprise Integrator 6.x.x—WSO2 Documentation*) que fornece diversos gráficos e estatísticas que permitem monitorizar os diferentes recursos disponibilizados pelos microsserviços. Este componente apresenta diversas informações acerca de tempos médios de resposta, latência, quantidade de chamadas por recurso, entre outras. Porém, dados acerca do ambiente de execução dos microsserviços (como, por exemplo, memória utilizada e CPU utilizado) não são fornecidos.

Relativamente à documentação das funcionalidades das API, o WSO2 API Manager contém o componente API Documentation. Ao contrário das outras abordagens, este componente permite que a documentação seja fornecida de diferentes tipos, nomeadamente, "In-line" (onde a documentação é colocada diretamente através de um editor de texto disponibilizado), "URL" (permite colocar ligações para documentos localizados noutras páginas *web*), "File" (submeter ficheiros que contém a documentação) e "Markdown" com documentação adicionada através de documentos *markdown* (*Add API Documentation—WSO2 API Manager Documentation 3.0.0*).

2.5.3 Message Broker

Como abordado anteriormente (secção 2.1.2.6), para estabelecer a comunicação assíncrona entre microsserviços, a forma mais comum é recorrer à utilização de um *Message Broker* (Fowler & Lewis, 2014). Para realizar a comunicação assíncrona entre microsserviços serão, de seguida, apresentados dois *Message Broker* como possíveis alternativas, RabbitMQ e Apache Kafka.

2.5.3.1 RabbitMQ

O RabbitMQ é um *Message Broker* gratuito que permite efetuar troca de mensagens assíncrona com suporte para diversos protocolos de mensagens (*Messaging that just works—RabbitMQ*).

Para além de permitir efetuar troca assíncrona de mensagens, o RabbitMQ também permite que seja realizada comunicação síncrona. A sua arquitetura de comunicação segue um modelo onde o *broker* é inteligente enquanto os consumidores não o são. Deste modo, o *broker* monitoriza o estado do consumidor e encontra-se focado em garantir uma entrega de mensagens consistente (Levy, 2019).

O RabbitMQ é também uma plataforma madura disponível para ser utilizada em diversas linguagens de programação. Ao nível do desempenho, para processar grandes quantidades de dados por segundo, o RabbitMQ necessita de mais recursos computacionais para o conseguir (Levy, 2019) (Humphrey, 2017).

2.5.3.2 Apache Kafka

Como alternativa ao RabbitMQ, o Apache Kafka é uma plataforma de *streaming* gratuita que permite a troca de mensagens com recurso ao protocolo TCP (*Apache Kafka*).

O Apache Kafka foi projetado para processar grandes volumes de dados de forma rápida e escalável sendo que, desta forma, ao contrário do RabbitMQ, permite processar grandes quantidades de mensagens por segundo com pouca sobrecarga. Deste modo, o Apache Kafka apresenta mais benefícios ao nível de desempenho permitindo uma maior taxa de transferência de mensagens por segundo sem necessidade de utilização de muitos recursos computacionais (Humphrey, 2017).

O modelo utilizado pelo Kafka é diferente do modelo do RabbitMQ na medida em que o consumidor é inteligente ao contrário do *broker*. Desta forma, o Kafka apenas mantém as mensagens não lidas não se preocupando em identificar as mensagens lidas pelos consumidores. Por outro lado, permite que os registos sejam mantidos por um período configurável servindo como uma espécie de *log* de mensagens, ao contrário do RabbitMQ onde as mensagens são eliminadas após serem entregues (Levy, 2019). Dado este armazenamento de mensagens, é possível repetir o fluxo de eventos (Humphrey, 2017) uma vez que as mensagens são armazenadas em conjunto com um registo temporal que permite que as mesmas sejam, assim, armazenadas seguindo uma ordem temporal (Levy, 2019).

2.5.4 Ferramentas de Integração Contínua

As ferramentas de integração contínua permitem automatizar diversas tarefas afetas ao ciclo de vida do produto. Estas ferramentas serão importantes para a manutenibilidade do produto sendo que o intuito será a sua utilização para a automação de tarefas tais como, compilação do projeto e consequente geração de artefactos (binários para execução da aplicação), execução de testes, implantação, entre outras.

De seguida, serão mencionadas duas ferramentas que poderão auxiliar neste processo que são o caso do Jenkins e do Bitbucket Pipes.

2.5.4.1 Jenkins

O Jenkins é um servidor de automação "que que pode ser usado para automatizar todos os tipos de tarefas relacionadas à criação, teste e entrega ou implantação de software" (*Jenkins User Documentation*).

Carateriza-se por ser um servidor de código aberto e, por isso, pode ser utilizado de forma gratuita se hospedado internamente. O Jenkins é uma ferramenta bastante flexível com diversos *plugins* para auxiliarem nas diversas tarefas que os *pipelines* podem ter e, ainda, a possibilidade de criação dos próprios *plugins*. Através do Jenkinsfile, é possível a criação de um *script* com as tarefas a realizar pelo Jenkins que poderá ser incluído no repositório de controlo de versões junto com o código fonte da aplicação e, assim, efetuar controlo de versões sob o mesmo (*Jenkins*) (*Bitbucket Pipelines vs Jenkins Pipeline*).

Apesar de ser gratuito, necessita de ser hospedado o que implica a utilização de servidores internos ou a alocação em *Cloud* que implicará custos extra. A necessidade de ter de ser hospedado implica, também, que se tenha de preparar todo o ambiente necessário para a

execução do mesmo como, por exemplo, as suas dependências e instalação do Docker que será necessário para a execução de algumas tarefas do *pipeline.*

Embora possua diversos *plugins*, a gestão do Jenkins poderá implicar algum conhecimento de configuração devido ao facto de existir alguma complexidade para efetuar a gestão do mesmo, o que poderá implicar a necessidade de despender mais tempo para o fazer (*Bitbucket Pipelines vs Jenkins Pipeline*). A integração com o JIRA poderá ser um aspeto importante no sentido de otimizar o processo de desenvolvimento do produto e, apesar de existir um *plugin* para integrar o Jenkins com o JIRA, atualmente este ainda é muito limitado e oferece poucas funcionalidades de integração.

2.5.4.2 Bitbucket Pipes

Desenvolvido pela Atlassian, o Bitbucket Pipes (ou Bitbucket Pipelines) é um serviço incorporado no Bitbucket que permite automatizar diversas tarefas como "criar, testar e implantar código (..) com base num arquivo de configuração (…)" que ficará no repositório de controlo de versões (*Introdução aos Pipelines Bitbucket—Atlassian Documentation*).

Este é um serviço pago, porém, a Flowinn, apesar de não o usar, já dispõe de licenças do Bitbucket do tipo *Standard* que fornece 2500 minutos por mês para utilização do mesmo (Atlassian). Este serviço pressupõe a execução em ambiente externo sendo que o Bitbucket Pipes executa sob *containers* Docker o que simplifica a configuração do ambiente necessário para as tarefas a realizar permitindo, ainda, que se utilizem diferentes imagens Docker para as diferentes etapas do *pipeline* (*Introdução aos Pipelines Bitbucket—Atlassian Documentation*).

Ao nível de gestão, o Bitbucket Pipes surge com vantagem devido à simplicidade que apresenta para efetuar toda a configuração. Este serviço dispõe de uma interface para efetuar a configuração do *pipeline* simples e intuitiva sendo apenas necessário selecionar e ativar a configuração pretendida para o *pipeline* de entre as opções fornecidas. Assim, o Bitbucket Pipes apresenta-se como um serviço de simples configuração e com ferramentas intuitivas para efetuar a sua configuração. (*Bitbucket Pipelines vs Jenkins Pipeline*) (*Bitbucket vs Jenkins | What are the differences?*).

Outro aspeto a considerar no Bitbucket Pipes é a integração com o JIRA. Esta integração apresenta-se como mais completa do que a apresentada pelo Jenkins com a possibilidade de possuir uma maior interação com esta plataforma (*Bitbucket vs Jenkins | What are the differences?*). Este poderá ser um aspeto importante para a Flowinn dada a utilização do JIRA para auxiliar no processo de monitorização e gestão das tarefas afetas aos projetos. Assim sendo, o Bitbucket Pipes poderá ser utilizado para, de forma automatizada, interagir com as tarefas do JIRA podendo, inclusive, alterar os respetivos estados após implantação das respetivas funcionalidades.

3 Análise de valor

Análise de valor é um processo sistemático de revisão e avaliação de um produto que utiliza diversas técnicas com o intuito de aumentar o valor do produto com o menor custo possível e sem comprometer a confiabilidade do mesmo (Rich & Holweg, 2000).

Posto isto, antes do desenvolvimento da solução, será necessário extrair a oportunidade de negócio a partir da identificação do contexto e do problema bem como a identificação de alternativas e consequente análise e avaliação para, desta forma, tentar selecionar as mais adequadas para o problema em questão.

3.1 Fuzzy Front End

Fuzzy Front End é um processo a partir do qual se identificam e analisam as oportunidades antes de se efetuar o desenvolvimento do produto. Surge, assim, como o ponto de partida auxiliando na extração da oportunidade bem como da identificação, análise e seleção de alternativas sendo importante no processo de tomada de decisão (Wikipedia contributors, 2018).

De facto, para a fase inicial do processo de inovação, o *Fuzzy Front End* define cinco elementos de atividade controlada que são os seguintes: identificação de oportunidade, análise de oportunidade, geração de ideias, seleção de ideias e definição de conceito (Koen et al., 2002).

Assim sendo, será, de seguida, apresentado o contexto atual e o respetivo problema para, a partir do mesmo, se iniciar a identificação e análise da oportunidade. Posteriormente, será apresentada a proposta de valor que se pretende atingir e serão, também, apresentadas possíveis abordagens e respetiva seleção para a resolução do problema.

3.2 Contexto e problema

A plataforma eDocuments é uma aplicação *web* assente numa arquitetura monolítica responsável por faturação eletrónica e EDI, que tem como principal funcionalidade a troca de ficheiros de dados entre várias entidades/empresas que se encontram registadas na plataforma.

Esta plataforma apresenta uma interface com o utilizador desenvolvida em AngularJS, uma API do lado do servidor implementada com recurso a Spring Boot e um sistema de gestão de base de dados utilizando MySQL.

Nos últimos tempos, esta aplicação tem apresentado um crescimento de entidades/empresas o que resultou num aumento de sistemas a integrar e dados/documentos a processar. Consequentemente, com este crescimento têm surgido problemas de desempenho e de memória da aplicação na sequência do processamento de dados/documentos. Para além disso, o crescimento da aplicação pressupõe a existência de uma evolução constante com a recorrente necessidade de melhoria, manutenção e incremento de novas funcionalidades no sistema. Porém, a arquitetura atual tem dificultado este processo devido ao elevado acoplamento entre serviços que dificulta a tarefa de efetuar modificações/adições de código fonte.

Simultaneamente, são reconhecidas limitações no processo de desenvolvimento de software que, entre outros aspetos, tem dificultado, também, a manutenção e evolução da aplicação. Atualmente, é utilizado o Jenkins para automatizar partes do processo de desenvolvimento, porém, o mesmo só tem sido utilizado para efetuar as seguintes tarefas:

- recolha do código fonte presente no repositório de controlo de versões;
- recolha de artefactos necessários para a execução do projeto (dependências);
- compilação do projeto e consequente geração de artefactos (binários para execução da aplicação).

Dada a importância que os processos de software têm (Sharma, 2017), existe a necessidade melhorar este processo de desenvolvimento, automatizando mais tarefas tornando o processo mais ágil. Deste modo, tarefas como execução de testes e implantação da aplicação devem ser automatizadas no sentido de reduzir a parte manual deste processo tornando-o menos suscetível a erros e, ainda, reduzindo o tempo necessário para a execução das mesmas.

3.3 Identificação da oportunidade de negócio

Após efetuada a identificação do contexto e do problema, um dos objetivos passa por analisar de que forma podemos resolver o problema, gerando valor que corresponda às necessidades do cliente.

De facto, apesar do crescimento da aplicação, os problemas identificados inibem que a aplicação possa crescer em maior escala. Deste modo, surge a oportunidade de efetuar uma reestruturação na arquitetura da mesma de forma a torná-la escalável, com um melhor desempenho e um melhor nível de manutenibilidade que permita evoluir a aplicação da melhor forma. Consequentemente, emerge a oportunidade de atingir novos clientes devido ao melhor serviço prestado pelo sistema e pelo aumento da capacidade da aplicação em suportar um maior volume de processamento em virtude da sua escalabilidade.

Como apresentado no capítulo Estado da Arte (secção 2.4) diversas empresas de diferentes dimensões já passaram por processos de reestruturação idênticos devido a problemas também eles semelhantes. Nos casos abordados nessa secção, essa reestruturação revelou-se um passo importante para o crescimento e permitiu que essas empresas resolvessem os seus problemas com sucesso entre os quais problemas de escalabilidade, desempenho e manutenibilidade. Deste modo, estes casos apresentados evidenciam-se como referências no sentido em que representam casos de sucesso no mercado com apresentação de tendências tecnológicas pertinentes para o problema em questão.

3.4 Proposta de valor

Dado o problema existente, a principal finalidade do projeto proposto consiste na reestruturação da aplicação existente de forma a torná-la escalável e elástica no sentido de ser mais eficiente dando resposta aos problemas de desempenho, memória e crescimento aplicacional que existem atualmente. O sistema deverá apresentar um aumento do desempenho em especial no processamento de dados/documentos. Pretende-se, também, aumentar a manutenibilidade do software de modo a facilitar a sua evolução (desde correção até à adição de novas funcionalidades), a permitir reduzir a complexidade de integração dos sistemas dos clientes e a simplificar a adição e manutenção de processamentos a efetuar aos documentos/dados da aplicação. Simultaneamente, também se pretende melhorar a qualidade do processo de desenvolvimento de forma a facilitar a evolução do produto ao longo do seu ciclo de vida. Assim, pretende-se que seja desenvolvida e implementada uma estratégia de *Continuous Delivery* e *Continuous Integration* com automação de diversas tarefas como, por exemplo, *build*, testes e *deploy* que, atualmente, são efetuadas manualmente. Posto isto, é possível reduzir o número de tarefas que atualmente são manuais, automatizando as mesmas para tornar este processo mais ágil e eficiente.

Deste modo, a proposta de valor deste projeto resume-se aos seguintes aspetos:

- Sistema escalável e elástico;
- Aumento do desempenho;
- Aumento da manutenibilidade;
- Aumento da automação e qualidade do processo de desenvolvimento (estratégia de *Continuous Delivery* e *Continuous Integration*).

3.4.1 Business Model Canvas

Tabela 1 - Business Model Canvas

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

• Flowinn</td><td>Atividades chave

• Investigação
• Desenvolvimento da solução
• Implementação de pipelines de Continuous Delivery e Continuous Integration
• Configurações de software</td><td rowspan="2">Proposta de valor

• Escalabilidade e elasticidade
• Desempenho
• Manutenibilidade
• Automação e Qualidade do processo de desenvolvimento</td><td>Relação com o cliente

• Acompanhamento dos resultados do produto
• Recolha e análise constante de feedbacks do cliente</td><td rowspan="2">Segmento de clientes

• Flowinn
• Clientes Flowinn</td></tr>
<tr><td>Recursos chave

• Cloud
• Bitbucket</td><td>Canais de comunicação

• N.A.</td></tr>
<tr><td colspan="2">Estrutura de custos

• Licenças de Software (Cloud e Bitbucket)</td><td colspan="3">Fontes de receita

• Venda do produto</td></tr>
</table>

3.5 Seleção de Alternativas

Com base nas alternativas apresentadas (secção 2.5) serão, de seguida, apresentadas as comparações entre as possíveis abordagens com intuito de efetuar a seleção das ferramentas mais adequadas para o desenvolvimento deste projeto.

3.5.1 Arquitetura

3.5.1.1 Seleção

Tabela 2 - Microsserviços vs Bibliotecas

	Bibliotecas	Microsserviços
Reutilizável	✓	✓
Heterogeneidade tecnológica	×	✓
Escalável	×	✓
Facilidade de manutenção	✓	✓
Implantação independente	×	✓

Após a análise efetuada, a abordagem recorrendo a microsserviços surge como a alternativa que mais benefícios pode trazer para a resolução do problema em questão.

De facto, a abordagem recorrendo a microsserviços oferece, ao contrário da utilização de bibliotecas, a possibilidade de obter heterogeneidade tecnológica, uma maior escalabilidade e a possibilidade de implantação independente. Estes aspetos permitem escalabilidade, um maior desempenho, um aumento da manutenibilidade da plataforma e simplificar o processo de entrega e implantação de novas funcionalidades/alterações da plataforma.

3.5.2 API Gateway, API Management e Service Discovery

Para auxiliar na seleção das alternativas apresentadas será utilizado o método AHP (*Analytic Hierarchy Process*).

Analytic Hierarchy Process (AHP), é um método de decisão multicritério que, através de técnicas numéricas, auxilia na tomada de decisão da seleção de uma opção entre um conjunto de alternativas. Com o intuito de dar apoio à decisão, este método auxilia no processo de avaliação de alternativas através do uso de critérios qualitativos e/ou quantitativos (Saaty, 1990).

Posto isto, este método será utilizado para auxiliar na seleção de uma opção entre as três alternativas apresentadas (JHipster API Gateway + JHipster Registry, JHipster API Gateway + HashiCorp Consul e WSO2 API Manager).

Tabela 3 - Objetivo, critérios e alternativas

Objetivo	**Selecionar opção para API Gateway, API Management e Service Discovery**
Critérios	• Monitorização • Documentação • Facilidade de configuração • Customização • Verificação de integridade dos microsserviços
Alternativas	• JHipster API Gateway + JHipster Registry • JHipster API Gateway + HashiCorp Consul • WSO2 API Manager

Para facilitar a compreensão e a avaliação do problema de tomada de decisão, o aspeto principal deste método consiste em dividir o problema de decisão em níveis hierárquicos (Saaty, 1990).

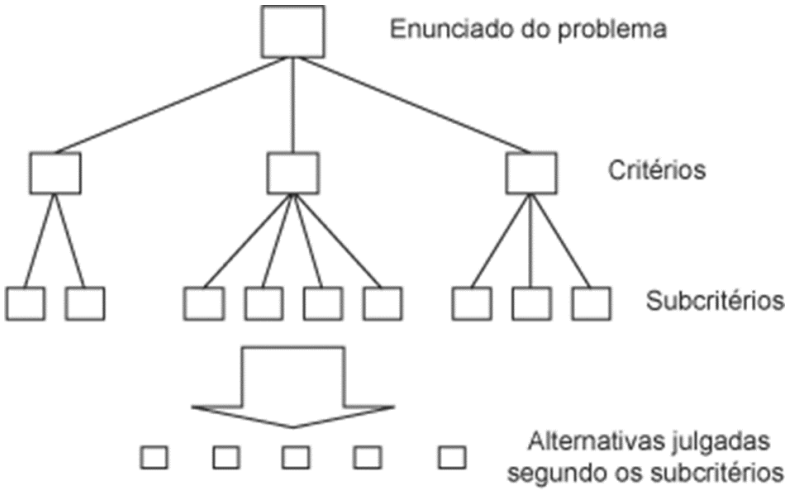

Figura 10 - Divisão hierárquica (Saaty, 1990)

Para o problema em questão, a árvore hierárquica de decisão é representada da seguinte forma:

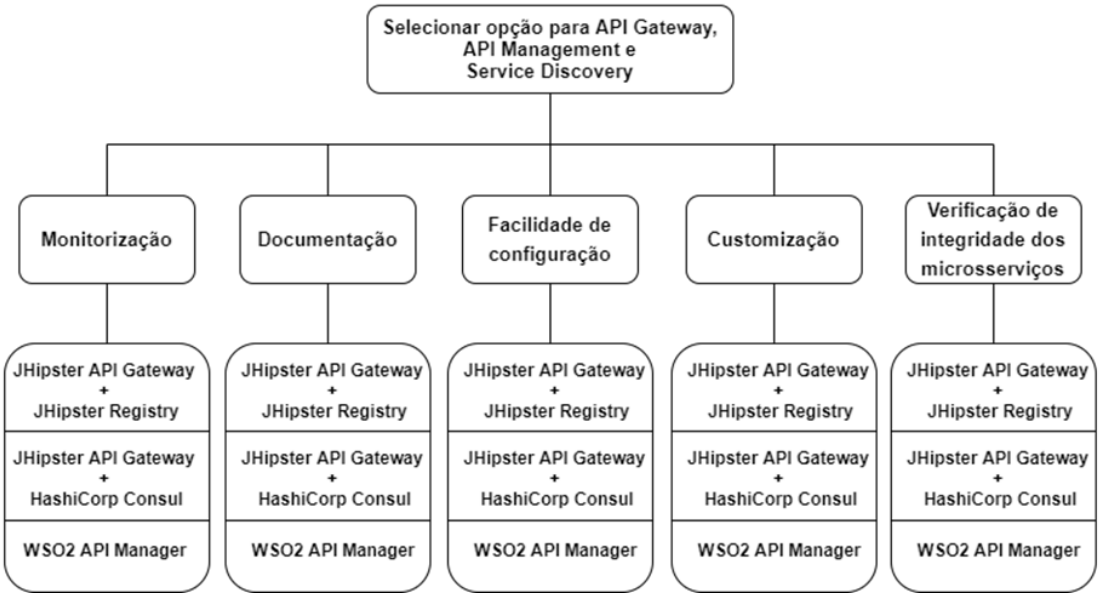

Figura 11 - Árvore hierárquica de decisão

Depois de construída a árvore hierárquica de decisão, é necessário representar a comparação entre os critérios especificados com a atribuição de prioridades entre ambos (Saaty, 1990). As prioridades atribuídas devem seguir a seguinte escala:

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 12 - Escala Fundamental (Saaty, 1990)

Tabela 4 - Comparação entre critérios

	Monitorização	Documentação	Facilidade de configuração	Customização	Verificação de integridade dos microsserviços
Monitorização	1	5	2	3	2
Documentação	1/5	1	1/3	1	1/3
Facilidade de configuração	1/2	3	1	2	1
Customização	1/3	1	1/2	1	1/2
Verificação de integridade dos microsserviços	1/2	3	1	2	1
Soma	38/15	13	29/6	9	29/6

Através da tabela podemos extrair a seguinte matriz de comparação:

$$A=\begin{bmatrix} 1 & 5 & 2 & 3 & 2 \\ \frac{1}{5} & 1 & \frac{1}{3} & 1 & \frac{1}{3} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \\ \frac{1}{3} & 1 & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \end{bmatrix}$$

Após atribuídas as prioridades como representação da comparação entre critérios, é necessário normalizar os valores das prioridades atribuídas para que estes sejam representados sob a mesma unidade. Para tal, os valores das prioridades devem ser divididos pelo somatório da respetiva coluna. Depois de todos os valores serem normalizados pretende-se identificar a ordem de importância de cada critério e, para isso, calcula-se a Prioridade Relativa (calculada a partir da média aritmética dos valores de cada linha previamente normalizada) (Saaty, 1990).

Tabela 5 - Normalização da comparação entre critérios e Prioridade Relativa

Prioridade Relativa	Monitorização	Documentação	Facilidade de configuração	Customização	Verificação de integridade dos microsserviços
Monitorização = 0,388054353	15/38	5/13	12/29	1/3	12/29
Documentação = 0,080982518	3/38	1/13	2/29	1/9	2/29
Facilidade de configuração = 0,212830595	15/76	3/13	6/29	2/9	6/29
Customização = 0,105301937	5/38	1/13	3/29	1/9	3/29
Verificação de integridade dos microsserviços = 0,212830595	15/76	3/13	6/29	2/9	6/29

Através da tabela podemos extrair a matriz normalizada e, com base na coluna da Prioridade Relativa, podemos extrair o vetor de prioridades que corresponde ao peso de cada um dos critérios.

$$\text{Matriz de Comparação - A} = \begin{bmatrix} 1 & 5 & 2 & 3 & 2 \\ \frac{1}{5} & 1 & \frac{1}{3} & 1 & \frac{1}{3} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \\ \frac{1}{3} & 1 & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \end{bmatrix}$$

$$\text{Matriz Normalizada - N} = \begin{bmatrix} \frac{15}{38} & \frac{5}{13} & \frac{12}{29} & \frac{1}{3} & \frac{12}{29} \\ \frac{3}{38} & \frac{1}{13} & \frac{2}{29} & \frac{1}{9} & \frac{2}{29} \\ \frac{15}{76} & \frac{3}{13} & \frac{6}{29} & \frac{2}{9} & \frac{6}{29} \\ \frac{5}{38} & \frac{1}{13} & \frac{3}{29} & \frac{1}{9} & \frac{3}{29} \\ \frac{15}{76} & \frac{3}{13} & \frac{6}{29} & \frac{2}{29} & \frac{6}{29} \end{bmatrix}$$

$$\text{Vetor de Prioridades V} = \begin{bmatrix} 0{,}39 \\ 0{,}08 \\ 0{,}21 \\ 0{,}11 \\ 0{,}21 \end{bmatrix}$$

Extraídas as Prioridades Relativas, é necessário avaliar a consistência das mesmas. Assim sendo, o próximo passo é o cálculo da Razão de Consistência (RC) que nos indica quanto consistentes foram os valores das Prioridades Relativas (Saaty, 1990).

Considerando $[\, A.V = \lambda \max .V \,]$ onde A é a matriz de comparação e V o vetor de prioridades:

$$\text{A} = \begin{bmatrix} 1 & 5 & 2 & 3 & 2 \\ \frac{1}{5} & 1 & \frac{1}{3} & 1 & \frac{1}{3} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \\ \frac{1}{3} & 1 & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{2} & 3 & 1 & 2 & 1 \end{bmatrix} \quad \text{V} = \begin{bmatrix} 0{,}39 \\ 0{,}08 \\ 0{,}21 \\ 0{,}11 \\ 0{,}21 \end{bmatrix} \quad \cong \quad \lambda\, max \quad \text{V} = \begin{bmatrix} 0{,}39 \\ 0{,}08 \\ 0{,}21 \\ 0{,}11 \\ 0{,}21 \end{bmatrix} \qquad (1)$$

$$\text{A.V} = \begin{bmatrix} 1{,}96 \\ 0{,}41 \\ 1{,}08 \\ 0{,}53 \\ 1{,}08 \end{bmatrix} \quad \cong \quad \lambda\, max \quad \text{V} = \begin{bmatrix} 0{,}39 \\ 0{,}08 \\ 0{,}21 \\ 0{,}11 \\ 0{,}21 \end{bmatrix} \qquad (2)$$

Assim, $\lambda\, max = \overline{x}\, \{\frac{1{,}96}{0{,}39} + \frac{0{,}41}{0{,}08} + \frac{1{,}08}{0{,}21} + \frac{0{,}53}{0{,}11} + \frac{1{,}08}{0{,}21}\}$, sendo $\overline{x}\ a\ m\acute{e}dia$ aritmética.

$$\text{IC} = \frac{(\lambda \max - n)}{(n-1)} = \frac{(5{,}05-5)}{5-1} = 0{,}0125 \qquad (3)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura 13 - Valores de IR para matrizes quadradas de ordem n (Saaty, 1990)

De acordo com a figura anterior, sendo n o número de critérios, para n=5 temos o valor 1,12.

$$RC = \frac{IC}{1,12} = \frac{0,0125}{1,12} = 0,01 \quad (4)$$

Sendo RC < 0,1, podemos concluir que os valores das prioridades relativas do exemplo utilizado estão consistentes (Saaty, 1990)

O próximo passo passa por construir a matriz de comparação para cada critério tendo por base cada uma das alternativas do problema em questão.

- Monitorização

Tabela 6 - Comparação entre alternativas - monitorização

	JHipster API Gateway + JHipster Registry	**JHipster API Gateway + HashiCorp Consul**	**WSO2 API Manager**	**Prioridade Relativa**
JHipster API Gateway + JHipster Registry	1 15/23	5 5/11	3 5/7	0,607001694
JHipster API Gateway + HashiCorp Consul	1/5 3/23	1 1/11	1/5 1/21	0,089654307
WSO2 API Manager	1/3 5/23	5 5/11	1 5/21	0,303343999
Soma	23/15	11	21/5	

- Documentação

Tabela 7 - Comparação entre alternativas - documentação

	JHipster API Gateway + JHipster Registry	**JHipster API Gateway + HashiCorp Consul**	**WSO2 API Manager**	**Prioridade Relativa**
JHipster API Gateway + JHipster Registry	1 3/13	3 3/8	1/3 4/19	0,272098516
JHipster API Gateway + HashiCorp Consul	1/3 1/13	1 1/8	1/4 3/19	0,119939271
WSO2 API Manager	1/3 9/23	4 1/2	1 12/19	0,607962213
Soma	13/3	8	19/12	

- Facilidade de configuração

Tabela 8 - Comparação entre alternativas - facilidade de configuração

	JHipster API Gateway + JHipster Registry	**JHipster API Gateway + HashiCorp Consul**	**WSO2 API Manager**	**Prioridade Relativa**
JHipster API Gateway + JHipster Registry	1 5/11	1 5/11	5 5/11	0,454545455
JHipster API Gateway + HashiCorp Consul	1 5/11	1 5/11	5 5/11	0,454545455
WSO2 API Manager	1/5 1/11	1/5 1/11	1 1/11	0,090909091
Soma	11/5	11/5	11	

- Customização

Tabela 9 - Comparação entre alternativas - customização

	JHipster API Gateway + JHipster Registry	JHipster API Gateway + HashiCorp Consul	WSO2 API Manager	Prioridade Relativa
JHipster API Gateway + JHipster Registry	1 7/9	7 7/9	7 7/9	0,777777778
JHipster API Gateway + HashiCorp Consul	1/7 1/9	1 1/9	1 1/9	0,111111111
WSO2 API Manager	1/7 1/9	1 1/9	1 1/9	0,111111111
Soma	9/7	9	9	

- Verificação de integridade dos microsserviços

Tabela 10 - Comparação entre alternativas - verificação de integridades dos microsserviços

	JHipster API Gateway + JHipster Registry	JHipster API Gateway + HashiCorp Consul	WSO2 API Manager	Prioridade Relativa
JHipster API Gateway + JHipster Registry	1 5/31	1/5 7/47	5 5/13	0,231613959
JHipster API Gateway + HashiCorp Consul	5 25/31	1 35/47	7 7/13	0,696531334
WSO2 API Manager	1/5 1/31	1/7 5/47	1 1/13	0,071854707
Soma	31/5	47/35	13	

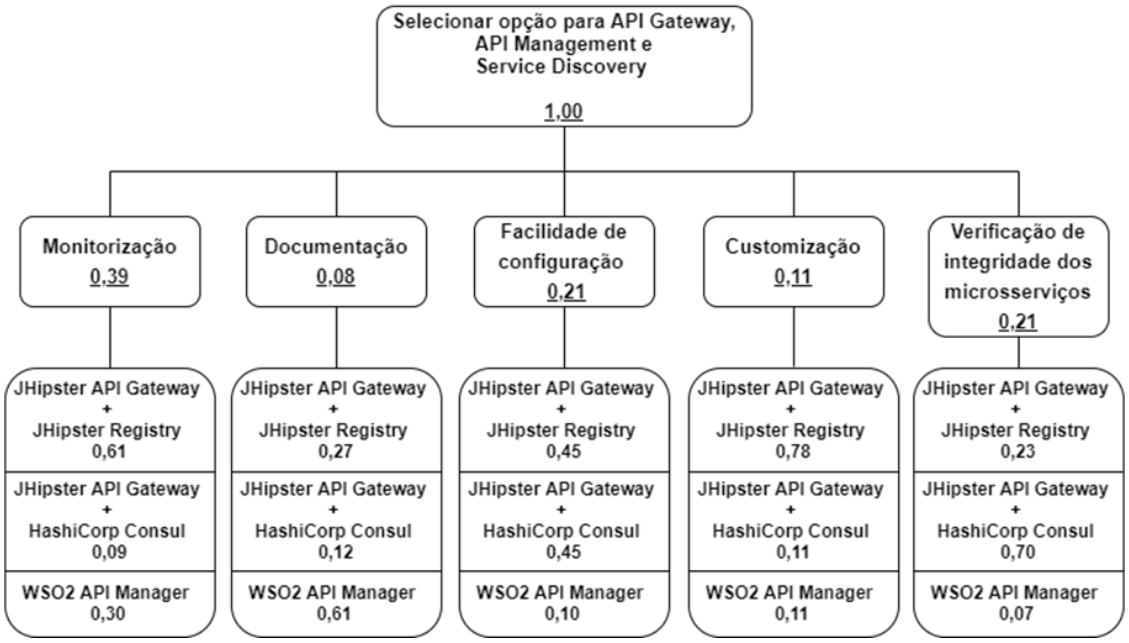

Figura 14 - Árvore hierárquica de decisão com pesos atribuídos

Posto isto, a partir da árvore hierárquica apresentada na Figura 14 pode ser extraída a matriz M (Matriz Prioridade das alternativas) e o vetor de prioridades V (indicativo dos pesos dos critérios) que podem ser representados da seguinte forma:

$$M=\begin{bmatrix}0{,}61 & 0{,}27 & 0{,}45 & 0{,}78 & 0{,}23\\ 0{,}09 & 0{,}12 & 0{,}45 & 0{,}11 & 0{,}70\\ 0{,}30 & 0{,}61 & 0{,}10 & 0{,}11 & 0{,}07\end{bmatrix} \text{ e } \quad V=\begin{bmatrix}0{,}39\\ 0{,}08\\ 0{,}21\\ 0{,}11\\ 0{,}21\end{bmatrix}$$

Então, $M \times V = \begin{bmatrix}0{,}49\\ 0{,}30\\ 0{,}21\end{bmatrix}$ indica a prioridade composta para as alternativas.

Tabela 11 - Alternativas e respetivas importâncias

JHipster API Gateway + JHipster Registry	**0,49**
JHipster API Gateway + HashiCorp Consul	0,30
WSO2 API Manager	0,21

Assim, como indicado na Tabela 11, utilizando o método AHP sob os critérios estabelecidos, a alternativa JHipster API Gateway + JHipster Registry aparece como a mais indicada.

3.5.3 Message Broker

Tabela 12 - RabbitMQ vs Kafka

	RabbitMQ	Apache Kafka
Melhor desempenho sob grandes quantidades de dados.	×	✓
Ordenação de mensagem	×	✓
Armazenamento de mensagens	×	✓
Reprocessamento de fluxos	×	✓
Diversidade de protocolos	✓	×

Apresentadas as duas alternativas, o Apache Kafka surge com mais benefícios devido ao seu melhor desempenho. A capacidade de processar grandes quantidades de dados por segundo sem necessidade de aumentar os recursos computacionais a utilizar torna-se um ponto relevante para a decisão. Outro aspeto importante é o facto de no Apache Kafka as mensagens poderem ser armazenadas, de forma ordenada, por um período configurável ao contrário do RabbitMQ que remove as mensagens das suas filas após as mesmas serem entregues. Consequentemente, surge a possibilidade de reprocessar os fluxos armazenados que, para o problema em questão, poderá ser uma funcionalidade útil na medida em que, em determinadas situações, poderá ser necessário repetir fluxos de processamento efetuados aos documentos.

3.5.4 Ferramentas de integração contínua

Tabela 13 - Jenkins vs Bitbucket Pipes

	Jenkins	Bitbucket Pipes
Execução externa e com ambiente preparado para utilização do Docker	×	✓
Facilidade de configuração	×	✓
Controlo de versões sob o Pipeline	✓	✓
Integração com JIRA	✓	✓
Flexibilidade	✓	×
Maior quantidade de funcionalidades de integração com o JIRA	×	✓
Facilidade de integração com o JIRA	×	✓
Gratuito	✓	**2500 min/mês 10$ / mês por 1000 min**
Integração com diversos repositórios de controlo de versões	✓	×

Apesar da flexibilidade (com a existência de diversos *plugins* para auxiliar nas diversas tarefas), o preço e a integração com diferentes repositórios de controlo de versões apresentada pelo Jenkins, o Bitbucket Pipes apresenta mais benefícios para o problema em questão.

Ainda que não seja gratuito como o Jenkins, o facto de a Flowinn possuir licença de utilização do Bitbucket faz com que tenha à sua disposição 2500 minutos por mês o que se prevê como suficiente para este projeto. Todavia, a possibilidade de possuir mais 1000 minutos por mês por mais 10$ não se torna um obstáculo para a Flowinn.

O facto de não ser possível integrar com diferentes repositórios de controlo de versões (apenas repositórios Bitbucket) não torna a utilização do Bitbucket Pipes uma contrariedade, visto que a Flowinn contém todos os seus projetos em repositórios Bitbucket.

Ainda que mais flexível, o Jenkins apresenta maior complexidade de configuração em relação ao Bitbucket Pipes que, com uma interface apelativa e intuitiva, torna a configuração bastante simplificada e consequentemente mais rápida. Outro aspeto relevante é que, apesar da sua flexibilidade e da existência de diversos *plugins* para o Jenkins, a sua integração com o JIRA é possível, porém, limitada relativamente ao Bitbucket Pipes. Este é um aspeto importante dado o facto da Flowinn utilizar o JIRA para auxiliar na monitorização e gestão de atividades relativas aos projetos e, deste modo, a integração com o JIRA poderá auxiliar na simplificação do processo de desenvolvimento da Flowinn.

Apesar de gratuito, o Jenkins necessita de ser alocado para ser executado. Para além disso, o ambiente para a sua execução tem de ser preparado com a instalação, por exemplo, das suas dependências e instalação do Docker que será necessário para a execução de algumas tarefas do *pipeline.* Em contrapartida, o Bitbucket Pipes executa como um serviço na plataforma Bitbucket não sendo necessário efetuar a sua alocação. Para além disso, pode-se fazer uso do Docker visto que a plataforma Bitbucket já dispõe do ambiente necessário para a sua utilização. Posto isto, o Bitbucket Pipes tem a vantagem de ser executado externamente e com o ambiente preparado para utilização de Docker ao contrário do Jenkins que necessita de ser hospedado e necessita que o ambiente para a execução do Docker seja preparado.

4 *Design* selecionado

Com base na análise das alternativas apresentadas (secção 2.5) e respetivas comparações e seleções (secção 3.5) serve o presente capítulo para apresentar um resumo das escolhas efetuadas assim como o modo de aplicação das mesmas no âmbito deste projeto.

De facto, selecionadas as diferentes abordagens e tecnologias a utilizar, apresenta-se, em seguida, a aplicação/função das mesmas no desenvolvimento e implementação deste projeto:

- Reestruturação da aplicação para uma arquitetura baseada em microsserviços;
- JHipster API Gateway e JHipster Registry como API Gateway (secção 2.1.2.4) e Serviço de Descoberta de Serviços (secção 2.1.2.5) respetivamente;
- Apache Kafka como *Message Broker*; e
- Bitbucket Pipes como ferramenta de integração contínua.

A arquitetura selecionada poderá ser representada pelo seguinte diagrama de componentes (Figura 15):

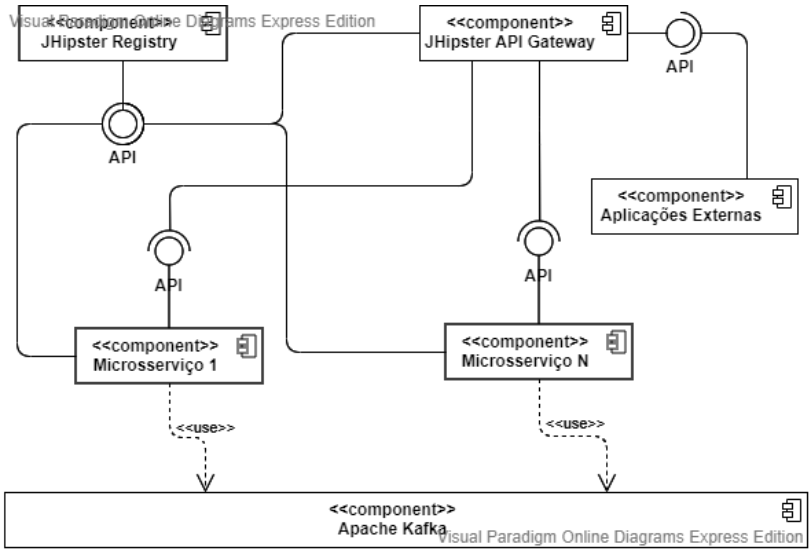

Figura 15 - Diagrama de componentes da arquitetura selecionada

4.1 Ambiente de implantação

No que diz respeito à implantação deste projeto, todos os componentes representados anteriormente (JHipster Registry, JHipster API Gateway, microsserviços e Apache Kafka) irão executar sob *containers* Docker (imagens Docker armazenadas no Docker Hub) na plataforma Kubernetes alocada na DigitalOcean Cloud.

A Figura 16 representa a implantação referida.

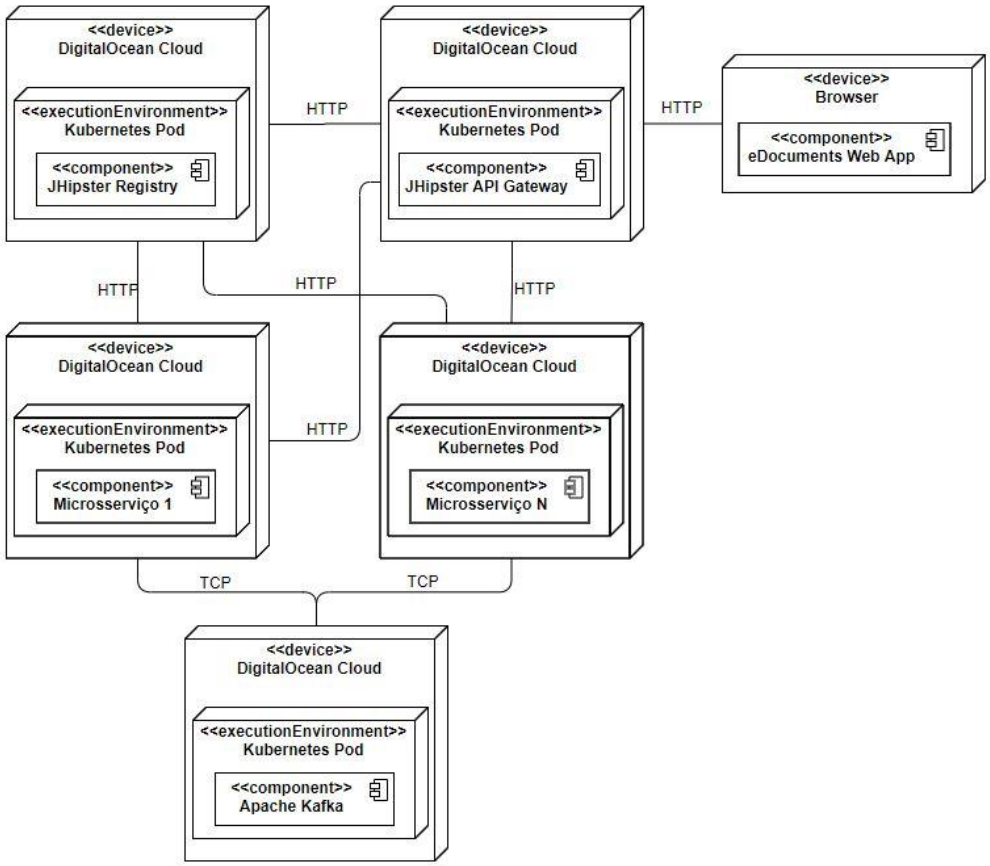

Figura 16 - Diagrama de Implantação

4.1.1 Container Docker

Um *container* é uma unidade de software que agrupa o código fonte e respetivas dependências para, dessa forma, separar as dependências da infraestrutura e, assim, ser possível executar a aplicação de forma rápida em qualquer ambiente computacional (*What is a Container?*).

Assim, uma imagem de um *container* Docker é uma unidade de software "leve, independente e executável que inclui tudo o que é necessário para executar uma aplicação: código, tempo de execução, ferramentas do sistema, bibliotecas e configurações do sistema." (*What is a*

Container?). Deste modo, um *container* garante que a aplicação seja executada independentemente do ambiente no qual seja alocado.

Posto isto, a utilização de *containers* Docker visa auxiliar o processo de implantação tornando-o mais simples e mais seguro visto que, assim, poderá executar em qualquer ambiente computacional e sem necessidade de configuração deste ambiente. Os *containers* Docker permitem, também, reduzir os custos com a alocação das aplicações devido ao facto de não ser necessário configurar o ambiente para a execução das mesmas (*What is a Container?*).

4.1.2 Docker Hub

O Docker Hub é um repositório gratuito de imagens de *containers* Docker. O Docker Hub permite o acesso a repositórios públicos ou repositórios privados que permite que as imagens lá colocadas sejam partilhadas pela equipa de desenvolvimento (*Docker Hub—Container Image Library | Docker*).

Assim sendo, o Docker Hub será utilizado para armazenar as imagens criadas e, desta forma, permitir que as mesmas sejam compartilhas com a equipa de desenvolvimento, recolhidas e executadas no ambiente computacional onde devem executar.

4.1.3 DigitalOcean Cloud

Atualmente, a Flowinn já possui licenças para utilização da DigitalOcean Cloud e tem como objetivo a curto prazo a migração de todas as suas aplicações para esta plataforma.

Assim sendo, os serviços desenvolvidos serão implantados na DigitalOcean Cloud que, assim, irá ser utilizada para alocar a plataforma desenvolvida.

4.1.4 Kubernetes

Kubernetes é uma plataforma de código aberto que efetua a gestão de *containers* responsáveis pela execução das aplicações. Posto isto, em ambientes de produção, o Kubernetes "fornece uma estrutura para executar sistemas distribuídos de forma resiliente" auxiliando na gestão da implantação, balanceamento de carga, escalonamento e na recuperação de falhas dos mesmos (What Is Kubernetes?).

A implantação no Kubernetes é efetuada num *cluster* que representa um conjuntos de máquinas/instâncias denominadas "Nós" ("Nodes"). Por sua vez, cada "Nó" contém um ou mais "Pods" que são componentes onde são executados os *containers* com a aplicação (Kubernetes Components).

A plataforma Kubernetes dispõe de diversos componentes que permitem diversas funcionalidades (What Is Kubernetes?)(Kubernetes Components), como, por exemplo, as seguintes:

- Permite a configuração, para cada *container*, dos recursos (CPU e memória) pretendidos;
- Gestão da implantação de "Pods" nos "Nodes" – Os "Pods" criados são automaticamente atribuídos a um determinado nó em execução com o intuito de efetuar uma melhor gestão dos recursos;
- Permite automatizar a implantação de novas versões dos *containers*;
- Deteta falhas de "Nodes" e de "Pods" e atua no sentido de recuperar as falhas mantendo o número de "Pods" correto;
- Permite escalonamento automático;
- Permite monitorizar métricas relativas ao estado de cada *container*;
- Permite visualização de *logs* de cada *container.*

Posto isto, a plataforma Kubernetes será utilizada com o intuito de auxiliar na implantação da plataforma e de melhor tirar partido das vantagens fornecidas pela *Cloud*.

5 Desenvolvimento da Solução

5.1 Análise da solução atual

A plataforma eDocuments é uma aplicação *web* de faturação eletrónica e EDI cujo intuito é efetuar a troca de ficheiros/documentos entre as entidades registadas na plataforma (secção 1.1.2). Deste modo, de forma a cumprir os requisitos pretendidos, a aplicação apresenta o modelo de domínio ilustrado na Figura 17.

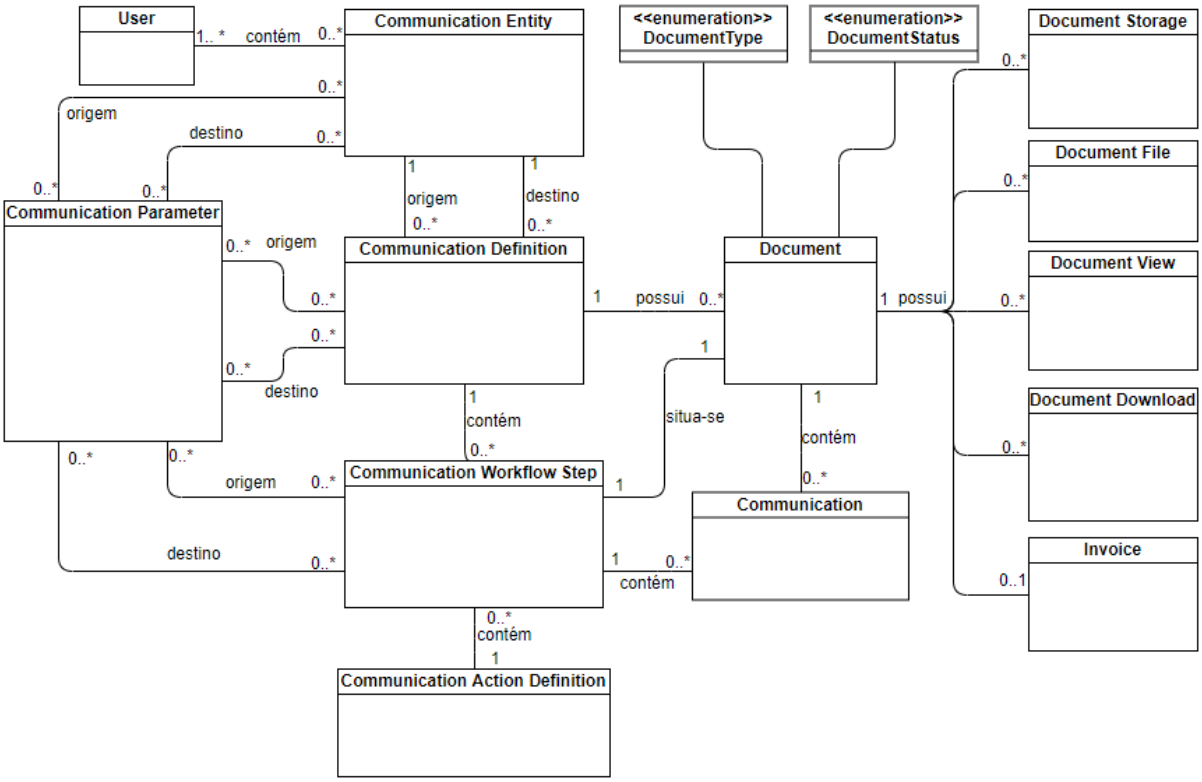

Figura 17 - Modelo de Domínio

Com base no diagrama apresentado, pode-se destacar o seguinte:

- "*Communication Entity*" – representa as entidades que se encontram registadas na plataforma eDocuments e que contém os seus utilizadores que poderão ter acesso às informações relativas à entidade;
- "*Communication Definition*" – define a comunicação que existe entre 2 entidades (*Communication Entity*) e contém todos os passos necessários (*workflow)* a efetuar aos documentos relativos à comunicação existente entre essas mesmas entidades;
- "*Communication Workflow Step"* – carateriza um passo do *workflow* definido para uma "*Communication Definition*" que será necessário efetuar no processamento dos documentos dessa mesma definição de comunicação;
- "*Communication Parameter*" – define parâmetros necessários para as entidades a que se encontra associada. Permite armazenar parâmetros necessários para o processamento/integração de documentos para uma dada entidade (*Communication Entity*), definição de comunicação (*Communication Definition*) ou passo do *workflow* (*Communication Workflow Step*) como, por exemplo, endereços de servidores e respetivas credenciais para recolha/integração de documentos;
- "*Communication Action Definition*" – define a ação necessária para um dado "*Communication Workflow Step".* Desta forma, indica qual a ação/serviço que terá de ser efetuado ao documento como, por exemplo, assinatura digital, envio de documentos por email, integração no sistema de alguma entidade, entre outros.
- "*Communication*" – Regista informação do estado da execução de um determinado passo ("*Communication Workflow Step")* para um dado documento. Desta forma, permite obter a informação do sucesso da execução deste passo assim como da data e hora em que esse passo foi executado;
- "*Document*" - entidade representativa dos documentos integrados na plataforma eDocuments e que serão processados e integrados nas entidades destino. Encontra-se relacionada com entidades que permitem armazenar informações relativamente ao ficheiro que deu entrada na plataforma ("*Document File*" e "*Document Storage*") assim como entidades que armazenam informações da interação (visualização e *download*) dos utilizadores com o documento.

Tal como referido anteriormente (secção 1.1.2), a solução atual apresenta-se como uma aplicação monolítica com um *frontend* desenvolvido em AngularJS e um *backend* desenvolvido sob uma API implementada com recurso à *framework* Spring Boot. A aplicação encontra-se organizada da seguinte forma:

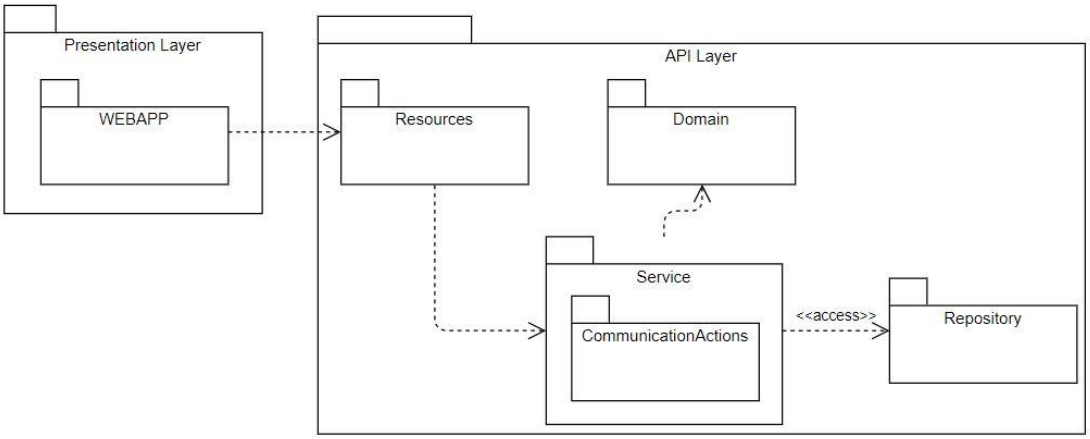

Figura 18 - Diagrama de *packages*

Como é possível observar no diagrama anterior (Figura 18), a aplicação possui uma camada de apresentação que contém a todo o código *frontend* desenvolvido em AngularJS e é responsável pela apresentação das páginas aos utilizadores (*webapp*).

De facto, a camada de apresentação é responsável por exibir informações ao utilizador, informações essas que são fornecidas pela camada API. A API é composta pelos seguintes *packages*:

- *Resources* – Contém as classes responsáveis por receber os pedidos REST da camada de apresentação e retorna as respostas aos mesmos. Para retornar as respostas aos pedidos efetuados, os *Resources* utilizam os serviços necessários que se encontram no *package Services* para obterem a informação de retorno;
- *Services* – Contém os serviços que dão resposta aos pedidos efetuados definindo (e aplicando), as regras de negócios associadas aos requisitos. Para fornecer as respostas necessárias, os dados a recolher são obtidos com acesso ao *package Repository*;
- *Repository* – Contém as classes responsáveis pelos acessos à camada de persistência. Definem as *queries* a serem utilizadas para o efeito;
- *Domain* – Contém as classes representativas do modelo de domínio;
- *CommunicationActions* – contém todos os serviços responsáveis por executar as ações sobre os documentos. Estas ações estão associadas à classe *Communication Action Definition* abordada no modelo de domínio.

Para além dos serviços responsáveis por efetuar ações sobre os documentos, existem outros 2 serviços muito importantes para o funcionamento desta aplicação sendo eles os seguintes:

- *ProcessNextWorkflowStep* – serviço responsável por atribuir, a um determinado documento, qual a próxima ação necessária a executar tendo por base o *workflow* associado ao documento. Deste modo, após o término com sucesso do serviço de uma determinada ação de comunicação, este serviço será executado de forma a identificar e atribuir a próxima ação necessária a executar colocando, ainda, o documento com o estado "*Processing*" (para posteriormente ser processado pelo

serviço *Scheduler*). No caso do *workflow* estar concluído, e por consequência não existir próxima ação necessária, o documento será dado como finalizado sendo atribuído o estado "*Finished*";

- *Scheduler* – serviço executado em determinados períodos definidos (e parametrizados) que é responsável por recolher todos os documentos que se encontram no estado "*Processing*" e os direcionar para a execução do serviço associado à ação de comunicação que se encontra atribuída ao documento em questão. No caso de existirem *workflows* onde o primeiro *step* requer execução por parte do eDocuments (com o intuito de integrar novos documentos na aplicação), este serviço é, também, responsável por recolhê-los e direcioná-los para a execução do respetivo serviço.

Posto isto, o exemplo de um fluxo será de seguida representado pela Figura 19.

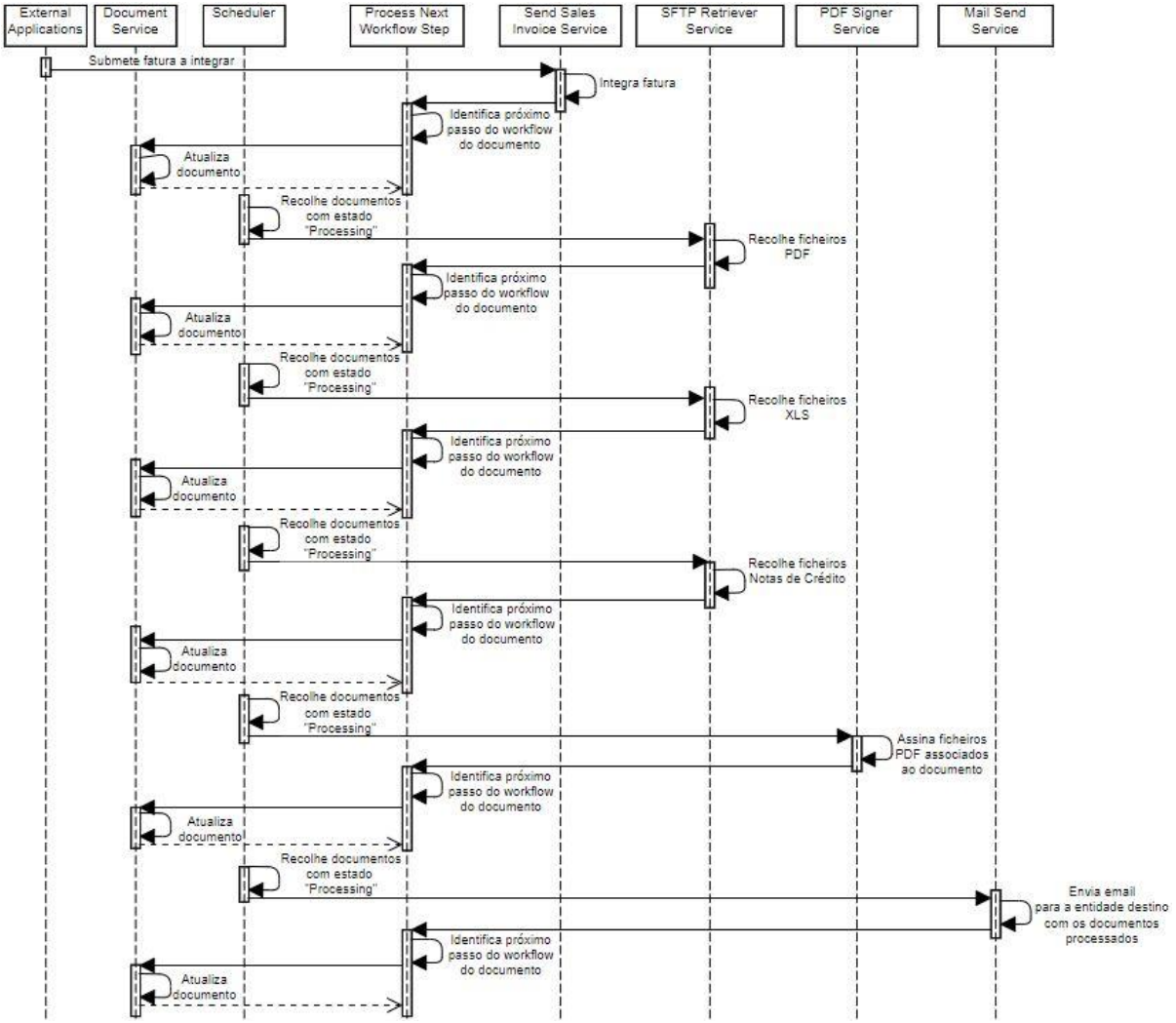

Figura 19 - Diagrama de Sequências (exemplo de funcionamento atual)

A figura anterior representa a sequência que existe, na aplicação atual, para a definição de comunicação mais utilizada mensalmente. Esta definição de comunicação (*workflow*), responsável por processar cerca de 7000 documentos no início de cada mês, contém as seguintes ações:

- Integração via "*Send Sales Invoice Service*" - onde documentos comerciais provenientes de ERP comercializados pela Flowinn são integrados na plataforma eDocuments;
- Recolha, por SFTP, de ficheiros PDF que representam faturas, notas de crédito ou autofaturas;
- Recolha, por SFTP, de ficheiros XLS que contém a descrição dos movimentos efetuados pela entidade destino;
- No caso de um documento ser uma nota de crédito, recolha, por SFTP, de ficheiros correspondentes às cartas de notas de crédito que, como requisito legal, acompanham as notas de crédito para assinatura do cliente (entidade de destino);
- Assinatura digital dos ficheiros PDF associados a um determinado documento;
- E, por último, o envio de email para a entidade de destino (definida na definição de comunicação) com os documentos que foram integrados pela entidade de origem e, posteriormente, processados pela aplicação.

5.2 Migração para arquitetura baseada em microsserviços

Partindo da análise ao modelo de domínio apresentada anteriormente, a separação do mesmo em possíveis blocos/estruturas independentes é apresentada na Figura 20.

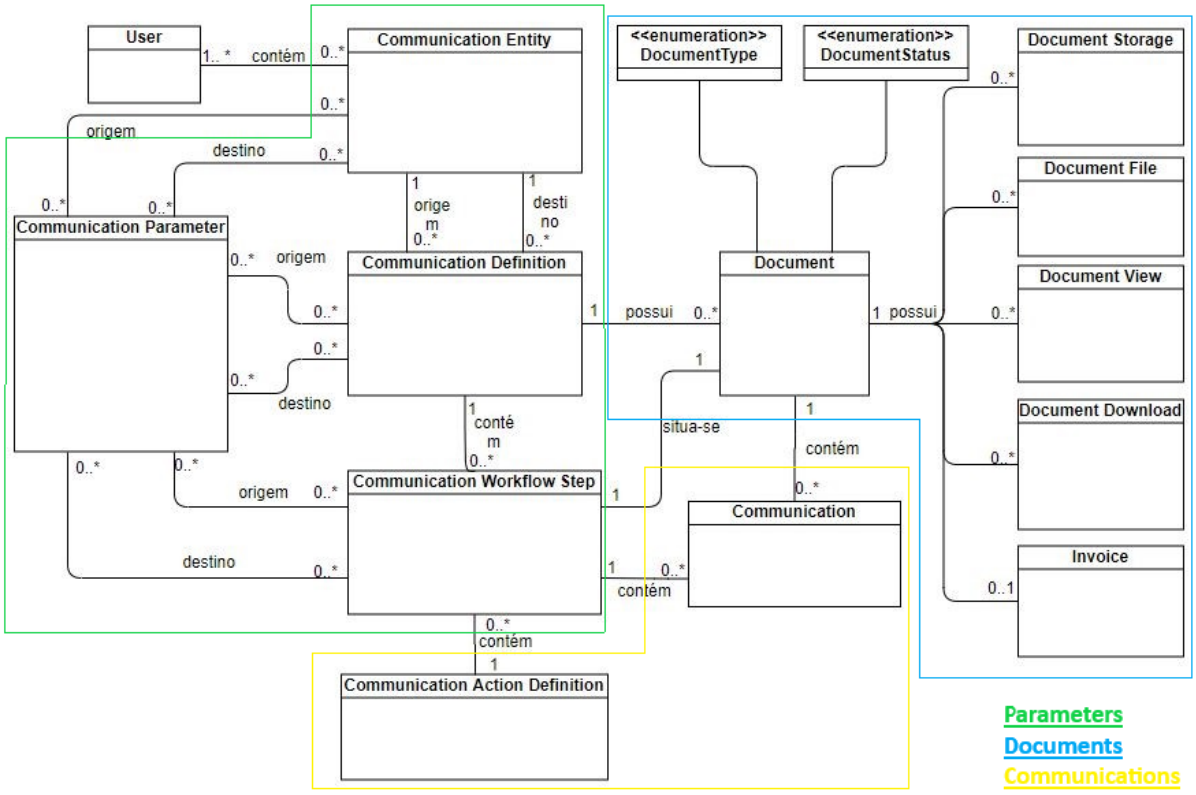

Figura 20 - Modelo de Domínio (separação)

Desta forma, e com base na análise da aplicação monolítica apresentada na secção anterior e no *design* selecionado (capítulo 4), da separação da aplicação monolítica existente em microsserviços resultou a divisão de componentes ilustrada pela Figura 21.

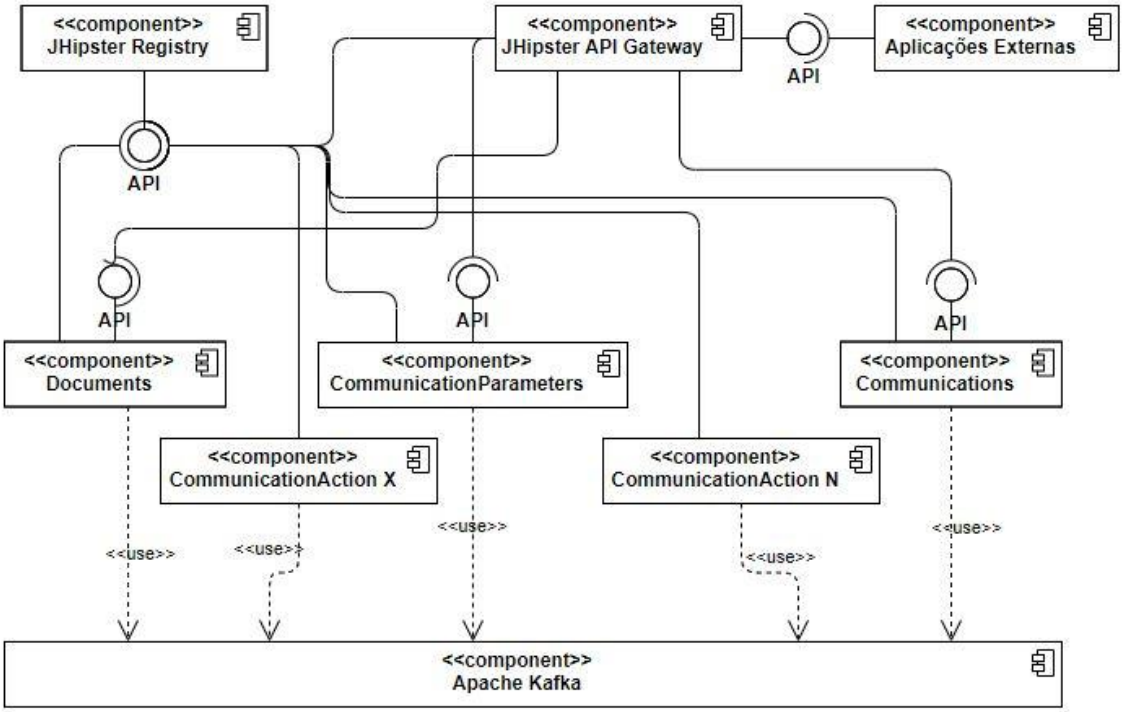

Figura 21 - Diagrama de Componentes (Microsserviços)

5.2.1 Documents

Componente responsável pela gestão dos documentos armazenando e gerindo a informação relativa aos documentos integrados na plataforma. Contém API REST que permite a criação, remoção e obtenção de documentos bem como das entidades relacionadas (Figura 20) como, por exemplo, ficheiros associados, faturas, registos de downloads e de visualização.

5.2.2 Communication Parameters

Componente que contém a API REST responsável por gerir todas as entidades consideradas "parametrizações". Deste modo, permite a criação, remoção e obtenção das seguintes entidades:

- Empresas/entidades existentes na plataforma (*Communication Entity*);
- Definições de Comunicação (*Communication Definition*) entre entidades registadas na plataforma;
- *Workflows* (e respetivos passos) definidos para uma determinada definição de comunicação entre duas entidades;

- Parâmetros associados às entidades, definições de comunicação e passos do *workflow* como, por exemplo, endereços de servidores para recolha/integração de documentos, credenciais, coordenadas para disposição da assinatura digital, entre outros.

5.2.3 Communications

Componente responsável por gerir os registos de comunicações efetuadas bem como das definições de ações de processamento existentes na plataforma eDocuments. As definições de ações representam características das respetivas ações que são microsserviços independentes e responsáveis pelos processamentos de ações sobre documentos. Os microsserviços das ações serão apresentados na secção seguinte.

Este componente encontra-se, também, à escuta do término das ações de processamento de documentos (componente "*CommunicationAction X*", "*CommunicationAction N*", entre outros) com o intuito de efetuar a identificação e atribuição da próxima ação necessária a executar tendo por base o *workflow* associado ao documento. Posto isto, o documento será então direcionado para o respetivo serviço de processamento ou ser dado como finalizado (estado "*Finished*") como consequência de não existir próxima ação necessária.

Para além disto, esta componente contém um serviço (executado em períodos definidos) responsável por recolher ações que requerem execução (ações que são o 1º passo do *workflow* e que necessitam de ser despoletadas pela plataforma eDocuments para efetuar a integração/criação dos documentos na plataforma eDocuments) e direciona para a execução do serviço associado à respetiva ação de comunicação.

5.2.4 Communication Action

Cada ação de comunicação é um componente (microsserviço) independente que contém a lógica associada à sua função específica de integração/processamento de documentos. Desta forma, estes serviços encontram-se à escuta de pedidos de processamento para que possam efetuar o seu processamento e posterior indicação do sucesso da operação.

Derivado do facto de que cada serviço de integração/processamento de documentos dar origem a um microsserviços independente, é possível, desta forma, alocar diferentes recursos de *hardware* para diferentes serviços. Assim sendo, cada microsserviço de integração/processamento de documentos é escalável e elástico existindo a possibilidade de reajustar os recursos conforme a sua necessidade de utilização. Consequentemente, as ações de comunicação (microsserviços) mais utilizadas poderão dispor de mais recursos alocados e, consoante os seus picos de utilização, esses recursos podem ser aumentados de forma independente e retirados após diminuição da sua necessidade.

De facto, cada ação de comunicação está associada a um microsserviço e, como exemplos de microsserviços de ações de comunicação, destacam-se os seguintes:

- *SFTP Retriever Service* – serviço responsável por, através do protocolo SFTP, recolher ficheiros de uma determinada diretoria de um servidor (dados parametrizados) e de os integrar na plataforma eDocuments;
- *PDF Signer Service* – para um dado documento, este serviço é responsável por assinar digitalmente todos os ficheiros com extensão ".pdf" a ele associado;
- *Mail Send Service* – serviço que envia os documentos para a entidade destino da definição de comunicação atribuída ao documento.

5.2.5 Exemplo de funcionamento

Tendo em conta a separação e as alterações efetuadas, o fluxo que fora apresentado como exemplo (Figura 19) poderá ser representado da seguinte forma:

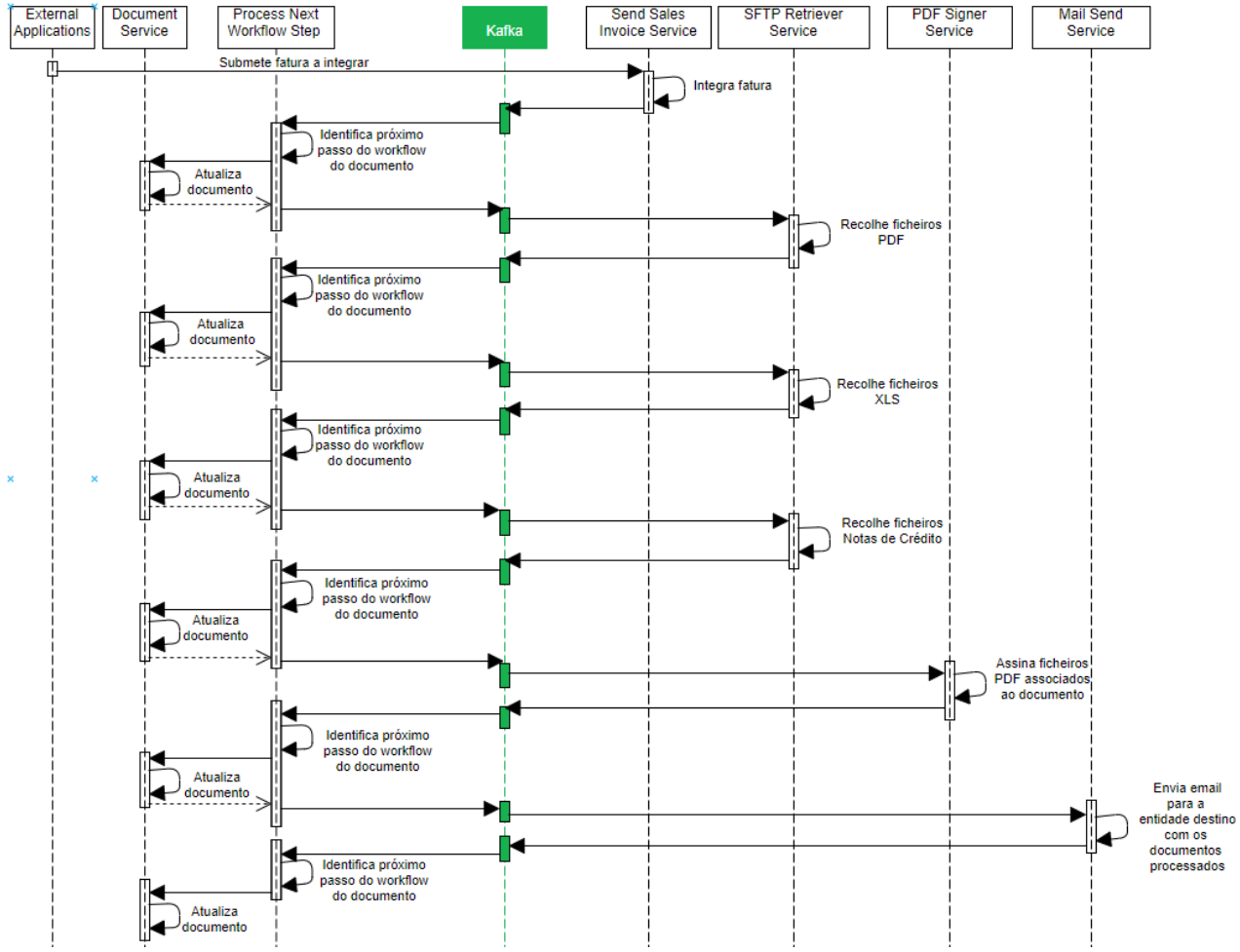

Figura 22 - Diagrama de Sequências (exemplo com microsserviços)

Com base na Figura 22, após as alterações efetuadas, destaca-se a presença do Apache Kafka que, como foi abordado na secção 2.5.3, será responsável por estabelecer a comunicação assíncrona entre microsserviços efetuando, desta forma, a troca de mensagens entre os diferentes serviços apresentados. Destaca-se, também, a ausência do serviço "*Scheduler*" que

foi substituído pelo facto dos documentos serem redirecionados para processamento através da troca de mensagens entre o Apache Kafka e o serviço relativo à ação de comunicação que se segue no *workflow* do documento (ação identificada pela serviço "*Process Next Workflow Step"*).

Relativamente aos serviços apresentados na Figura 22, todos eles estão dispostos em microsserviços independentes. O serviço "*Document Service*" pertence ao microsserviço "*Documents*" (secção 5.2.1), o serviço "*Process Next Workflow Step*" ao microsserviço "*Communications*" (secção 5.2.3) e os restantes são microsserviços independentes do tipo "*Communication Action*" (secção 5.2.4).

5.3 Processo de Desenvolvimento

Seguindo as normas já existentes na Flowinn, para o desenvolvimento deste projeto foram seguidas metodologias ágeis com o objetivo de verificar, de forma rápida, que os requisitos estão a ser cumpridos.

Desta forma, para uma melhor gestão deste processo, a Flowinn utiliza a ferramenta Jira como suporte à monitorização das tarefas afetas a um determinado projeto seguindo, para o efeito, a metodologia *Kanban* (Atlassian). A Figura 23 apresenta as diferentes fases pelas quais as tarefas poderão passar ao longo do processo de desenvolvimento.

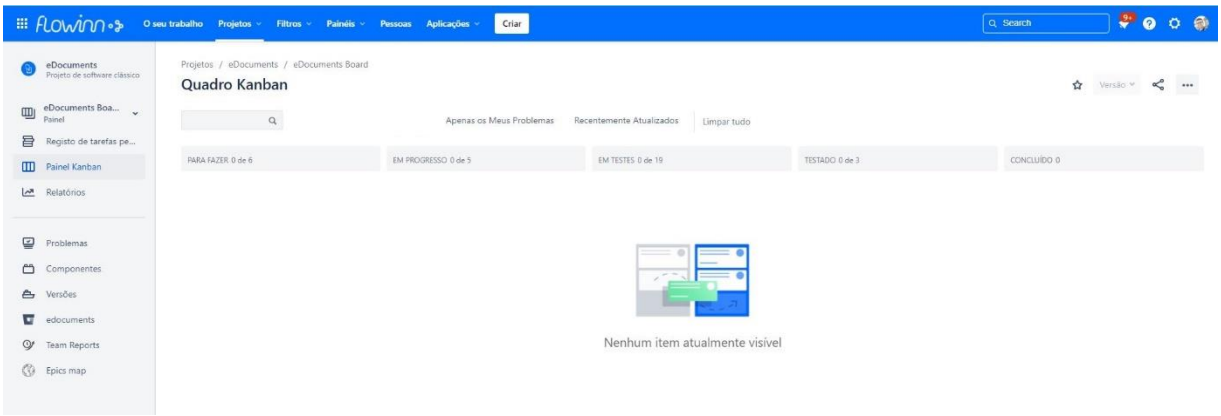

Figura 23 - Quadro Kanban na plataforma Jira

Com base na figura anterior, as diversas fases de cada tarefa durante o seu processo de desenvolvimento são as seguintes:

- Registo de tarefas pendentes – Tarefas identificadas que são para ser realizadas no âmbito do projeto;
- Para fazer - Tarefas selecionadas (e com colaborador atribuído) para realização na iteração em que a equipa se encontra;
- Em progresso – Tarefas que se encontram em execução;
- Em testes - Tarefas que se encontram em fase de testes;

- Testado – Tarefas já testadas e que se encontram prontas para serem instaladas em produção;
- Concluído – Tarefas já concluídas e instaladas em Produção.

A ferramenta Jira encontra-se integrada com o repositório de controlo de versões (Bitbucket) para ser possível, entre outras funcionalidades, relacionar as tarefas do quadro do Jira com os *branchs* e *commits* efetuados no repositório de controlo de versões.

De forma a melhor usufruir desta integração entre Jira e Bitbucket, foram definidas, no âmbito deste projeto, as seguintes regras que deverão ser cumpridas ao longo do processo de desenvolvimento de software:

- Criação de *branch* com referência ao identificador da tarefa no quadro *Kanban*;
- Após concluída uma tarefa, deve ser criado um *pull request* para o *branch development* – bloqueados os *commits* diretamente para este *branch*;
- O *pull request* deverá ser analisado pelo(s) *reviewer(s)* e, caso seja aceite, a tarefa será instalada em testes;
- Após a tarefa ser testada com sucesso, deverá ser criado um *pull request* para o *branch* master – bloqueados os *commits* diretamente para este *branch*;
- O *pull requests* deverá ser analisado pelo(s) *reviewer(s)* e, caso seja aceite, a tarefa será instalada em produção e dada como concluída.

5.3.1 DevOps

De forma a simplificar e automatizar processo de entrega de modificações efetuadas ao código fonte, foi desenvolvida uma estratégia de *Continuous Delivery* e *Continuous Integration*.

De facto, procedeu-se à implementação de um *pipeline* de *Continuous Delivery* e *Continuous Integration* que visa a automação de tarefas como, por exemplo, *build*, testes e *deploy* da aplicação. Este *pipeline* foi desenvolvido com recurso ao serviço Bitbucket Pipes (secção 2.5.4.2) resultando num ficheiro de configuração presente no repositório de controlo de versões de cada microsserviço. O *pipeline* resultante encontra-se dividido em 3 grupos que representam 3 fases distintas do processo de desenvolvimento para as quais será despoletado um diferente conjunto de ações. Desta forma, as fases presentes no *pipeline* e as respetivas ações despoletadas são as seguintes:

- Criação de *pull request:*
    - *Build and Tests* – com utilização do Maven, é gerada a build da aplicação e executados os testes de software (arquitetura, integração e unitários).
- Aceitação de *pull request* para *branch development*:
    - *Build and Tests* – com utilização do Maven, é gerada a build da aplicação e executados os testes de software (arquitetura, integração e unitários);
    - *Build Docker Image* – constrói imagem Docker com a aplicação;
    - *Save Docker Image to Docker Hub*: publica imagem Docker construída na ação anterior no repositório Docker Hub;

- *Testing Deployment* – efetua a implantação da aplicação em testes;
- *Smoke Tests* – verifica se a aplicação implantada está a responder.

- Aceitação de *pull request* para *branch master*:
  - *Build and Tests* – com utilização do Maven, é gerada a build da aplicação e executados os testes de software (arquitetura, integração e unitários);
  - *Build Docker Image* – constrói imagem Docker com a aplicação;
  - *Save Docker Image to Docker Hub*: publica imagem Docker construída na ação anterior no repositório Docker Hub;
  - *Production Deployment* – efetua a implantação da aplicação em produção;
  - *Smoke Tests* – verifica se a aplicação implantada está a responder.

Segue, na Figura 24, o exemplo da execução do *pipeline*, na plataforma Bitbucket, para o *branch development*.

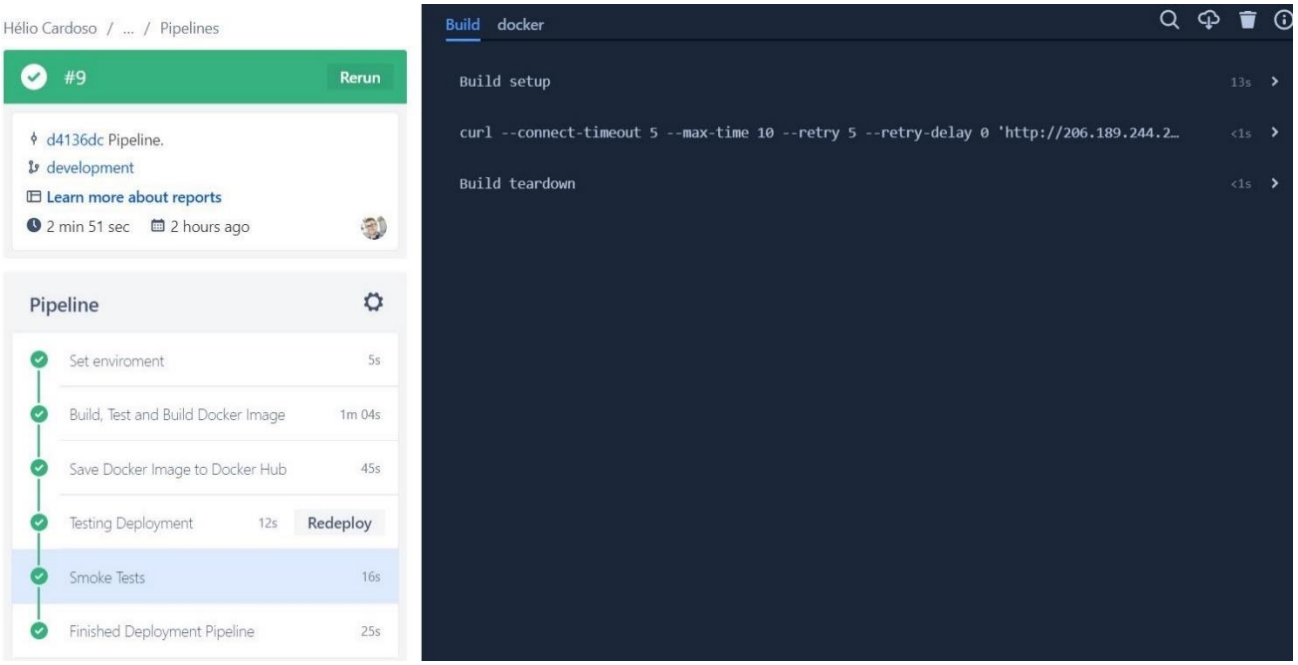

Figura 24 - Exemplo de execução do *pipeline* no *branch development*

6 Experimentação e Avaliação

6.1 Hipótese de Investigação

Dado o problema em questão (secção 1.2), pretende-se mostrar que, como consequência da reestruturação do sistema para uma arquitetura baseada em microsserviços, possamos usufruir de um sistema com melhor desempenho e com melhores níveis de manutenibilidade. Assim, dadas as características apresentadas pela arquitetura baseada em microsserviços (secção 2.1), espera-se que o sistema se torne mais rápido no processamento de documentos fruto da escalabilidade e elasticidade inerente às arquiteturas baseadas em microsserviços. Para além disso, dada a independência dos microsserviços, espera-se que a manutenibilidade seja melhorada simplificando e agilizando o processo de resolução de problemas e de incremento de funcionalidades.

Outro aspeto a mostrar é a melhoria da qualidade do processo de desenvolvimento de software com a implementação de um *pipeline* de *Continuous Delivery* e *Continuous Integration.* Espera-se, com a implementação deste *pipeline*, simplificação e automação do processo de entrega de modificações efetuadas ao código fonte face à automação de tarefas que atualmente são manuais como, por exemplo, a implantação e verificação da execução da aplicação (*smoke tests*).

6.2 Indicadores

Tendo em consideração os objetivos definidos (secção 1.3), os aspetos a avaliar são os seguintes:

- Desempenho;
- Escalabilidade e Elasticidade;
- Manutenibilidade;
- Qualidade do Processo de Desenvolvimento.

6.2.1 Desempenho

Dado o problema atual de diminuição do desempenho no processamento de documentos face ao crescimento da aplicação, pretende-se, com este projeto, aumentar os níveis de desempenho do sistema.

De facto, perante este objetivo, pretende-se reduzir o tempo médio necessário para processar o *workflow* mais utilizado pela aplicação (Figura 19) em pelo menos 10%. Como foi referido anteriormente, este *workflow* é responsável pelo processamento de cerca de 7000 documentos durante os primeiros dias de cada mês sendo que o número estimado de documentos que são processados em simultâneo situa-se entre os 400 e os 700.

Assim sendo, pretende-se efetuar uma análise exploratória de dados com o intuito de verificar se os dados refletem os valores esperados. Desta forma, considerando o processamento simultâneo estimado entre os 400 e 700 documentos, pretende-se efetuar um planeamento de casos de teste em torno desses valores e, com isso, calcular o tempo total de processamento e tempo médio de processamento por documento por parte do sistema atual e pela nova solução. Posto isto, pretende-se que, com este projeto, para a nova solução o tempo médio de processamento e o tempo médio de processamento por documento seja reduzido em 10% face ao sistema atual.

Assim, para o desenho de experiências efetuado foram considerados os seguintes fatores:

- Execução dos testes efetuados sob os mesmos recursos de *hardware* quer para a solução atual quer para a nova solução desenvolvida;
- Execução dos testes efetuados sob o mesmo *workflow* (Figura 19);
- Ficheiros utilizados para os testes contendo o mesmo tipo e tamanho;
- Ficheiros utilizados para os testes recolhidos de um servidor SFTP a executar sob o mesmo *hardware* para evitar períodos de latência distintos ao longo dos testes;
- Plano de testes considerando o processamento simultâneo de 100, 250, 500 e 1000 documentos com execução, para cada número de documentos, de 3 ensaios.

Com base nestes fatores, pretende-se, com as experiências efetuadas, retirar os seguintes resultados:

- Tempo médio de processamento – média (resultante dos 3 ensaios efetuados) do tempo total que as aplicações precisaram para processar o número de documentos a testar;
- Tempo médio de processamento por documento – média resultante do tempo que cada documento necessitou para ser processado (valor extraído pela média resultante dos 3 ensaios efetuados). O tempo que cada documento demora para ser processado é o intervalo de tempo desde a sua entrada/integração na aplicação até ao mesmo ser dado como finalizado (estado “FINISHED”).

Executados os testes com base no plano de testes retratado, os resultados obtidos são apresentados de seguida.

Tabela 14 - Resultados obtidos no processamento de 100 documentos (solução atual)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:02:25	00:01:16
Ensaio 2	00:03:21	00:01:52
Ensaio 3	00:02:28	00:01:23
Média	00:02:45	00:01:30

Tabela 15 - Resultados obtidos no processamento de 100 documentos (nova solução)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:03:29	00:02:45
Ensaio 2	00:03:03	00:02:44
Ensaio 3	00:02:49	00:02:31
Média	00:03:07	00:02:40

Tabela 16 - Solução Atual vs Nova Solução - 100 documentos em simultâneo

	Tempo médio de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Solução Atual	00:02:45	00:01:30
Nova Solução	00:03:07	00:02:40

Após serem efetuados 3 ensaios sob o processamento de 100 documentos em simultâneo, por observação da Tabela 16, verifica-se que a nova solução apresenta tempos superiores em relação à solução atual. Para uma quantidade de documentos relativamente baixa (comparada com o pico de processamentos em simultâneo em torno de 400-700 que se verifica atualmente em produção) a solução atual apresenta, em média, um tempo total de processamento de menos 22 segundos. Para além disso, relativamente ao tempo médio de processamento por cada documento, a diferença temporal é ainda maior com a solução atual a apresentar um tempo médio inferior em 1 minuto e 10 segundos.

Tabela 17 - Resultados obtidos no processamento de 250 documentos (solução atual)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:41:00	00:11:54
Ensaio 2	00:58:03	00:14:24
Ensaio 3	00:36:29	00:05:48
Média	00:45:11	00:10:42

Tabela 18 - Resultados obtidos no processamento de 250 documentos (nova solução)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:07:20	00:05:41
Ensaio 2	00:06:40	00:05:16
Ensaio 3	00:07:28	00:05:39
Média	00:07:09	00:05:32

Tabela 19 - Solução Atual vs Nova Solução - 250 documentos em simultâneo

	Tempo médio de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Solução Atual	00:45:11	00:10:42
Nova Solução	00:07:09	00:05:32

Para 250 documentos em simultâneo (valor mais próximo do pico de 400 a 700 documentos estimados atualmente) os resultados registados apresentam um conclusão diferente. Por observação da Tabela 19 verifica-se que a nova solução apresenta valores temporais significativamente inferiores. Em média, a nova solução demorou menos 38 minutos e 2 segundos a processar todos os 250 documentos e cada documento demorou, em média, menos 5 minutos e 10 segundos para ser processado. Para o processamento simultâneo de 250 documentos verifica-se uma diferença significava entre as 2 aplicação mesmo não estando ainda dentro dos valores estimados como pico de processamento (400-700 documentos).

Tabela 20 - Resultados obtidos no processamento de 500 documentos (solução atual)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:56:55	00:20:34
Ensaio 2	00:51:59	00:21:33
Ensaio 3	00:38:56	00:14:33
Média	00:49:17	00:18:53

Tabela 21 - Resultados obtidos no processamento de 500 documentos (nova solução)

	Tempo total de Processamento (hh:mm:ss)	**Tempo médio de processamento por documento (hh:mm:ss)**
Ensaio 1	00:14:06	00:12:00
Ensaio 2	00:14:58	00:12:42
Ensaio 3	00:15:46	00:13:48
Média	00:14:57	00:12:50

Tabela 22 - Solução Atual vs Nova Solução - 500 documentos em simultâneo

	Tempo médio de Processamento (hh:mm:ss)	Tempo médio de processamento por documento (hh:mm:ss)
Solução Atual	00:49:17	00:18:53
Nova Solução	00:14:57	00:12:50

Observando a Tabela 22 verifica-se que, embora a solução atual não tenha apresentado um aumento elevado relativamente ao tempo de processamento (em comparação com o processamento simultâneo de 250 documentos), a diferença entre as 2 aplicações continua elevada. Em média, a nova aplicação apresentou um tempo total de processamento inferior em 34 minutos e 20 segundos e em 6 minutos e 3 segundos relativamente ao tempo médio para processar cada documento. Considerando que 500 documentos em simultâneo é um valor dentro do intervalo entre 400 e 700 documentos que fora estimado como pico de processamento, estes são valores importantes a considerar no âmbito deste projeto.

Tabela 23 - Resultados obtidos no processamento de 1000 documentos (solução atual)

	Tempo total de Processamento (hh:mm:ss)	Tempo médio de processamento por documento (hh:mm:ss)
Ensaio 1	01:38:18	00:27:20
Ensaio 2	01:55:04	00:39:08
Ensaio 3	01:54:02	00:42:10
Média	01:49:08	00:36:13

Tabela 24 - Resultados obtidos no processamento de 1000 documentos (nova solução)

	Tempo total de Processamento (hh:mm:ss)	Tempo médio de processamento por documento (hh:mm:ss)
Ensaio 1	00:27:46	00:24:32
Ensaio 2	00:31:40	00:26:30
Ensaio 3	00:28:20	00:24:02
Média	00:29:15	00:25:01

Tabela 25 - Solução Atual vs Nova Solução - 1000 documentos em simultâneo

	Tempo médio de Processamento (hh:mm:ss)	Tempo médio de processamento por documento (hh:mm:ss)
Solução Atual	01:49:08	00:36:13
Nova Solução	00:29:15	00:25:01

Considerando o crescimento que se verifica atualmente da plataforma eDocuments (secção 1.1.2), o valor de 1000 documentos processados em simultâneo poderá ser alcançado num curto espaço de tempo. Desta forma, os dados para este número de documentos poderão ser importantes num futuro breve. Posto isto, a nova solução apresenta tempos inferiores, realçando-se a diferença significativa de 1 hora, 19 minutos e 53 segundos e de 11 minutos e 12 segundos, em média, para o tempo total de processamento e de tempo de processamento por documento, respetivamente.

Tabela 26 – Resultados de Tempo Total de Processamento (hh:mm:ss)

	Solução atual	Nova solução	Evolução Percentual
100 documentos	00:02:45	00:03:07	+ 13,33%
250 documentos	00:45:11	00:07:09	- 84,18%
500 documentos	00:49:17	00:14:57	- 69,67%
1000 documentos	01:49:08	00:29:15	-73,20%

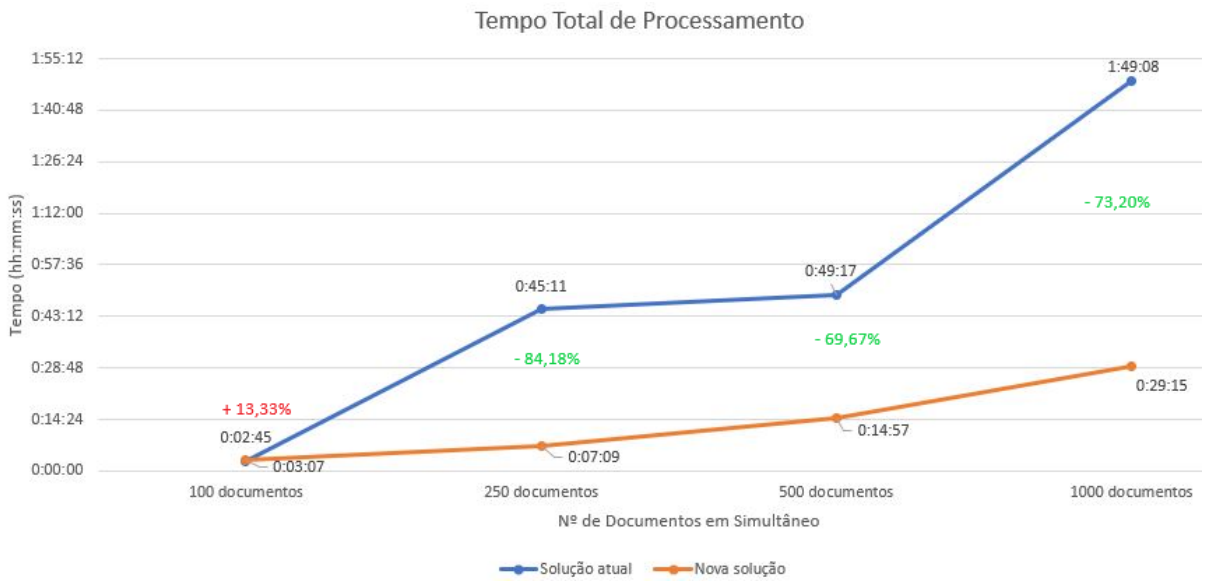

Figura 25 - Solução Atual vs Nova Solução – Tempo Total de Processamento

A Tabela 26 e a Figura 25 apresentam uma visão geral dos resultados obtidos relativamente ao tempo total de processamento. Através dos mesmos, verifica-se que, apesar de piores resultados no processamento de 100 documentos, a nova solução apresenta uma melhoria considerável relativamente ao desempenho no processamento de um número de documentos em simultâneo mais próximo, dentro e até superior ao pico atual estimado. Para o processamento de 100 documentos a nova solução apresentou um tempo superior em 13,33% sendo que, com o aumento do número de documentos, a tendência inverteu-se e a nova solução apresentou tempos inferiores em 84,18%, 69,67% e 73,20% para o processamento de 250, 500 e 1000 documentos, respetivamente. Observa-se, também, que a nova solução, face ao aumento do número de documentos a processar em simultâneo, apresenta uma tendência de crescimento menos acentuada e mais constante. Pelo contrário,

a solução atual apresenta uma tendência de crescimentos mais acentuada e, como mostra a Figura 25, apresenta picos de crescimento significativos aquando do aumento do número de documentos como, por exemplo, o pico registado para o processamento de 250 documentos.

Em suma, verifica-se que, para um número de documentos processados em simultâneo em torno do pico estimado (e até superior), a nova solução registou um diferença percentual dentro do objetivo estabelecido.

Tabela 27 – Resultados de Tempo Médio de Processamento por Documento (hh:mm:ss)

	Solução atual	Nova solução	Evolução Percentual
100 documentos	00:01:30	00:02:40	+ 77,78%
250 documentos	00:10:42	00:05:32	- 48,29%
500 documentos	00:18:53	00:12:50	- 32,04%
1000 documentos	00:36:13	00:25:01	- 30,92%

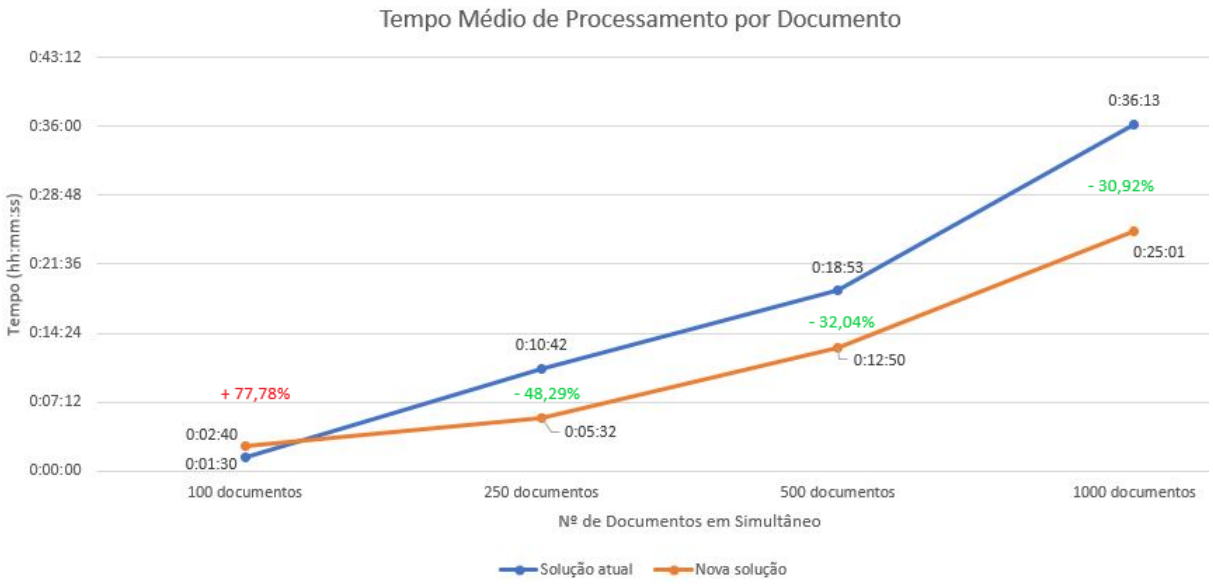

Figura 26 - Solução Atual vs Nova Solução – Tempo Médio de Processamento por Documento

A Tabela 27 e a Figura 26 representam uma visão geral dos resultados obtidos relativamente ao tempo médio de processamento por documento. Tal como no tempo total de processamento, para o processamento de 100 documentos em simultâneo, a nova solução apresenta um tempo superior em 77,78% ao registado pela solução atual. Para o processamento de 250 documentos os resultados voltam a inverter-se sendo que a nova solução apresentou um tempo inferior em 48,29% face à atual solução. Com o aumento do número de documentos a diferença percentual foi reduzida, porém, a nova solução voltou a apresentar tempos inferiores desta vez em 32,04% e 30,92% para o processamento de 500 e 1000 documentos, respetivamente. Quer para a solução atual quer para a nova solução, com o aumento do número de documentos a tendência de crescimento temporal apresenta-se constante e sem picos de crescimento significativos.

Posto isto, os resultados obtidos pela nova aplicação alcançam o objetivo estabelecido tendo em consideração o número de documentos processados em simultâneo em torno do pico estimado de 400 a 700 documentos.

6.2.2 Escalabilidade e Elasticidade

Abordado na secção 1.3, um dos objetivos pretendidos com este projeto é tornar a aplicação escalável e elástica no sentido de dar resposta aos problemas de desempenho e memória que se verificam atualmente.

Posto isto, pretende-se efetuar um controlo de qualidade de forma a avaliar esta componente utilizando, para o efeito, o modelo QEF (*Quantitative Evaluation Framework*). Desta forma, o fator a avaliar nesta componente é a funcionalidade e os requisitos (e respetivas métricas de avaliação) são apresentados na Tabela 28.

Tabela 28 - Escalabilidade e Elasticidade – Requisitos e Métricas de Avaliação

Requisito	**Métrica de Avaliação**
EEF01 - Possibilidade de aumentar e diminuir recursos de *hardware* da aplicação.	A implantação da plataforma deverá possibilitar quer o aumento quer a diminuição de recursos de *hardware* alocados.
EEF02 - Separação das ações de processamento de documentos em microsserviços distintos.	As ações de processamento de documentos devem ser implementadas em microsserviços independentes.
EEF03 - Possibilidade de obter escalabilidade e elasticidade para cada ação de processamento de documentos de forma independente.	As ações de processamento de documentos devem ser implantadas de forma a possibilitar a escalabilidade e elasticidade dos seus recursos de *hardware* alocados de forma independente.

De forma a avaliar a componente através do modelo QEF é necessário, para cada requisito, indicar o peso do mesmo para o fator a avaliar (2,4,6,8 e 10 como valores possíveis) e o respetivo cumprimento (de 0% a 100%). Deste modo, as informações para os requisitos apresentados encontram-se representadas na Tabela 29.

Tabela 29 - Escalabilidade e Elasticidade – Requisitos, Peso e Cumprimento

		Cumprimento (%)	
Requisito	**Peso [2,4,6,8,10]**	**0%**	**100%**
EEF01 - Possibilidade de aumentar e diminuir recursos de *hardware* da aplicação.	10	Não	Sim
EEF02 - Separação das ações de processamento de documentos em microsserviços distintos.	10	Não	Sim
EEF03 - Possibilidade de obter escalabilidade e elasticidade para cada ação de processamento de documentos de forma independente.	10	Não	Sim

Com base nos requisitos e nas métricas retratadas, para cada requisito será, de seguida, apresentado o respetivo cumprimento.

- EEF01 - Possibilidade de aumentar e diminuir recursos de *hardware* da aplicação:

Considerando o ambiente de implantação abordado anteriormente (secção 4.1), a possibilidade de obter escalabilidade e elasticidade (aumentar e diminuir os recursos de *hardware* alocados) é uma funcionalidade fornecida pela implantação em *Cloud*. Posto isto, com a implantação da solução na plataforma DigitalOcean Cloud (secção 4.1.3) este requisito é atingido na totalidade.

- EEF02 - Separação das ações de processamento de documentos em microsserviços distintos:

Como referido na secção 5.2, a arquitetura da solução desenvolvida considera a separação das ações de processamento de documentos em microsserviços independentes o que implica a obtenção do cumprimento deste requisito.

- EEF03 - Possibilidade de obter escalabilidade e elasticidade para cada ação de processamento de documentos de forma independente:

Com o cumprimento dos requisitos EEF01 e EEF02, tirando partido das funcionalidades da *Cloud* e da separação das ações de processamento de documentos em microsserviços independentes, obtemos a possibilidade de obter escalabilidade e elasticidade para cada ação de processamento de documentos de forma independente. Desta forma, o cumprimento deste requisito é atingido.

6.2.3 Manutenibilidade

Como referido no problema, face ao crescimento da plataforma, têm surgido problemas em efetuar a manutenção da mesma. Assim sendo, pretende-se, para a nova solução, efetuar um controlo de qualidade de forma a avaliar esta componente utilizando, para o efeito, o modelo QEF (*Quantitative Evaluation Framework*).

Posto isto, deverão ser considerados diferentes critérios/fatores com o intuito de efetuar o controlo de qualidade sob a dimensão/componente manutenibilidade.

6.2.3.1 Monitorização

Em virtude de simplificar a verificação do estado do sistema em determinado momento, pretende-se obter dados de auditoria e de diagnóstico de anomalias na aplicação. Assim, os requisitos a implementar e as respetivas métricas de avaliação são apresentados na Tabela 30.

Tabela 30 - Monitorização – Requisitos e Métricas de Avaliação

Requisito	Métrica de Avaliação
MM01 - Desenvolvimento de componente que permita apresentar estado atual dos microsserviços.	Desenvolvimento de ecrã onde deverá ser possível visualizar informações de estado do microsserviço como, por exemplo, se um determinado microsserviço está ou não em execução.
MM02 - Desenvolvimento de componente que permita apresentar informações sobre os microsserviços.	Desenvolvimento de ecrã onde deverá ser possível visualizar informações como, por exemplo, local onde se encontram em execução os microsserviços, CPU e memória consumida, tempos médios de resposta por recurso, entre outros.
MM03 - Desenvolvimento de componente que permita apresentar os *logs* dos microsserviços.	Desenvolvimento de ecrã onde deverá ser possível visualizar, por microsserviço, os respetivos *logs* que indicam os registos de eventos sucedidos.

De forma a avaliar a monitorização através do modelo QEF é necessário, para cada requisito, indicar o peso do mesmo para o fator a avaliar (2,4,6,8 e 10 como valores possíveis) e o respetivo cumprimento (de 0% a 100%). Posto isto, as informações para os requisitos apresentados encontram-se representadas na Tabela 31.

Tabela 31 - Monitorização – Requisitos, Peso e Cumprimento

		Cumprimento (%)	
Requisito	**Peso [2,4,6,8,10]**	**0%**	**100%**
MM01 - Desenvolvimento de componente que permita apresentar estado atual dos microsserviços.	10	Não	Sim
MM02 - Desenvolvimento de componente que permita apresentar informações sobre os microsserviços.	10	Não	Sim
MM03 - Desenvolvimento de componente que permita apresentar os *logs* dos microsserviços.	8	Não	Sim

Como analisado na secção 2.5.2.1, a utilização do JHipster Registry (como Serviço de Descoberta de Serviços) permite a obtenção de componentes de apresentação de diversas métricas. De entre todas as métricas disponibilizadas pelo JHipster Registry destacam-se a apresentação do estado do microsserviço (ativo ou não ativo), endereço de execução do microsserviço, CPU e memória consumida, tempos médios de resposta, entre outros. Para além disso, informações acerca de endereço de execução do microsserviço, CPU e memória consumida e tempos médios de resposta são também funcionalidades disponibilizadas pela utilização da *Cloud.* Assim sendo, os requisitos MM01 e MM02 encontram-se cumpridos na sua totalidade.

Para além disso, o JHipster Registry disponibiliza, também, através de configuração, um componente de visualização de *logs* da aplicação por seleção do microsserviço pretendido. Posto isto, também o requisito MM03 se encontra cumprido na sua totalidade.

6.2.3.2 Documentação

Por forma a facilitar o acesso e recolha de informação acerca do funcionamento da aplicação será necessário implementar os requisitos e cumprir as respetivas métricas de avaliação apresentados na Tabela 32.

Tabela 32 - Documentação – Requisitos e Métricas de Avaliação

Requisito	Métrica de Avaliação
MD01 - Documentação das interfaces expostas pelos microsserviços.	Desenvolvimento de ecrã onde deverá ser possível visualizar a documentação das interfaces expostas pelos microsserviços para permitir que os programadores obtenham informação acerca das funcionalidades das mesmas.
MD02 - Construção de documento de arquitetura de software.	Construção de documento de arquitetura de software para a aplicação desenvolvida com o intuito de fornecer um documento base com as informações sobre a estrutura e funcionamento da aplicação.
MD03 – Qualidade do documento de arquitetura de software.	Questionário respondido pelos programadores afetos ao projeto (após concluído o documento) relativamente à satisfação, qualidade e adequação de conteúdos do documento desenvolvido. Pontuação de cada pergunta varia entre 1-5.

De forma a avaliar a documentação através do modelo QEF é necessário, para cada requisito, indicar o peso do mesmo para o fator a avaliar (2,4,6,8 e 10 como valores possíveis) e o respetivo cumprimento (de 0% a 100%). Assim, as informações para os requisitos apresentados encontram-se representadas na Tabela 33.

Tabela 33 - Documentação – Requisitos, Peso e Cumprimento

		Cumprimento (%)		
Requisito	**Peso [2,4,6,8,10]**	**0%**	**50%**	**100%**
MD01 - Documentação das interfaces expostas pelos microsserviços.	10	Não	-	Sim
MD02 - Construção de documento de arquitetura de software.	10	Não	-	Sim
MD03 – Qualidade do documento de arquitetura de software.	10	Média da pontuação < 2	Média da pontuação 2-3	Média da pontuação ≥ 4

Relativamente ao requisito MD01, a documentação das interfaces expostas pelas API dos microsserviços é uma prática importante a seguir (secção 2.1.2.8). Deste modo, como referido na secção 2.5.2.1, a utilização do JHipster Registry (como Serviço de Descoberta de Serviços) e do JHipster API Gateway (como API Gateway) permite a obtenção de um Swagger que

automaticamente expõe em ambas as aplicações informações relativas às interfaces expostas pelos microsserviços.

Por outro lado, em relação aos requisitos MD02 e MD03, a Figura 27 apresenta as perguntas e respetivas respostas obtidas (através dos programadores afetos ao projeto) relativamente ao questionário acerca da qualidade do documento de arquitetura de software. Ilustrada na Figura 27, a avaliação média obtida foi 5. Desta forma, ambos os requisitos foram cumpridos na sua totalidade.

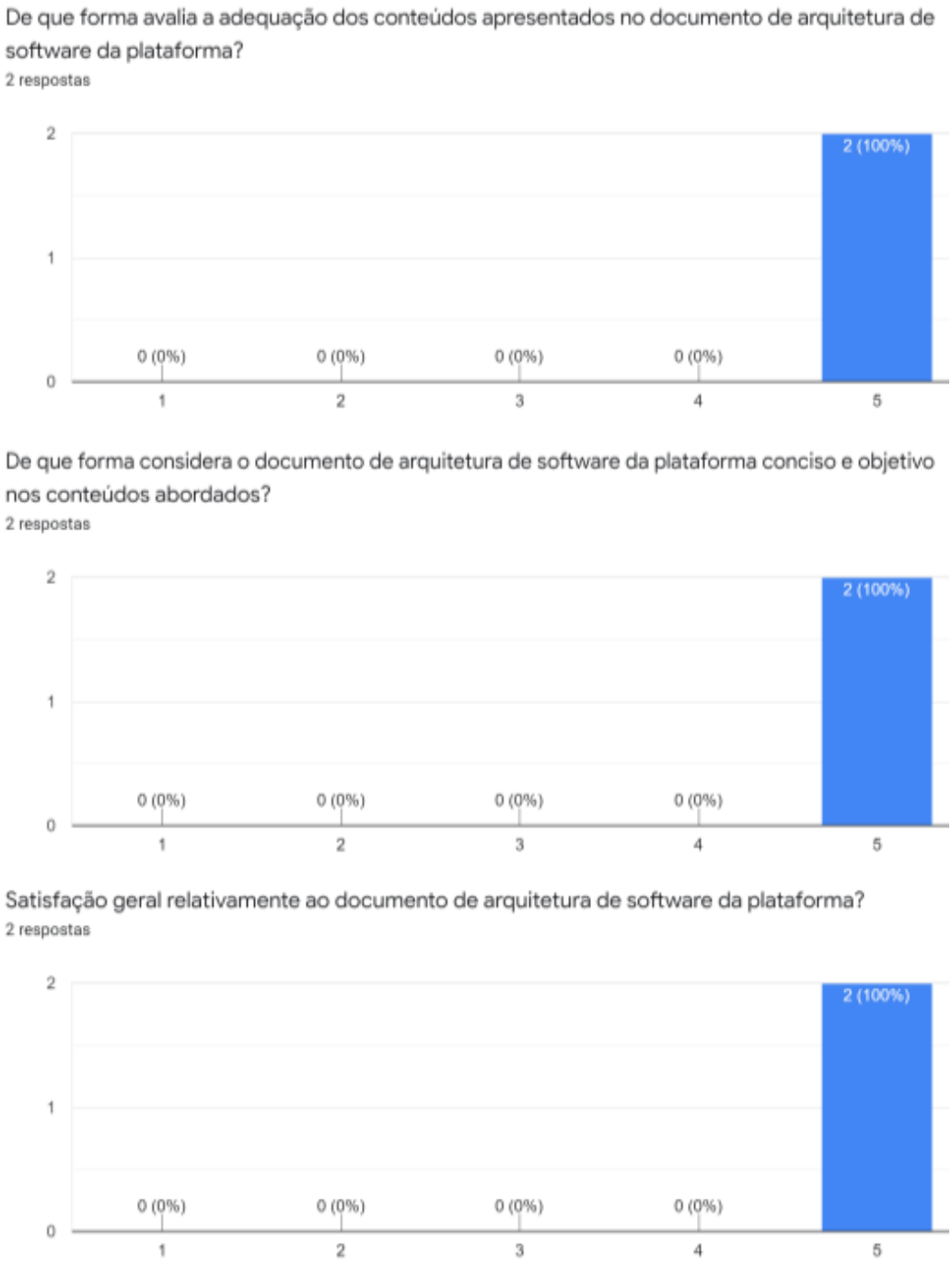

Figura 27 - Questionário e Respostas – Requisito MD03

6.2.3.3 Qualidade e Legibilidade do Código Fonte

Com o intuito de avaliar a qualidade e legibilidade do código fonte pretende-se a implementação de requisitos que visam assegurar que este fator seja respeitado.

Posto isto, os requisitos e respetivas métricas de avaliação são apresentados na Tabela 34:

Tabela 34 - Qualidade e Legibilidade do Código Fonte – Requisitos e Métricas de Avaliação

Requisito	**Métrica de Avaliação**
MQLCF01 – Estrutura em camadas das API dos microsserviços respeitada.	Todos os microsserviços devem apresentar 100% de sucesso para os testes de arquitetura desenvolvidos (testes que validam que as regras definidas para as camadas da API são respeitadas).
MQLCF02 –Qualidade e adequação da estrutura e funcionamento da aplicação desenvolvida.	Questionário respondido pelos programadores afetos ao projeto (após desenvolvida a nova solução) relativamente à qualidade da estrutura da aplicação desenvolvida e adequação do modo de funcionamento de forma a efetuar a manutenção e o incremento de funcionalidades (com particular destaque para os serviços relativos às ações de processamento de documentos). Pontuação de cada pergunta varia entre 1-5.

De forma a avaliar a qualidade e legibilidade do código fonte através do modelo QEF é necessário, para cada requisito, indicar o peso do mesmo para o fator a avaliar (2,4,6,8 e 10 como valores possíveis) e o respetivo cumprimento (de 0% a 100%). Assim, as informações para os requisitos apresentados encontram-se representadas na Tabela 35.

Tabela 35 - Qualidade e Legibilidade do Código Fonte – Requisitos, Peso e Cumprimento

		Cumprimento (%)		
Requisito	**Peso [2,4,6,8,10]**	**0%**	**50%**	**100%**
MQLCF01 – Estrutura em camadas das API dos microsserviços respeitada.	8	< 100 % sucesso para testes de arquitetura	-	100 % sucesso para testes de arquitetura
MQLCF02 –Qualidade e adequação da estrutura e funcionamento da aplicação desenvolvida.	8	Média da pontuação < 2	Média da pontuação 2-3	Média da pontuação ≥ 4

De modo a garantir o cumprimento do requisito MQLCF01, para validar que a estrutura definida para as API de cada microsserviço é respeitada foi utilizada a ferramenta ArchUnit (Gafert). Esta ferramenta possibilita a definição de regras que devem ser respeitadas pelas camadas da API como, por exemplo, que camadas podem conter regras de negócio, que camadas podem aceder à camada de persistência de dados, qual a interligação/comunicação

permitida entre camadas, entre outras. Posto isto, no momento da *build* do código fonte, a ferramenta é responsável por analisar e validar se as regras definidas são respeitadas. Desta forma, se existir incumprimento de alguma das regras a *build* do código fonte não ocorrerá com sucesso.

Relativamente ao requisito MQLCF02, como ilustrado na Figura 28, para o questionário relativo à qualidade e adequação da estrutura e funcionamento da aplicação desenvolvida, a avaliação média obtida foi 5. Posto isto, após obtenção da avaliação através dos programadores afetos ao projeto, o requisito MQLCF02 foi cumprido na totalidade.

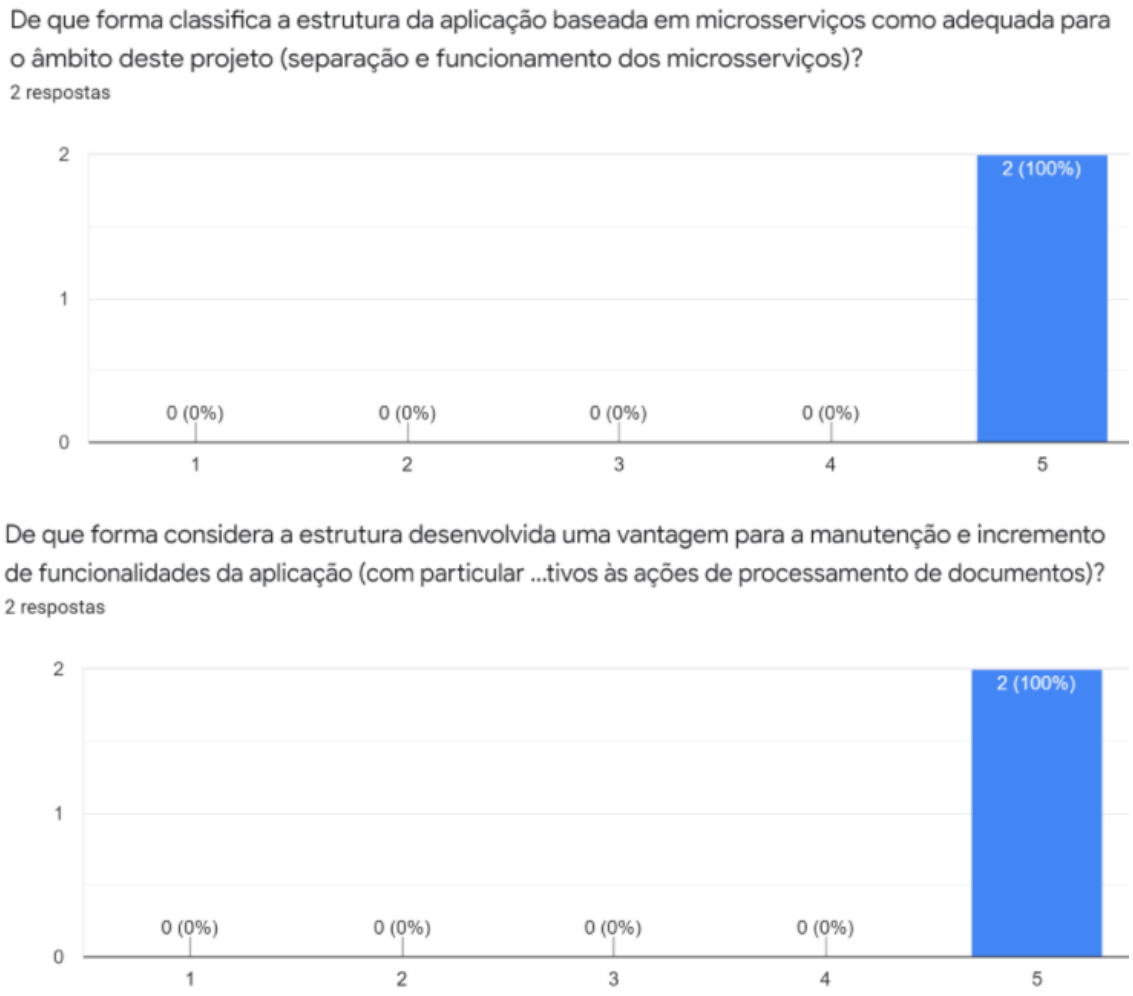

Figura 28 - Questionário e Respostas – Requisito MQLCF02

6.2.4 Qualidade do Processo de Desenvolvimento

Com o intuito de colmatar as limitações no processo de desenvolvimento, pretende-se, desenvolvida uma estratégia de *Continuous Delivery* e *Continuous Integration* com automação de tarefas como *build*, testes e *deploy.*

Assim sendo, pretende-se efetuar um controlo de qualidade de forma a avaliar esta componente. Posto isto, o fator a avaliar no âmbito da qualidade do processo de desenvolvimento passa pela automação de tarefas que será efetuada com recurso à implementação de um *pipeline* que implemente os requisitos e respetivas métricas de avaliação apresentados na Tabela 36.

Tabela 36 - Qualidade do Processo de Desenvolvimento – Requisitos e Métricas de Avaliação

Requisito	Métrica de Avaliação
QPDAT01 - Efetuar *build* do código fonte.	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá efetuar a *build* do código fonte.
QPDAT02 - Execução dos testes de software.	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá executar os testes de software implementados.
QPDAT03 - Construção de imagem Docker com a aplicação.	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá construir uma imagem Docker contendo a aplicação correspondente.
QPDAT04 - Publicação da imagem no Docker Hub.	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá publicar a imagem Docker (construída anteriormente) no repositório Docker Hub.
QPDAT05 - *Deploy* da aplicação na DigitalOcean Cloud.	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá efetuar a implantação da aplicação na DigitalOcean Cloud.
QPDAT06 - Execução de *Smoke Tests.*	O *pipeline* de *Continuous Delivery* e *Continuous Integration* deverá verificar se a aplicação se encontra em execução após a implantação da mesma.

Para efetuar a avaliação através do modelo QEF é necessário, para cada requisito, indicar o peso do mesmo para o fator a avaliar (2,4,6,8 e 10 como valores possíveis) e o respetivo cumprimento (de 0% a 100%). Estas informações encontram-se representadas na Tabela 37.

Tabela 37 - Qualidade do Processo de Desenvolvimento – Requisitos, Peso e Cumprimento

		Cumprimento (%)	
Requisito	**Peso [2,4,6,8,10]**	**0%**	**100%**
QPDAT01 - Efetuar *build* do código fonte.	10	Não implementado	Implementado
QPDAT02 - Execução dos testes de software.	10	Não implementado	Implementado
QPDAT03 - Construção de imagem Docker com a aplicação.	10	Não implementado	Implementado
QPDAT04 - Publicação da imagem no Docker Hub.	10	Não implementado	Implementado
QPDAT05 - Deploy da aplicação na DigitalOcean Cloud.	10	Não implementado	Implementado
QPDAT06 - Execução de *Smoke Tests.*	8	Não implementado	Implementado

Referido na secção 5.3, o *pipeline* desenvolvido possui todos estes passos o que, desta forma, permite o cumprimento de todos os requisitos funcionais no âmbito da qualidade do processo de desenvolvimento de software.

6.2.5 Modelo QEF - Resumo

Abordados nas secções anteriores, os indicadores Escalabilidade e Elasticidade, Manutenibilidade e Qualidade do Processo de Desenvolvimento foram avaliados através de um controlo de qualidade utilizando, para o efeito, o modelo QEF.

De facto, após apresentados os resultados individuais de cada indicador, a Figura 29 apresenta os resultados obtidos pelo modelo QEF para todos os indicadores abordados.

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]
100%	0,00	100	Escalabilidade e Elasticidade	100	1,00	Funcionalidade	10	EEF01 - Possibilidade de aumentar e diminuir recursos de hardware da aplicação.	100
							10	EEF02 - Separação das ações de processamento de documentos em microsserviços distintos.	100
							10	EEF03 - Possibilidade de escalar cada ação de processamento de documentos de forma independente.	100
		100	Manutenibilidade	100	0,38	Monitorização	10	MM01 - Desenvolvimento de componente que permita apresentar estado atual dos microsserviços.	100
							10	MM02 - Desenvolvimento de componente que permita apresentar informações sobre os microsserviços.	100
							8	MM03 - Desenvolvimento de componente que permita apresentar os logs dos microsserviços.	100
				100	0,38	Documentação	10	MD01 - Documentação das interfaces expostas pelos microsserviços.	100
							10	MD02 - Construção de documento de arquitetura de software.	100
							10	MD03 – Qualidade do documento de arquitetura de software.	100
				100	0,25	Qualidade do Código Fonte	8	MQLCF01 – Estrutura em camadas das API dos microsserviços respeitada.	100
							8	MQLCF02 –Qualidade e adequação da estrutura e funcionamento da aplicação desenvolvida.	100
		100	Qualidade do Processo de Desenvolvimento	100	1,00	Automação de Tarefas	10	QPDAT01 - Efetuar build do código fonte.	100
							10	QPDAT02 - Execução dos testes de software.	100
							10	QPDAT03 - Construção de imagem Docker com a aplicação.	100
							10	QPDAT04 - Publicação da imagem no Docker Hub.	100
							10	QPDAT05 - Deploy da aplicação na DigitalOcean Cloud.	100
							8	QPDAT06 - Execução de Smoke Tests.	100

Figura 29 – Modelo QEF

Em suma, como apresentado na Figura 29, os resultados pretendidos para os indicadores Escalabilidade e Elasticidade, Manutenibilidade e Qualidade do Processo de Desenvolvimento foram cumpridos na totalidade. Deste modo, através do modelo QEF apresentado, os resultados englobando todos os indicadores foram atingidos a 100%.

7 Conclusões

Tal como referido na secção 1.3, pretendia-se, com esta dissertação, reestruturar a aplicação monolítica atual para uma arquitetura que colmata-se os problemas existentes (secção 1.2).

De facto, um dos objetivos desta dissertação era a reestruturação da aplicação existente para uma arquitetura que possibilitasse a obtenção de escalabilidade e elasticidade, aumentar o desempenho e aumentar os níveis de manutenibilidade. O outro objetivo passava pelo desenvolvimento de uma estratégia de *Continuous Delivery* e *Continuous Integration* de forma a melhorar a qualidade do processo de desenvolvimento de software.

Posto isto, foi desenvolvida uma aplicação sob uma arquitetura baseada em microsserviços com implantação em *Cloud* de forma a colmatar as lacunas da aplicação eDocuments existente. Desta forma, foi analisada a solução existente e o seu modelo de domínio de forma a efetuar uma separação da mesma em diversos microsserviços geridos e implantados de forma independente com o intuito de aumentar a manutenibilidade e obter escalabilidade (e elasticidade), respetivamente. Além da separação em microsserviços, foram necessários a investigação e o uso de ferramentas que permitissem a comunicação entre os mesmos de forma a cumprir as funcionalidades/requisitos da plataforma.

Para além disso, foi desenvolvido um *pipeline* de *Continuous Delivery* e *Continuous Integration* que permitiu automatizar diversas tarefas que eram efetuadas de forma manual. Com utilização do Bitbucket Pipelines como ferramenta de integração contínua, foi desenvolvido um *pipeline* que, de forma automatizada, efetua desde a *build* do código fonte até à implantação do software.

Após desenvolvida a reestruturação para uma arquitetura baseada em microsserviços foi avaliado o desempenho da nova aplicação em comparação com o desempenho da solução existente em termos de tempos de processamentos de documentos em simultâneo. Para o indicador desempenho, foi comparado o tempo de processamento e o tempo médio de processamento por documento para o *workflow* mais utilizado pela aplicação. Comparados os tempos obtidos por ambas as aplicações, os objetivos estabelecidos de redução de 10% do

tempo total de processamento e do tempo médio de processamento por documento foram superados. Para tempo total de processamento, a nova aplicação obteve tempos inferiores em 84,18%, 69,67% e 73,20% para o processamento de 250, 500 e 1000 documentos em simultâneo, respetivamente. No que diz respeito ao tempo médio de processamento por documento, a nova aplicação conseguiu um redução temporal de 48,29%, 32,04% e 30,92% para o processamento de 250, 500 e 1000 documentos em simultâneo, respetivamente.

Para além do desempenho, indicadores como Escalabilidade e Elasticidade, Manutenibilidade e Qualidade do Processo de Desenvolvimento apresentaram, também, resultados positivos. Avaliados segundo o modelo QEF, todos os fatores e respetivos requisitos mencionados para cada um destes indicadores foram atingidos na totalidade.

Em suma, com esta dissertação foi possível obter uma aplicação sob uma arquitetura baseada em microsserviços que conseguiu atingir os objetivos estabelecidos no âmbito deste projeto. Para além da reestruturação da arquitetura, foi obtida, também, uma maior qualidade no processo de desenvolvimento de software com a automação de diversas tarefas através do desenvolvimento e implementação de um *pipeline* de *Continuous Delivery* e *Continuous Integration* que permitiu automatizar diversas tarefas que eram efetuadas de forma manual.

7.1 Limitações

A principal limitação existente prende-se com o facto dos tempos de processamento de documentos dependerem, também, do tempo de latência de rede na comunicação síncrona que por vezes sucede entre microsserviços. Para os testes efetuados, todos os microsserviços (e também a aplicação existente) foram executados sob o mesmo *hardware* de modo a serem garantidas as mesmas condições. Deste modo, os resultados obtidos apresentam um reduzido tempo de latência de rede que, após a aplicação ser implantada em *Cloud*, poderá resultar em tempos ligeiramente superiores.

Outra limitação é o facto de, apesar de cumpridos os objetivos estabelecidos para a Manutenibilidade, no futuro, correções ou incrementos de funcionalidades poderão ser dificultados devido à maior complexidade e dificuldade de monotorização derivado das comunicações assíncronas.

7.2 Trabalho futuro

Futuramente, com o intuito de dar continuidade ao crescimento e melhoria da plataforma eDocuments, poderá ser explorada a heterogeneidade existente na solução desenvolvida (secção 2.1.1.2). Deste modo, poderão ser investigadas e analisadas outras linguagens de programação assim como sistemas de gestão de base de dados de forma a encontrar as ferramentas mais adequadas tendo em consideração as funcionalidades definidas para cada microsserviço.

Outro aspeto importante, deverá passar pela monitorização e análise contínua da aplicação de forma a tentar ajustar os recursos computacionais alocados na *Cloud* de forma a obter uma maior eficiência dos mesmos.

Com o crescimento deste projeto, poderá, também, ser analisada outra vertente da manutenibilidade. Poderá ser explorada de que forma a solução desenvolvida permite uma diminuição dos tempos de resolução de problemas/incremento de novas funcionalidades relativamente à aplicação atual. Para isto, a plataforma Jira (já utilizada no processo de desenvolvimento de software) oferece ferramentas que permitirão comparar os tempos relativos à resolução de problemas da nova solução com o histórico de tempos correspondentes à resolução de problemas da solução atualmente existente.

Referências

Abdullin, R. (2010). CQRS Documents by Greg Young. 56.

Add API Documentation—WSO2 API Manager Documentation 3.0.0. (sem data). Obtido de https://apim.docs.wso2.com/en/latest/Learn/DesignAPI/APIDocumentation/add-api-documentation/

Apache Kafka. (sem data). Obtido de Apache Kafka website: https://kafka.apache.org/intro

API Gateway. (sem data). Obtido de https://www.jhipster.tech/api-gateway/

API Management—On-Premise and in the Cloud. (sem data). Obtido de https://wso2.com/api-management/

Atlassian. (sem data-a). Bitbucket—Pricing. Obtido de Atlassian website: https://bitbucket.org/product/pricing

Atlassian. (sem data-b). What is a Kanban Board? Obtido de Atlassian website: https://www.atlassian.com/agile/kanban/boards

Behara, S. (2018, Setembro 13). Monolith to Microservices Using the Strangler Pattern—DZone Microservices. Obtido de Dzone.com website: https://dzone.com/articles/monolith-to-microservices-using-the-strangler-patt

Bitbucket Pipelines vs Jenkins Pipeline. (sem data). Obtido de https://hackernoon.com/bitbucket-pipelines-vs-jenkins-pipeline-f3b7c0e1c198

Bitbucket vs Jenkins | What are the differences? (sem data). Obtido de StackShare website: https://stackshare.io/stackups/bitbucket-vs-jenkins

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S., & Mazzara, M. (2017). *From Monolithic to Microservices: An experience report*. https://doi.org/10.13140/RG.2.2.34717.00482

Case Study: From Monolith to Microservices With Micronaut. (sem data). Obtido 30 de Novembro de 2019, de https://objectcomputing.com/about/our-clients/case-studies/case-study-from-monolith-to-microservices-with-micronaut

Consul. (sem data). Obtido de https://www.jhipster.tech/consul/

Consul vs Eureka | What are the differences? (sem data). Obtido de https://stackshare.io/stackups/consul-vs-eureka

Docker Hub—Container Image Library | Docker. (sem data). Obtido 8 de Fevereiro de 2020, de https://www.docker.com/products/docker-hub

Doerrfeld, B. (sem data). From monolith to microservices: Horror stories and best practices. Obtido 30 de Novembro de 2019, de TechBeacon website: https://techbeacon.com/app-dev-testing/monolith-microservices-horror-stories-best-practices

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. Em M. Mazzara & B. Meyer (Eds.), *Present and Ulterior Software Engineering* (pp. 195–216). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-67425-4_12

EDI. (2020). Em *Wikipédia, a enciclopédia livre*. Obtido de https://pt.wikipedia.org/w/index.php?title=EDI&oldid=58476987

Flowinn – Business Inovation. (sem data). Obtido 16 de Fevereiro de 2020, de https://www.flowinn.biz/

Fowler, M. (2005). Event Sourcing. Obtido de Martinfowler.com website: https://martinfowler.com/eaaDev/EventSourcing.html

Fowler, M. (2011). CQRS. Obtido de Martinfowler.com website: https://martinfowler.com/bliki/CQRS.html

Fowler, M. (2013, Maio 30). Continuous Delivery. Obtido de Martinfowler.com website: https://martinfowler.com/bliki/ContinuousDelivery.html

Fowler, M., & Lewis, J. (2014). Microservices. Obtido de https://www.martinfowler.com/articles/microservices.html

Gafert, P. (sem data). Unit test your Java architecture. Obtido 8 de Fevereiro de 2020, de ArchUnit website: https://www.archunit.org/

Humphrey, P. (2017, Abril 26). Understanding When to use RabbitMQ or Apache Kafka. Obtido de https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

Introdução aos Pipelines Bitbucket—Atlassian Documentation. (sem data). Obtido de https://confluence.atlassian.com/bitbucket/get-started-with-bitbucket-pipelines-792298921.html
Introduction—Consul by HashiCorp. (sem data). Obtido de https://www.consul.io/intro/index.html
Jenkins. (sem data). Obtido de Jenkins website: https://jenkins.io/index.html
Jenkins User Documentation. (sem data). Obtido de Jenkins User Documentation website: https://jenkins.io/doc/index.html
JHipster Registry. (sem data). Obtido de https://www.jhipster.tech/jhipster-registry/
Koen, P. A., Ajamian, G. M., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S., … Seibert, R. (2002). Effective Methods, Tools, and Techniques. *The PDMA ToolBook for New Product Development*, 32.
Kubernetes Components. (sem data). Obtido 19 de Setembro de 2020, de Kubernetes website: https://kubernetes.io/docs/concepts/overview/components/
Levy, E. (2019, Maio 7). Kafka vs. RabbitMQ: Architecture, Performance & Use Cases. Obtido de https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case
Messaging that just works—RabbitMQ. (sem data). Obtido de https://www.rabbitmq.com/
Microservices at Netflix: Lessons for Architectural Design. (2015, Fevereiro 19). Obtido 1 de Dezembro de 2019, de NGINX website: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/
Newman, S. (2015). *Building Microservices*. 473.
Pacheco, V. F. (2018). Microservice Patterns and Best Practices: Explore Patterns Like CQRS and Event Sourcing to Create Scalable, Maintainable, and Testable Microservices.
Rich, N., & Holweg, M. (2000). *Value analysis, Value engineering*. 32.
Richardson, C. (2018). *Microservices patterns*.
Saaty, T. L. (1990). *How to make a decision: The analytic hierarchy process*.
Scaling Wix to 60M Users—From Monolith to Microservices—Wix Tech Stack. (sem data). Obtido 7 de Dezembro de 2019, de https://stackshare.io/wix/scaling-wix-to-60m-users-from-monolith-to-microservices
Scaling wix with microservices architecture devoxx London 2015. (sem data). Obtido 13 de Dezembro de 2019, de https://www.slideshare.net/aviranwix/scaling-wix-with-microservices-architecture-devoxx2015
Service Discovery with WSO2 API Microgateway. (sem data). Obtido de https://wso2.com/blogs/thesource/2019/08/service-discovery-with-wso2-api-microgateway/
Sharma, S. (2017). *The DevOps Adoption Playbook: A Guide to adopting DevOps in a multi-speed IT Enterprise*. Indianapolis, Indiana: John Wiley & Sons, Inc. https://doi.org/10.1002/9781119310778
Swagger (software). (2020). Em *Wikipedia*. Obtido de https://en.wikipedia.org/w/index.php?title=Swagger_(software)&oldid=977456358
Thennakoon, T. (2019, Março 20). Microservices implementation—Netflix stack. Obtido 8 de Dezembro de 2019, de Medium website: https://medium.com/@tharanganilupul/microservices-implementation-netflix-stack-ba4f4a57a79f
Throttling Use-Cases—WSO2 API Manager Documentation 3.0.0. (sem data). Obtido de https://apim.docs.wso2.com/en/latest/Learn/RateLimiting/introducing-throttling-use-cases/
What is a Container? | App Containerization | Docker. (sem data). Obtido 8 de Fevereiro de 2020, de https://www.docker.com/resources/what-container
What is Kubernetes? (sem data). Obtido 19 de Setembro de 2020, de Kubernetes website: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience – HYS Enterprise. (sem data). Obtido 1 de Dezembro de 2019, de https://www.hys-enterprise.com/blog/why-and-how-netflix-amazon-and-uber-migrated-to-microservices-learn-from-their-experience/
Wikipedia contributors. (2018). Front end innovation. Em *Wikipedia*. Obtido de https://en.wikipedia.org/w/index.php?title=Front_end_innovation&oldid=819255121
Working with Analytics—WSO2 Enterprise Integrator 6.x.x—WSO2 Documentation. (sem data). Obtido de https://docs.wso2.com/display/EI6xx/Working+with+Analytics