INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema centralizado de monitorização de logs orientado a microsserviços

RAFAEL GOMES ALMEIDA SIL RIBEIRO
Junho de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Sistema centralizado de monitorização de logs orientado a microsserviços

Rafael Gomes Almeida Sil Ribeiro

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

Orientador: Alexandre Bragança

Porto, 29 de junho de 2022

À minha família e amigos que me incentivaram e apoiaram durante esta etapa da minha vida.

Rafael Ribeiro

Resumo

Recentemente os sistemas baseados em microsserviços têm demonstrado uma forte adesão no mundo organizacional. Inclusivamente empresas como a Netflix, Amazon e o The Guardian estão a adotar este estilo arquitetural como forma de evoluir as suas aplicações.

Se bem que existem vantagens associadas ao uso de microsserviços no desenvolvimento de um sistema (maior modularidade e escalabilidade, melhor separação de responsabilidades, entre outros), existem também fatores que afetam negativamente o dito sistema. Um desses pontos negativos remete para o tempo despendido a diagnosticar problemas ou anomalias no sistema.

Assim sendo, a presente dissertação foca-se no desenvolvimento de uma solução centralizada de monitorização de *logs* que visa diminuir o consumo de tempo e a taxa de erro humano que existe quando se efetua a depuração de sistemas baseados em microsserviços. De maneira a atingir este objetivo, analisaram-se soluções semelhantes à pretendida nesta dissertação e projetos anteriormente elaborados de modo a se obter um ponto de referência aquando da definição dos requisitos da solução.

Apresentam-se diferentes arquiteturas passíveis de se aplicar à solução e finalmente após uma escolha devidamente fundamentada da arquitetura, detalha-se o processo de implementação da mesma tirando proveito de tecnologias *open-source* pertencentes à *ELK stack*. Por último, avalia-se os atributos de qualidade da solução tais como a eficiência, usabilidade e funcionalidade da mesma.

Conclui-se que a solução final responde de um modo satisfatório aos requisitos estabelecidos, que incluem não só mas também as funções de alertas automáticos e de visualização de informação do sistema monitorizado.

Palavras-chave: Microsserviços, *Logging* centralizado, ELK *stack*, Monitorização, Alertas

Abstract

Recently microservices based systems have shown strong adoption in the enterprise world. Even companies like Netflix, Amazon and The Guardian are now deploying this architectural style as a way to evolve their applications.

While there are advantages associated with the use of microservices in the development of a system (greater modularity and scalability, better separation of responsibilities, among others), there are also factors that negatively affect the system. One of these negative points refers to the time spent in order to diagnose problems or anomalies in the system.

Therefore, this dissertation focuses on developing a centralized log monitoring solution that aims to decrease the time consumption and human error rate that exists when debugging microservices-based systems. To achieve this goal, solutions similar to the one intended in this dissertation and previously developed projects were analyzed in order to obtain a reference point when defining the solution's requirements.

Different architectures that can be applied to the solution are portrayed and finally after a well-founded choice of architecture, the process of its implementation is detailed while taking advantage of open-source technologies belonging to the ELK stack. Lastly, the quality attributes of the solution are evaluated, such as efficiency, usability and functionality.

It was concluded that the final solution satisfactorily meets the established requirements, which include not only but also the functions of automatic alerts and the visualization of information from the monitored system.

Keywords: Microservices, Centralized Logging, ELK stack, Monitoring, Alerts

Agradecimentos

Começo por agradecer à minha família, especialmente aos meus pais e à minha avó materna pelo apoio incondicional que têm proporcionado ao longo deste projeto.

Aos meus colegas de trabalho que desde início suportaram a minha decisão de colocar o percurso académico como primeira prioridade.

Aos meus colegas de grupo e parceiros académicos pelos momentos partilhados, sejam eles bons ou menos bons. Especialmente ao meu colega Paulo Pereira com quem partilhei não menos de 90% das experiências vivenciadas no ISEP, um brinde às noites em branco passadas na cantina.

Por último venho agradecer ao meu orientador Alexandre Bragança por aceitar este desafio e por demonstrar uma tremenda disponibilidade sempre que necessitei de esclarecimentos. É indubitável que todo o *feedback* fornecido fomentou o trabalho desenvolvido nesta dissertação.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Lista de Acrónimos

ACID	*Atomicidade Consistência Isolamento e Durabilidade*.
AHP	*Analytic Hierarchy Process*.
API	Interfaces de Programação de Aplicação.
BASE	Basicamente disponível, estado leve e eventualmente consistente.
DSL	*Domain Specific Language*.
DSR	*Design Science Research Methodology*.
E2E	*End-to-End*.
ELK	ElasticSearch, Logstash e Kibana.
FDD	*Feature-Driven Development*.
FFE	*Fuzzy Front End*.
HTTP	Protocolo de transferência de hipertexto.
JSON	Notação de Objeto JavaScript.
MV	Máquina Virtual.
QEF	*Quantitative Evaluation Framework*.
QFD	*Quality Function Deployment*.
RESP	*REdis Serialization Protocol*.
RGPD	Regulamento Geral de Proteção de Dados.
SCV	Sistema de Controlo de Versões.
SCVC	Sistema de Controlo de Versões Centralizados.
SCVD	Sistema de Controlo de Versões Distribuído.
SGBD	Sistema de Gestão de Base de Dados.
SGBD	*Test-Driven-Development*.
SI	Sistema Ideal.
SR	Sistema Real.
SSL	*Secure Sockets Layer*.
SUS	*System Usability Scale*.
UML	Linguagem de Modelação Unificada.

Capítulo 1

Introdução

Neste capítulo introdutório, é exposto o panorama geral do projeto elaborado no contexto da unidade curricular de Tese/Dissertação/Estágio (TMDEI) do Mestrado em Engenharia Informática (MEI) do Instituto Superior de Engenharia do Porto (ISEP). O projeto pormenoriza e contextualiza a temática de sistemas de monitorização orientados a arquiteturas de microsserviços.

São abordados tópicos que incluem o problema em causa, bem como os objetivos e as respetivas metodologias empregues de forma a cumprir na totalidade com os objetivos definidos. Este capítulo é concluído com os resultados esperados onde é aludida a importância do trabalho juntamente com uma breve descrição dos capítulos que se seguem.

1.1 Enquadramento e Contexto

Nos últimos anos e em particular atualmente, os microsserviços são uma tendência que apresenta um crescimento acelerado no mundo empresarial, empresas como Netflix, Amazon, The Guardian, evoluíram as suas aplicações para uma arquitetura de microsserviços [1].

Este estilo arquitetural permite fragmentar grandes soluções de software em pequenos serviços independentes e fracamente acoplados. Cada um destes serviços é executado no seu próprio processo e o mecanismo de comunicação geralmente adotado é assíncrono e envolve API's com recurso ao protocolo HTTP. Como resultado tem-se um maior nível de escalabilidade, manutibilidade e de isolamento de falhas no sistema como um todo [2].

Uma das principais formas de averiguar o comportamento de um determinado sistema em ambiente de produção é através do uso de sistemas de monitorização. A informação que é fornecida pelos *logs* aplicacionais é essencial para corretamente detetar e diagnosticar comportamentos indesejados no intuito de melhorar a fiabilidade do sistema [3].

Os sistemas de monitorização atuais focam-se acima de tudo nas seguintes vertentes [4]:

- **Monitorização do desempenho**: Este tipo de monitorização preocupa-se com questões como o tempo de inicialização de uma aplicação ou o tempo de resposta;
- **Monitorização da disponibilidade**: Consiste em garantir que o servidor responde aos pedidos efetuados;
- **Monitorização dos recursos**: Verifica se os recursos dos servidores não estão a ultrapassar a sua capacidade;
- **Monitorização de erros**: Responsável por alertar os administradores de sistemas sempre que a frequência ou natureza dos erros se desviar da norma;

- **Monitorização dos logs**: Processo pelo qual os *logs* são avaliados à medida que são registados;
- **Monitorização da base de dados**: Preocupa-se com perigos passíveis de ocorrer no sistema de base de dados;
- **Monitorização da segurança**: Procura ativamente por atividades suspeitas quer da parte do sistema quer do utilizador.

Das vertentes supracitadas, o presente relatório foca-se predominantemente na monitorização de *logs*.

Denota-se também que uma vez que o projeto não vai ser desenvolvido no contexto de uma organização, escolheu-se um cenário hipotético de um sistema composto por vários microsserviços onde os *logs* são armazenados de forma distinta, o dito cenário encontra-se detalhado numa fase posterior do relatório.

1.2 Descrição do Problema

Apesar das vantagens da utilização de microsserviços enumeradas na secção anterior, existem também vários inconvenientes associados ao seu uso. A execução de um sistema de microsserviços pode abranger um grande numero de interações, onde parte delas são assíncronas e envolvem uma cadeia complexa de invocações. Esta situação pode-se complicar devido à natureza dinâmica dos microsserviços, visto que podem existir milhares de instâncias de um microsserviço a serem executadas em diversos ambientes [5].

Deste modo, é possível inferir que a complexidade e o dinamismo de sistemas baseados em microsserviços representam desafios grandes e únicos no que toca a questões de *debugging*, pois os desenvolvedores necessitam de raciocinar sobre o comportamento concorrente de diferentes microsserviços bem como precisam de compreender a topologia de todo o sistema [6].

Posto isto, o problema a ser solucionado pelo projeto em causa passa pelo desenvolvimento de uma solução capaz de tratar e apresentar *logs* gerados pelos diferentes microsserviços, de modo a reduzir o consumo de tempo e a taxa de erro humano quando se efetua *debugging* do sistema.

1.3 Objetivos

A finalidade do projeto traduz-se no desenvolvimento de um sistema capaz de extrair, processar, armazenar e apresentar *logs* provenientes de microsserviços, que não sejam necessariamente sistemas homogéneos, isto é, aplicações desenvolvidas em diferentes linguagens com sistemas de armazenamento de *logs* distintos (ficheiros de texto, base de dados, etc.).

Numa vertente mais científica, optou-se por expressar os objetivos do trabalho através da formulação de perguntas de pesquisa [7] que se encontram abaixo identificadas:

RQ.1 : Quais são as semelhanças e as diferenças das plataformas/ferramentas atualmente existentes no mercado que lidam com tópicos como centralização de *logs*?

RQ.2 : Quais os requisitos funcionais e não funcionais que se devem de ter em consideração no desenvolvimento de um sistema de monitorização de *logs* centralizado?

RQ.3 : Como deve ser desenhada uma solução de monitorização de *logs* centralizada, de modo a se obter uma arquitetura robusta e escalável seguindo boas práticas de desenvolvimento de software?

RQ.4 : Quais os fatores que devem ser considerados no momento da avaliação de uma solução de monitorização de *logs* centralizada?

As questões previamente referidas são abordadas e respondidas no decorrer do relatório, sempre tendo em consideração o âmbito do projeto.

1.4 Metodologia

Devido à natureza do projeto, existe uma necessidade inerente de se efetuar pesquisas. Por consequência, há uma metodologia empregue mesmo que esta não se encontre explícita. Ao explicar-se a metodologia utilizada, dá-se a entender ao leitor o porquê de se efetuar pesquisas de determinada maneira, tal como facilita-se a compreensão do que efetivamente está a ser realizado [8].

Durante o decorrer do projeto, o autor decidiu utilizar a metodologia *Design Science Research Methodology* (**DSRM**). Esta metodologia é direcionada no desenvolvimento de novos artefactos de modo a tratar um tipo generalizado de problema, avaliando a utilidade dos artefactos para resolver problemas desse tipo [9].

Para complementar a análise, a tabela 1.1 detalha os passos subjacentes da metodologia em questão.

Tabela 1.1: Passos correspondentes à metodologia DSRM [10]

Passo	**Preocupações**	**Output para o próximo passo**	**Ponto de partida**
1. Identificar Problema	- Definir problema - Evidenciar importância	Inferência	Iniciação centrada no problema
2. Definir objetivos de uma solução	- O que realizaria melhor um novo artefacto?	Teoria	Iniciação centrada no objetivo
3. Desenho & Desenvolvimento	- Artefacto	Conhecimento de como fazer	Iniciação centrada no desenho e desenvolvimento
4. Demonstração	- Encontrar contexto apropriado - Usar artefacto para resolver o problema	Métricas, Conhecimento de análises	Cliente/Contexto iniciado
5. Avaliação	- Observar eficácia e eficiência - Iterar novamente para o Desenho	Conhecimento disciplinar	
6. Comunicação	- Publicações académicas - Publicações profissionais		

Como se pode observar, é possível fazer um enquadramento dos passos da metodologia com as diferentes fases do projeto.

A primeira fase do projeto denominada de **Análise**, remete para a contextualização do problema em conjunto com os objetivos a atingir, também inclui o estudo de aplicações baseadas em microsserviços especialmente o modo como geram e armazenam os respetivos *logs*. Além disso, são analisadas ferramentas e tecnologias atuais que auxiliem na resolução do problema e em seguida define-se os requisitos da solução a desenvolver. Por último, averigua-se o valor de negócio associado ao projeto. Esta fase vai de encontro com a descrição dos primeiros dois passos da metodologia.

A segunda e terceira fase, **Desenho** e **Implementação** estão em conformidade com o terceiro passo da metodologia e consiste na identificação de diferentes arquiteturas capazes de colmatar os problemas identificados. Após a seleção da arquitetura que melhor se adequa ao projeto, as fases em questão são finalizadas através da implementação da arquitetura escolhida segundo padrões e boas práticas de desenvolvimento de software.

Após a implementação da solução é necessário encontrar um contexto apropriado para a aplicar, isto significa que é necessário simular um cenário de caso real onde seja possível demonstrar o funcionamento do artefacto. Esta fase coincide com o quarto passo da metodologia.

O quinto passo da metodologia reflete-se na fase de **Testes** do projeto. Esta fase equivale não só a avaliar a solução implementada mediante o modelo de avaliação escolhido, como também na elaboração de um plano de testes que inclua quer requisitos funcionais como os não funcionais do projeto.

O sexto e último passo da metodologia é referente ao presente documento, uma vez que este encontra-se compreendido na preocupação "Publicações académicas". Este ponto é concebido desde da data de início até à data de fim do projeto.

Por último e de maneira a se efetuar um paralelismo com as fases previamente mencionadas, a figura 1.1 apresenta de uma forma concisa o planeamento do trabalho a ser empregue durante a totalidade do projeto.

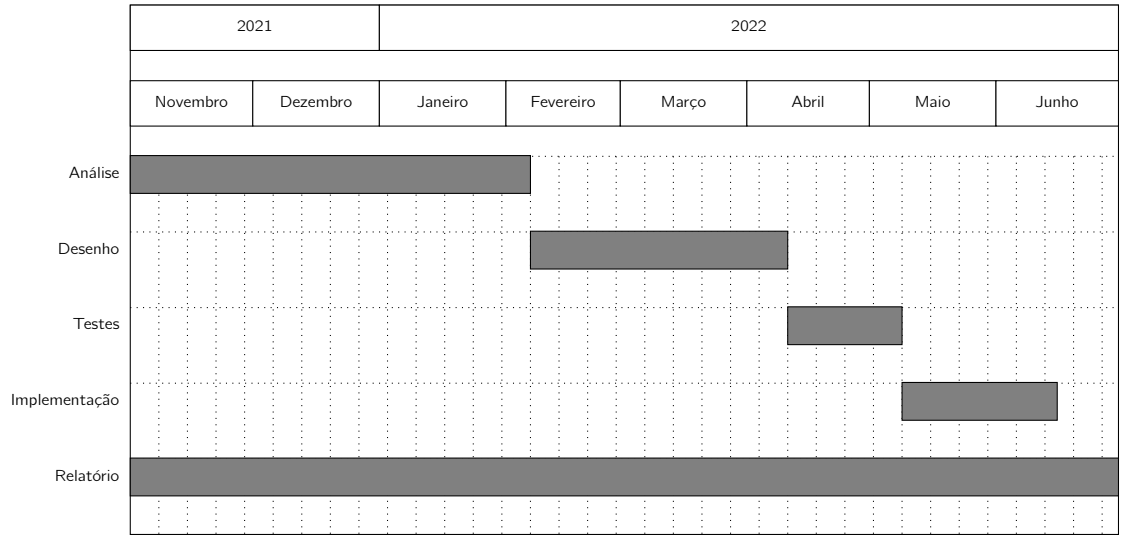

Figura 1.1: *Roadmap* do projeto

1.5 Contributos

Um dos contributos principais do desenvolvimento deste projeto envolve a utilização da solução desenvolvida. Espera-se que seja reduzido substancialmente o tempo despendido a depurar erro ou anomalias em sistemas distribuídos, particularmente microsserviços. Não só o desenvolvedor poupa tempo que pode administrar noutras tarefas, como também reduz a suscetibilidade de erros associada à análise e conjugação manual de *logs* de diferentes aplicações.

Complementarmente, expõe-se de seguida os aspetos consolidados no decorrer do projeto que se consideram importantes de assimilar:

- Estado da arte das tecnologias e ferramentas atualmente no mercado que melhor respondem aos objetivos definidos. São efetuadas comparações entre as diversas plataformas de forma a reter os pontos positivos e evitar os pontos negativos de cada uma;
- Aplicação da metodologia de análise de valor à solução no intuito de reduzir o custo e aumentar o valor associado;
- Levantamento e definição dos requisitos funcionais e não funcionais que devem estar contemplados na solução;
- Argumentação da escolha da arquitetura que melhor responde aos critérios delineados;
- Escolha justificada de um modelo de avaliação que é posteriormente empregue à solução desenvolvida.

Não menos importante e tendo em conta que no ciclo de vida de uma aplicação de software, quase 70% do tempo e recursos são alocados para atividades de manutenção [11], espera-se que este projeto contribua na redução de tempo e recursos destinados a efetuar a manutibilidade destes sistemas.

1.6 Estrutura do Relatório

O relatório em questão, incluindo a introdução, é composto por mais seis capítulos que são em seguida enumerados.

O capítulo 2, cujo título é **Contextualização e Estado da Arte**, aborda conceitos-chaves para o projeto, tendo um foco especial em aplicações de monitorização centralizadas e tecnologias *open-source* que atendem às necessidades do projeto.

O capítulo 3, cujo título é **Análise**, apresenta os diferentes métodos de análise adotados ao projeto no intuito de determinar a real necessidade do mesmo bem como as funcionalidades que este deve deter.

O capítulo 4, cujo título é **Desenho da Solução**, expõe os artefactos que refletem a arquitetura empregue à solução, tendo em conta os requisitos enunciados no capítulo anterior.

O capítulo 5, cujo título é **Implementação da Solução**, denota os pormenores técnicos da implementação aplicada ao projeto, juntamente com excertos e ilustrações que corroborem essa mesma implementação.

O capítulo 6, cujo título é **Avaliação da Solução**, apresenta a metodologia de avaliação adotada na solução. Também são detalhados os testes efetuados tal como o comportamento da solução perante os mesmos.

O sétimo e último capítulo, cujo título é **Conclusão**, expõe as conclusões obtidas do projeto como um todo. São também enfatizados os objetivos que se concretizaram e possíveis melhorias para um trabalho futuro.

Por último e igualmente importante, tem-se no final do presente relatório uma secção dedicada aos anexos do documento. Estes anexos expõem excertos e ilustrações que visam auxiliar a interpretação dos capítulos previamente mencionados.

Capítulo 2

Contextualização e Estado da Arte

De forma a melhorar a compreensão do projeto este capítulo enquadra o contexto real do mesmo, além disso também detalha o estado da arte da área de atuação. Numa fase inicial é apresentada uma contextualização do problema e dos conceitos de negócio subjacentes, adicionalmente são analisadas plataformas existentes no mercado de forma a verificar o que estas oferecem. Por último e de modo a escolher as tecnologias que mais se adequem ao problema, estas são comparadas e avaliadas segundo métricas que façam sentido no contexto do projeto.

2.1 Área de atuação

Quase todas as aplicações executadas num ambiente de servidor geram *logs* automaticamente. Estes registos são uma parte crucial de qualquer sistema pois são eles que fornecem informações essenciais sobre o seu comportamento no momento atual ou no passado. Ao analisar a informação dos *logs* é possível detetar eventuais problemas, erros e tendências. Todavia, o processo de analise dos ditos *logs* pode ser demoroso e frustrante devido a ser necessário procurar manualmente por um erro específico em centenas, ou até mesmo milhares de servidores que albergam inúmeros *logs* [12].

Com isto em mente e de forma a extinguir os problemas previamente mencionados optou-se por abordar sistemas de monitorização centralizados de *logs*. Um sistema que se enquadre nessa categoria, consiste num tipo de solução projetado para recolher *logs* de diferentes servidores e consolidar os dados. Estes sistemas apresentam os dados numa interface gráfica que deve ser acessível e de fácil usabilidade. Por conseguinte, o objetivo destes sistemas é agilizar e automatizar o processo de gestão de *logs* provocando uma redução do tempo e do esforço frequentemente envolvidos na manutenção de um grande repositório de *logs* [13].

As principais etapas referentes a sistemas de monitorização centralizados de *logs* são as seguintes [14]:

1. **Agrupar**: Os *logs* provenientes de diferentes locais e ambientes são agrupados;
2. **Transportar**: Transferir os *logs* até ao local centralizado;
3. **Armazenar**: Mecanismo de armazenamento escalável que implemente rotação de *logs* [1] e controlo de acesso;

[1] Processo automático utilizado na administração de sistemas em que os ficheiros de *logs* são comprimidos, arquivados, renomeados ou apagados a partir de uma certa data ou quando excedem um determinado tamanho [15]

4. **Analisar**: Ferramentas e componentes com interface gráfica capazes de analisar a informação dos *logs* de uma maneira conveniente.

Opcionalmente, existe também uma quinta etapa que consiste na configuração de um sistema de alertas tendo por base a análise dos *logs*. Estes sistemas provam a sua utilidade quando se gere centenas de servidores, onde não se consegue identificar facilmente comportamentos anómalos devido ao vasto volume de dados.

2.2 Conceitos de negócio

De modo a expandir o tema retrato neste projeto, esta secção aborda os diferentes conceitos de negócio que estão implícitos quando se fala de sistemas de monitorização centralizados em particular sistemas centralizados de *logs*.

2.2.1 Logs

Na área da computação, um *log* é definido como um registo dos eventos que ocorreram nos sistemas ou redes de uma organização. Os *logs* são compostos por entradas de registo onde cada uma destas contém informações relacionadas a um evento em específico. Originalmente os *logs* eram primordialmente utilizados na depuração de problemas, contudo agora servem diversas funções tais como otimizar o desempenho da rede e do sistema, registar as ações dos utilizadores e fornecer informação útil na perscruta de atividades maliciosas [16].

Ao incluir-se *logs* no desenvolvimento de software é essencial decidir qual a informação a registar e como a registar. Os "melhores" *logs* informam exatamente o que aconteceu, como, onde e quando. Cada entrada de *log* deve registar o que aconteceu, quando aconteceu, quem despoletou o evento e o porquê de ter acontecido. Como resultado destas diretrizes, uma entrada de *log* deve, sempre que possível, responder às seguintes questões [17]:

1. **O que aconteceu?**: Deve ser possível distinguir o tipo da ocorrência (aviso, informação, erro) bem como os elementos afetados (e.g componentes do sistema);
2. **Quando aconteceu?**: Presença do registo da data e da hora da ocorrência juntamente com o fuso horário;
3. **Onde aconteceu?**: Indicação de qual sistema, aplicação, classe em causa. A resposta a esta pergunta deve conter a entidade de origem e a entidade de destino/alvo;
4. **Quem acionou o evento?**: Existência do nome de utilizador que por sua vez informa quem ativamente provoca essas ações;
5. **Quais foram as partes envolvidas?**: Revela se a ocorrência afetou mais utilizadores para além daquele que a causou;
6. **Porquê que aconteceu?**: Explicita a razão da ocorrência, e.g palavra-passe incorreta, parâmetros obrigatórios não preenchidos, etc.

Em suma, estas questões devem sempre ser respondidas através dos *logs*, sendo que algumas destas respostas são bastantes diretas.

2.2.2 Tracing

Tracing, que traduzido do inglês significa rastreamento, remete para o uso específico de *logs* com o objetivo de registar informações sobre a execução de um determinado programa. O *tracing* pode ser definido como o processamento completo de um dado pedido. Ele representa o caminho percorrido pela *stack* da aplicação ou do sistema e também armazena informações mais detalhadas do que as geralmente presentes em *logs* "comuns"[16].

Em muitos casos, o *tracing* representa o caminho de um único utilizador pela totalidade da *stack* de uma aplicação. O seu propósito não é reativo, mas sim focado na otimização. Ao realizar-se o *tracing* da *stack*, os desenvolvedores conseguem identificar locais passíveis de melhorias. Quando ocorre uma anomalia, o *tracing* permite identificar o porquê da ocorrência dela bem como outros fatores [18]:

- Qual a função em causa;
- A duração da função;
- Os parâmetros passados à função;
- A profundidade atingida na função.

A figura 2.1 que representa o *trace* de um registo de erro, ilustra o caminho percorrido pelo programa até ao momento da exceção.

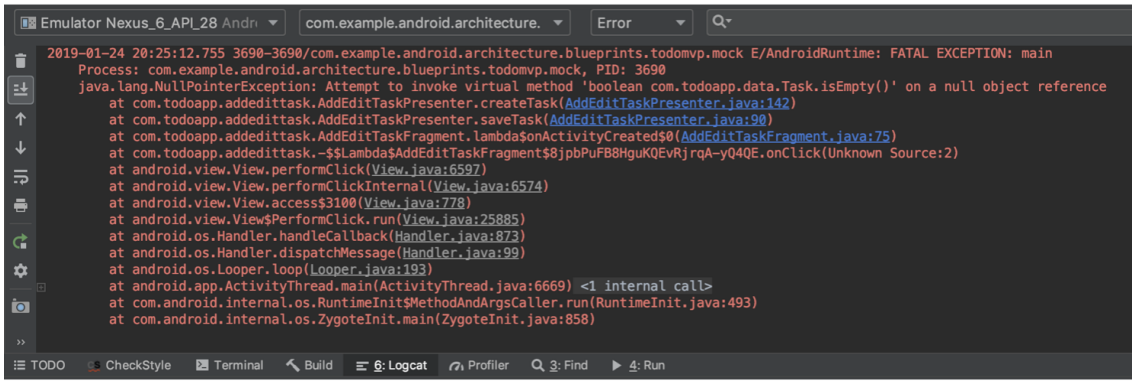

Figura 2.1: *Trace* de uma aplicação Android

Fonte: [19]

2.2.3 Logging como um serviço

Logging como um serviço, também conhecido como LaaS (*Logging as a Service*), consiste num serviço de gestão centralizado de *logs* na nuvem. Ele garante um mecanismo rápido e fiável de agregação e análise de *logs*. Este serviço é utilizado para diagnosticar problemas e compreender as tendências do sistema atualmente em monitorização [20]. O LaaS aumenta a eficiência dos desenvolvedores e torna as operações mais confiáveis atendendo a três requisitos principais: acesso, agregação e alertas [21].

A figura 2.2 apresenta numa perspectiva de granularidade grossa a arquitetura de um sistema LaaS composto não só mas também por serviços da Amazon.

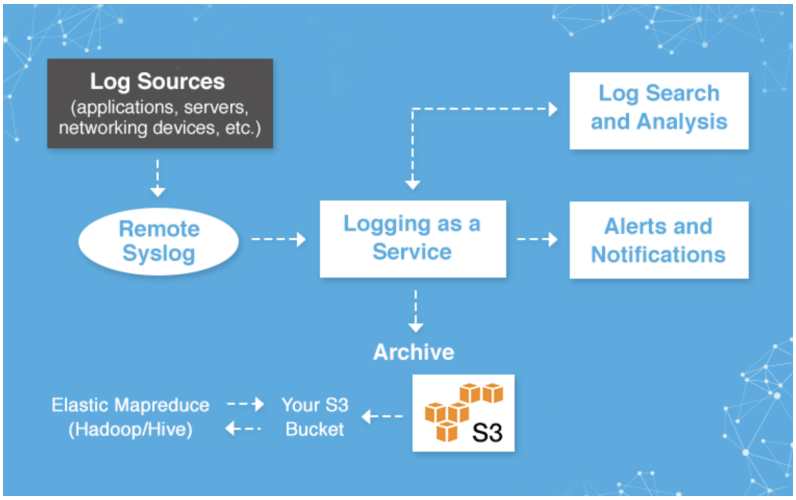

Figura 2.2: Arquitetura de LaaS

Fonte: [22]

Um dos benefícios do uso deste serviço diz respeito à garantia que se obtém relativamente à consistência de todo o sistema. Se for optada uma arquitetura de microsserviços no desenvolvimento de um determinado sistema, é uma mais valia se o processo de inspecionar *logs* for uniforme e consistente.

Na medida que o serviço LaaS está apenas concentrado em receber, processar e arquivar *logs* por consequência aumenta-se a coesão do sistema [23]. Com este conceito em mente, espera-se que a solução final satisfaça os critérios de um produto LaaS.

2.3 Trabalhos Relacionados

Neste secção são analisadas e exploradas várias aplicações que apresentam funcionalidades semelhantes às pretendidas neste projeto, mais concretamente aplicações que sejam capazes de agrupar, filtrar, armazenar e visualizar *logs* provenientes de diversas aplicações, nomeadamente produtos LaaS.

As plataformas abordadas nesta secção encontram-se abaixo enumeradas:

- **Splunk**: Plataforma focada na pesquisa, análise e monitoramento de dados. Atualmente acessível através do website https://www.splunk.com [24];
- **Loggly**: Plataforma provedora de serviços de análise e gestão de *logs* baseado na nuvem. Atualmente acessível através do website https://www.loggly.com [25];
- **LogDNA**: Plataforma que fornece soluções de gestão de *logs*. Atualmente acessível através do website https://www.logdna.com [26];
- **SumoLogic**: Plataforma de análise de dados baseada na nuvem focada em segurança, operações e casos de uso de BI (*Business Intelligence*). Atualmente acessível através do website https://www.sumologic.com [27];
- **Logz.io**: Plataforma que disponibiliza serviços de monitorização na nuvem. Atualmente acessível através do website https://logz.io [28];

- **Seq**: Servidor inteligente de pesquisa, análise e alertas criado especificamente para dados de *logs* estruturados que pode ser hospedado localmente ou na nuvem. Atualmente acessível através do website https://datalust.co/seq [29].

Estas plataformas foram seleccionadas com base numa lista relativamente recente (7 de janeiro de 2022) que enuncia as melhores plataformas e tecnologias de gestão de *logs*. Esta lista pode ser consultada a partir do website https://sematext.com/best-log-management-tools/ [30].

Todas as plataformas previamente expostas são soluções comerciais, no sentido em que oferecem diferentes planos de pagamento com base nas necessidades de cada cliente. Tendo em conta que o contexto deste projeto é meramente académico e como não se pretende suportar custos financeiros, achou-se pertinente comparar as plataformas a partir do plano gratuito que estas oferecem.

É importante notar que o autor não conseguiu identificar plataformas *open-source* que cumprissem na totalidade os critérios de um produto LaaS. Existem contudo tecnologias *open-source* que quando integradas e utilizadas em conjunto conseguem satisfazer as necessidades de um produto LaaS, estas tecnologias são pormenorizadas na secção 2.4.

2.3.1 Funcionalidades

As plataformas em questão oferecem funcionalidades distintas aos seus clientes. Enquanto que algumas têm um leque alargado de funcionalidades incluídas no plano de pagamento gratuito, outras já são mais restritas.

As funcionalidades que se consideraram importantes de comparar tendo por base aplicações LaaS, encontram-se abaixo detalhadas:

- **Alertas**: Possibilidade de configurar notificações via e-mail ou SMS;
- **Volume de indexação**: Quantidade de dados permitida indexar por dia;
- **Durabilidade da informação**: Intervalo de tempo em que informação está armazenada e disponível para consulta ou análise;
- **Suporte adicional**: Possibilidade de contactar a plataforma de forma a esclarecer dúvidas ou obter ajudar;
- **Filtragem da informação**: Existência de um mecanismo de pesquisa e filtragem da informação proveniente dos *logs*;
- **Uso concorrente**: Quantidade de utilizadores que podem simultaneamente aceder à mesma instância da plataforma;
- **Métricas personalizáveis**: Capacidade de adicionar métricas adaptadas às necessidades do sistema a ser monitorizado.

A tabela 2.1 apresenta o resultado da comparação entre as plataformas previamente mencionadas.

Após uma análise da tabela é possível averiguar que a maioria das plataformas não fornece suporte de alertas e quando o fazem, limitam o número de alertas como é o caso do SumoLogic que oferece 50 alertas e o caso do Logz.io que oferece 10. A única plataforma que oferece o uso ilimitado de alertas é o Seq.

Tabela 2.1: Comparação entre plataformas LaaS

	Splunk	**Loggly**	**LogDNA**	**SumoLogic**	**Logz.io**	**Seq***
Alertas	✖	✖	✖	Limitado	Limitado	✔
Volume da informação	500MB	200MB	Ilimitado	1GB	1GB	Ilimitado
Durabilidade da informação	8 dias	7 dias	0 dias	7 dias	1 dia	Ilimitado
Suporte	✖	✖	✖	✖	✖	✖
Filtragem da informação	✔	✔	✔	✔	✔	✔
Uso concorrente	1	Ilimitado	1	Ilimitado	Ilimitado	1
Métricas personalizáveis	✖	✖	✖	✖	✖	✔

* O plano gratuito do Seq requer hospedagem local e não na nuvem

Relativamente a volume e durabilidade da informação as plataformas variam ligeiramente entre si, sendo que todas são bastante restritas. Para os critérios referidos anteriormente, a melhor é mais uma vez o Seq.

No que toca a suporte personalizado das plataformas nenhuma fornece tal funcionalidade visto que os plano de pagamento utilizado para cada plataforma foi o gratuito. No entanto todas as plataforma, como expectado, disponibilizam ferramentas para filtrar e pesquisar a informação dos logs. Estas ferramentas tornam-se mais sofisticadas na análise e pesquisa de *logs* consoante o plano de pagamento escolhido.

A maior parte das plataformas viabiliza o seu uso de forma concorrente, com exceção das plataformas Splunk, LogDNA e Seq. Por último, as plataformas só proporcionam métricas personalizáveis em planos de pagamento não gratuitos, à exceção da plataforma Seq.

De um modo geral, a plataforma que apresenta uma maior gama de funcionalidades é o Seq, por outro lado esta plataforma ao contrário das restantes necessita de ser hospedada localmente. Isto tem a vantagem inerente de se obter volume e durabilidade da informação ilimitado, só que também arrecada os custos de manutenção da plataforma.

É seguro afirmar que estas plataformas no caso de se adquirir um plano de pagamento robusto, isto é que forneça bastantes funcionalidades, são capazes de resolver o problema em mão. Todavia, existem dois grandes obstáculos associados a esta abordagem:

1. **Custo monetário**: O alto custo de aquisição e manutenção destas plataformas pode ser um fator decisivo na compra ou não deste;
2. **Flexibilidade**: Pode existir uma falta de flebixilidade das plataformas o que pode comprometer as necessidades dos clientes. Por exemplo, o processo de adaptar e enviar da informação dos *logs* para estas plataformas, reside no lado do cliente (mais concretamente no sistema que está a ser monitorizado). Em contrapartida, ao implementar-se um sistema de monitorização de raiz o dito processo ficaria agora da responsabilidade do sistema de monitorização.

Por estas razões, optou-se por desenvolver uma solução de LaaS do zero, tirando partido de tecnologias *open-source* de modo a satisfazer os requisitos funcionais e não funcionais do projeto em mão.

2.4 Tecnologias

Esta secção aborda as diversas tecnologias que auxiliem na resolução do problema. Estas são enquadradas no âmbito de sistemas de monitorização de *logs* e finalmente são comparadas de forma a escolher a que melhor responde aos requisitos do projeto.

Mais especificamente, esta secção foca-se em analisar não só, mas também tecnologias capazes de realizar as etapas associadas a um sistema de monitorização centralizado de *logs* (ver secção 2.1). A figura 2.3 efetua uma alusão a essas mesmas etapas.

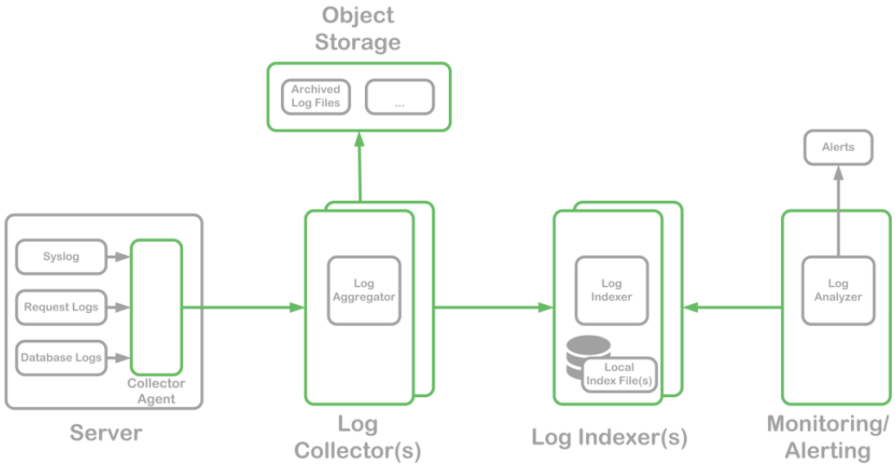

Figura 2.3: Etapas associadas a um sistema de monitorização centralizado de *logs*

Fonte: [31]

Como tal, é necessário garantir que das múltiplas tecnologias abordadas, incluem-se tecnologias de **agrupamento**, **tratamento**, **armazenamento** e **visualização** de dados.

2.4.1 Tecnologias de Versionamento

Os sistemas de controlo de versões (SCV), também denominados de sistemas de gestão de código-fonte, foram introduzidos no início da década de 70. Estes sistemas fazem parte de uma categoria de ferramentas de software destinadas a ajudar equipas de desenvolvimento a gerir alterações do código-fonte ao longo do tempo [32].

Existem duas abordagens diferentes relativamente a SCVs, nomeadamente SCVs centralizados (SCVC) e SCVs distribuídos (SCVD). SCVC é um modelo centralizado que contém um repositório central, enquando que o modelo SCVD é um modelo distribuído onde não existe um repositório central mas sim um repositório local para cada utilizador [33].

A tabela 2.2 apresenta um resumo da comparação entre ambas as abordagens (SCVC e SCVD).

Tabela 2.2: Comparação entre sistemas centralizados e distribuídos de controlo de versões [33]

Sistema de Controlo de Versões	**SCVC**	**SCVD**
Repositório	O servidor atua como o único repositório central	Cada utilizador tem no seu computador o repositório na sua íntegra, também chamado de repositório local
Acesso repositório	Qualquer utilizador que precise de aceder ao repositório deve ter acesso à internet	Permite aos utilizadores trabalharde forma completamente offline. É necessário internet para partilhar o repositório com outros utilizadores
Exemplo de ferramentas	Subversion, Perforce Revision Control System	Git, Mercurial, Bazaar, BitKeeper
Características de software que se adequem	- Projetos que permitam apenas certos utilizadores contribuírem para o desenvolvimento do software - Equipas localizadas num só lugar	- Adequado para um ou mais desenvolvedores - Equipas localizadas em diferentes locais ou países - Adequado para pequenos ou grandes projetos pois o processo de contribuição de código é mais fácil

Uma vez que não existem requisitos sobre qual o sistema de controlo de versões a usar e tendo por base a experiência do autor e a comparação da tabela acima, escolheu-se utilizar um sistema de controlo de versões distribuído através do uso da ferramenta Git.

Uma das mais valias do Git, é a ajuda que este fornece a acompanhar as alterações feitas no código. Se em algum momento do desenvolvimento for encontrado um erro e não se encontrar a causa dele, muito rapidamente pode-se reverter para um estado estável da aplicação [34].

Das várias plataformas de hospedagem Git decidiu-se analisar as três mais populares. Começando pela mais popular a plataforma GitHub contém cerca de 73 milhões de utilizadores [35], logo a seguir tem-se a plataforma GitLab que contém uma estimativa de 30 milhões de utilizadores registados [36], por último tem-se a plataforma Bitbucket que contém aproximadamente 10 milhões de utilizadores [37].

Estas plataformas foram comparadas segundo determinados critérios que se encontram evidenciados na tabela 2.3.

Finalmente optou-se por utilizar a plataforma GitLab não só pelas vantagens relativamente às outras (ver tabela 2.3) como também pela experiência que o autor detém da plataforma.

Tabela 2.3: Comparação dos serviços gratuitos de plataformas de hospedagem Git

Funcionalidades	GitHub	GitLab	Bitbucket
Open-Source	✖	✔	✖
Nº de repositórios privados	Ilimitado	Ilimitado	Ilimitado
Nº de colaboradores	3	Ilimitado	5
Capacidade	1GB	10GB	2GB
Integração Contínua	✔	✔	✔
Entrega Contínua	✔	✔	✔
Integração com o Jira	✔	✔	✔

2.4.2 Tecnologias de Virtualização

Atualmente existem diversas abordagens que incentivam a criação de projetos de grande escala. Uma das estratégias mais utilizadas consiste em tirar partido do uso de máquinas virtuais (MV). Estas máquinas têm os seus próprios recursos (processador, memória RAM, sistema operativo, etc.). É comum as MV lidarem com problemas de virtualização tais como a congestão da rede e falhas de hardware que por consequência reduzem o desempenho das MV [38].

Como alternativa às máquinas virtuais, existe uma tecnologia denominada de Conteinerização que combina as aplicações com as suas dependências (e.g bibliotecas do sistema) de forma a construir um *container*. Estas aplicações são construídas e organizadas de tal modo que podem ser executadas e implantadas como um *container*. Esta plataforma, também conhecida como Docker, garante que a aplicação funciona em qualquer ambiente, de forma rápida e leve [39].

A figura 2.4 denota as arquiteturas das tecnologias de virtualização (MV) e das tecnologias de Conteinerização (Docker).

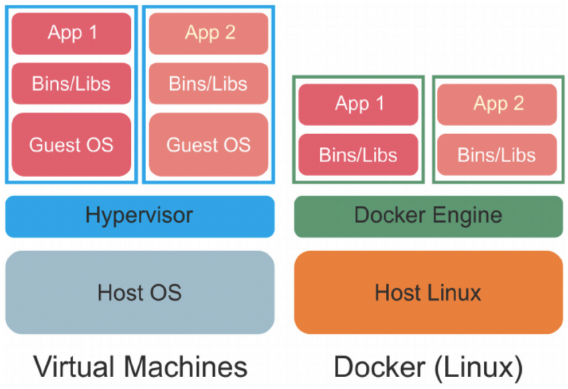

Figura 2.4: Comparação entre arquitetura de MV e Docker

Fonte: [40]

Como se pode observar, a arquitetura da esquerda (MV) utiliza uma camada adicional (*Hypervisor*) para simular o sistema operativo de cada MV. Tanto a aplicação, dependências e sistema operativo estão contidos na MV. A arquitetura da direita (Docker) em contraste das MV, não necessita de uma camada adicional para simular o sistema operativo, dado que recursos do sistema operativo são partilhados entre os diferentes containers.

Muitas das vezes, o Docker é referido como uma máquina virtual "leve", mas ele de facto não é nem pode ser considerado uma MV. A tabela 2.4 compara as características subjacentes das MV e de Docker containers.

Tabela 2.4: Comparação dos atributos das MV e Docker [39]

	Máquinas Virtuais	**Docker Containers**
Nível de isolamento	Hardware	Software
Sistema operativo	Separado	Partilhado
Tempo de iniciação	Longo	Curto
Utilização de recursos	Maior	Menor
Imagens pré-construídas	Difícil de encontrar e manter	Disponíveis em grande quantidade
Tamanho	Maior devido a conter um sistema operativo separado	Menor devido a existir apenas o motor do Docker em cima do sistema operativo
Portabilidade	Fácil de mover para outro sistema operativo	Em vez de serem movidos são criados e destruídos
Tempo de criação	Mais demorado	Em segundos

Através da análise da tabela, é possível deduzir que a tecnologia Docker quando comparada com MV apresenta vantagens tais como o espaço em disco utilizado, os recursos gastos ou tempo despendido quer a criar quer a iniciar a aplicação. No entanto reconhece-se que no que toca a questões de segurança, devido ao facto dos containers partilharem o mesmo sistema operativo da máquina física, qualquer vulnerabilidade existente nessa máquina pode ser aproveitada para sabotar e consequentemente comprometer os *containers*.

No contexto do projeto e pelas razões supracitadas, decidiu-se empregar a tecnologia Docker na criação e implantação de todos os componentes contidos na solução a desenvolver.

2.4.3 Tecnologias de Base de Dados

Um dos pontos cruciais deste projeto compreende o armazenamento persistente de *logs*, o que implica a escrita e consulta de um grande volume de informação. Como tal foram analisadas tecnologias de **SGBD** de maneira a se escolher a que melhor responde a estes parâmetros.

Dos diferentes tipos de base de dados existentes, os principais são as base de dados relacionais e não relacionais. As base de dados relacionais são baseadas no modelo ACID que por sua vez é caracterizado pelos seguintes pontos [41]:

- **Atomicidade**: Garante a integridade das transações;
- **Consistência**: Garante a consistência da informação na base de dados;
- **Isolamento**: Garante a independência de múltiplas transações executadas ao mesmo tempo;
- **Durabilidade**: Garante que a informação armazenada não é alterada, mesmo em casos de falha.

O modelo ACID proporciona consistência e disponibilidade como fortes propriedades e ao mesmo tempo este tipo de base de dados oferece uma linguagem de consulta estruturada, no entanto têm pouca escalabilidade, desempenho fraco, custam mais e sofrem de problemas de disponibilidade quando existe um grande número de utilizadores [42].

Como alternativa, as bases de dados não relacionais estão assentes no modelo BASE, este modelo indica que devido à natureza distribuída das base de dados não relacionais existe a possibilidade da informação estar parcialmente disponível quando partes da base de dados não estão operacionais. É de salientar que este modelo parte do princípio da consistência eventual, que indica que a informação vai ficar consistente no futuro e não imediatamente após a operação [43].

As características enfatizadas pelas base de dados não relacionais são, melhor escalabilidade, flexibilidade e desempenho quando comparadas a base de dados relacionais. Contudo as base de dados não relacionais são desprovidas de uma linguagem de consulta estruturada e não providenciam mecanismos de segurança adequados [42].

Em seguida, apresenta-se a tabela 2.5 que de uma forma concisa compara ambos os tipos de base de dados segundo determinados critérios.

Após análise da tabela supracitada e tendo em conta que para o desenvolvimento do projeto, critérios como **Disponibilidade**, **Desempenho**, **Volume de informação**, **Escalabilidade** e **Custo** são de grande relevância optou-se por analisar tecnologias *open-source* de bases de dados não relacionais.

Salienta-se que foi dada uma maior atenção às base de dados não relacionais com características similares a motores de pesquisa. Pois estas base de dados além de estarem otimizadas para esse tipo de operações, oferecem também as seguintes funcionalidades [44]:

- Suporte para expressões de pesquisa complexas;
- Pesquisa de texto completo;
- Stemização[2]
- Classificação e agrupamento dos resultados da pesquisa;
- Pesquisa distribuída de forma a se obter alta escalabilidade.

[2]Processo que consiste em reduzir uma palavra ao seu radical. E.g as palavras "gato", "gata"reduziriam-se para "gat"[45]

Tabela 2.5: Comparação entre base de dados relacionais e não relacionais

	Base de dados relacional	**Base de dados não relacional**
Escalabilidade	Escala verticalmente	Escala horizontalmente
Custo	Cara para armazenamento de dados	Mais barato uma vez que são maioritariamente open-source e as atualizações são baratas
Volume de informação	Lida com um volume limitado	Lida com grandes volumes de dados
Disponibilidade	Sofre de pontos únicos de falha	Devido à natureza distribuída proporciona maior disponibilidade para os utilizadores na presença de falhas de hardware
Desempenho	Requere mais tempo para processar a informação o que torna o processo mais lento	Melhor desempenho nas operações de consulta
Complexidade	Utiliza estruturas de dados complexas para armazenar a informação	Armazena os dados de forma semi-estruturada ou não estruturada
Linguagem de Consulta	SQL é a única linguagem de manipulação de dados, podendo existir pequenas variações	Cada implementação da base de dados tem a sua linguagem de manipulação dos dados
Consistência	Consistência forte com esquemas bem definidos	Fraca consistência sem a necessidade de existir um esquema bem definido
Segurança	Contém fortes mecanismos de segurança a fim de proteger a informação	Segurança não faz parte da base de dados, é necessário ser efetuada por um middleware

Estas características são interessantes no contexto do projeto dado que há a necessidade inerente de se efetuar consultas/leituras a partir de texto introduzido pelo utilizador.

MongoDB

Apesar do MongoDB não ser uma base de dados propriamente focada em operações de pesquisa de texto, visto que esta é a base de dados não relacional que apresenta a melhor classificação consoante critérios como, interesse geral, nº de ofertas de emprego, nº de referências em web sites, etc. [46] escolheu-se fazer uma breve análise desta.

MongoDB é uma base de dados *open-source* orientada a documentos que oferece alto desempenho, alta disponibilidade e dimensionamento automático. Um registo no MongoDB consiste num documento, que é representado por uma estrutura de dados composta por pares de chave-valor. Estes documentos assemelham-se a objetos JSON. Os valores associados a estas chaves, podem conter tipos primitivos, outros documentos, *arrays* e *arrays* de documentos [47].

Os conceitos básicos do MongoDB encontram-se em seguida enumerados [48]:

1. O documento é a unidade base de informação e é equivalente a uma linha de uma tabela de um sistema de bases de dados relacional;

2. Do mesmo modo, uma coleção pode ser equiparada a uma tabela com uma estrutura dinâmica;
3. Uma única instância de MongoDB pode hospedar várias base de dados independentes, onde cada uma contém as suas coleções;
4. Todos os documentos têm uma chave especial "_id"que é única em toda a coleção.

A figura 2.5 representa um documento da coleção 'post' que contém várias chaves de diferentes tipos de valores (primitivos, outros documentos e *arrays*).

documento da coleção 'post'

```
{
   _id: 1,
   titulo: 'Vendo carro',
   comentarios: [
      {
         'titulo: Qual o preço?'
      },{
         'titulo: Qual o ano?'
      }
   ],
   data_publicacao: 00/00/0000,
   seção: 'carro'
}
```

Figura 2.5: Documento de uma coleção do MongoDB

Fonte: [49]

Apache Solr

A tecnologia Apache Solr atua como um motor de pesquisa e uma base de dados não relacional com suporte transacional. Esta base de dados, de forma semelhante ao MongoDB, é orientada a documentos e oferece suporte a uma linguagem de consulta estruturada e é executada de forma distribuída [50].

Além das características das base de dados não relacionais, esta tecnologia devido a ser baseada na biblioteca Lucene[3] oferece recursos poderosos tais como a pesquisa de texto completa, indexação muito próxima de tempo real, alta disponibilidade, integração com diversas ferramentas de *Big Data* e a capacidade de lidar com documentos de texto enriquecido (e.g Word e PDF) [53].

Um conceito importante a reter, que está presente em diversas base de dados orientadas para motores de pesquisa, é o conceito de *Sharding*. Isto significa que sempre que é inserido um novo documento na base de dados, este é replicado e automaticamente distribuído para as instâncias do Solr. Isto é realizado de forma a melhorar o desempenho das operações de consulta bem como para garantir a redundância da informação [50].

Tendo em conta que o Apache Solr também é um motor de busca, faz sentido analisar a interface de comunicação que este disponibiliza. Relativamente a integração com outras aplicações, dado que esta tecnologia é também um motor de busca, existe uma API de pedidos JSON já integrada com o Apache Solr. Com o uso de parâmetros de *query* ou

[3] Apache Lucene é uma biblioteca de software de pesquisa de texto que fornece uma plataforma de pesquisa e indexação baseada em Java [51, 52]

diretamente no corpo do pedido HTTP, é possível consultar ou criar documentos, como demonstrado na figura 2.6.

```
curl -XGET http://localhost:8983/solr/books/query -d '
{"query": "content:(solr lucene)"}'
```

Figura 2.6: Pedido de consulta à API do Solr

Fonte: [54]

Elasticsearch

Lançado inicialmente em 2010 o Elasticsearch é um motor de análise e pesquisa, esta tecnologia é completamente *open-source* e é construída com base na linguagem de programação Java. O Elasticsearch também pode ser considerado uma base de dados não relacional, contudo ao contrário da maioria dessas base de dados, o Elasticsearch tem um foco particular nas funcionalidades de pesquisa. No contexto de análise de informação, esta tecnologia é normalmente usada em conjunto com os restantes componentes da pilha ELK (Elasticsearch, Logstash e Kibana) que desempenham o papel de indexação e armazenamento de dados [55].

Do mesmo modo que a tecnologia Apache Solr, o Elasticsearch tira partido da biblioteca Lucene para realizar operações de indexação e pesquisa de texto. O propósito do Elasticsearch é tornar a pesquisa de texto completo um processo fácil, escondendo a complexidade da biblioteca Lucene por meio de uma API RESTful simples e coerente. Contudo o Elasticsearch é muito mais do que somente a biblioteca Lucene ou somente um motor de pesquisa. Ele pode ser descrito da seguinte forma [56]:

- Um armazém de documentos distribuído em tempo real onde cada campo é indexado e pesquisável;
- Um motor de pesquisa com análises em tempo real;
- Capaz de escalar para centenas de servidores e para petabytes de informação estruturada e não estruturada.

A tabela 2.6 expõe de uma forma resumida as principais diferenças entre a tecnologia Elasticsearch e a tecnologia analisada previamente (Apache Solr) [57].

Adicionalmente, através da análise da figura 2.7 onde as tecnologias foram alvo de testes de desempenho, ambas no mesmo ambiente e para as mesmas circunstâncias (1000 queries enviadas em simultâneo) pode-se inferir que a tecnologia Elasticsearch oferece um melhor tempo de resposta [58].

Posto isto e tendo em consideração a tabela e figura anterior (ver tabela 2.6 e figura 2.7), escolheu-se utilizar a tecnologia Elasticsearch não só como tecnologia de base de dados mas também como motor de pesquisa.

Tabela 2.6: Comparação entre Apache Solr e Elasticsearch

Funcionalidade	Apache Solr	Elasticsearch
Cache	Global, que se torna inválida a cada mudança de segmento	Por segmento, ideal para dados que mudam dinamicamente
Motor de análise	Agregações poderosas	Agregações sofisticadas e altamente flexíveis
Execução de queries otimizadas	Inexistente	Intervalo de queries mais rápidas, dependendo do contexto
Velocidade de pesquisa	Melhor para dados estáticos, devido à cache e leitor não invertido	Boa para dados que mudam rapidamente, devido à cache por segmento
Desempenho do motor de análise	Ótimo para dados estáticos através de cálculos exatos	Exatidão dos resultados depende do posicionamento dos dados
Ecossistema	Modesto	Rico - Kibana, Grafana, e outros componentes
Funcionalidade de pesquisa de texto completo	Múltiplas sugestões, corretores ortográficos, suporte de realce avançado	API de sugestões, Suporta realce

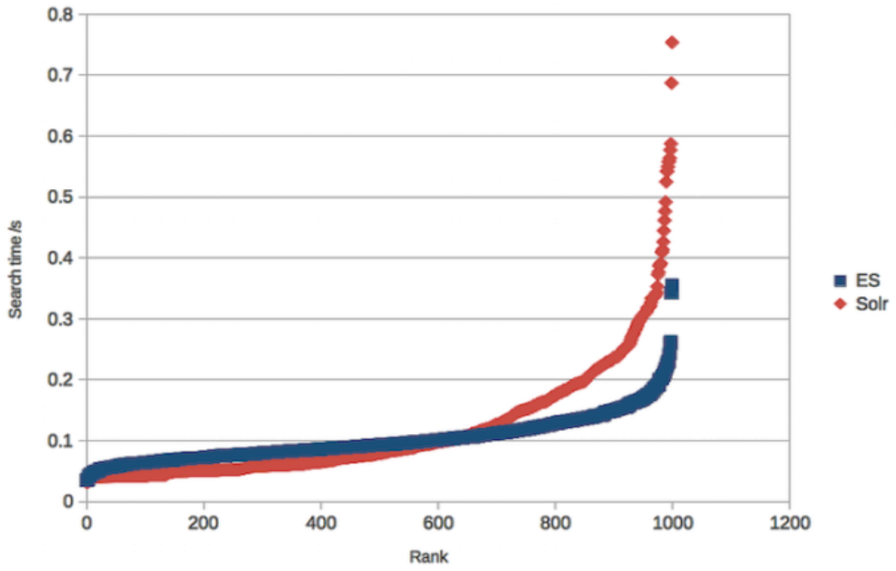

Figura 2.7: Tempo despendido de ambas as tecnologias a processar 1000 queries

Fonte: [58]

2.4.4 Tecnologias de Visualização de Dados

Uma das componentes da solução a desenvolver envolve a visualização da informação proveniente dos *logs*. Deste modo, optou-se por analisar tecnologias que fornecessem um vasto leque de representações gráficas (e.g gráfico de barras, circulares, de linhas, etc.) e textuais.

Grafana

A tecnologia Grafana é uma solução *open-source* que visa analisar dados e obter métricas que dão sentido à enorme quantidade de informação existente. Também facilita o processo de monitorização das aplicações mediante o uso de *dashboards* customizáveis. O Grafana oferece a possibilidade de se conectar a várias fontes de dados, normalmente denominadas de base de dados, tais como o Elasticsearch. As principais funções desta tecnologia consistem no rastreamento do comportamento tanto da aplicação como dos seus utilizadores. Também permite analisar a frequência e os tipos de erros, fornecendo sempre dados que contextualizem os possíveis cenários [59].

A figura 2.8 apresenta um exemplo de uma *dasbhoard* da tecnologia Grafana.

Figura 2.8: *Dashboard* do Grafana

Fonte: [60]

Conforme consta na figura previamente mencionada, é possível visualizar a informação do sistema em diversos formatos e.g gráfico de linhas, velocímetros, histogramas, tabelas, etc. Contudo esta tecnologia é mais focada em analisar e visualizar métricas do sistema tais como [61]:

- Percentagem de utilização do processador;
- Quantidade de memória utilizada;
- Espaço utilizado no disco;
- Utilização de operações I/O.

Outro fator que se considerou negativo tendo em conta o contexto do problema, é que esta tecnologia além de não se focar tanto em análise e visualização de *logs* aplicacionais também não permite pesquisa de texto completo [61].

Kibana

O Kibana é uma tecnologia que permite explorar, visualizar e criar uma *dashboard* sobre a informação de *logs* proveniente do Elasticsearch. A característica principal do Kibana é a

consulta e análise de dados. Além disso, os recursos de visualização do Kibana permitem visualizar dados de diversas maneiras usando mapas de calor, gráficos de linha, histogramas, gráficos circulares, etc. Através destes métodos é possível pesquisar os dados armazenados no Elasticsearch de forma a se diagnosticar problemas no sistema [62].

O Kibana através da instalação do módulo Metricbeat e da mesma forma que o Grafana, também é capaz de monitorizar métricas do sistema conforme representado na figura 2.9.

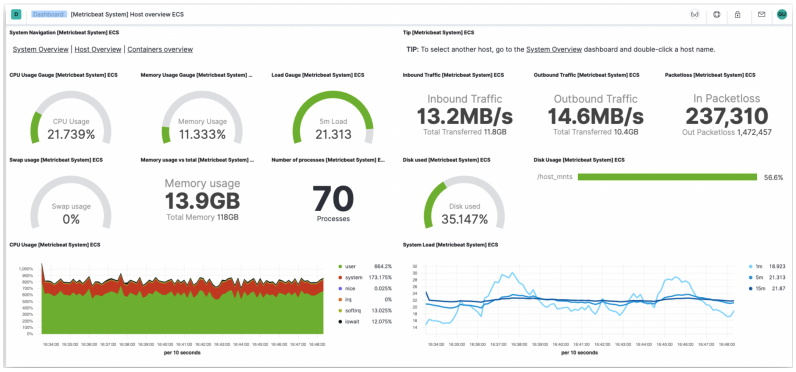

Figura 2.9: Métricas do sistema no Kibana

Fonte: [63]

Adicionalmente e de acordo com a figura 2.10, o Kibana permite pesquisar e filtrar a informação de logs armazenados no Elasticsearch.

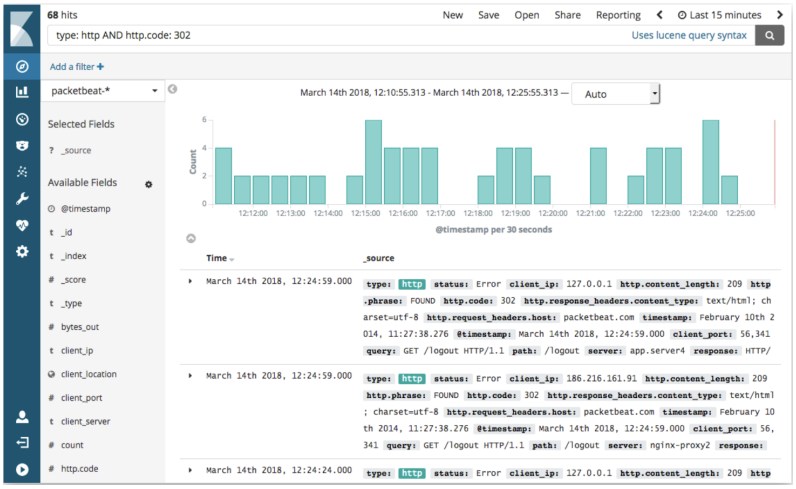

Figura 2.10: Análise e pesquisa de *logs* no Kibana

Fonte: [64]

Por último e tendo em conta que a tecnologia Kibana é mais vocacionada para análise e pesquisa de *logs* do que a tecnologia Grafana, optou-se por utilizar o Kibana como tecnologia de visualização de dados.

2.4.5 Tecnologias de Processamento de Dados

De maneira a se implementar um sistema de *logs* centralizado que seja eficaz, é necessário uma ferramenta que consiga extrair e processar dados de diferentes fontes. Esta ferramenta deve ser capaz de extrair e receber informação de diferentes sistemas, transformando-a em conjuntos de dados estruturados e posteriormente transportá-los para um mecanismo de armazenamento persistente [65].

Fluentd

O Fluentd é uma tecnologia de agregação de *logs* eficiente. Foi desenvolvida em Ruby e é altamente escalável. O Fluentd retira os *logs* de um determinado conjunto de fontes, processando-os e convertendo-os num formato de dados estruturados que são posteriormente enviados para outros serviços, como por exemplo o Elasticsearch. Esta tecnologia é especialmente flexível no que toca a integrações, inclusive funciona com mais de 300 serviços de análise e de armazenamento de *logs* [66].

A figura 2.11 representa de uma forma simplificada a arquitetura do Fluentd.

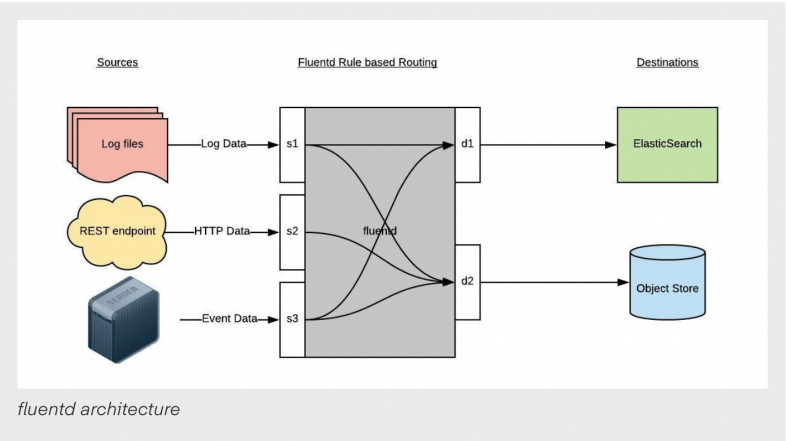

Figura 2.11: Arquitetura do Fluentd

Fonte: [66]

Numa primeira fase o Fluentd obtém a informação de diversas fontes (logs, *endpoints* REST, etc.). Após capturar toda a informação o próximo passo envolve estruturá-la e etiquetá-la e por último enviar a informação já transformada para os múltiplos destinos.

Logstash

O Logstash pode ser definido como uma *pipeline open-source* de processamento de dados provenientes de um ou mais servidores. Esta tecnologia pode ingerir simultaneamente dados de uma ampla gama de fontes, pode também analisar, filtrar, transformar e enriquecer os dados e finalmente enviá-los para um sistema *downstream* [67]. Muitas das vezes esse sistema é o Elasticsearch mas não tem que o ser obrigatoriamente.

O comportamento desta tecnologia encontra-se retratado na figura 2.12.

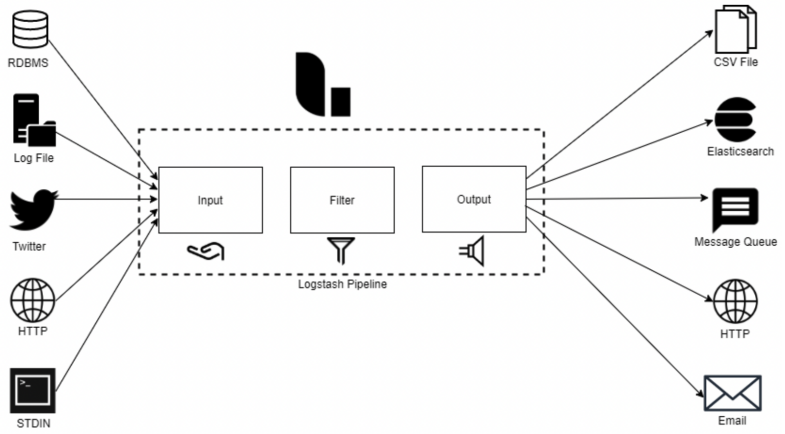

Figura 2.12: Arquitetura do Logstash

Fonte: [67]

De uma forma muito similar à tecnologia Fluentd, o Logstash é composto por 3 fases:

1. **Fase de entrada**: Os dados de diversas fontes são ingeridos para o Logstash. O Logstash por si só não acede ao sistemas que contêm a informação para agregar. Essa tarefa é realizada por módulos que integram com o Logstash;
2. **Fase de filtragem**: Nesta fase os dados que se acharem relevantes são extraídos e potencialmente alterados ou enriquecidos;
3. **Fase de saída**: Os dados processados são enviados para respetivos sistemas destinatários.

No contexto deste projeto foi decido utilizar a tecnologia Logstash de forma a completar a ELK *stack*. Uma das vantagens da utilização desta *stack* é existência de um documentação extensa e a facilidade com que se integra os seus componentes (Elasticsearch, Logstash e Kibana). Contudo reconhece-se que a tecnologia Fluentd é mais performante no sentido em que consome menos recursos do sistema segundo indica a figura 2.13.

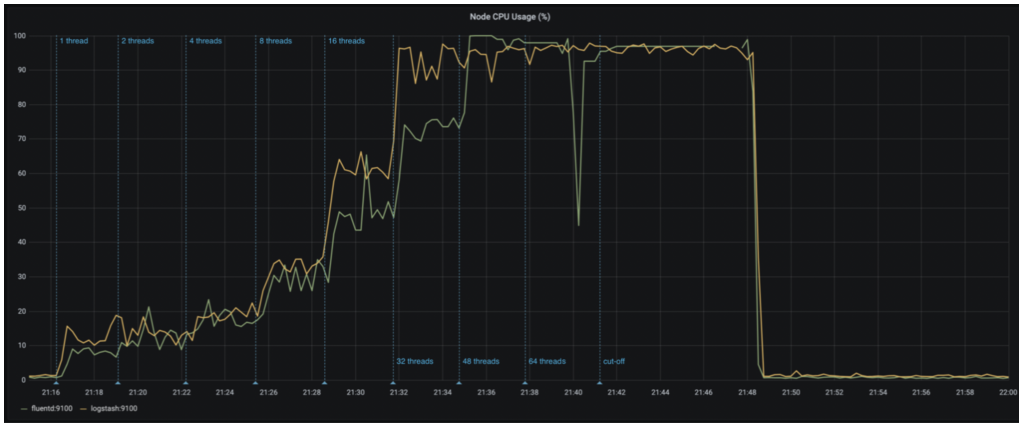

Figura 2.13: Utilização do CPU por um nó do Logstash e do Fluentd

Fonte: [68]

Filebeat

O Filebeat é um dos componentes de envio de dados disponíveis como parte da ELK *stack*, também conhecidos como Beats. Estes "carregadores de dados"têm uma única finalidade e foram concebidos para serem instalados na máquina que gera os dados. O Filebeat lê ficheiros de *logs* baseados em texto e envia-os diretamente para o ElasticSearch ou para o Logstash [69].

À primeira vista, pode-se argumentar que o Logstash já desempenha a função de ler ficheiro de *logs*, o que inviabiliza o uso do Filebeat. No entanto e conforme exposto na tabela 2.7 rapidamente deteta-se as vantagens associadas ao uso do Filebeat.

Tabela 2.7: Comparação dos atributos entre o Logstash e o Filebeat [70]

Atributos	**Filebeat**	**Logstash**
Plataformas	Mac, Windows e Linux	Mac, Windows e Linux
Plugins	Descentralizado	Centralizado
Desempenho	Consome menos memória	Consome mais memória
Gestão de Tráfico e Transporte	Fiabilidade já incorporada	Utiliza Redis para preocupações de fiabilidade

No que toca a leitura de ficheiros de texto gerados pelo sistema a monitorizar, tanto o Logstash como o Filebeat estão aptos para desempenhar essa tarefa, sendo que qualquer um dos dois componentes necessita de se encontrar alojado na máquina física do sistema que gera os *logs*. Por este motivo e com base na tabela anteriormente referida escolheu-se utilizar o Filebeat na leitura de ficheiros de *logs*, pelo simples facto de consumir menos recursos.

Em suma, as tecnologias Filebeat e Logstash não são mutuamente exclusivas pois o Filebeat vem complementar as funcionalidades prestadas pelo Logstash.

Capítulo 3

Análise

O capítulo de Análise aborda e detalha três temáticas associadas ao desenvolvimento do projeto. Inicialmente descreve-se o cenário hipotético no qual o desenvolvimento da solução se vai centrar, como tal apresenta-se a área de negócio e a arquitetura do sistema que vai ser alvo de monitorização. Logo após e com base na literatura e em projetos previamente elaborados, são enunciados os requisitos funcionais e não funcionais da solução. Esta secção finaliza numa vertente mais teórica onde é efetuada uma análise de valor ao projeto, que inclui a identificação das forças, fraquezas, oportunidades e ameaças deste, bem como uma avaliação do valor percecionado.

3.1 Cenário Hipotético

Tendo em conta que o projeto em mão não está a ser desenvolvido no contexto de um cenário real (e.g para uma organização), existe a necessidade de simular o dito cenário. Para tal foram ponderadas duas abordagens, nomeadamente o desenvolvimento de raiz de um sistema baseados em microsserviços, ou a utilização de um já existente.

A primeira abordagem, que corresponde ao desenvolvimento de um sistema do "zero", apesar de garantir uma maior flexibilidade relativamente às tecnologias e padrões empregues, implica um maior consumo de tempo e uma maior complexidade quando comparada à segunda abordagem que remete para a utilização de um sistema existente.

Além disso, o foco e o alvo de avaliação deste projeto remete para a solução de monitorização de *logs* a desenvolver e não necessariamente no sistema baseado em microsserviços, que por si só poderia ser um trabalho para uma dissertação ou tese.

Com base nos fatores anteriormente referidos, optou-se então por um sistema de microsserviços já existente. De seguida são apresentadas as características que foram tidas em consideração na escolha do sistema:

- **Open-source**: É necessário que o sistema seja *open-source*, não só por questões monetárias mas também para possibilitar eventuais alterações ao sistema;
- **Heterogeneidade**: Idealmente os microsserviços devem ser desenvolvidos em plataformas diferentes, o que implica o uso de várias linguagens de programação ou de mecanismos de persistência diversificados;
- **Complexidade**: O sistema deve ser composto no mínimo por três microsserviços de modo a se obter *logs* de diversas fontes com diferentes tipos de informação;

- **Conteinerização**: O sistema deve possuir mecanismos de conteinerização (e.g Docker, Docker Compose, etc.) de forma a simplificar a implantação do sistema;
- **Funcionalidades**: O sistema deve ser minimamente diversificado, isto significa que idealmente o sistema disponibiliza várias funcionalidades sobre diferentes aspetos do domínio de atuação;
- **Escrita de logs**: O sistema deve-se prover de mecanismos de escrita de *logs* sempre que é efetuada uma ação no sistema por parte dos utilizadores;
- **Documentação de instalação**: Para agilizar o processo de instalação e execução do sistema, este deve conter documentação referente a esses mesmos tópicos.

Os sistemas que foram analisados constam de uma lista com mais de 60 repositórios de projetos *open-source* baseados em microsserviços [71]. Esta lista pode ser consultada em https://github.com/davidetaibi/Microservices_Project_List.

Da lista mencionada, escolheu-se o projeto **eShopOnContainers** [72]. Este projeto disponibiliza uma solução *open-source* cuja arquitetura é assente em microsserviços e Docker *containers*.

O repositório que alberga o dito projeto além de conter o código-fonte também possui uma documentação bastante minuciosa não só do modo como o sistema se encontra implementado mas também de como instalá-lo e executá-lo localmente, por consequência cumpre-se com a característica **Documentação de instalação**.

O eShopOnContainers é uma aplicação de comércio eletrónico baseado na Web, onde os utilizadores podem pesquisar e comprar acessórios e utensílios conforme ilustrado na figura 3.1. Por esta razão e por questões de legibilidade optou-se denominar o projeto eShopOnContainers de **Online Boutique** sempre que este é mencionado no decorrer do relatório.

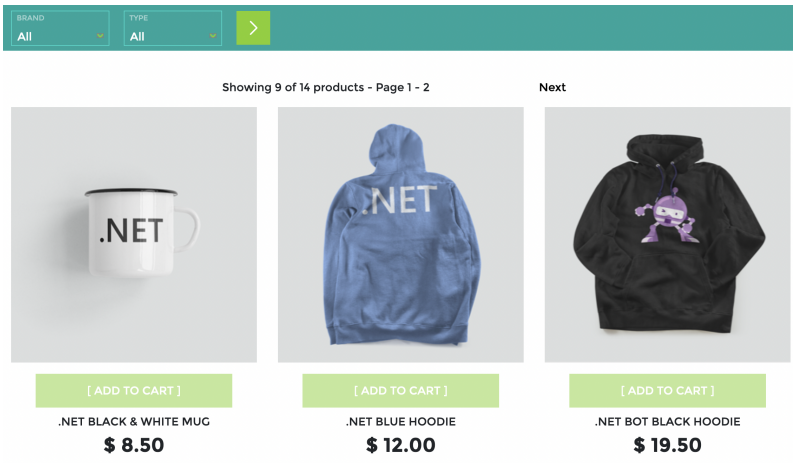

Figura 3.1: Plataforma eShopOnContainers

Fonte: [72]

De seguida enumeram-se algumas das funcionalidades prestadas pela plataforma em causa:

- Listagem de itens do catálogo;
- Filtração dos itens por tipo ou por marca;

- Adição, remoção ou atualização de itens do carrinho de compra virtual;
- Registo de uma conta de utilizador;
- Autenticação;
- Efetuação de compras;

Com isto em mente, considerou-se que os conceitos e as operações de negócio que estão ligadas ao Online Boutique satisfazem a característica de **Funcionalidades**.

Relativamente à arquitetura do sistema, este é constituído por sete "contextos"que estão representados na figura 3.2.

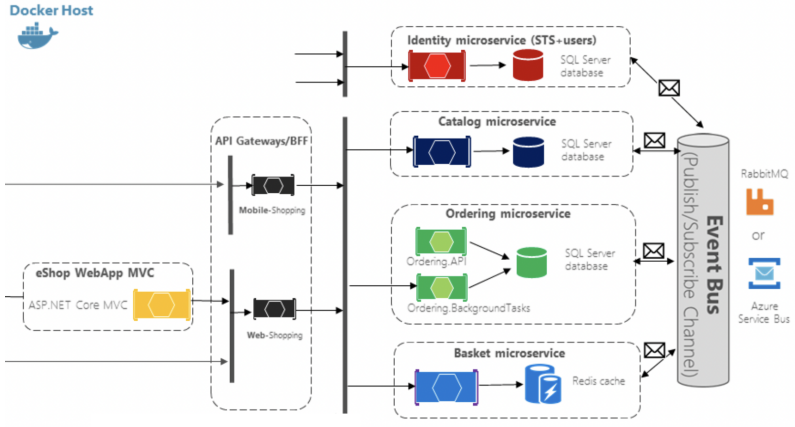

Figura 3.2: Arquitetura do Online Boutique

Fonte: [72]

A responsabilidade de cada um dos contextos pode ser sintetizada da seguinte forma:

1. **eShop WebApp MVC**: Representa um componente desenvolvido em C# [73] que expõe um servidor HTTP que é consumido pelo navegador web, representa a interface gráfica do utilizador;
2. **API Gateway/BFF**: Representa um componente que desempenha as funções de um *reverse-proxy*, por outras palavras é responsável por receber pedidos do exterior e reencaminhá-los para os respetivos microsserviços;
3. **Identity microservice**: Representa um microsserviço desenvolvido em C# responsável por lidar com as preocupações de autenticação e autorização sobre os utilizadores do sistema. Utiliza como mecanismo de persistência uma base de dados SQL Server [74];
4. **Catalog microservice**: Representa um microsserviço desenvolvido em C# responsável por disponibilizar operações CRUD sobre os catálogos. Utiliza como mecanismo de persistência uma base de dados SQL Server;
5. **Ordering microservice**: Representa um microsserviço desenvolvido em C# responsável por disponibilizar operações CRUD sobre os pedidos de compra. Utiliza como mecanismo de persistência uma base de dados SQL Server;

6. **Basket microservice**: Representa um microsserviço desenvolvido em C# responsável por disponibilizar operações CRUD sobre os carrinho de compra. Utiliza como mecanismo de persistência uma base de dados em memória Redis [75];
7. **Event Bus**: Representa um componente cuja função é reencaminhar os eventos nele publicados para os respetivos destinatários (microsserviços previamente citados).

Além dos serviços armazenarem os *logs* nas respetivas base de dados e uma vez que o sistema é *open-source* teve-se a liberdade de adicionar um novo mecanismo de escrita de *logs*, nomeadamente um mecanismos de escrita para ficheiros de texto. Assim através desta alteração é possível demonstrar o agrupamento de *logs* de diversas fontes (ficheiro de texto e base de dados).

Em suma, o autor considerou que o **Online Boutique** responde de um modo satisfatório às diretrizes inicialmente estabelecidas. Isto inclui preocupações como a heterogeneidade e complexidade do sistema uma vez que existem sete componentes sendo quatro deles microsserviços que utilizam mecanismos de persistência de *logs* diversificados. Por fim, escolheu-se este sistema como o cenário hipotético no qual se vai desenvolver a solução preconizada neste relatório.

3.2 Análise de Domínio e Requisitos

Esta secção visa pormenorizar o domínio e os requisitos que estão assentes na solução. Para este efeito, são utilizados diagramas em notação UML[1] que evidenciam as entidades de negócio bem como os cenários de interação do utilizador com a solução. Por último são abordados quer os requisitos funcionais como os não funcionais da solução. Este requisitos incluem preocupações como restrições tecnológicas, tempos de resposta, entre outros fatores que influenciem o desenho e consequentemente a implementação do sistema.

3.2.1 Modelo de Domínio

O modelo de domínio é adequado para detalhar e modelar as entidades do mundo real e as relações entre elas. Estes dois conceitos em conjunto descrevem o domínio do problema. Com recurso à modelação do modelo de domínio pode-se eliminar eventuais lacunas entre a compreensão do domínio do problema e a interpretação dos requisitos [77]. De uma forma resumida, a modelação do domínio pressupõe a solução como um conjunto de entidades de domínio que colaboram entre si para satisfazer os diferentes cenários do sistema.

No contexto deste projeto, foi elaborado o modelo de domínio que se encontra ilustrado na figura 3.3.

Trata-se de um modelo relativamente simples onde foram considerados nove conceitos que estão subjacentes ao domínio do problema:

- **Colaborador**: Representa a entidade que interage com a solução e que tem como finalidade diagnosticar problemas que ocorrem no sistema;
- **Sistema Monitorado**: Representa de um modo holístico o sistema que vai ser alvo de monitorização, neste caso em concreto representa o sistema **Online Boutique**;

[1] UML é uma linguagem de modelação visual dominante no desenvolvimento de software orientado a objetos [76].

- **Microsserviço**: Representa os serviços do sistema que são responsáveis por produzir *logs*;
- **Log**: Representa a informação relevante de eventos que ocorrem nos microsserviços;
- **Formato**: Representa a estrutura da informação presente nos *logs*;
- **Sistema Monitorização**: Representa o sistema responsável por consumir os *logs* dos microsserviços e por produzir notificações;
- **Notificação**: Representa uma notificação que contém os detalhes de anomalias que são posteriormente enviados para os colaboradores;
- **Regra de notificação**: Representa uma regra na qual o sistema de monitorização se baseia para gerar notificações, esta regra é sustentada pelos *logs* dos microsserviços;
- **Administrador**: Representa a entidade responsável por configurar as regras de notificação e estende os comportamentos do colaborador;

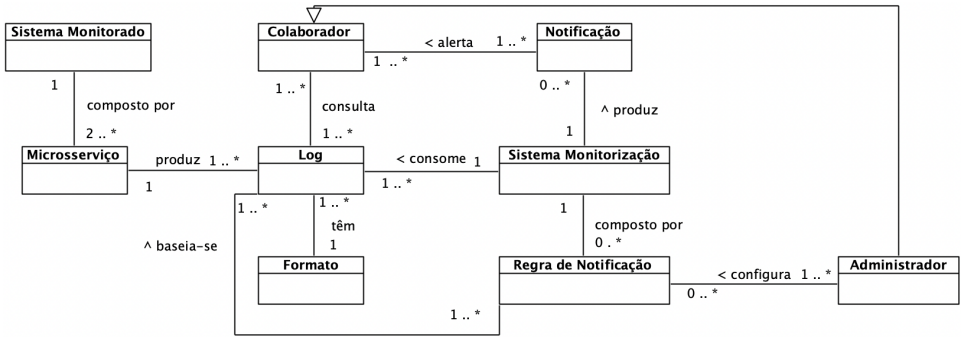

Figura 3.3: Modelo de domínio da solução

Como dito anteriormente, este modelo de domínio além de ser simples denota conceitos já existentes, isto é, entidades e relações que já se encontravam presentes mesmo antes do desenvolvimento deste projeto. Isto é uma consequência esperada tendo em conta que a natureza do projeto compete ao tratamento, armazenamento e análise de informação atualmente gerada e mantida pelo sistema.

No que concerne os novos conceito eles centram-se essencialmente no domínio das notificações, quer seja o administrador a configurar regras de notificações quer seja o próprio sistema a automaticamente alertar os colaboradores.

3.2.2 Requisitos

Um requisito é uma condição ou capacidade possuída pelo software de maneira a resolver um problema do mundo real. Estes problemas podem-se relacionar com a necessidade de automatizar partes de um sistema, corrigir deficiências ou implementar novas funcionalidades. IEEE define um requisito como [78]:

1. Capacidade ou condição necessária para um utilizador resolver um problema ou atingir um determinado objetivo;
2. Capacidade ou condição que deve ser atendida ou possuída por um sistema para satisfazer um determinado contrato, padrão, especificação ou qualquer outro documento imposto;

3. Uma representação documentada de uma condição ou capacidade referida nos pontos anteriores (1 e 2).

Os requisitos descrevem como um sistema se deve comportar e como tal diferem de um cliente para outro, bem como de um processo de negócio para outro [78].

No âmbito do projeto o processo de análise e levantamento de requisitos teve em consideração as características das soluções estudadas na secção 2.3, como também dissertações e relatórios técnicos que abordam a temática de centralização de *logs* [16, 79–81].

Abaixo encontram-se listados todos os requisitos levantados:

R.1 O sistema deve afetar o mínimo possível as aplicações já existentes;

R.2 O sistema deve ser desenvolvido de forma a facilitar alterações ao mesmo;

R.3 O sistema deve ser capaz de tratar 100 *logs* por segundo sem que exista perda de informação [2];

R.4 O sistema deve ser capaz de agrupar e processar *logs* provenientes de diferentes fontes e formatos;

R.5 O sistema deve ser capaz de manipular e armazenar *logs* sem perder informações nos casos de falha no sistema, problemas de conexão ou congestionamento de rede;

R.6 O sistema deve disponibilizar um meio para analisar e visualizar a informação dos *logs* na web (*dashboard*);

R.7 O sistema deve ser capaz de apresentar a informação dos *logs* de uma maneira facilmente acessível e legível;

R.8 O sistema deve suportar a pesquisa de *logs* através de texto introduzido pelo colaborador;

R.9 O sistema deve notificar atividades suspeitas ou anomalias aos colaboradores via e-mail ou SMS;

R.10 O sistema deve garantir a integridade dos *logs* quando estes são transportados;

R.11 O sistema deve garantir a autenticidade dos *logs* quando estes são transportados;

R.12 O sistema deve ser desenvolvido seguindo boas práticas de desenvolvimento de software;

R.13 O sistema deve suportar diversos navegadores de web (Chrome, Safari, Edge, Firefox, Opera);

R.14 O sistema deve implementar um mecanismo de rotação de logs;

R.15 O sistema deve ser capaz de ordenar os *logs* por grau de importância;

R.16 O sistema deve permitir aos colaboradores efetuar consultas complexas, isto é que envolvam diferentes filtros;

R.17 O sistema deve ser capaz de apresentar a informação dos *logs* graficamente, por meio da utilização de gráficos de linhas, circular, histogramas, etc.;

[2] Este requisito em particular é referente a uma característica de performance proveniente de uma dissertação de mestrado [16] relacionada com o tópico de monitorização de logs

R.18 O sistema deve permitir escalar sem grandes dificuldades;

R.19 O sistema deve apresentar os dados quase em tempo real por isso a latência dos servidores deve ser na ordem dos segundos;

R.20 O sistema deve expor uma interface de comunicação de uso interno, nomeadamente uma API RESTful;

R.21 O sistema deve implementar um mecanismo de autenticação que deve ser utilizado pelos colaboradores quando acedem à *dashboard*.

R.22 O sistema deve disponibilizar uma interface gráfica, que apenas pode ser acedida pelo administrador, que permita configurar o mecanismo de notificações/alertas;

R.23 O sistema deve disponibilizar uma interface gráfica, que apenas pode ser acedida pelo administrador, que permita registar novos colaboradores no sistema;

R.24 O sistema deve implementar um mecanismo de autorização de modo a restringir funcionalidades do sistema consoante o papel do utilizador autenticado;

Requisitos Não Funcionais

A complexidade de um software é determinada parcialmente pelas suas funcionalidades (aquilo que o sistema faz) e parcialmente pelos requisitos durante a fase de desenvolvimento como por exemplo os custos operacionais, desempenho, fiabilidade, manutibilidade, portabilidade, robustez, entre outros. Estes requisitos não funcionais têm um papel crítico durante o desenvolvimento do sistema pois servem como critérios de seleção na escolha de tecnologias ou arquiteturas [82].

Tendo por base os requisitos supracitados o próximo passo consiste em identificar os requisitos não funcionais e posteriormente categorizá-los. Com isto em mente, decidiu-se utilizar o modelo FURPS+. Este modelo é um sistema de classificação de requisitos, o acrónimo representa categorias que são utilizadas na definição de requisitos assim como representa atributos de qualidade do software [83]. Analisando o acrónimo letra a letra obteve-se as seguintes categorias:

F (Funcionalidade), representa as principais funcionalidades do produto que se enquadram no domínio do problema. Estes requisitos também podem possuir uma vertente mais técnica. Algumas das preocupações que se encaixam nesta categoria são, auditoria, localização, segurança, e-mail, entre outras [84]. No contexto deste projeto foram identificados os requisitos **R.4**, **R.9**, **R.14**, **R.21**, **R.24** como requisitos de funcionalidade.

U (Usabilidade), consiste em observar, capturar e declarar requisitos com base em questões da interface gráfica do utilizador. Estas questões podem remeter para a acessibilidade, estética e consistência da interface gráfica [84]. No contexto deste projeto foi identificado o requisito **R.7** como requisito de usabilidade.

R (Confiabilidade), refere-se a integridade, conformidade e interoperabilidade do software. Alguns dos requisitos que podem ser considerados são, frequência e gravidade de falha, tempo médio entre falhas, possibilidade de recuperação de dados, entre outros [83]. No contexto deste projeto foram identificados os requisitos **R.5**, **R.10**, **R.11** como requisitos de confiabilidade.

P (Desempenho), preocupa-se em avaliar o desempenho do sistema e envolve questões como a taxa de transferência de informação, tempo de resposta, consumo de memória entre

outros [83]. No contexto deste projeto foram identificados os requisitos **R.3**, **R.18**, **R.19** como requisitos de desempenho.

S (Suportabilidade), diz respeito a preocupações como a testabilidade, manutibilidade, compatibilidade, escalabilidade entre outras [84]. No contexto deste projeto foram identificados os requisitos **R.1**, **R.2**, **R.13** como requisitos de suportabilidade.

Para finalizar, tem-se o símbolo + do acrónimo FURPS+ que abrange restrições que não se enquadram nas categorias previamente mencionadas e que se encontram de seguida enunciadas:

- **Restrições de desenho**: Como o próprio nome indica, limita o desenho do sistema como por exemplo ao restringir-se o tipo de base de dados a utilizar [84]. No âmbito do projeto o requisito **R.12** enquadra-se nesta categoria;
- **Restrições de implementação**: Requisitos que obriguem o uso de uma determinada linguagem de programação, biblioteca ou de determinados padrões [84]. No âmbito deste projeto não foram identificados requisitos que se enquadram nesta categoria.
- **Restrições de interface**: Restrições que indicam os mecanismos de interação com sistemas externos ou internos [84]. No âmbito do projeto o requisito **R.20** enquadra-se nesta categoria;
- **Restrições físicas**: Limitações físicas que afetam o hardware que alberga o sistema, como por exemplo o tamanho, forma e peso [84]. No âmbito deste projeto não foram identificados requisitos que se enquadram nesta categoria.

Requisitos Funcionais

Um requisito funcional é um recurso ou funcionalidade do sistema que é diretamente visível para os atores do sistema [85]. Tratam-se de requisitos que descrevem as funcionalidades principais ou o comportamento de um sistema [86].

Dos requisitos listados anteriormente estes foram os que se consideraram como requisitos funcionais, **R.6**, **R.8**, **R.15**, **R.16**, **R.17**, **R.22** e **R.23**. Todos estes requisitos estão associados a uma ação específica do utilizador, que no contexto do projeto tem o papel de colaborador. O diagrama de casos de uso da figura 3.4 ilustra não só as ditas ações como também algumas ações mencionadas na fase dos requisitos não funcionais.

Ao analisar-se a figura previamente mencionada é possível detetar quatro atores, o colaborador não autenticado, o colaborador autenticado, o administrador e a fonte de *logs*. Salienta-se que as interações subjacentes aos casos de uso "Notificar anomalia"e "Agrupar logs", são desencadeadas/inicializadas pelo sistema de monitorização de *logs*. Os restantes casos de uso são espoletados pelo respetivo ator.

Nos capítulos que se seguem, estes casos de usos são detalhados e explicados em pormenor mas ainda assim considerou-se pertinente abordar de uma forma breve os casos de uso previamente retratados:

(R4) Agrupar logs O sistema de monitorização consome os *logs* que são continuamente gerados pelo sistema monitorado. Além de os consumir, o sistema de monitorização também processa os ditos *logs* adaptando-os para um formato padronizado a fim de armazená-los num mecanismo de persistência de dados;

(R9) Notificar anomalia O sistema de monitorização procura periodicamente por situações anómalas nos *logs* do sistema tendo como base as regras definidas pelos administradores. No caso de existirem anomalias então o sistema envia um notificação via e-mail para todas as partes interessadas (colaboradores);

(R21) Efetuar autenticação O colaborador não autenticado navega até à página de autenticação e o sistema solicita as credenciais (utilizador e palavra-passe). Uma vez introduzidas as credenciais o sistema valida-as e no caso de serem legítimas autentica o colaborador redirecionando-o para a página inicial do sistema;

(R22) Configurar alertas O administrador navega até à página de configuração de alertas e o sistema apresenta todos os alertas existentes. O administrador seleciona um alerta e o sistema redireciona-o para a página de configuração de um dado alerta. Nesta página o sistema solicita as condições nas quais deve ser enviado um alerta, a periodicidade de verificação das dadas condições e as ações que devem ser tomadas quando se verifica uma condição. O administrador introduz os dados solicitados e o sistema guarda a nova configuração e informa-o do sucesso da operação;

(R23) Configurar colaboradores O administrador navega até à página de configuração de utilizadores e o sistema apresenta todos os utilizadores existentes. O administrador seleciona a opção de registar um novo utilizador, o que causa uma resposta do servidor a solicitar os dados (nome de utilizador, palavra-passe, cargo). O administrador introduz os dados solicitados enviando-os para o sistema e este por sua vez valida os dados, persiste o novo utilizador e informa o sucesso da operação ao administrador.

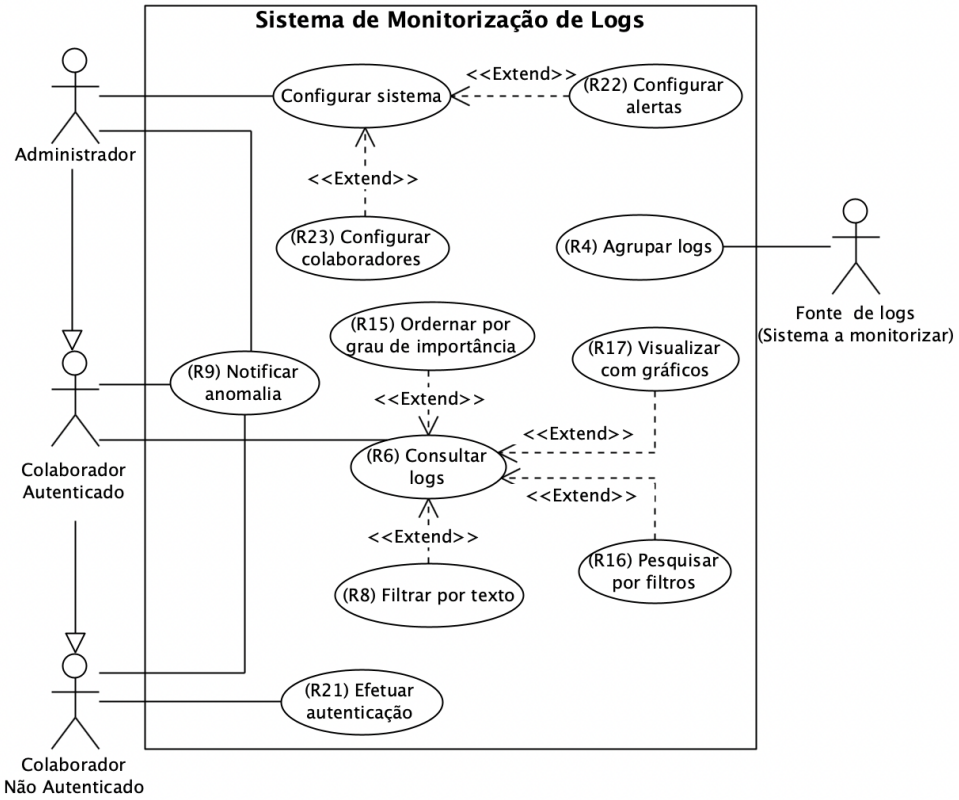

Figura 3.4: Diagrama de Casos de Uso

Por último, decidiu-se enfatizar o comportamento do caso de uso **(R6) Consultar logs** conforme representado na figura 3.5.

Ator Principal:

- Colaborador autenticado.

Partes interessadas e seus interesses:

- Colaborador autenticado: tem como interesse visualizar e analisar a informação dos *logs* armazenados.

Pré-Condições

- O colaborador deve estar registado no sistema;
- O colaborador deve estar autenticado;
- O sistema deve conter *logs* armazenados a fim de serem consultados.

Cenário de sucesso principal

1. O colaborador acede à *dashboard* através de um navegador web;
2. O sistema apresenta a *dashboard* na página inicial;
3. O colaborador seleciona a opção de consulta de *logs*;
4. O sistema devolve uma lista com informação relativa aos últimos 20 logs registados.

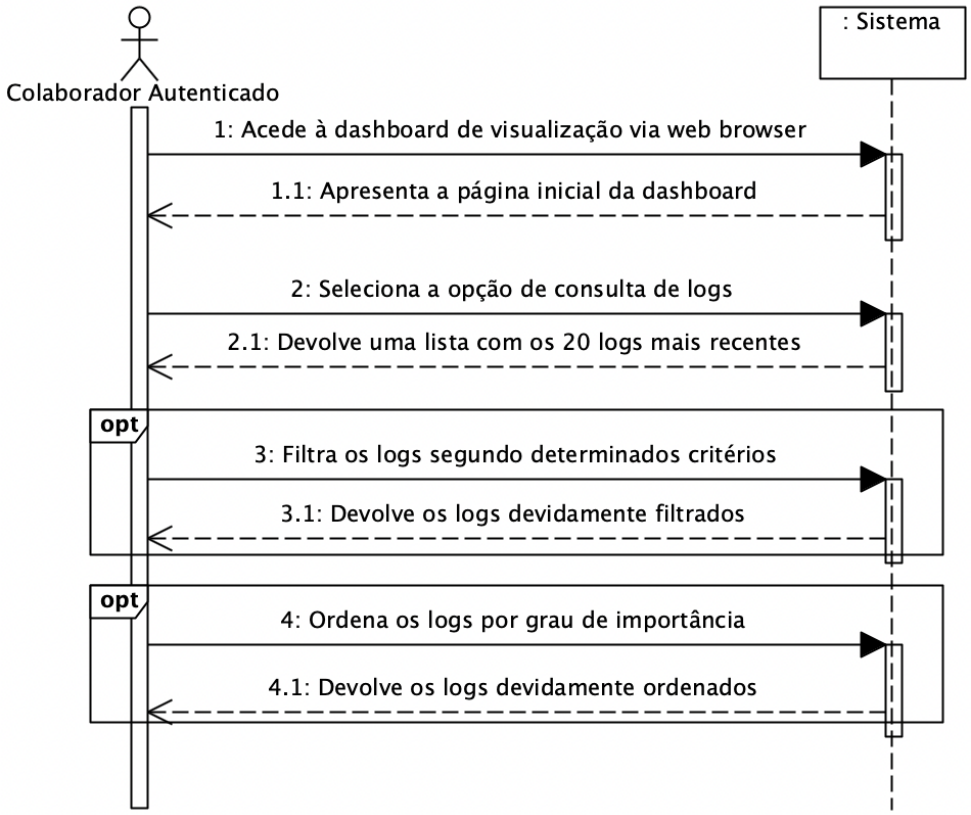

Figura 3.5: Diagrama de Sequência do Sistema

3.3 Análise de Valor

Antes de se proceder ao desenho e implementação da solução é necessário averiguar se a oportunidade denotada pelo problema é de facto válida. De uma forma similar, é importante garantir que a solução escolhida é a mais adequada para resolver o problema. De maneira a verificar se estas diretrizes são cumpridas na totalidade, a presente secção aborda e detalha a metodologia de análise de valor no contexto da solução a desenvolver.

A metodologia mencionada anteriormente, análise de valor, visa identificar, quantificar e retificar fraquezas em processos e produtos e ao mesmo tempo fornecer um conjunto de funções a um custo mínimo. Os princípios subjacentes são os seguintes [87]:

1. Os clientes focam-se nas propriedades funcionais dos produtos;
2. No desenho de um produto todos os custos incidem no provimento de funcionalidades.

A análise de valor também pode ser referida como uma forma de analisar o custo/benefício de um determinado produto, onde as funções são entendidas como características benéficas [88].

3.3.1 Processo de Inovação

A maioria das atividades associadas ao processo de desenvolvimento de um novo produto são realizadas num ambiente probabilístico. A incerteza caracteriza particularmente as fases iniciais do processo de desenvolvimento de um novo produto, pois as organizações têm dificuldade em prever o sucesso no desenvolvimento do produto. Mesmo que o produto venha a ser desenvolvido existem também dificuldades em prever se os objetivos de marketing serão atingidos [89].

O processo de inovação pode ser dividido em 3 partes distintas conforme retratado na figura 3.6.

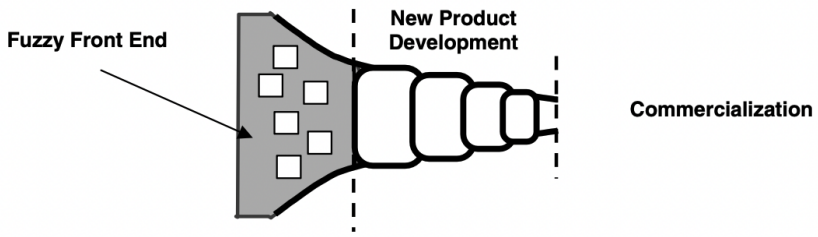

Figura 3.6: Partes do processo de inovação

Fonte: [90]

A primeira parte é relativa ao *front end* da inovação, também conhecido como *Fuzzy Front End* (FFE), e apresenta uma das maiores oportunidades para melhorar o processo de inovação [91]. A importância do FFE reside no facto de que a execução eficaz de atividades de *front end* podem contribuir diretamente para o sucesso de um novo produto. Assim, a competência na manutenção do FFE é importante para garantir o sucesso do novo produto [92].

A segunda parte corresponde ao desenvolvimento do novo produto, denominada de *New Product Development* (NPD), e relaciona-se com a materialização da ideia num produto,

serviço ou funcionalidade. A terceira e última parte, *Commercialization*, compreende todo o processo de promoção e venda do produto final ao consumidor alvo.

O gráfico representado na figura 3.7 detalha os níveis de imprecisão ou turbulência durante o desenvolvimento do novo produto.

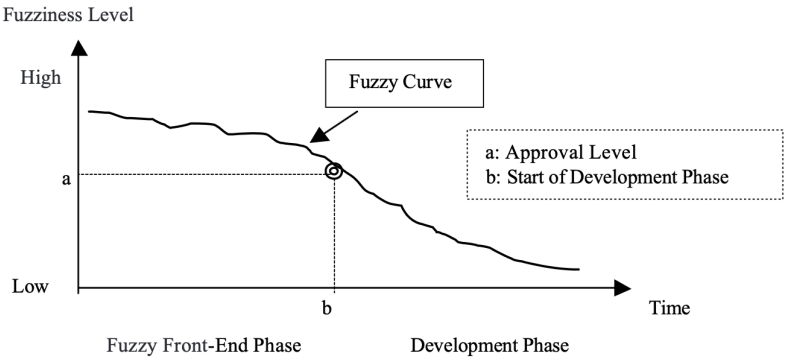

Figura 3.7: Nível de imprecisão do NPD

Fonte: [92]

Como se pode concluir através da observação do gráfico supracitado, o nível de imprecisão diminui gradualmente à medida que o processo de NPD avança. O momento onde existe um maior nível de instabilidade é na fase de FFE devido à natureza incerta dos processos que lhe estão subjacentes.

3.3.2 New Concept Development

O modelo New Concept Development ou NCD foi criado por uma equipa do *Industral Research Institute* composta por oito empresas [91]. Numa fase inicial as empresas estudaram o *front end* com o objetivo de desenvolver uma lista de melhores práticas para o FFE. Contudo os membro da equipa rapidamente concluíram que era impossível determinar as melhores práticas em cada empresa. Comparar os processos de *front end* de uma empresa com os de outra provou ser algo insuportável devido ao facto de não existir uma linguagem comum ou uma definição dos principais elementos do *front end* [91].

De forma a colmatar as deficiências detetadas, a equipa optou por desenvolver um modelo conhecido por NCD no intuito de fornecer discernimento e uma terminologia comum para o FFE. Este modelo, que se encontra ilustrado na figura 3.8, consiste em três partes cruciais:

- **Motor**: representa a liderança, cultura e estratégias de negócio da organização que estimula os cinco elementos-chave controláveis pela organização [90];
- **Elementos-Chave**: Cinto elementos de atividade controláveis do FFE, nomeadamente, identificação da oportunidade, análise da oportunidade, geração de ideias, selecções de ideias e por último definição do conceito [90];
- **Fatores de Influência**: Estes fatores podem ser as capacidades da organização, o mundo exterior (leis, política governamental, concorrentes, políticas económicas, etc.) e as ciências capacitadoras (internas ou externas) que possam estar envolvidas. Estes fatores influenciam todo o processo de inovação e não podem ser controlados pelas organizações [90].

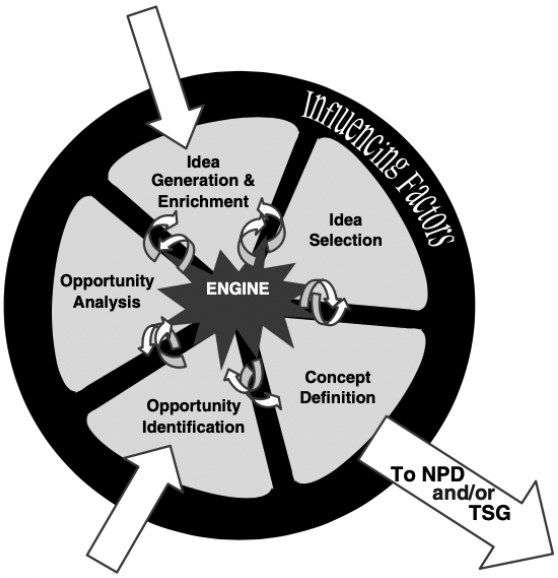

Figura 3.8: Modelo NCD

Fonte: [90]

O restante conteúdo desta subsecção foca-se sobretudo nos **Elementos-Chave** do modelo NCD, já que estes são influenciados pela organização.

Identificação da Oportunidade

É neste elemento que a organização identifica as oportunidades que pretende alcançar. Oportunidades de negócios e tecnológicas são consideradas de maneira a que os recursos da organização sejam alocados para novas áreas de crescimento ou para melhorar a eficácia e eficiência de processos já existentes. Alguns exemplos destas oportunidades podem ser, a resposta de curto prazo a uma ameaça competitiva, uma possibilidade de capturar vantagem competitiva ou um meio de reduzir os custos operacionais [91]. Este elemento é tipicamente motivado pelos objetivos do negócio.

O projeto retratado neste documento, que é centrado no desenvolvimento de um sistema centralizado de *logs*, vai de encontro com um dos exemplos de oportunidade referidos previamente, nomeadamente "um meio de reduzir os custos operacionais". Esta afirmação assenta em tópicos previamente abordados, em particular o facto de que atualmente o processo de manutenção de uma aplicação consome cerca de 70% dos recursos de uma organização [11].

Além disso e de modo a reforçar a ideia de este projeto se trata de uma oportunidade válida, os trabalhos analisados na secção 2.3 apresentam lacunas ou ausências de funcionalidades que tornariam a aplicação a desenvolver mais vantajosa do que as atualmente existentes no mercado.

Análise de Oportunidade

Este elemento avalia uma oportunidade a fim de confirmar se esta deve ou não ser perseguida, como tal é necessária informação adicional para traduzir a identificação da oportunidade em oportunidades específicas de negócio e tecnologias [90].

Para este efeito, escolheu-se efetuar uma análise SWOT ao projeto. A análise SWOT (abreviação de forças, fraquezas, oportunidades e ameaças) é uma ferramenta de estratégia de negócio usada para avaliar como uma organização se compara à concorrência, esta análise inclui fatores internos e externos à organização [93].

De seguida encontra-se definida a terminologia sustentada pela ferramenta [94]:

- **Fatores internos**: Fatores que a pessoa ou a organização tem controlo sobre;
- **Fatores externos**: Fatores que a pessoa ou a organização tem pouco ou nenhum controlo sobre;
- **Fatores benéficos**: Fatores que ajudam a alcançar o sucesso;
- **Fatores prejudiciais**: Fatores que bloqueiam ou impedem o sucesso.

Por último, tem-se as 4 quadrantes da análise SWOT que são representadas pelas letras do acrónimo [94]:

1. **S (Forças)**: São fatores **internos** e **benéficos** que contribuem uma oportunidade ou que ultrapassam uma dada ameaça. As forças podem ser de natureza económica, tecnológica, pessoal, entre outras;
2. **W (Fraquezas)**: São fatores **internos** e **prejudiciais** que significam as incapacidades de tirar proveito de uma oportunidade ou as vulnerabilidades a uma dada ameaça;
3. **O (Oportunidades)**: São fatores **externos** e **benéficos** que podem resultar de avanços tecnológicos, tendências sociais, entre outros;
4. **T (Ameaças)**: São fatores **externos** e **prejudiciais** que podem derivar de novos competidores no mesmo segmento de mercado, regulações restritivas, entre outros.

A figura 3.9 representa as 4 quadrantes da análise SWOT no contexto deste projeto, isto é, analisa-se segundo as diretrizes da ferramenta SWOT a oportunidade detetada no desenvolvimento de um sistema centralizado de *logs*.

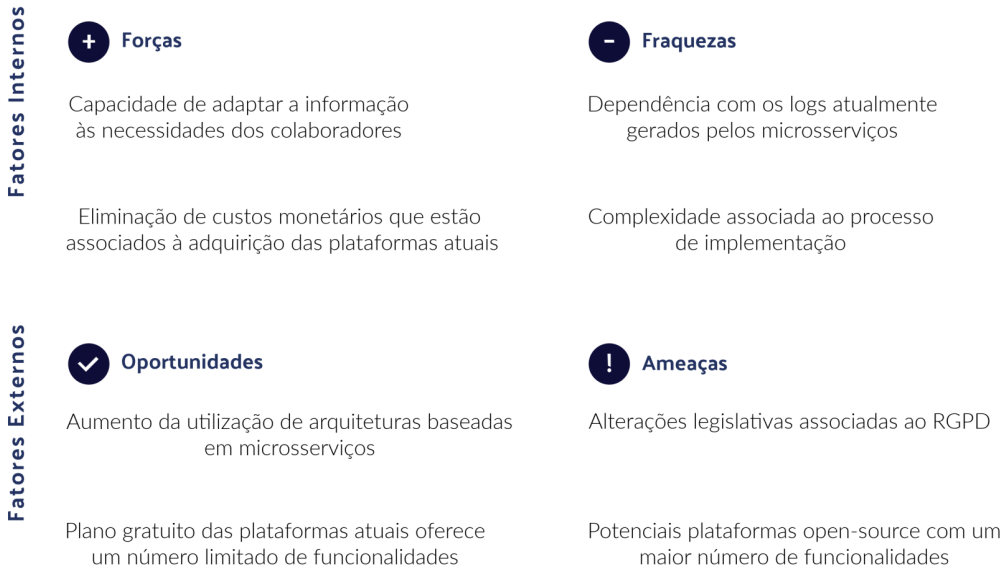

Figura 3.9: Análise SWOT no contexto do projeto

Conforme ilustrado na figura acima, as forças assinaladas estão em conformidade com as desvantagens associadas ao uso de plataformas já existentes, esta informação pode ser revista na secção 2.3. Com isto em mente, os pontos fortes são de facto a flexibilidade inerente ao desenvolvimento de uma aplicação do "zero", pois pode-se adaptar a solução consoante as necessidades do sistema e por último o custo monetário que está ligado à compra de plataformas atuais.

Por outro lado, entende-se que a solução a desenvolver está de certa forma acoplada aos *logs* atualmente gerados pelos microsserviços. Deste modo, aumenta-se a complexidade de implementação não só pelo facto de existir diversos componentes em jogo (interface gráfica, motor de busca, etc.) como também é necessário adaptar o mecanismo de agrupamento e processamento dos *logs* para cada um dos microsserviços. Sendo estas as fraquezas principais associadas ao projeto.

No que concerne a oportunidades, este projeto beneficia da tendência de se desenvolver software com base em arquiteturas de microsserviços, pois trata-se de uma tendência que apresenta um crescimento acelerado nos últimos anos, conforme referenciado no capítulo introdutório. Outra oportunidade é relacionada com o plano gratuito das plataformas abordadas na secção 2.3 que consiste no facto destas restringem certas funcionalidades, o que dá origem à oportunidade de se eliminar essas restrições com a nova solução.

Finalmente, as principais ameaças são as alterações às políticas de proteção de dados nomeadamente o RGPD, isto porque os *logs* estão maioritariamente associados a questões de auditoria e essas alterações podem provocar uma modificação acrescida no sistema. Se por ventura, forem desenvolvidas plataformas *open-source* que oferecem o mesmo número de funcionalidades ou inclusive mais funcionalidades, então estas podem também se considerar ameaças.

Após efetuada a análise SWOT o autor considera a oportunidade associada a este projeto como válida. As fraquezas e ameaças identificadas não são impeditivas nem superam as forças e oportunidades.

Geração de ideias

Uma vez analisada e validada a oportunidade o próximo passo reflete-se na geração de potenciais soluções passíveis de se implementar e capazes de alcançar os objetivos associados à oportunidade identificada.

No processo de geração de potenciais soluções deu-se particular atenção aos trabalhos relacionados (ver secção 2.3), no sentido em que foi analisado o modo como estes combatem o problema de centralização de *logs* bem como as falhas, lacunas e potenciais melhorias que apresentam.

Posto isto, apresentam-se de seguida três possíveis ideias:

- **Desenvolvimento de uma plataforma nova**, de raiz, que compreenda todas as funcionalidades das plataformas atualmente existentes;
- **Desenvolvimento de uma API** que deve ser consumida pelos microsserviços existentes, esta API é responsável por receber e posteriormente tratar os *logs* enviados pelos microsserviços;
- **Utilização do plano gratuito das plataformas**, que em todos os casos implica desenvolver e integrar as funcionalidades em falta;

As ideias expostas conseguem atender na totalidade com os objetivos pretendidos, contudo existem diferenças no que diz respeito à fase de desenvolvimento e implementação entre cada uma delas. É com base nestas diferenças que surge a necessidade de as avaliar segundo um conjunto de critérios de forma a averiguar a melhor solução.

Seleção de Ideias

Na maioria das vezes, o problema não é ter ideias novas. O problema para a maior parte das empresas, é decidir quais as ideias que devem ser seguidas de forma de alcançar um maior valor comercial. Efetuar uma boa seleção de ideias é um passo vital para assegurar o sucesso prolongado de um determinado negócio [90]. De modo a atingir este objetivo, o autor decidiu empregar o método de apoio à decisão *Analytic Hierarchy Process* (AHP).

Este método, desenvolvido por Saaty, visa quantificar prioridades relativas para um determinado conjunto de alternativas numa dada escala. Esta escala tem por base a opinião e a intuição do decisor. Deste modo é necessário garantir a consistência da comparação de alternativas no processo de tomada de decisão [95].

O método pode ser divido em 3 partes distintas: **Divisão Hierárquica**, **Definição de Prioridades** e **Avaliação da Consistência** [96].

A primeira parte, **divisão hierárquica**, centra-se principalmente em 3 pontos chaves: o objetivo, os critérios e as alternativas. O objetivo relaciona-se com a escolha da melhor abordagem de implementação do sistema. Os critérios que serão empregues às alternativas encontram-se de seguida enumerados:

1. **Tempo de desenvolvimento**: Este critério diz respeito ao tempo de desenvolvimento do sistema. Visto que projeto está a ser realizado num contexto académico, existem prazos que devem ser cumpridos. Para este efeito, considera-se uma alternativa superior a outra se o tempo despendido na fase de implementação for menor;
2. **Complexidade**: Este critério compreende a dificuldade associada à implementação dos componentes de cada uma das alternativa como também os custos ligados aos processos de manutenção. Para este efeito, considera-se uma alternativa superior a outra se a dificuldade e os custos de implementar e manter a solução for menor;
3. **Desempenho**: Este critério preocupa-se com questões de desempenho da alternativa, tais como a velocidade na pesquisa de informação ou o quão recente é a informação que está a ser visualizada. Para este efeito, considera-se uma alternativa superior a outra se os tempos de resposta forem menores e se os dados apresentados forem mais recentes.

As alternativas consistem nas três ideias referidas anteriormente, quando se abordou a temática de geração de ideias. Estas alternativas, que por questões de clareza foram renomeadas, são respetivamente:

- Plataforma nova;
- API integração;
- Integração com plataforma.

A figura 3.10 expõe os três pontos-chave supracitados através de uma estrutura hierárquica. O objetivo encontra-se no topo do diagrama, os critérios no meio e as diferentes alternativas no fundo.

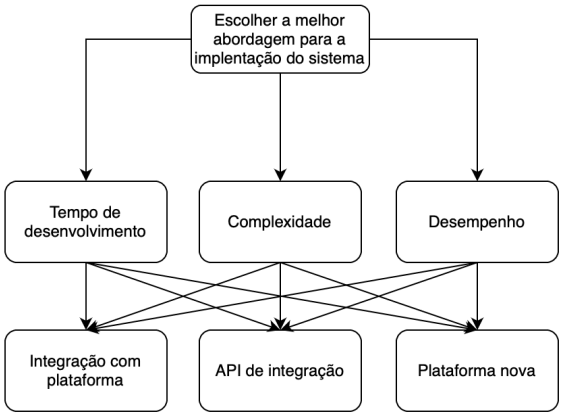

Figura 3.10: Estrutura Hierárquica do AHP

A segunda parte do método AHP, **definição de prioridades**, consiste em definir prioridades para cada um dos critérios previamente estabelecidos com recurso à escala apresentada na tabela 3.1 [95].

Tabela 3.1: Escala de preferências AHP

Classificação Numérica	**Definição**
9	Extremamente preferido
8	Muito fortemente a extremamente preferido
7	Muito fortemente preferido
6	Fortemente a muito fortemente preferido
5	Fortemente preferido
4	Moderadamente a fortemente preferido
3	Moderadamente preferido
2	Igualmente a moderadamente preferido
1	Mesma preferência

Destaca-se que as classificações numéricas ímpares são os níveis principais:

- **Classificação 1**: Ambas as atividades contribuem de igual forma no alcance dos objetivos;
- **Classificação 3**: A preferência de uma das atividades é moderadamente mais forte do que outra;
- **Classificação 5**: A preferência de uma das atividades é mais forte do que outra;
- **Classificação 7**: A preferência de uma das atividades é muito mais forte do que outra;
- **Classificação 9**: A preferência de uma das atividades é extremamente mais forte do que outra, sendo esta a classificação máxima da escala.

Os restantes níveis (2, 4, 6 e 8) correspondem a valores que se intercalam com os classificações ímpares.

A tabela 3.2 apresenta a matriz de prioridades tendo por base os critérios de **complexidade (A)**, **desempenho (B)** e **tempo de desenvolvimento (C)**.

Tabela 3.2: Prioridade relativa de cada critério

	A	B	C	Prioridade Relativa
A	1	1/2	4	0.33
B	2	1	5	0.57
C	1/4	1/5	1	0.10

A tabela acima evidencia a ordem de prioridade relativa de cada um dos critérios: **Desempenho** -> **Complexidade** -> **Tempo de desenvolvimento**. Além disso, é possível constatar que:

- O critério complexidade é quatro vezes mais preferencial do que o critério tempo de desenvolvimento;
- O critério desempenho é duas vezes mais preferencial do que o critério complexidade;
- O critério desempenho é cinco vezes mais preferencial do que o critério tempo de desenvolvimento.

Todos estes resultados vão de encontro com as expectativas do autor. Acredita-se que o **critério desempenho é fortemente preferencial** pois no contexto do projeto, o objetivo principal da solução é diminuir o tempo despendido a diagnosticar problemas no sistema. Imediatamente a este critério, tem-se o fator de **complexidade que se considerou moderadamente preferencial**, uma vez que quanto menos complexa for a solução, mais facilmente esta é manutenível e alterada. Por último considerou-se o **tempo de desempenho como o critério menos preferencial**, isto porque o autor tem a possibilidade de estender o prazo de entrega e também pelo facto de se preferir investir mais tempo no desenvolvimento da solução de forma a evitar alterações no futuro.

O próximo passo traduz-se na priorização das alternativas com base nestes critérios. Para tal, é apresentada uma tabela de prioridades relativas para cada critério tendo por base as alternativas propostas. De maneira a aumentar o grau de clareza das tabelas, utilizou-se a seguinte terminologia no que diz respeito às diferentes alternativas:

- **X**: Plataforma nova;
- **Y**: API de integração;
- **Z**: Integração com plataforma.

De seguida, apresenta-se a tabela 3.3 que prioriza as alternativas pelo critério de tempo de desenvolvimento.

Tabela 3.3: Prioridade relativa das alternativas segundo o critério - Tempo de desenvolvimento

	X	Y	Z	Prioridade Relativa
X	1	1/4	1/7	0.08
Y	4	1	1/3	0.26
Z	7	3	1	0.66

A tabela acima evidencia a ordem de prioridade relativa de cada uma das alternativas: **Integração com plataforma** -> **API de integração** -> **Plataforma nova**. Além disso, é possível constatar que:

- A alternativa API de integração é quatro vezes mais preferencial do que a alternativa de desenvolver uma plataforma nova;
- A alternativa integração com plataforma é três vezes mais preferencial do que a alternativa de API de integração;
- A alternativa integração com plataforma é sete vezes mais preferencial do que a alternativa de desenvolver uma plataforma nova.

As prioridades obtidas estão conforme o esperado, pois o desenvolvimento de uma aplicação de raiz é a que indubitavelmente consome mais tempo. A seguir, tem-se a alternativa de uma API de integração que consome menos tempo do que a alternativa anterior, pois a responsabilidade de enviar os *logs* para a API deixa de recair nesta solução. Por último, a alternativa mais adequada para o critério em questão é a integração com uma plataforma já existente, pois neste caso a maioria das funcionalidades já se encontram implementadas.

Logo após, na tabela 3.4 encontram-se denotadas as prioridades relativas de cada alternativa tendo como base o critério de complexidade.

Tabela 3.4: Prioridade relativa das alternativas segundo o critério - Complexidade

	X	**Y**	**Z**	**Prioridade Relativa**
X	1	2	5	0.57
Y	1/2	1	4	0.33
Z	1/5	1/4	1	0.10

A tabela acima evidencia a ordem de prioridade relativa de cada uma das alternativas: **Plataforma nova** -> **API de integração** -> **Integração com plataforma**. Além disso, é possível constatar que:

- A alternativa API de integração é quatro vezes mais preferencial do que a alternativa de integrar com uma plataforma já existente;
- A alternativa de desenvolver uma nova plataforma é duas vezes mais preferencial do que a alternativa de API de integração;
- A alternativa de desenvolver uma nova plataforma é cinco vezes mais preferencial do que a alternativa de integrar com uma plataforma já existente.

Para o critério de complexidade, obteve-se como alternativa menos preferencial a integração com plataforma, isto acontece pois a complexidade associada a criar, integrar e manter componentes novos e componentes de terceiros é com base na experiência do autor superior às das outras alternativas. Dentro das alternativas restantes, API de integração e nova plataforma, a primeira apresenta uma complexidade superior pois nos casos onde se é necessário transformar a informação dos *logs* antes de os receber, a API de integração não suporta esta alteração, o que provoca um ligeiro aumento de complexidade quando comparada à alternativa mais preferencial (desenvolvimento de uma nova plataforma).

Para finalizar a comparação dos critérios com as diferentes alternativas, a tabela 3.5 apresenta as prioridades relativas assentes no critério de desempenho.

Tabela 3.5: Prioridade relativa das alternativas segundo o critério - Desempenho

	X	Y	Z	Prioridade Relativa
X	1	3	6	0.63
Y	1/3	1	5	0.29
Z	1/6	1/5	1	0.08

A tabela acima evidencia a ordem de prioridade relativa de cada uma das alternativas: **Plataforma nova** -> **API de integração** -> **Integração com plataforma**. Além disso, é possível constatar que:

- A alternativa API de integração é cinco vezes mais preferencial do que a alternativa de integrar com uma plataforma já existente;
- A alternativa de desenvolver uma nova plataforma é três vezes mais preferencial do que a alternativa de API de integração;
- A alternativa de desenvolver uma nova plataforma é seis vezes mais preferencial do que a alternativa de integrar com uma plataforma já existente.

A alternativa menos preferencial, integração com uma plataforma já existente, é a que o autor crê ser a menos performante devido à necessidade de se integrar novos componentes com a plataforma, provocando uma maior latência de todo o sistema. Das outras duas alternativas a que aparenta ser mais performante é a de desenvolvimento de uma nova plataforma, pois ao contrário da API de integração o processo de agrupar e transformar os logs é tratado único e exclusivamente pela solução a desenvolver e como já referido no capítulo das tecnologias (ver secção 2.4) a *ELK stack* apresenta os melhores tempos de resposta.

Uma vez efetuada a comparação dos critérios com as alternativas, o próximo passo remete na identificação da alternativa que melhor responde ao objetivo inicialmente definido. Como tal, elaborou-se a matriz da tabela 3.6 que relaciona todas as alternativas com todos os critérios.

Tabela 3.6: Prioridades relativas das soluções

	A	B	C	Prioridade Relativa
X	0.57	0.63	0.08	0.56
Y	0.33	0.29	0.26	0.30
Z	0.10	0.08	0.66	0.14
Prioridade Relativa	0.33	0.57	0.10	

A coluna mais à direita da tabela apresenta a prioridade relativa final de cada solução. Estes valores foram obtidos a partir da multiplicação da matriz de prioridades das soluções com o vetor de pesos de critério calculado na tabela 3.2 que se encontra representado na última linha da tabela 3.6. Posto isto, é possível inferir que a melhor alternativa é a **X, que está relacionada com o desenvolvimento de uma nova plataforma**, esta afirmação está fundamentada nos resultados obtidos, em particular pelo maior valor de prioridade relativa que foi de 0.56.

Ainda assim, resta efetuar a última parte do AHP que tem como interesse garantir a consistência dos resultados obtidos. Esta validação de consistência pode ser comprovada através do cálculo da razão de consistência (RC), que se encontra representada na fórmula 3.1:

$$RC = \frac{IC}{IR} \tag{3.1}$$

Se o valor obtido de RC for superior a 0.1 então todos os cálculos anteriormente efetuados não são fidedignos. Isto porque a avaliação de consistência está assente na lei da transitividade nas prioridades atribuídas aos critérios. Por exemplo, se o critério A é mais preferencial que o B e se o B é mais preferencial que o C então o critério A é mais preferencial que o C.

Para o cálculo do IC, que se encontra na fórmula 3.2, é necessário atribuir valores de n e λmax.

$$IC = \frac{\lambda max - n}{n - 1} \tag{3.2}$$

O valor n é relativamente simples de se atribuir, pois esta incógnita corresponde à quantidade de critérios utilizados ou seja n=3. O valor de λmax pode ser calculado através da equação $Ax = \lambda\text{max}\ x$, onde x representa o vetor de prioridades de cada critério e Ax a multiplicação deste vetor com a matriz de prioridades **não normalizada**.

Os resultados da dita multiplicação podem ser visualizados na matriz de consistência representada pela tabela 3.7.

Tabela 3.7: Matriz de consistência

A	1.01
B	1.73
C	0.30

Com base nestes valores já é possível calcular o quociente de λmax, conforme ilustrado na equação 3.3

$$\lambda max = \frac{1.01/0.33 + 1.73/0.57 + 0.30/0.10}{3} = 3.03 \tag{3.3}$$

Com os valores de n e de λmax calculados, resta agora substituí-los na fórmula 3.4 para se obter o valor de IC:

$$IC = \frac{\lambda max - n}{n - 1} = \frac{3.03 - 3}{3 - 1} = 0.01 \tag{3.4}$$

A última etapa da validação da consistência remete para o cálculo do valor de RC. Agora que se tem o valor de IC, sobra calcular o valor de índice aleatório (IR). Como o IR trata-se de um valor tabelado, este relaciona-se com o valor de n utilizado no processo AHP [95]. A tabela 3.8 indica os valores a utilizar no IR de acordo com o respetivo valor de n.

Tabela 3.8: Valor de índice de aleatoriedade

Tamanho da matriz	1	2	3	4	5	6	7	8	9	10
Índice de aleatoriedade	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

Sendo $n = 3$, conclui-se então que IR = 0.58. Desta forma torna-se possível calcular o rácio de consistência através da equação 3.5:

$$RC = \frac{0.01}{0.58} \approx 0.02 \tag{3.5}$$

Uma vez que o valor obtido de RC (0.02) é inferior a 0.1, é seguro afirmar que os resultados obtidos ao longo da aplicação do método AHP são fiáveis. Por conseguinte, **garante-se que a melhor alternativa remete para o desenvolvimento de uma plataforma nova**.

3.3.3 Valor da solução

Em virtude da análise da secção 3.3.2 resultou uma possível solução para o projeto em causa. Cabe agora ao autor perceber de que forma esta solução contribui com um acréscimo de valor para uma entidade organizacional e para os utilizadores finais. Para atingir este objetivo, são empregues processos e ferramentas que analisem a proposta de valor da solução como também as necessidades dos clientes.

Valor

De forma a se expressar a relevância de um produto de um modo simples e conciso, considerou-se importante definir os conceitos que estão subjacente quando se põe em causa o valor de um produto.

O conceito de **valor** pode ser definido como uma medida expressa em dinheiro ou esforço que reflete o desejo de se reter ou obter um determinado produto ou serviço. Outra definição de valor, indica que este é a troca equivalente de serviço ou bens que um proprietário ou consumidor recebem em troca de dinheiro [97]. Como consequência destas definições, é possível concluir que existe uma componente subjetiva associada ao preço de um produto.

É a partir dessa característica intrínseca dos clientes que surge o **valor para o cliente**. Este conceito indica que o valor que um cliente atribui a um dado produto varia conforme a sua percepção pessoal [97], ou seja, o mesmo produto para clientes diferentes pode apresentar valores distintos. Por último tem-se o conceito de **valor percecionado** que incide na diferença entre a avaliação que um potencial cliente faz de todos os benefícios e custos associados a um produto ou serviço [98].

No contexto deste projeto, os clientes/consumidores finais correspondem ao colaboradores de uma determinada organização. Considera-se como o maior benefício da solução a aceleração do processo de diagnosticar e corrigir problemas ou anomalias no sistema informático. Apesar disso, estima-se que o maior sacrifício envolvido na utilização deste produto, remete para a fase inicial onde os colaboradores ainda não se encontram familiarizados com o produto.

Proposta de Valor

A proposta de valor traduz-se na visão geral do conjunto de produtos e serviços de uma organização que são de valor para o cliente [99]. Deste modo é necessário evidenciar os benefícios associados à aquisição do produto em causa. Para tal, a proposta de valor é realizada em dois diferentes formatos - **Value Proposition Canvas** e **Cadeia de Valor de Porter**.

O **Value Proposition Canvas** , também conhecido como Canvas da proposta de valor, é composto por duas partes distintas. Uma dessas partes diz respeito ao perfil do cliente e visa esclarecer questões como dores e ganhos do cliente. A restante parte consiste no mapa de valor que descreve como se pretende gerar valor para o cliente [100].

A figura 3.11 expõe no contexto do projeto as duas partes previamente mencionadas.

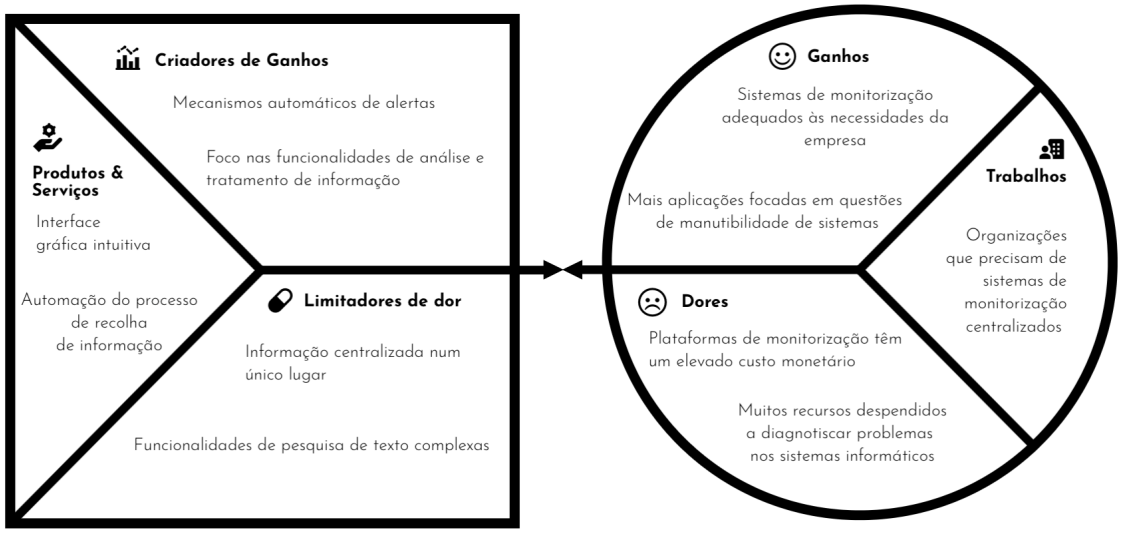

Figura 3.11: Canvas de proposta de valor

Partindo da óptica do utilizador é possível observar os seguintes atributos:

- **Dores**: Os principais problemas detetados vão de encontro com tópicos já previamente abordados, nomeadamente o custo das plataformas de centralização de *logs* bem como o tempo gasto a depurar sistemas baseados em arquiteturas de microsserviços;
- **Trabalhos**: Os trabalhos procurados por organizações com sistemas informáticos complexos encontram-se ligados ao desenvolvimento de plataformas que reduzam o recursos utilizados em processos de manutenção;
- **Ganhos**: Os clientes, que neste caso em concreto representam os colaboradores, procuram sistemas de monitorização centralizados adequados às necessidades da organização e aplicações focadas na manutibilidade do sistema.

Por último tem-se as seguintes características na perspetiva do mapa de valor:

- **Limitadores de Dor**: Os principais obstáculos destes utilizadores são ultrapassados através de funcionalidades de pesquisa de texto completo e de um local centralizado que contém toda a informação do sistema;

- **Criadores de Ganhos**: A proposta pretende gerar valor para os utilizadores por meio da implementação de novos mecanismos (e.g alertas) e através de funcionalidades de análise e tratamento de informação;
- **Produtos & Serviços**: O produto oferecido disponibiliza uma interface gráfica de fácil uso e intuitiva. Além disso também fornece mecanismos de recolha e tratamento automático da informação do sistema.

O próximo passo consiste na aplicação da **Cadeia de Valor de Porter** ao projeto em mão. Esta cadeia de valor, concebida por Michael Porter em 1985, trata-se de uma ferramenta para gerir processos que revela todas as atividades que uma organização efetua e como estas se relacionam com o objetivo de gerar valor para os clientes [101].

Para complementar essa definição, a cadeira de valor é uma série de atividades interligadas que são necessárias para garantir uma perceção positivas dos clientes relativamente aos produtos e serviços de uma organização [101].

A figura 3.12 denota as atividades previamente referidas, dividindo-as em duas categorias: atividades primárias e atividades de apoio.

Figura 3.12: Cadeia de Valor de Porter

Fonte: [102]

As atividades primárias são as que se relacionam diretamente com a criação física, venda, manutenção e suporte de um dado produto ou serviço [102]. Estas atividades são nomeadamente:

1. **Logística de entrada**: Envolve a compra de matéria prima ou a contratação de serviços que mais tarde serão transformados em produtos [101];
2. **Operações**: Atividades que transformam o *input* no produto final [102];
3. **Logística de saída**: Atividades associadas com o processo de entrega do produto ao cliente alvo [102];
4. **Marketing e Vendas**: Processos utilizados pelas organizações a fim de convencer os clientes a adquirirem os seus produtos ou serviços [102];
5. **Serviços**: Atividades que aumentam o valor dos produtos ou serviços após a compra [102].

As atividades de apoio geram valor de uma forma indireta na medida em que apoiam as atividades primárias da organização [101]. Estas atividades encontram-se de seguida enumeradas:

1. **Infraestrutura**: Sistemas de apoio que a organização necessita a fim de manter as operações diárias (Gestão financeira, contabilística, legal, etc.) [102];
2. **Gestão de Recursos Humanos**: Envolve atividades como recrutar e selecionar novos colaboradores ao mesmo tempo que se articula programas de desenvolvimento [101];
3. **Desenvolvimento tecnológico**: Atividades que apoiam os processos primários através de intervenções tecnológicas, como a automação de processos e o uso de ferramentas digitais [101];
4. **Aquisição/Compra**: Todos os processos realizados pela organização de maneira a adquirir recursos necessários no desempenho das tarefas [102].

A margem, representada pelo único triângulo da figura supracitada, simboliza a diferença entre o valor percebido pelo utilizador e o custo total de produção do serviço ou produto, como consequência a margem está diretamente ligada à rentabilidade da empresa [101].

É possível descrever as principais atividades que estão associadas ao projeto em mão e interligá-las com as atividades primárias. Desta forma define-se como e onde é gerado valor com o desenvolvimento deste projeto.

As atividades principais que foram detetadas encontram-se abaixo enunciadas:

- **Operações**: Desenvolvimento do projeto em causa;
- **Logística de Saída**: O produto que se vai desenvolver deve ser integrado com o cenário hipotético abordado na secção 3.1;
- **Serviços**: Manutenção da aplicação desenvolvida e caso haja necessidade o desenvolvimento de novas funcionalidades.

As atividades de apoio identificadas são as que se seguem:

- **Infraestrutura**: Implementação de medidas para assegurar que os *logs* estão de acordo com a legislação de protecção de dados (RGPD);
- **Gestão de Recursos Humanos**: Formações que lecionem os colaboradores a utilizar o novo produto;
- **Aquisição/Compra**: Aquisição de servidores que serão utilizados para alojar/hospedar todos os componentes desenvolvimentos neste projeto.

É a partir destas atividades que o autor crê ser possível gerar valor para a organização e os seus utilizadores. O objetivo passa por desenvolver uma nova plataforma, de raíz, que atue como um sistema de monitorização de *logs* centralizado. Esta solução vai permitir à organização preencher as lacunas detetadas na secção 2.3 e como efeito oferecer valor para a organização.

Necessidades do Cliente

Durante a fase de desenvolvimento de software em grande escala, a gestão eficaz e eficiente dos requisitos do cliente é um dos problemas mais prevalentes e também um dos menos compreendido. Os problemas encontrados nos requisitos tipicamente não são reconhecidos

até ao final do processo de desenvolvimento, onde os impactos negativos são substanciais [103].

Em vista disso, é necessário tirar partido de métodos ou ferramentas que ajudem os desenvolvedores de software a gerir melhor os requisitos e as respetivas necessidades do cliente. Um desses métodos, intitulado de **Quality Function Deployment** (QFD) foi aplicado ao projeto em mão.

O QFD trata-se de um processo e de um conjunto de ferramentas utilizadas para definir os requisitos do cliente e convertê-los em especificações detalhadas de engenharia. Este processo é usado para traduzir as necessidades do cliente em metas de projeto mesuráveis. Para tal, fornece um conjunto bem definido de matrizes que devem ser utilizadas de forma a auxiliar a progressão dos requisitos [104].

Este processo compreende quatro fases onde cada uma delas é composta por atividades que devem ser executadas no decorrer do desenvolvimento do produto [104]. No âmbito deste projeto vai-se analisar e aplicar apenas a primeira fase, denominada de **Definição do Produto**, pois foi a que se considerou mais apropriada.

A fase supracitada consiste na tradução dos desejos e necessidades do cliente em especificações do produto. Também envolve uma componente competitiva no sentido em que avalia a eficácia de produtos concorrentes que satisfaçam as necessidades do cliente [104].

De maneira a se aplicar a fase de definição do produto é necessário definir os seguintes parâmetros:

1. Necessidades do cliente;
2. Importância das necessidades do cliente;
3. Características de Engenharia;
4. Escalas de correlação;
5. Concorrentes.

Como o autor não tem contacto direto com o cliente, uma vez que este documento retrata o desenvolvimento de um projeto académico no intuito de resolver problemas de manutibilidade de uma organização fictícia, foi necessário efetuar um estudo das soluções existentes conforme abordado na secção 2.3. Adicionalmente e como detalhado na secção 3.2.2, analisou-se dissertações de mestrado e relatórios técnicos para averiguar quais as necessidades que tipicamente as organizações têm quando se desenvolvem soluções de monitorização de *logs* centralizadas. Abaixo encontram-se listadas essas mesmas necessidades:

- **Informação relevante**: A informação que é apresentada aos colaboradores da organização deve auxiliar no máximo possivel o processo de depuração do sistema;
- **Logs em diversos formatos**: A informação que provém de *logs* de microsserviços deve ser tratada e analisada independentemente do seu formato original;
- **Facilidade de utilização** - A plataforma deve ser fácil e intuitiva de utilizar sem existir a necessidade de possuir grandes conhecimentos técnicos;
- **Filtragem da informação** - O colaborador deve ser capaz de filtrar e pesquisar informação segundo os seus critérios, devendo a informação ser coerente com os critérios introduzidos pelo colaborador;

- **Velocidade da informação** - A informação que é apresentada aos colaboradores deve ser obtida de uma forma rápida, não devendo o colaborador aguardar vários minutos até obter a informação desejada.

Definidas as necessidades o próximo passo compete à classificações destas, onde se atribuiu um índice de importância numa escala de 1 a 5, sendo que o 5 corresponde ao grau de maior importância. A classificação das necessidades encontra-se ilustrada na tabela 3.9.

Tabela 3.9: Necessidades do cliente e respetivas importâncias.

Necessidades	Importância (Absoluta)	Importância (Relativa %)
Informação relevante	5	25%
Logs em diversos formatos	4	20%
Facilidade de utilização	3	15%
Filtragem da informação	4	20%
Velocidade da informação	4	20%

Uma vez identificas e classificadas as necessidades do cliente, é necessário traduzi-las em características de engenharia. Estas são as características que a solução deve obedecer e incorporar de maneira a responder às necessidades do cliente:

- **Tempo de Resposta**: Característica associada ao tempo que o sistema demora a responder a um pedido do colaborador, neste caso o pedido de consulta de informação de *logs*;
- **Usabilidade**: Característica associada à facilidade de utilização da plataforma de modo a que as necessidades dos colaboradores sejam atingidas de forma eficaz e eficiente;
- **Adaptabilidade**: Característica associada à dificuldade de adaptar a informação proveniente de *logs* cujo formato é diferente. Esta dificuldade deve ser a mínima possível;
- **Categorização e Ordenação**: Característica associada à capacidade do sistema ordenar e categorizar a informação dos *logs* segundo vários critérios.

Por último e de modo a relacionar estas características com as necessidades do cliente previamente estabelecidas, é necessário definir duas escalas. Uma dessas escalas encontra-se ilustrada na figura 3.13 e representa as relações entre as características de engenharia e as necessidades do cliente, a outra escala denotada pela figura 3.14 representa a correlação entre as diferentes características de engenharia.

No que diz respeito à análise de plataformas concorrentes, apenas se considerou as plataformas que respondiam de uma forma minimamente satisfatória às necessidades do cliente, nomeadamente a plataforma SumoLogic e Seq. Estas foram classificadas de 1 a 5 no que remete às necessidades do cliente, onde a classificação 5 é a melhor delas. É importante sublinhar que estas classificações advêm do próprio autor, através da experiência de utilização de cada uma das plataformas.

De acordo com a figura 3.15, é possível retirar as seguintes conclusões:

- A necessidade do cliente Informação Relevante é a que apresenta o maior índice de importância;
- As características de engenharia Categorização/Ordenação e Tempo de Resposta são as que apresentam um maior grau de importância respetivamente;

Matriz de relação		
	Forte	9
	Média	3
	Fraca	1
	Sem Relação	0

Figura 3.13: Matriz relação de características e necessidades.

Matriz de correlação	
+ +	Muito Positiva
+	Positiva
–	Negativa
– –	Muito Negativa
	Sem relação

Figura 3.14: Matriz de correlação de características.

- A plataforma Seq apresenta um maior grau de satisfação do que a plataforma Sumo-Logic, algo já expectável visto que o Seq fornece o maior número de funcionalidades entre todas as plataformas comparadas na secção 2.3;
- A necessidade facilidade de utilização e a característica usabilidade são as que apresentam o menor índice de importância nas suas respetivas categorias;
- A característica Usabilidade e Tempo de resposta e Adaptabilidade têm uma correlação negativa, uma vez que quanto maior o número de formatos de *logs* a suportar maior será o tempo de processamento;
- A característica de Tempo de resposta e Categorização/Ordenação têm uma correlação negativa, visto que quanto mais critérios de ordenação se utilizar maior será o tempo de processamento;
- A característica de Usabilidade e Categorização/Ordenação têm uma correlação positiva, pois um maior número de opções de categorização e ordenação dos resultados contribui para uma melhor experiência de utilização da aplicação.

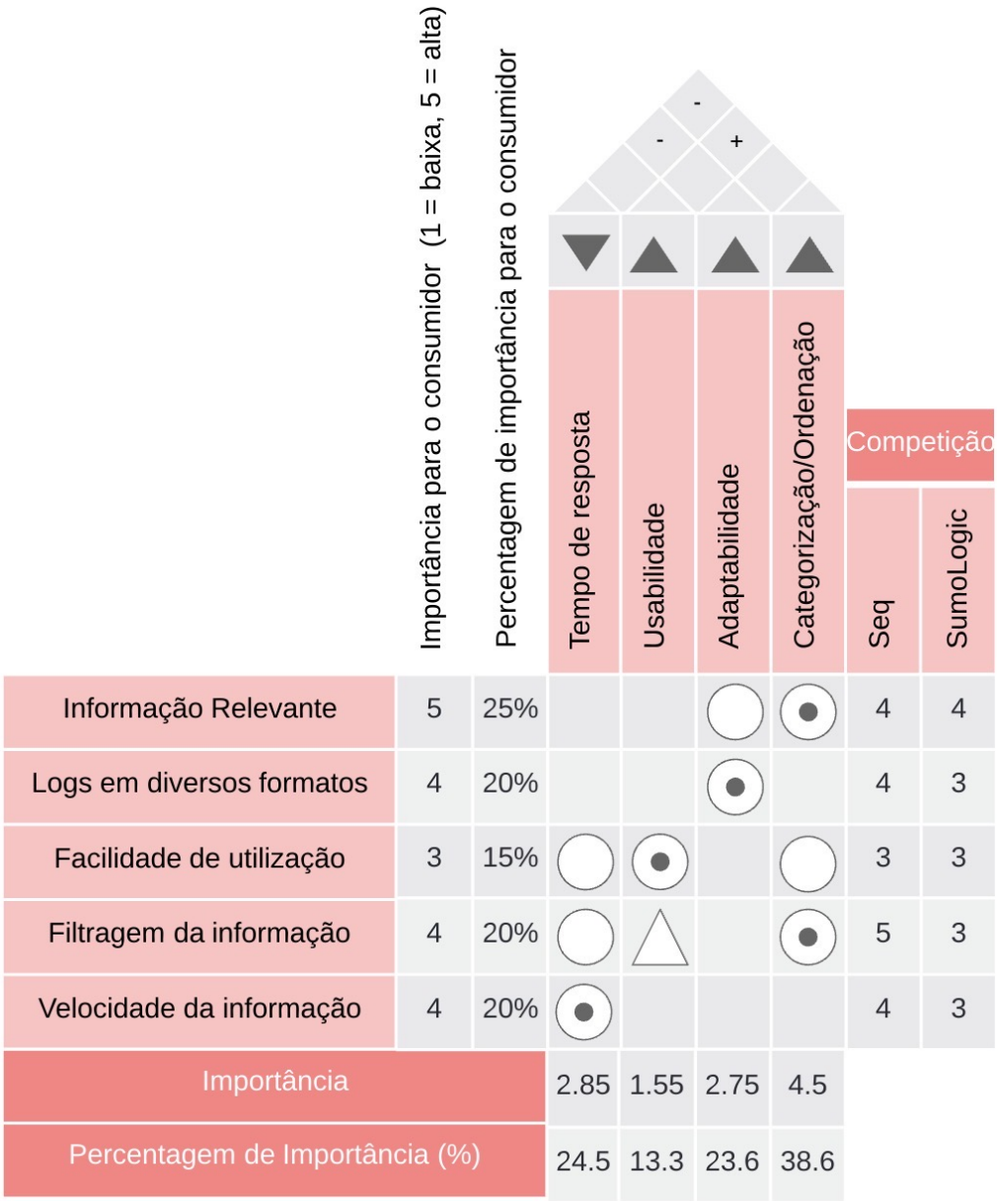

Figura 3.15: QDF aplicado ao projeto

Capítulo 4

Desenho da Solução

Uma vez definidos e categorizados os requisitos funcionais e não funcionais da solução e seguindo o trajeto inicialmente delineado, atinge-se a fase de desenho da solução. Esta fase, cujo o presente capítulo aborda, visa garantir que o desenvolvimento do projeto decorre de forma eficiente e eficaz. Para este efeito, é necessário tomar decisões que se relacionam com a arquitetura do sistema. Como tal, deve-se produzir artefactos que evidenciem os componentes do sistema e a forma como estes interagem. Estes artefactos apresentam uma visão estática do sistema, através de diagramas de componentes e implantação, como também apresentam uma visão dinâmica do sistema recorrendo a diagramas de sequência.

4.1 Arquitetura

Esta secção aborda arquiteturas passíveis de se aplicar à solução a desenvolver. Estas arquiteturas são analisadas e comparadas no intuito de escolher a que melhor se adequa ao problema em causa. Martin Fowler denota a arquitetura de software como um conjunto de decisões importantes e difíceis de alterar [105].

Por esse motivo, é necessário que as decisões relacionadas com a arquitetura da solução estejam devidamente fundamentadas. Neste sentido, decidiu-se tirar partido do modelo de vista arquitetónica 4+1. Este modelo é compreendido sobretudo pelas seguintes vistas [106]:

- **Vista Lógica**: Esta vista preocupa-se com as funcionalidades que o sistema proporciona aos utilizadores finais. Podem-se utilizar diagramas UML para representar esta vista, nomeadamente diagrama de classes e de estados;
- **Vista de Desenvolvimento**: Esta vista, também conhecida como vista de implementação, detalha o sistema do ponto de vista do desenvolvedor e preocupa-se com a gestão de software. Os diagramas UML que representam esta vista incluem os diagramas de componentes e de pacotes;
- **Vista Física**: Esta vista, também denominada de vista de implantação, descreve o sistema da perspetiva de um engenheiro de sistemas. Preocupa-se com a topologia dos componentes de software na camada física, incluindo as ligações físicas e protocolos utilizados entre estes. Utiliza-se o diagrama de implantação para representar esta vista;
- **Vista de Processos**: Esta vista lida com aspetos dinâmicos do sistema, explica os seus processos e foca-se no comportamento do sistema em *runtime*. Os diagramas UML utilizados para representar esta vista são os diagramas de sequência, comunicação e atividades;

- **Cenários**: Esta quinta e última vista, trata-se de uma vista que tendo em conta as anteriores é redundante, daí resultar o "+1"da nomenclatura do modelo. Ainda assim, serve como condutor para descobrir os elementos arquiteturais durante a conceção da arquitetura e desempenha o papel de validação e ilustração após a conclusão do projeto.

Das vistas previamente enunciadas, tanto a **vista lógica** como a **cenários** encontram-se destacadas na secção 3.2, em específico na definição do modelo de domínio e do diagrama de casos de uso respetivamente.

É de referir que por questões de legibilidade optou-se por representar os microsserviços do projeto **Online Boutique** através de um só componente nos casos em que o projeto é ilustrado nos diagramas.

No que se refere às subsecções que de seguida são apresentadas, é importante frisar que as alternativas ilustradas em cada subsecção (e.g alternativa A, alternativa B, etc.) **não se relacionam e são totalmente independentes de subsecção para subsecção**. Por exemplo, a alternativa A da vista de processos não impacta nem se correlaciona com a alternativa A da vista física.

4.1.1 Vista de Desenvolvimento

A partir da vista de desenvolvimento e tendo por base uma análise de granularidade grossa, é possível constatar quais os componentes/aplicações que fazem parte do sistema, os serviços que disponibilizam para o exterior e também a forma como estes são consumidos.

Alternativa A

Numa fase bastante inicial do projeto, o autor considerou implementar o sistema com base no diagrama representado na figura 4.1.

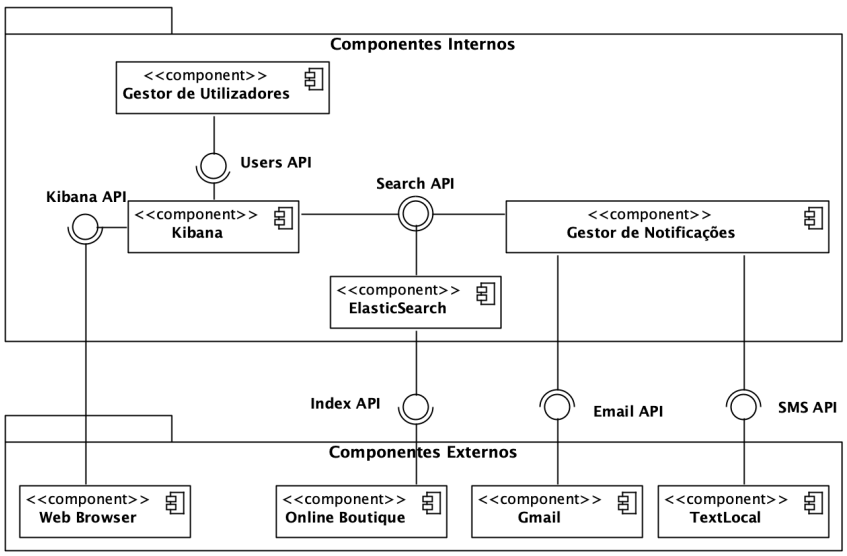

Figura 4.1: Diagrama de componentes da alternativa A

O diagrama em questão efetua uma divisão lógica dos componentes (internos e externos) e é compreendido por oito componentes. Os componentes internos representam as partes

do sistema que são desenvolvidas ou configuradas no âmbito deste projeto e são as que se seguem:

1. **Kibana**: Componente responsável por disponibilizar uma interface gráfica de visualização de dados dos *logs*, para atingir este objetivo o Kibana consome os serviços de consulta de informação do ElasticSearch e de autenticação e autorização do Gestor de Utilizadores;
2. **ElasticSearch**: Representa a peça central do sistema, pois este componente é responsável por disponibilizar serviços de indexação e consulta de informação, nomeadamente operações CRUD relativas aos *logs* do sistema Online Boutique. Além disso também atua como um motor de pesquisa e um mecanismo de persistência de dados;
3. **Gestor de Notificações**: Este componente é responsável por enviar notificações aos colaboradores da organização, para tal necessita de consumir os serviços do ElasticSearch (para fins de consulta de informação) bem como os serviços dos componentes Gmail e TextLocal que oferecem funcionalidades de envio de e-mail e SMS respetivamente. Este componente não disponibiliza serviços para o exterior e é configurado através da linha de comandos ou ficheiros de configuração;
4. **Gestor de Utilizadores**: Trata-se de um componente responsável por disponibilizar uma API RESTful capaz de gerir e manter os utilizadores do sistema quer estes sejam colaboradores ou administradores. As funcionalidades disponibilizadas pelo componente são consumidas diretamente pelo Kibana.

Por outro lado os componentes externos constituem todas as aplicações desenvolvidas por terceiros, ou seja são componentes que não foram desenvolvidos ou configurados no contexto deste projeto e encontram-se abaixo enumerados:

1. **Web Browser**: Este componente, que já existe fora do âmbito deste projeto, representa um cliente HTTP que consome as funcionalidades de visualização de dados provenientes do Kibana;
2. **Online Boutique**: Representa os microsserviços do projeto que é alvo de monitorização e consome os serviços de indexação de dados do ElasticSearch, ou seja, envia os *logs* para o dito componente.
3. **Gmail**: Representa um componente que expõe para o exterior funcionalidades referentes a envio de e-mails;
4. **TextLocal**: Este componente disponibiliza para o exterior serviços de envio de SMS's.

O uso desta abordagem implica alterações aos microsserviços do Online Boutique. Isto deve-se ao facto do componente Online Boutique ser responsável por indexar a informação dos *logs* no ElasticSearch. Outro aspeto a considerar é que os componentes Kibana e ElasticSearch já possuem mecanismos de notificações, autenticação e autorização que são facilmente parametrizáveis. Os ditos mecanismos são abordados e minuciados em maior extensão no capítulo de Implementação. Deste modo não se justifica implementar os componentes Gestor de Utilizadores e Gestor de Notificações, quando a própria ELK *stack* satisfaz na totalidade com os requisitos **R.9**, **R.21** e **R.24**.

Outro fator que inviabiliza o uso desta alternativa resulta do **R.1**, este requisito especifica que o sistema deve de afetar o mínimo possível as aplicações já existentes, ao implementar-se esta alternativa infringe-se em grande parte o dito requisito. Por estas razões o autor decidiu descartar a possibilidade de implementar o sistema com base nesta alternativa.

Alternativa B

O diagrama de componentes ilustrado na figura 4.2 destaca os componentes que fazem parte desta alternativa.

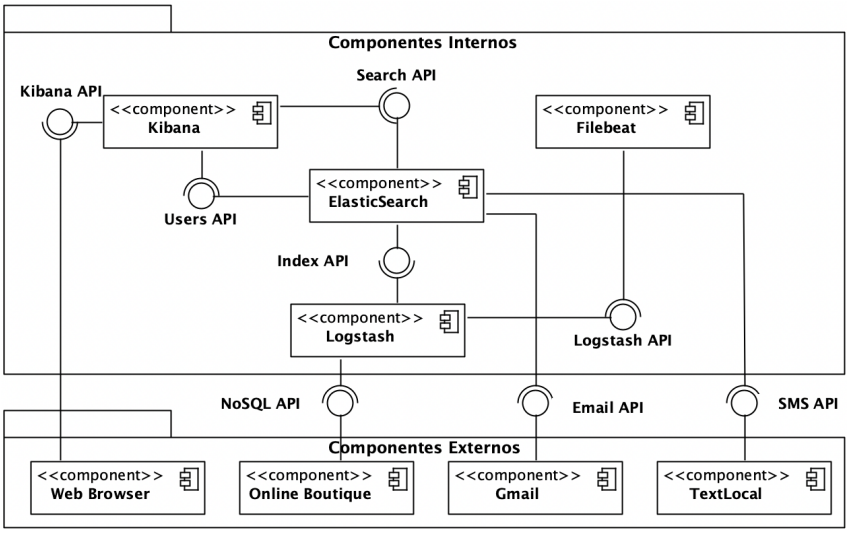

Figura 4.2: Diagrama de componentes da alternativa B

Para esta alternativa eliminaram-se os componentes Gestor de Utilizadores e Gestor de Notificações. Também é possível observar a existência de dois novo componentes, nomeadamente o Logstash e o Filebeat. De seguida são explicadas as responsabilidades dos novos componentes e daqueles que sofreram alterações:

1. **Logstash**: Este novo componente é responsável por agrupar, tratar e enviar os *logs* armazenados nas base de dados não relacionais do Online Boutique, para o ElasticSearch (através da API de indexação);
2. **Filebeat**: Este novo componente é responsável por agrupar e enviar os *logs* armazenados nos ficheiros de texto da máquina do Online Boutique para o Logstash;
3. **Online Boutique**: Nesta alternativa este componente deixa de desempenhar um papel pró-ativo no que concerne o envio de *logs* para o ElasticSearch. Mais especificamente, este componente representa os mecanismos de persistência onde os *logs* do Online Boutique encontram-se depositados;
4. **ElasticSearch**: Além das funções referenciadas na alternativa A, este componente agora disponibiliza serviços de gestão de utilizadores, nomeadamente operações *CRUD* sobre os utilizadores do sistema. Também passa a comunicar de modo direto com os serviços de envio de e-mail e SMS;

5. **Kibana**: Uma vez que nesta alternativa deixa de existir o componente Gestor de Utilizador, o Kibana passa a consumir as funcionalidades de gestão de utilizadores a partir do componente ElasticSearch.

Nota-se que os componentes que não foram mencionados na enumeração anterior mantém as mesmas características e desempenham as mesmas funções conforme descrito na alternativa A.

Para a presente alternativa o autor preocupou-se essencialmente em eliminar a dependência dos microsserviços do Online Boutique relativamente ao envio de *logs*. Com recurso aos novos componentes Logstash e Filebeat, poucas ou nenhumas alterações são necessárias de se efetuar às aplicações já existentes. Deste modo, elimina-se o problema inicialmente detetado na alternativa A e reduz-se o número de componentes necessários de implementar, o que por sua vez promove uma diminuição da complexidade do sistema sem comprometer os requisitos inicialmente estabelecidos.

Alternativa C

Ponderou-se também uma terceira alternativa, que de certa forma complementa a arquitetura retratada na figura 4.2 da alternativa B. A alternativa C encontra-se evidenciada no diagrama da figura 4.3.

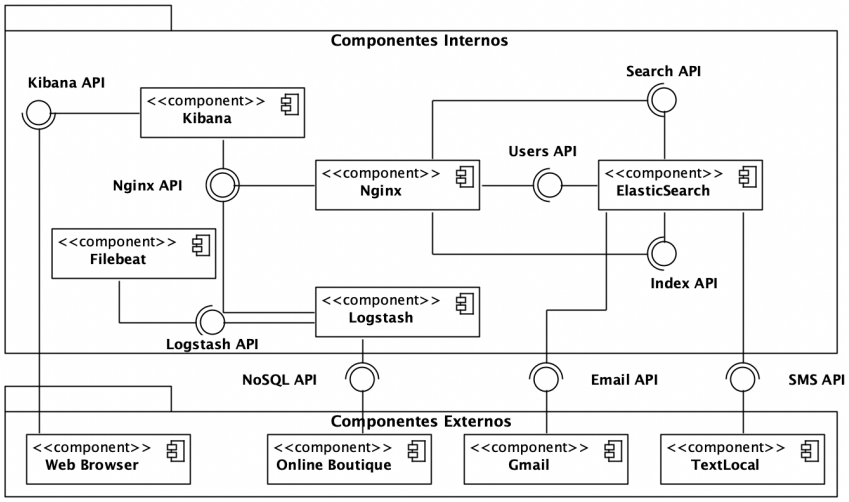

Figura 4.3: Diagrama de componentes da alternativa C

Esta alternativa adiciona o componente Nginx aos componentes presentes na alternativa anterior. A responsabilidade do novo componente pode ser descrita da seguinte maneira:

1. **Nginx**: Este componente tem duas grandes responsabilidades, sendo uma delas efetuar o balanceamento de carga (*load balancer*) e a restante remete para o reencaminhamento dos pedidos (*reverse proxy*). Desta forma os componentes Kibana e Logstash passam a consumir os serviços do Nginx, ficando este responsável por utilizar os serviços do ElasticSearch.

Esta alternativa à primeira vista parecer ser mais robusta que a alternativa B, contudo a tecnologia ElasticSearch já é provida de capacidades de balanceamento de carga (e.g *sharding* ver secção 2.4). Outro aspecto a considerar é o facto que as tecnologias Kibana e Logstash foram desenhadas para trabalhar exclusivamente com a tecnologia ElasticSearch.

Dadas essas razões, optou-se por não utilizar esta alternativa uma vez que se ganham poucas ou nenhumas vantagens ao utilizar-se o componente Nginx. Na verdade, até se pode considerar que o uso deste componente aumenta desnecessariamente o nível de complexidade do sistema.

No final e tendo como fundamento a análise efetuada nesta secção, escolheu-se implementar a solução com base na arquitetura representada na **alternativa B**.

4.1.2 Vista Física

Esta vista permite examinar ao detalhe o ambiente dentro do qual o sistema é executado. Também descreve a estrutura dos componentes de hardware que executam o software. Realça-se que conforme referido na secção 2.4, todos os componentes associados à solução devem ser executados num ambiente de conteinerização com recurso à tecnologia Docker.

Como previamente selecionou-se a **alternativa B** da vista de desenvolvimento, os diagramas de implantação denotados nesta subsecção expõem topologias do sistema que envolvem os componentes internos da dita alternativa.

Alternativa A

A figura 4.4 descreve o diagrama de implantação preconizado nesta alternativa.

Antes de se partir para uma análise aprofundada do diagrama, é importante salientar as seguintes características:

- Todos os componentes associados à solução encontram-se representados com o estereótipo *«artifact»*;
- Os nós assinalados com o estereótipo *«executionEnvironment»* representam o ambiente de execução dos componentes;
- Os nós retratados com o estereótipo *«device»* retraram as máquinas físicas de hardware que são responsáveis por hospedar os ambientes de execução.

Posto isto, ao observar-se o diagrama de implantação desta alternativa rapidamente se deteta que todos os componentes da solução situam-se alojados num único servidor físico, nomeadamente o nó **Main Server** (nomenclatura atribuída de forma livre pelo autor).

De maneira a tirar proveito das propriedades de balanceamento de carga do ElasticSearch, foram implantados dois *containers* adicionais que executam o componente ElasticSearch. Esta característica leva a um total de três instâncias do ElasticSearch a serem executadas em simultâneo.

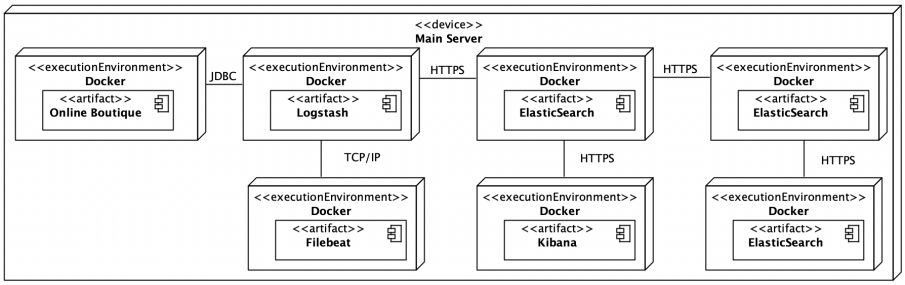

Figura 4.4: Diagrama de implantação da alternativa A

Acerca dos protocolos de comunicação e com base na figura 4.4 os *containers* que alojam o componente ElasticSearch e Kibana interagem unicamente através do protocolo HTTPS. Por outro lado, a comunicação do Logstash com o Online Boutique é efetuada por meio do protocolo JDBC (*driver* de conecção com bases de dados incluída com o Logstash). O *container* que hospeda o Filebeat por sua vez utiliza o protocolo TCP/IP para comunicar com o Logstash.

Por último, é necessário enfatizar que apesar do Logstash comunicar unicamente com uma das instâncias do ElasticSearch, todas estas vão receber e armazenar dados. Isto ocorre pois o ElasticSearch permite a definição de um "nó mestre"que está encarregue de distribuir trabalho pelos restantes nós, também conhecidos como "nós trabalhadores". Desta forma o Logstash apenas necessita de conhecer e comunicar uma instância do ElasticSearch, o que torna o mecanismo de distribuição de trabalho completamente transparente na perspetiva do componente Logstash.

Esta abordagem apresenta como grande vantagem a redução dos custos de manutenção do hardware, dado que existe apenas uma máquina física em causa (que inclui o sistema a monitorizar).

Todavia, no que toca a questões de segurança esta alternativa apresenta um falha notória. Se por algum motivo a segurança da máquina for comprometida (e.g ataque informático), todos os componentes que lhe estão associados são também afetados. Para agravar este cenário, a presente alternativa não é escalável pois todos os componentes do sistema estão associados a uma só máquina física.

Alternativa B

O diagrama da figura 4.5 apresenta a topologia proposta pela alternativa B.

Como se pode observar os protocolos de comunicação mantêm-se inalterados, porém os componentes do sistema encontram-se agora alojados em máquinas físicas individuais, desta forma a presente alternativa consegue amortecer as consequências de um eventual ataque informático a um nó físico (*«device»*).

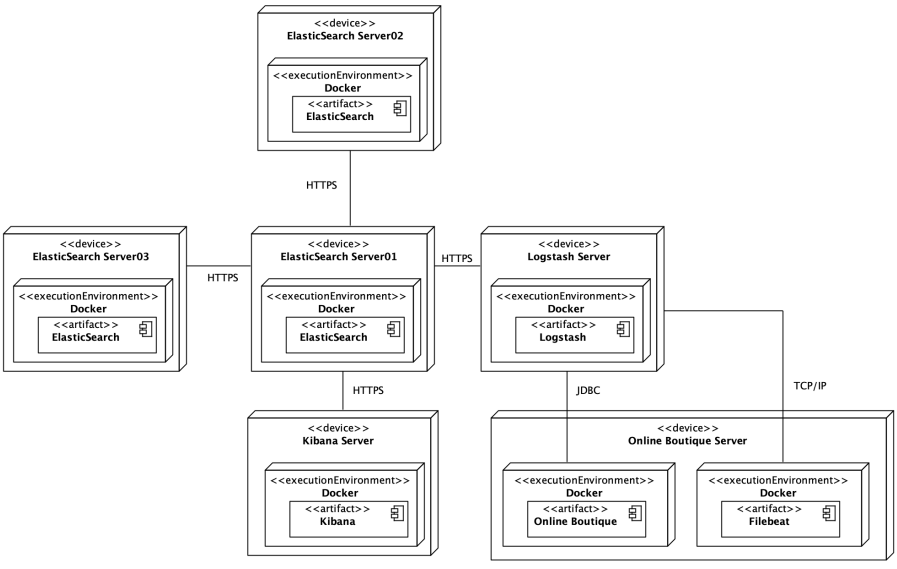

Figura 4.5: Diagrama de implantação da alternativa B

Além disso também permite escalar cada um dos componentes de um modo independente. Se for necessário escalar apenas um dos componentes (e.g um determinado componente recebe um número de pedidos superior aos restantes), consegue-se facilmente adicionar um novo *«device»* sem a necessidade de replicar os serviços cujo tráfego ainda é comparativamente menor.

Reforça-se que nesta alternativa o componente Filebeat tem que obrigatoriamente estar alojado na mesma máquina física do sistema Online Boutique. Isto surge devido ao facto do Filebeat não conseguir aceder a ficheiros remotos, somente a ficheiros da máquina onde o próprio Filebeat está a ser executado. Este ponto apenas foi levantado neste cenário pois na alternativa A só uma máquina física é colocada em uso e como consequência tanto o Filebeat e o Online Boutique partilham o mesmo sistema de ficheiros.

No entanto, esta abordagem implica um maior custo no processo de aquisição e manutenção tantos das máquinas físicas como do sistema em si. Por esta razão e pelo facto do presente projeto ser meramente académico, o autor optou implantar a solução com base na topologia da **alternativa A**.

Ainda assim, reconhece-se que a alternativa B representaria um cenário mais vantajoso se o custo monetário desta alternativa não fosse um fator impeditivo.

4.1.3 Vista de Processos

A vista de processos, como o próprio nome indica, é direcionada aos processos do sistema. Utiliza-se esta vista para ilustrar o comportamento do sistema em tempo de execução assim como para destacar os seus elementos dinâmicos.

Relativamente à presente vista e no que respeita os requisitos e os casos de uso da solução a desenvolver, o autor decidiu enfatizar o requisito **R.4**, **R.9**, o caso de uso **Consultar logs** e **Configurar sistema**, dado que estas funcionalidades são as que apresentam um maior impacto no sistema em termos de negócio quando comparadas às restantes. Por este motivo todas as decisões que afetem as ditas funcionalidades, devem-se encontrar devidamente explicadas e fundamentadas.

Note-se que os componentes demonstrados nesta subsecção derivam da alternativa B da vista de desenvolvimento, no entanto para a presente vista e para a alternativa A adicionou-se um novo componente (Message Queue) de forma a ilustrar como seria o comportamento do sistema num cenário onde se utilizasse filas de mensagens assíncronas.

Antes de se proceder a uma análise detalhada das funcionalidades supracitadas, o autor achou relevante efetuar uma divisão lógica das mesmas. Enquanto que os casos de uso **Consultar logs** e **Configurar sistema** encontram-se intrinsecamente ligados (relacionam-se com a interface gráfica do utilizador final), os requisito **R.4** e **R.9** focam-se num aspecto funcional do sistema que é completamente transparente ao utilizador final, que no caso deste projeto refere-se ao colaborador da organização ou ao próprio administrador.

Com isto em mente e fazendo uma analogia aos termos *back-end*[1] e *front-end*[2], categorizou-se os requisitos **R.4** e **R.9** como funcionalidades que se inserem na vertente de *back-end* e os casos de uso **Consultar logs** e **Configurar Sistema** como funcionalidades que se inserem na vertente de *front-end*.

[1] Refere-se a qualquer parte de um website ou programa de software que os utilizados não veem [107]
[2] Refere-se à interface do utilizador de um programa ou website [107]

Por último e antes de se proceder a uma análise detalhada dos componentes pertencentes ao *back-end* e ao *front-end*, é importante realçar que os diagramas de sequência expostos nesta subsecção apresentam uma granularidade grossa (também conhecida como granularidade de sistema). Isto acontece pois os componentes que fazem parte do sistema são todos implementações *open-source* de terceiros. Com isto em mente, o autor considerou mais importante enfatizar o modo como os componentes interagem entre si do que a forma de como foram implementados (também denominada de granularidade fina ou de aplicação).

Back-end

Segundo indica o requisito **R.4**, o sistema deve ser capaz de agrupar e processar *logs* provenientes de diferentes fontes e formatos. Com base na afirmação anterior, é possível identificar duas fases subjacentes a esta funcionalidade. A primeira fase preocupa-se com o **agrupamento** da informação sendo que a segunda concerne ao **processamento** dela.

No que toca à questão do **agrupamento** da informação o autor ponderou duas alternativas.

Alternativa A

A primeira alternativa, denominada de alternativa A, envolve implementar a solução com base nas características de sistemas de filas de mensagens assíncronas.

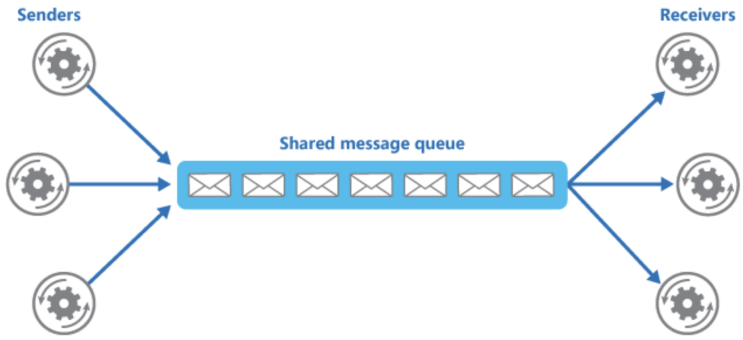

Figura 4.6: Fila partilhada de mensagens assíncronas

Fonte: [108]

Conforme se pode visualizar na figura 4.6, este tipo de sistema consiste no envio de mensagens para uma fila partilhada. Relativamente às mensagens recebidas, a fila tem a responsabilidade de as reencaminhar para os respetivos destinatários. No contexto do projeto em causa e segundo o diagrama de sequência da figura 4.7, tanto os destinatários como os remetentes de mensagens são aplicações/componentes do sistema.

Com base no diagrama supramencionado, é possível constatar que a primeira etapa do agrupamento da informação, implica subscrever o componente Logstash na fila de mensagens partilhada (denominada de *Message Queue* no diagrama). O próximo passo, denotado com o prefixo 2 no diagrama, consiste em processar pedidos do cliente (que não necessitam obrigatoriamente de ser operações POST). No final do processamento de cada pedido e após enviar *feedback* para o utilizador, os microsserviços do Online Boutique de forma **assíncrona** publicam mensagens para a fila de mensagens partilhada.

Estas mensagens além de outros dados, contêm informação relativa ao pedido recebido pelo microsserviço e à consequente resposta gerada pelo mesmo, isto significa que o conteúdo das mensagens corresponde efetivamente aos *logs* do microsserviço.

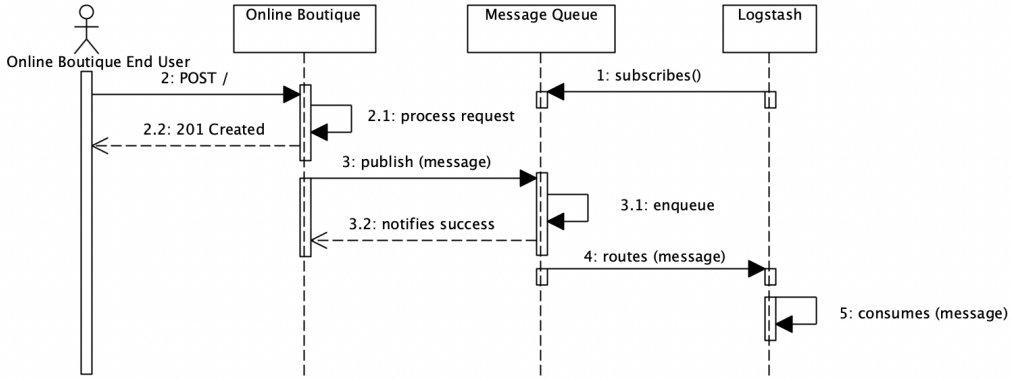

Figura 4.7: Diagrama de sequência relativo ao agrupamento da informação da alternativa A

Uma vez publicada a mensagem e colocada na fila com sucesso, o próximo passo traduz-se no roteamento da mensagem para o respetivo destinatário, que no caso em concreto coincide com o componente Logstash. Por último, assim que o componente Logstash recebe a mensagem, ele consome-a diretamente.

Um aspeto realçado pelo uso desta alternativa é a remoção do componente Filebeat, pois visto que o Online Boutique publica os *logs* diretamente na *Message Queue* elimina-se por completo a necessidade de ler os *logs* a partir de ficheiros de texto.

A abordagem recomendada pela presente alternativa evidencia uma grande vantagem, nomeadamente o **baixo acoplamento** entre os diferentes componentes. Os microsserviços do Online Boutique não sabem da existência do Logstash, nem dependem diretamente dele e vice-versa. Contudo o uso desta alternativa resulta na existência de um novo componente (Message Queue) e na alteração de componentes existentes (microsserviços do Online Boutique), trata-se de um *trade-off* onde as desvantagens (maior complexidade, alterações a componentes existentes) excedem as vantagens.

Alternativa B

A alternativa que se segue (alternativa B) visa colmatar as desvantagens associadas à alternativa A. A presente alternativa em vez de se focar em filas de mensagens assíncronas, opta por implementar o agrupamento da informação com base em mecanismos periódico de *polling* e de espera passiva. A figura 4.8 retrata o mecanismo de *polling*.

Figura 4.8: *Polling* periódico

Fonte: [109]

De um ponto de vista redutor, este mecanismo equivale a realizar pedidos por parte de uma aplicação cliente a um determinado servidor. Estes pedidos são efetuados rotineiramente (e.g cada 30 segundos) e é com base na resposta do servidor que a aplicação cliente realiza uma determinada ação.

O diagrama de sequência da figura 4.9, ilustra o mecanismo de *polling* periódico e de espera passiva aplicado ao projeto em mão.

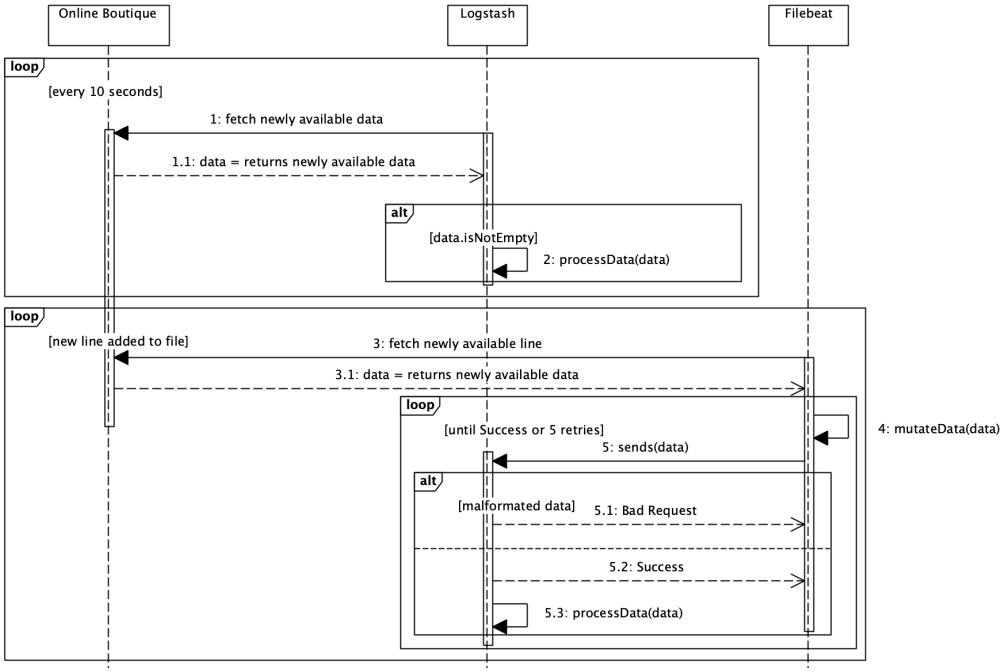

Figura 4.9: Diagrama de sequência relativo ao agrupamento da informação da alternativa B

A cada 10 segundos, o componente Logstash é responsável por verificar se existe nova informação nas base de dados dos microsserviços do Online Boutique. Caso exista informação disponível esta é processada e tratada pelo Logstash. Por outro lado, o Filebeat tira partido de mecanismos de espera passiva pois fica à "escuta"de alterações aos ficheiros de *logs*, nomeadamente adição de novas linhas. Deste modo, sempre que uma linha é adicionada ao ficheiro, o Filebeat obtém-na, altera-a (adiciona um campo cujo valor equivale ao nome do microsserviço em causa) e envia-a diretamente para o Logstash até obter uma resposta de sucesso ou após cinco tentativas.

Optou-se por utilizar esta alternativa pois além de apresentar uma implementação menos complexa, também remove a necessidade de alterar e adicionar componentes ao sistema.

Posto isto, o diagrama de sequência da figura 4.10 descreve o **agrupamento** da informação juntamento com o respetivo **processamento** dela.

Como analisado anteriormente, sempre que existe informação disponível o Logstash é responsável por tratá-la. Inicialmente efetua-se o mapeamento da informação recebida de modo a que esteja em conformidade com a estrutura de dados esperada pela API do ElasticSearch. Uma vez mapeada a informação, a próxima etapa remete para a adição de novos campos (e.g origem da informação, hora de receção da informação, etc.), estando a informação devidamente processada o último passo desempenhado pelo Logstash consiste no envio da

informação para a camada de persistência, que neste caso equivale à utilização da API de indexação do ElasticSearch de modo a armazenar a informação previamente tratada.

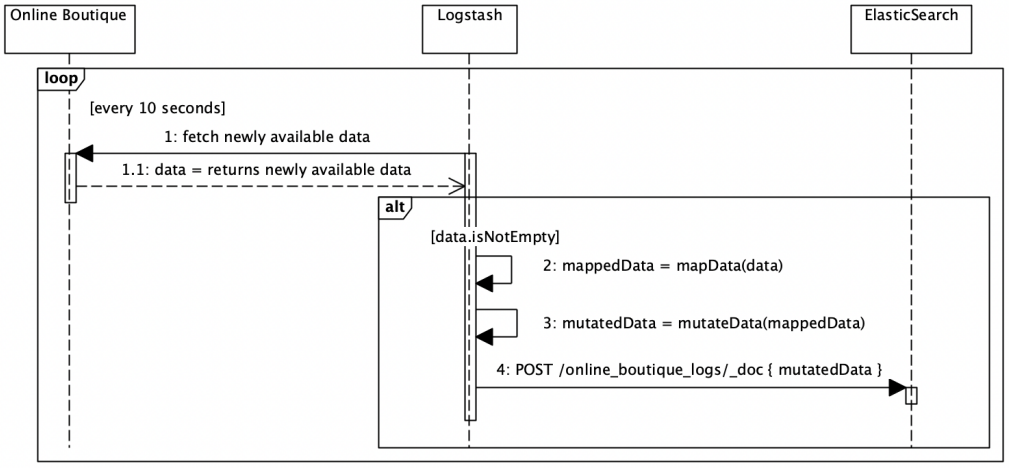

Figura 4.10: Diagrama de sequência relativo ao agrupamento e processamento da informação

Apesar de não se encontrar explícito no diagrama acima, a mesma lógica de processamento de informação é também aplicada quando esta é obtida a partir do Filebeat.

Para finalizar a dimensão do *back-end*, o diagrama retratado na figura 4.11 descreve as interações que dão origem ao envio de notificações de anomalias aos colaboradores da organização (**R.9**).

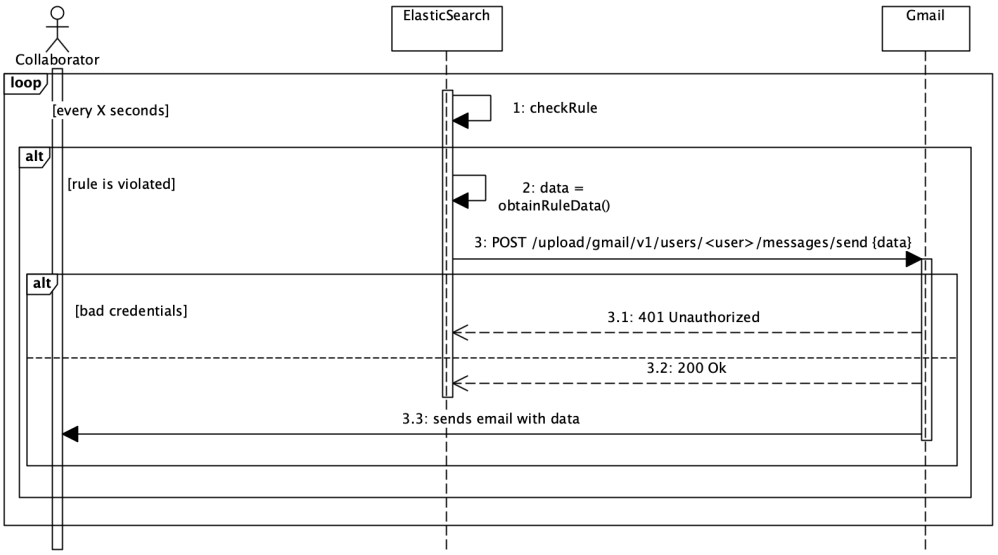

Figura 4.11: Diagrama de sequência relativo às notificações de anomalias

Com base no diagrama referido, é possível constatar que o componente ElasticSearch utiliza um mecanismo de *polling* para verificar periodicamente se as regras que constam nos alertas não são violadas. É importante realçar que o intervalo de tempo de espera que existe entre

as execuções do dito mecanismo de *polling* varia consoante a regra em causa (diferentes regras podem ter diferentes intervalos de tempo).

Se porventura a regra definida previamente no alerta for infringida, então o ElasticSearch utiliza os serviços do componente Gmail para notificar via e-mail os colaboradores da organização.

Front-end

Conforme denota o **R.6** e o caso de uso **Consultar logs**, o sistema deve disponibilizar um meio para analisar e visualizar a informação dos *logs* na web. Além disso, o colaborador autenticado deve ser capaz de consultar a dita informação e manipulá-la segundo os seguintes critérios:

- Filtrar os resultados da consulta com base em texto introduzido pelo colaborador;
- Filtrar os resultados da consulta com base em múltiplos filtros;
- Ordenar os resultados da consulta por grau de importância dos *logs*.

Deste modo identificou-se duas ações que necessitam ser analisadas em maior detalhe, nomeadamente a **consulta** e a **manipulação** da informação proveniente dos *logs*.

O diagrama de sequência retratado na figura 4.12, pormenoriza o fluxo das ações que ocorrem no sistema quando o colaborador autenticado acede à interface gráfica de visualização de *logs* com o objetivo de **consultar** a sua informação.

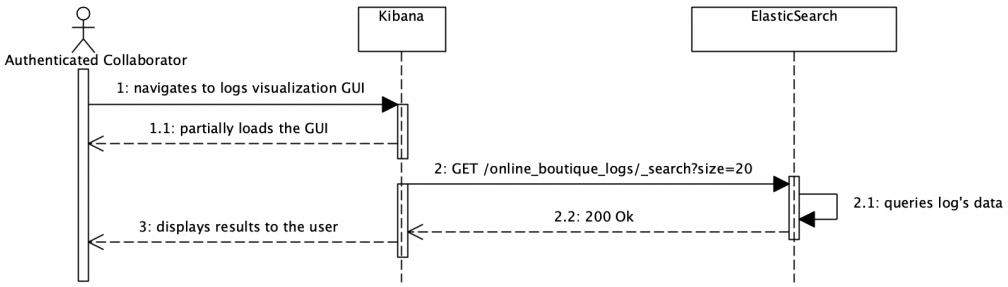

Figura 4.12: Diagrama de sequência relativo à consulta da informação dos *logs*

Inicialmente o colaborador acede à interface gráfica responsável por expor a informação dos *logs* (esta interface é representada pelo componente Kibana). O Kibana imediatamente devolve uma página web parcialmente carregada (pois ainda não tem a informação dos logs). Isto é um mecanismo crucial, pois permite ao colaborador interagir com a interface gráfica sem que esta fique bloqueada à espera da informação dos *logs*.

Logo após e de modo assíncrono o Kibana efetua um pedido GET à API RESTful do ElasticSearch no intuito de obter a informação dos *logs*. Logo que o ElasticSearch recebe o pedido, internamente efetua uma *query* a fim de obter os *logs*. Finalmente quando o ElasticSearch adquire os resultados da *query* retorna-os para o Kibana, por sua vez o Kibana insere a informação recebida na página previamente carregada.

O diagrama de sequência da figura 4.13 vem complementar o diagrama anterior. Além de evidenciar as interações entre os componentes do sistema aquando da **consulta** da informação, também expõe o processo de **manipulação** da informação dos *logs*.

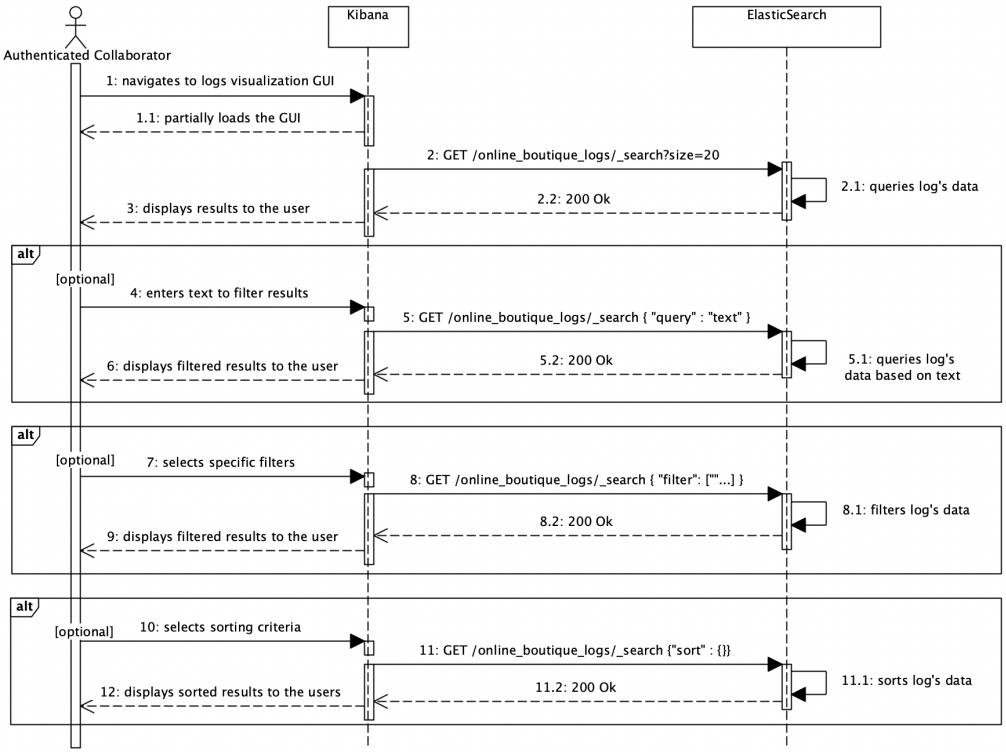

Figura 4.13: Diagrama de sequência relativo à manipulação da informação dos *logs*

Quando o Kibana disponibiliza os resultados para o colaborador (etapa nº 3 do diagrama), este tem agora a possibilidade de filtrar a informação consoante determinados critérios definidos pelo próprio.

O colaborador pode simultaneamente optar pelos três fluxos opcionais (etapa nº 4, 7 e 10 do diagrama) ou pode somente optar por um deles ou inclusive nenhum. Independentemente da escolha do colaborador, os três fluxos opcionais são praticamente idênticos. Para cada um deles, o Kibana efetua um pedido **assíncrono** à API do ElasticSearch passando no corpo do pedido os parâmetros introduzidos pelo colaborador. No caso de se introduzir texto, o parâmetro utilizado tem o nome de *query*, se forem selecionados filtros então o parâmetro passa a ter o nome de *filter*, por fim se o colaborador optar por ordenar os resultados então o parâmetro assume o nome de *sort*. Uma vez recebida a resposta do ElasticSearch o Kibana exibe os resultados na interface gráfica de maneira a serem visualizados pelo colaborador.

Em relação ao caso de uso **Configurar sistema** existem duas ações que podem ser realizadas, sendo uma delas a configuração de alertas. A figura 4.14 expõe as etapas que levam ao ato de persistir um novo alerta no sistema.

Estando o administrador autenticado ele navega até à interface gráfica de configuração de alertas. O primeiro passo do Kibana consiste em obter os alertas armazenados no ElasticSearch (passo 1.1 do diagrama). O ElasticSearch ao detetar que recebeu um pedido de

obtenção de alertas, começa por validar o *token* de acesso. No caso deste ser válido, o Elastic devolve os alertas para o Kibana que por sua vez apresenta-os para o administrador.

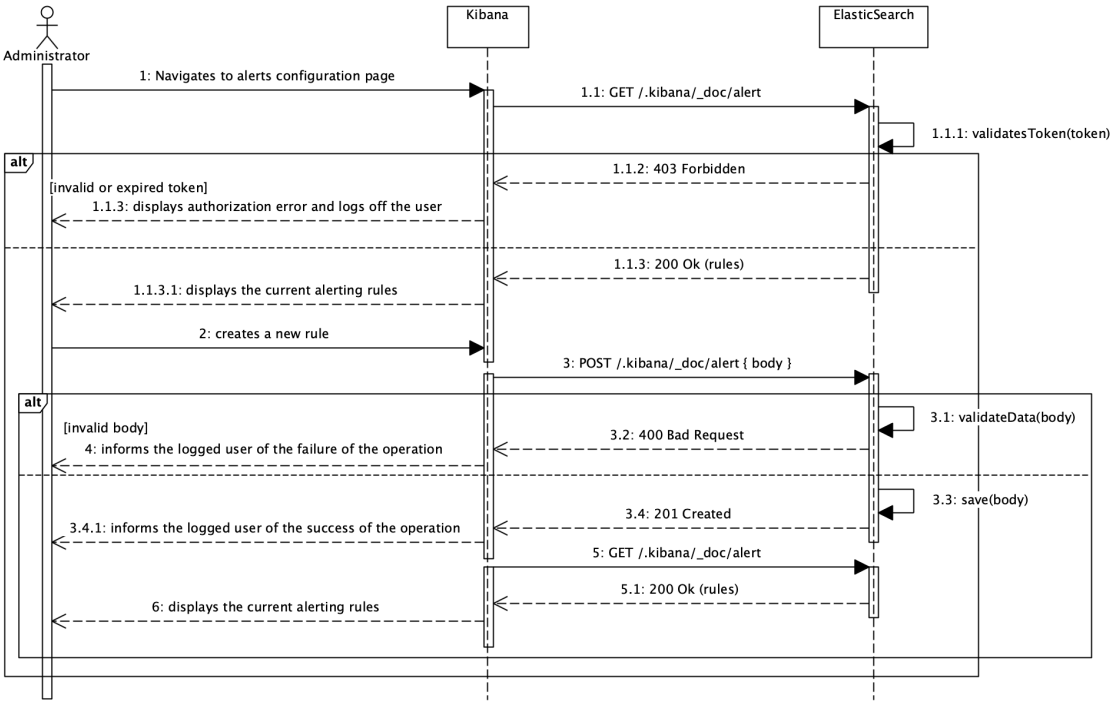

Figura 4.14: Diagrama de sequência relativo à configuração de alertas

Seguidamente e já com a interface da configuração dos alertas completamente carregada, o administrador introduz os dados necessários para a criação de um novo alerta (periodicidade, critérios para detetar anomalia, ações a efetuar no caso de anomalia, etc.). O Kibana de modo assíncrono efetua um pedido POST à API do ElasticSearch com os dados introduzidos pelo administrador. Se os dados forem válidos, o Elastic encarrega-se de persistir o novo alerta e de informar o Kibana o sucesso da operação.

Por último, o Kibana notifica o administrador que a operação foi bem sucedida e efetua um novo pedido ao Elastic de maneira a obter todos os alertas do sistema, que já inclui o alerta criado pelo administrador, apresentado-os na interface gráfica uma última vez.

Capítulo 5

Implementação da Solução

Finalizado o desenho da solução, onde resultou uma arquitetura concreta e passível de se aplicar ao projeto em mão, a próxima etapa remete para a implementação da dita arquitetura. Numa fase preliminar, este capítulo aborda a metodologia de desenvolvimento aplicada ao projeto. Logo após, descreve-se o processo de integração e entrega contínua empregue ao longo do desenvolvimento. Por último, este capítulo é finalizado com as peculiaridades mais relevantes associadas à implementação da solução. No decorrer do capítulo são evidenciados excertos de código de modo a corroborar não só a implementação da solução como também o desenho previamente elaborado.

5.1 Metodologia de Desenvolvimento

Uma metodologia de desenvolvimento é uma forma de gerir um projeto de software. Tipicamente aborda questões como a seleção de funcionalidades a incluir na versão atual do software, quando este deve ser lançado, que testes devem ser feitos, entre outros fatores. Uma escolha adequada de metodologia de desenvolvimento pode impactar positivamente o resultado final no que diz respeito a cumprimento de prazos, custos monetários e robustez do software [110].

A metodologia de desenvolvimento adotada ao projeto em causa denomina-se de **Feature-Driven Development** (FDD). O FDD caracteriza-se por ser um processo de desenvolvimento iterativo que atribui um maior ênfase no planeamento da arquitetura global, seguido pela implementação das funcionalidades numa ordem lógica [110]. Este processo faz parte da família de metodologias **Agile** que apesar de serem orientadas a pequenas equipas e a interação contínua com o cliente, condições que no contexto deste projeto são inexistentes, é possível tirar proveito das restantes características tais como o desenvolvimento incremental de software e o forte foco nos seus testes.

Numa vertente mais prática e conforme justificado na subsecção 2.4.1 o autor optou por utilizar o Git como tecnologia de versionamento simultaneamente com a tecnologia GitLab como plataforma de hospedagem Git. De forma a promover modularização **criou-se um repositório por componente do sistema**. Isto significa que os componentes Kibana, ElasticSearch, Logstash e Filebeat têm o seu próprio repositório que são inteiramente independentes entre si.

Adicionalmente, existe um quinto repositório denominado de *Online-Boutique-Log-Management-System* que contém ficheiros de configuração das imagens *Docker* dos repositórios supramencionados. Do mesmo modo, este repositório encarrega-se de orquestrar e efetuar o

deploy dos componentes do sistema numa máquina remota, que no caso deste projeto compete à máquina pessoal do autor.

Cada um destes repositórios segue a estratégia de **feature branching**. A figura 5.1 demonstra de uma forma concisa a estratégia em causa.

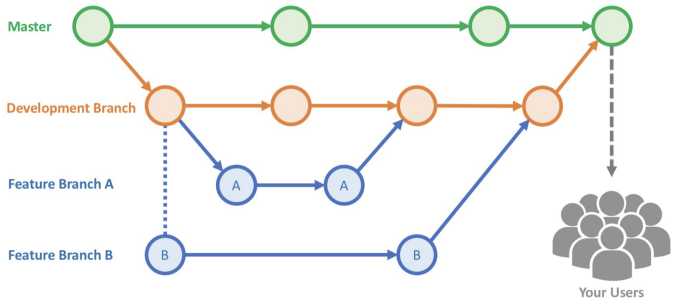

Figura 5.1: Estratégia de *feature branching*

Fonte: [111]

Nesta abordagem as funcionalidades do sistema são desenvolvidas em ramos isolados do ramo principal. Apenas quando a funcionalidade se encontra desenvolvida na sua totalidade é que é integrada no ramo de desenvolvimento e eventualmente no ramo principal. Esta estratégia além de facilitar o processo de *Code Review*, aumenta o nível de flexibilidade das *pipelines* de CI/CD.

Quanto ao processo de colocar novo código ou alterações no repositório, adotou-se uma ferramenta automática de versionamento de software que tem por base o conteúdo das mensagens de *commit*. Esta ferramenta tem o nome de **semantic release** e mediante a mensagem de *commit* consegue distinguir o impacto associado a cada commit (nova funcionalidade, testes, documentação, etc.) e incrementar a versão do software se for esse o caso.

Quanto aos ambientes nos quais se vai implantar o sistema, resolveu-se tirar proveito de uma prática que consiste em utilizar dois ambientes de implantação, nomeadamente os ambientes de **preparação** e de **produção**. Sempre que se efetuam commits ou *merge-requests* em ramos diferentes do "main", os componentes do sistema são publicados, implantados e testados no ambiente de **preparação** caso contrário utiliza-se o ambiente de **produção**.

Por último e não menos importante, o autor decidiu utilizar os *issues* do GitLab (ver anexo A.1) de maneira a gerir o tempo alocado a cada tarefa, bem como na priorização e categorização dessas mesmas tarefas. Em suma, a utilização da metodologia referida nesta secção torna o desenvolvimento das funcionalidades do sistema um processo simples e padronizado.

5.2 Integração e Entrega Contínua

Os termos integração e entrega contínua representam práticas destinadas a ajudar organizações a acelerar o desenvolvimento de software e a entregar funcionalidades de software sem comprometer a qualidade do mesmo. Enquanto que a integração contínua defende a integração de trabalho em progresso várias vezes ao dia, a entrega contínua relaciona-se com a capacidade de gerar valor de forma rápida e fiável aos clientes, trazendo apoio de automatização o tanto quanto possível [112].

Estas práticas proporcionam benefícios que se encontram abaixo enumerados [112]:

- Obter *feedback* do processo de desenvolvimento de software em maior quantidade e com maior celeridade;
- Lançamentos frequentes e fiáveis que consequentemente levam a uma maior satisfação do cliente e da qualidade do produto;
- Através de entrega contínua, a ligação entre as equipas de desenvolvimento e de operações (DevOps) é reforçada e as tarefas manuais podem ser eliminadas.

Por estas razões escolheu-se aplicar as práticas de integração e entrega contínua ao projeto em causa. Cada repositório do GitLab é composto por uma *pipeline* que implementa tarefas automatizadas tais como a execução de testes e implantação dos componentes na nuvem, que por sua vez são definidas no ficheiro **.gitlab-ci.yml** de cada repositório. Note-se que o dito ficheiro é escrito em formato YAML que de grosso modo é uma linguagem de serialização de dados frequentemente aplicada a ficheiros de configuração [113].

Um aspeto comum a todas as *pipelines* implementadas diz respeito à identificação do ambiente de implantação do sistema. Este mecanismo de diferenciação de ambientes encontra-se retratado no excerto 5.1.

```
1  variables:
2    TAG_NAME: "stable"
3
4  workflow:
5    rules:
6      - if: $CI_COMMIT_REF_NAME != "main"
7        variables:
8          TAG_NAME: "staging"
9      - when: always
```

Excerto de código 5.1: Definição e atribuição de uma variável global

Numa primeira fase começa-se por definir a variável TAG_NAME com o valor "stable"que visa representar o ambiente de produção. Logo após a *pipeline* inspeciona qual o ramo que deu origem à sua execução, se o ramo em causa diferir do ramo principal ("main") então o a variável TAG_NAME passa a assumir o valor de "staging"que representa o ambiente de preparação.

As variáveis que contêm dados sensíveis (credenciais, certificados, chaves privadas, etc.) são definidas à priori na interface gráfica do GitLab (ver figura 5.2). Estas variáveis podem ser utilizadas ao longo das etapas da *pipeline* através da utilização do símbolo **$** antes do nome da variável.

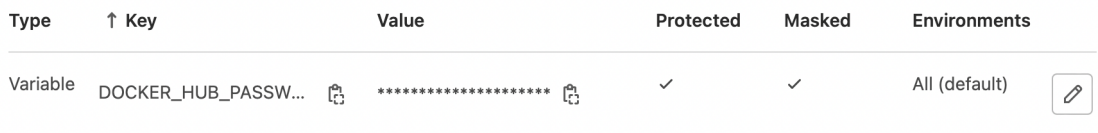

Figura 5.2: Definição de uma variável de ambiente no GitLab

5.2.1 Upstream Jobs

Upstream Jobs é um termo utilizado para designar *jobs* ou etapas que desencadeiam *pipelines* de outros projetos. É importante recordar este conceito, pois as *pipelines* dos componentes internos do sistema (Filebeat, Logstash, Elastic e Kibana) vão em determinada altura despoletar a execução da *pipeline* do repositório *Online-Boutique-Log-Management-System*.

As *pipelines* dos repositórios Filebeat, ElasticSearch, Kibana e Logstash seguem **todas** a sequência de etapas representadas na figura 5.3.

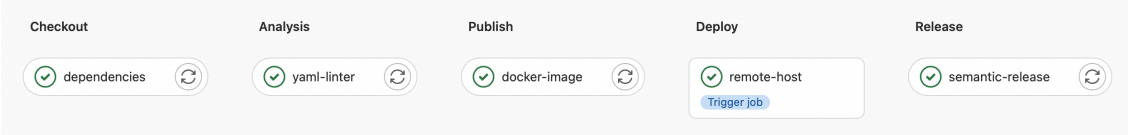

Figura 5.3: *Pipeline* dos componentes da solução

Checkout

A primeira fase intitulada de Checkout consiste em instalar as dependências associadas ao repositório. O excerto de código 5.2 expõe os comandos necessários para efetuar essa mesma ação.

```
1 dependencies:
2   image: node:alpine
3   stage: checkout
4   script:
5     - echo "Installing dependencies"
6     - npm install
```

Excerto de código 5.2: Definição da etapa de Checkout

Com recurso ao comando `npm install`, todas as dependências definidas no ficheiro **package.json** são devidamente instaladas, podendo estas ser utilizadas nas etapas seguintes.

Analysis

A fase que se segue, denominada de Analysis, tem como objetivo identificar anomalias/problemas nos ficheiros presentes no respetivo repositório conforme sugere o excerto de código 5.3.

```
1 yaml-linter:
2   image: node:alpine
3   stage: analysis
4   script:
5     - echo "Linting yaml files"
6     - yamllint ./
```

Excerto de código 5.3: Definição da etapa de Análise

O comando `yamllint ./` realiza uma análise aos ficheiros com a extensão .yml de modo a encontrar erros sintáticos ou problemas de indentação. No caso destes existirem, o yamllint imprime para o terminal do Gitlab a localização e a causa dos erros e termina de imediato a execução da *pipeline*.

Publish

A etapa de Publish constrói os artefactos do repositório e publica-os na nuvem. Os passos que estão subjacentes a este processo podem ser visualizados no excerto 5.4.

```
1 docker-image:
2   stage: publish
3   image: docker:20.10.16-alpine3.16
4   script:
5     - docker build -t $DOCKER_HUB_USERNAME/online-boutique-elasticsearch:$TAG_NAME .
6     - docker login -u $DOCKER_HUB_USERNAME -p $DOCKER_HUB_PASSWORD
7     - docker push $DOCKER_HUB_USERNAME/online-boutique-elasticsearch:$TAG_NAME
```

Excerto de código 5.4: Definição da etapa de Publicação

Inicia-se este processo por construir localmente a imagem do Docker *container* mediante o repositório em causa. O repositório do ElasticSearch é responsável por construir a imagem do ElasticSearch, o repositório do Kibana a imagem do Kibana e assim sucessivamente.

Construída a imagem a próxima etapa traduz-se na autenticação do utilizador no Docker Hub e na publicação da imagem nessa mesma plataforma que será mais tarde utilizada pelo sistema no momento da sua implantação.

Deploy

A fase de Deploy encarrega-se de iniciar a execução da *pipeline* do projeto *Online-Boutique-Log-Management-System*. O excerto 5.5 retrata a operação previamente mencionada.

```
1 remote-host:
2   variables:
3     ENVIRONMENT: $TAG_NAME
4   stage: deploy
5   trigger:
6     project: $GITLAB_USER/online-boutique-log-management-system
7     strategy: depend
```

Excerto de código 5.5: Definição da etapa de Implantação

Parte-se por definir uma variável que toma o valor de *staging* ou *stable*. Logo após e como último passo, despoleta-se a *pipeline* do projeto *Online-Boutique-Log-Management-System* que recebe como argumentos a variável definida inicialmente.

A *pipeline* que originalmente desencadeou este processo fica dependente do resultado da *pipeline* desencadeada. Caso esta por qualquer motivo termine abruptamente ou retorne um resposta de erro, então a *pipeline inicial* (*upstream*) cessa a sua execução. Este comportamento apenas é possível se a estratégia definida pela *pipeline upstream* for de dependência (instrução `strategy: depend` do excerto supracitado).

Release

Esta quinta e última fase da **pipeline** lida com as questões do *semantic-release* mencionadas no início do capítulo. Este processo encontra-se descrito no excerto de código 5.6.

Em primeiro lugar começa-se por efetuar uma instalação global do pacote *semantic-release*. Estando o dito pacote instalado, executa-se o comando `semantic-release` que de forma

automática analisa as mensagens de commit e com base nelas gera uma *tag* git, um ficheiro de alterações (changelog.md) e incrementa a versão do software.

```
semantic-release:
  image: node:latest
  stage: release
  before_script:
    - npm install -g semantic-release @semantic-release/gitlab
  script:
    - semantic-release
  only:
    - main
```

Excerto de código 5.6: Definição da etapa de *Release*

Uma vez realizadas as alterações, o *semantic-release* efetua um novo commit de forma a persistir essas mesmas alterações. Realça-se que esta fase é desempenhada somente quando se pretende lançar uma nova versão de software ou seja, apenas quando são colocadas alterações no ramo principal ("main").

5.2.2 Downstream Jobs

Os *downstream jobs* representam as fases inicializadas pelos *upstream jobs*. No contexto deste projeto tem-se as *pipelines* dos repositórios dos componentes internos como *upstream jobs* e o repositório *Online-Boutique-Log-Management-System* como *downstream job*.

Posto isto, a figura 5.4 explicita as fases da *pipeline* que estão relacionadas com o repositório *Online-Boutique-Log-Management-System*.

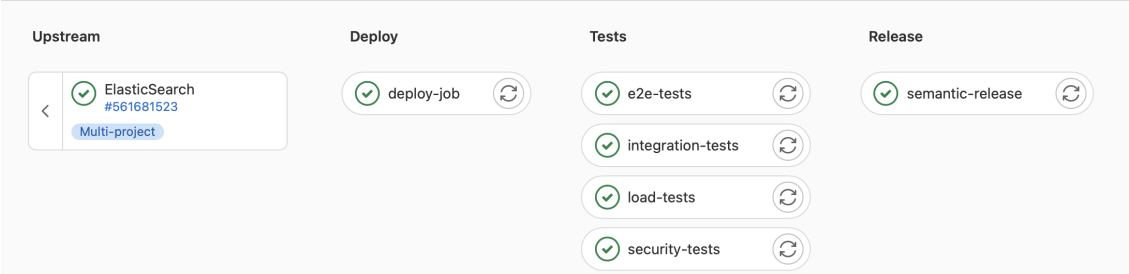

Figura 5.4: *Pipeline* do repositório Online-Boutique-Log-Management-System

A fase designada de Upstream pode ser ignorada pois não representa nenhuma ação executada pela *pipeline* do repositório em causa, apenas serve para indicar qual o projeto *upstream* que causou a execução da presente *pipeline*.

Deploy

A fase de Deploy é responsável por aceder à máquina remota do sistema de monitorização de *logs* do Online Boutique e substituir os componentes que sofreram alterações. Esta fase encontra-se exposta no excerto 5.7.

A fase em causa após configurar as chaves e as máquinas ssh que considera fidedignas, executa o ficheiro *deploy.sh* passando como argumentos as variáveis relacionadas com credencias e configurações de portas de acesso.

```
deploy-job:
  image: debian:latest
  stage: deploy
  before_script:
    # omitted ssh known hosts and ssh key creation

  script:
    - echo "Deploying application..."
    - chmod +x ./deploy.sh && ./deploy.sh $TAG $DOCKER_HUB_USER $HOST_KIBANA_PORT
      ↪ $HOST_ELASTIC_PORT
      $KIBANA_PASSWORD $ELASTIC_PASSWORD $LOGSTASH_PASSWORD
      ↪ $REMOTE_USER $REMOTE_HOST
    - echo "Application deployed"
```

Excerto de código 5.7: Definição da etapa de Implantação

O dito ficheiro, representado no anexo A.1 tem a responsabilidade de executar o seguintes passos de forma ordeira:

1. Substituir as variáveis de ambiente pelas variáveis configuradas no GitLab;
2. Desligar os *containers* que estão no momento a executar os componentes do sistema;
3. Substituir o docker-compose.yml e os ficheiros com a extensão .env pelos do repositório;
4. Iniciar os *containers* anteriormente desligados.

Tests

A etapa de testes contém todos os testes que se efetuam após a implantação do sistema (fase anterior). Também se pode considerar que esta fase realiza testes de regressão, pois averigua se as diversas funcionalidades do sistema encontram-se operacionais.

O excerto de código 5.8 demonstra a definição da tarefa que executa os testes de integração do sistema.

```
integration-tests:
  stage: tests
  image: node:latest
  before_script:
    - npm i
  script:
    - echo Running integration tests
    - node tests/integration/integration.js $REMOTE_HOST $HOST_ELASTIC_PORT
      ↪ $ELASTIC_PASSWORD
```

Excerto de código 5.8: Definição da tarefa que executa os testes de integração

Esta tarefa utiliza a imagem mais recente do NodeJS, instala as dependências definidas no ficheiro package.json e executa os testes de integração definidos no ficheiro integration.js.

A configuração da tarefa que executa os testes de carga do sistema encontra-se visível no excerto 5.9. A imagem utilizada pelo *container* responsável por efetuar os testes de carga corresponde sempre à versão mais recente do Python.

Antes de se iniciar os testes, instalam-se as dependências definidas no ficheiros de requisitos. Finalmente executam-se os testes com recurso à tecnologia Locust.

```
load-tests:
  stage: tests
  image: python:latest
  before_script:
    - pip install -r tests/load/requirements.txt
  script:
    - echo Running load tests
    - locust -f tests/load/load.py --headless -u 20 -r 20 --host http://$REMOTE_HOST:5100 --run-time
      ↪ 10s
      --elastic-host https://$REMOTE_HOST:$HOST_ELASTIC_PORT --elastic-user elastic
      ↪ --elastic-password $ELASTIC_PASSWORD
```

Excerto de código 5.9: Definição da tarefa que executa os testes de carga

No que diz respeito à tarefa que efetua os testes de segurança, esta encontra-se definida no excerto 5.10.

```
security-tests:
    stage: tests
    image: python:latest
    before_script:
        - pip install -r tests/security/requirements.txt
    script:
        - echo Running security tests
        - python3 -Wignore tests/security/tests.py $REMOTE_HOST $HOST_ELASTIC_PORT
```

Excerto de código 5.10: Definição da tarefa que executa os testes de segurança

Da mesma maneira que os testes de carga, os testes de segurança tirando partido da imagem mais recente do Python e instalam as dependências necessárias presentes no ficheiro de requisitos. A diferença é que esta tarefa executa os testes através do binário do Python (commando python3).

Por último os testes end-to-end são efetuados pela tarefa expressa no excerto de código 5.11.

```
e2e-tests:
  stage: tests
  image: cypress/base:latest
  before_script:
    - npm install cypress --location=global
  script:
    - echo Running end-to-end tests
    - cd tests/e2e && cypress run --headless --env
      ↪ HOST=$REMOTE_HOST,PORT=$HOST_KIBANA_PORT
```

Excerto de código 5.11: Definição da tarefa que executa os testes *end-to-end*

Esta tarefa começa por utilizar a imagem mais recente da tecnologia Cypress e por instalar o pacote Cypress no *container*, pois é este que possibilita-o de executar o comando "cypress". Uma vez descarregada a imagem e instalada as dependências, executam-se os testes *end-to-end* passando como argumento o endereço da máquina que aloja o Kibana bem como a sua respetiva porta.

Devido ao facto destas quatro definições partilharem a mesma fase (`stage: tests`), os testes abordados nesta subsecção são executados em paralelo, o que vem reduzir consideravelmente o tempo de execução da fase de testes e por consequência da *pipeline*.

Release

Esta fase é essencialmente igual à fase de Release da subsecção anterior (*upstream jobs*), a única diferença é que esta fase além de ser executada no ramo principal também a é quando recebe a variável ENVIRONMENT do *upstream job* com o valor de "stable".

5.3 Desenvolvimento da Solução

Esta secção tem como objetivo abordar e detalhar o processo de desenvolvimento da solução. Como tal, o processo de implementação dos requisito estabelecidos no capítulo de Análise é devidamente explicado e evidenciado através de excertos de código e de figuras ilustrativas das funcionalidades do sistema.

5.3.1 Tratamento da Informação

De maneira a satisfazer o requisito **R.4**, o sistema deve ser capaz de agrupar e processar a informação dos *logs* do Online Boutique. Para este fim e conforme denotado no capítulo de desenho utiliza-se o componente Logstash juntamente com o Filebeat.

Filebeat

O Filebeat como referido em capítulos anteriores, tem como função processar ficheiros de texto que contêm *logs* e enviá-los para o Logstash. Como tal, começou-se por criar uma imagem Docker deste mesmo componente, que pode ser visualizada no excerto de código 5.12.

```
1 FROM elastic/filebeat:8.2.2
2 COPY ./config/filebeat.yml  /usr/share/filebeat
```

Excerto de código 5.12: Configuração do Dockerfile do componente Filebeat

A imagem em questão começa por obter a versão oficial do Filebeat 8.2.2 e termina com a cópia do ficheiro de configuração local para o *container* responsável por executar a imagem. Este ficheiro de configuração encontra-se ilustrado no excerto 5.13.

Na secção de *input* definiu-se dois métodos de obtenção de dados que partilham o tipo *filestream* que indica ao Filebeat que os dados devem ser lidos de um determinado ficheiro. Ambos os métodos necessitam de especificar a localização dos ficheiro, um deles indica ao Filebeat que a fonte de logs provém do diretório do microsserviço "payment"enquanto que a outra fonte provém do microsserviço "webmvc". Nos dois casos, adiciona-se um novo campo aos *logs* denominado de "microservice"de modo a se distinguir a origem dos *logs* quando estes são visualizados na dashboard do sistema.

A secção de *output* instrui o Filebeat a enviar os *logs* para uma determinada instância do Logstash. Além disso e no caso de ocorrer alguma falha no envio, o Filebeat tenta reenviar a informação um máximo de cinco vezes, sendo que em cada uma delas espera cinco segundos até repetir o processo. Por intermédio da chave `hosts`, é possível definir várias instâncias do Logstash como destino da informação. No entanto e de acordo com a arquitetura previamente elaborada, apenas utiliza-se uma instância do componente Logstash. Neste sentido, a variável de ambiente LOGSTASH_HOSTS aponta unicamente para um só endereço que no caso em concreto representa o *container* que executa o Logstash.

```
filebeat.inputs:
  - type: filestream
    id: online-boutique-payment-logs
    paths:
      - "/home/payment/app_logs/*.txt"
    fields:
      microservice: payment

  - type: filestream
    id: online-boutique-webmvc-logs
    paths:
      - "/home/webmvc/app_logs/*.txt"
    fields:
      microservice: webmvc

output.logstash:
 hosts: ${LOGSTASH_HOSTS}
 max_retries: 5
 timeout: 5s
```

Excerto de código 5.13: Configuração do componente Filebeat

Logstash

O Logstash apresenta um comportamento bastante semelhante ao Filebeat, no sentido em que recolhe, processa e envia informação para outro componente, que no caso deste projeto é o ElasticSearch.

A imagem Docker utilizada para este componente pode ser observada no excerto 5.14.

```
FROM logstash:8.2.2
COPY ./config /usr/share/logstash/
```

Excerto de código 5.14: Configuração do Dockerfile do componente Logstash

Inicialmente obtém-se a imagem oficial do Logstash versão 8.2.2, logo após copiam-se os ficheiros de configuração locais para o *container* e finaliza-se com a exposição da porta 5044 para o exterior.

O Logstash é um componente que executa instruções definidas em uma ou mais pipelines. Estas pipelines regem-se pela DSL do Logstash e permitem configurar características como a entrada ou saída de dados e até mesmo os seus mapeamentos. O ficheiro de configuração representado no excerto 5.15 indica ao Logstash a localização dos ficheiros das *pipelines*.

```
- pipeline.id: ordering-database
  path.config: "/usr/share/logstash/pipeline/{ordering_in,ordering_filter,received_at_filter,
   ↪  elastic_out}.cfg"

- pipeline.id: catalog-database
  path.config: "/usr/share/logstash/pipeline/{catalog_in,catalog_filter,received_at_filter,
   ↪  elastic_out}.cfg"

- pipeline.id: filebeat-processor
  path.config: "/usr/share/logstash/pipeline/{filebeat_in,filebeat_filter,received_at_filter,
   ↪  elastic_out}.cfg"
```

Excerto de código 5.15: Definição da localização das *pipelines* do Logstash

É possível constatar a existência de três pipelines onde cada uma delas é constituída por quatro ficheiros e cada um destes ficheiro define uma etapa lógica da pipeline. O primeiro ficheiro, cujo nome tem o sufixo _in, define a etapa de entrada de dados, o segundo ficheiro define os filtros que se aplicam a uma determinada pipeline, o terceiro ficheiro descreve os filtros comum a todas as pipelines e por último o quarto ficheiro indica a lógica de saída de dados.

Também existe a possibilidade de implementar a lógica das três pipelines mencionadas num único ficheiro, contudo o autor considerou que a abordagem que se acabou por utilizar promove não só uma melhor modularização do sistema, como também uma separação concreta das responsabilidades de cada uma das pipelines.

A **entrada de dados**, para o componente Logstash, pode assumir duas formas:

1. *Logs* obtidos diretamente da base de dados dos microsserviços;
2. *Logs* recebidos por intermédio do Filebeat;

No que concerne a informação obtida da base de dados, foi necessário configurar vários parâmetros conforma denota o excerto de código 5.16.

```
input {
    jdbc {
        jdbc_driver_library => "/usr/share/logstash/mssql-jdbc-10.2.1.jre8.jar"
        jdbc_driver_class => "com.microsoft.sqlserver.jdbc.SQLServerDriver"
        jdbc_connection_string =>
        ↪  "jdbc:sqlserver://${DB_SERVER};encrypt=true;trustServerCertificate=true;
        ↪  databaseName=Microsoft.eShopOnContainers.Services.OrderingDb;"
        jdbc_user => "${DB_USER}"
        jdbc_password => "${DB_PASSWORD}"
        statement => "SELECT * from LogEvents;"
        schedule => "*/10 * * * * *"
    }
}
```

Excerto de código 5.16: Configuração da leitura de *logs* da base de dados

Começou-se por definir a classe e o binário responsáveis pela lógica de acesso à base de dados, depois indicou-se os parâmetros de conexão da base de dados juntamente com as respetivas credenciais e por último definiu-se a query que retorna a informação dos logs e a periodicidade de execução da query, que neste caso foi de 10 segundos.

Relativamente ao Filebeat, a configuração de entrada de dados já é comparativamente mais simples e pode ser visualizada no excerto 5.17.

```
input {
    beats {
        port => 5044
    }
}
```

Excerto de código 5.17: Configuração da leitura de dados do Filebeat

Para a etapa de leitura de dados do Filebeat, apenas foi necessário especificar a porta na qual o Logstash permanece à escuta de pedidos do Filebeat pois o resto da lógica necessária para a execução deste passo já é implementada e configurada nativamente pelo Logstash.

Dado que a informação dos *logs* já se encontra na posse do Logstash, a próxima fase consiste em **processá-la e tratá-la**.

O excerto de código 5.18 representa a configuração efetuada relativamente à filtragem de *logs* provenientes do microsserviço "ordering".

```
filter {
    mutate {
        add_field => { "microservice" => "ordering" }
    }
}
```

Excerto de código 5.18: Configuração da filtragem de dados do microsserviço ordering

O filtro em causa traduz-se na concatenação da chave "microservice"com o valor "ordering"aos *logs* recebidos. A mesma lógica acontece no cenário em que os *logs* são obtidos a partir do microsserviço "catalog"com a excepção do conteúdo da chave "microservice"que passa a assumir o valor de "catalog".

No que toca ao tratamento dos dados recebidos pelo Filebeat, o filtro que está em causa encontra-se explícito no excerto 5.19.

```
filter {
    if ![microservice] {
        mutate {
            add_field => { "microservice" => "unknown" }
        }
    }
}
```

Excerto de código 5.19: Configuração da filtragem de dados recebidos do Filebeat

Sempre que se recebe informação do Filebeat, verifica-se se existe alguma chave nomeada "microservice", se por qualquer motivo este não for o caso procede-se à adição dessa mesma chave com o valor de "unknown".

Finalmente o filtro que é comum a todas as pipelines, ou seja é executado de igual forma em qualquer uma das pipeline, encontra-se presente no excerto 5.20.

```
filter {
    if [level] {
        mutate {
            rename => { "level" => "severity" }
        }
    }
    mutate {
        add_field => { "receivedAt" => "%{@timestamp}" }
    }
}
```

Excerto de código 5.20: Configuração da filtragem de dados

Este último filtro encarrega-se de renomear o campo "level"para "severity"e de adicionar uma chave designada "receivedAt"cujo valor é a data e a hora na qual foi recebida e tratada a informação do *log* momentos antes de ser enviada para o ElasticSearch.

A etapa de **saída de dados** que é comum a todas as pipelines, equivale ao envio da informação anteriormente tratada para uma instância do ElasticSearch. Como tal o excerto de código 5.21 evidencia esse mesmo processo.

```
output {
    elasticsearch {
        hosts => "${ELASTIC_HOST}"
        user => logstash_internal
        password => "${LOGSTASH_INTERNAL_PASSWORD}"
        index => "online-boutique-log-data-%{+YYYY.MM.dd}"
    }
}
```

Excerto de código 5.21: Configuração da saída de dados

O processo de comunicação do Logstash para o ElasticSearch, requer a definição da instância atualmente ativa do ElasticSearch, das credenciais de acesso e do índice no qual será guardada a informação, caso este não exista o ElasticSearch cria-o de forma automática. É importar realçar que o nome do índice é concatenado com a data atual (ano, mês e dia), isto é uma aspeto crucial para mais tarde garantir-se o processo de rotação de *logs*.

5.3.2 Dashboard de Visualização de Informação

Um dos aspetos que se considera mais relevantes neste projeto diz respeito à visualização e filtração da informação dos *logs* via *dashboard*. Esta afirmação vai de encontro com o requisito **R.6** que por sua vez é estendido pelos requisitos **R.8**, **R.15**, **R.16** e **R.17**. Adicionalmente, a dashboard do sistema deve suportar operações de autenticação e autorização (**R.21** e **R.24**), bem como operações de configuração de alertas ou de utilizadores do sistema (**R.22** e **R.23**).

Em vista disso é necessário configurar uma imagem Docker do Kibana para que se possa cumprir com os requisitos supramencionados. O excerto de código 5.22 demonstra a configuração efetuada para a imagem do Kibana.

```
FROM kibana:8.2.2
COPY ./config /usr/share/kibana/config
EXPOSE 5601
```

Excerto de código 5.22: Configuração do Dockerfile do componente Kibana

A primeira linha do excerto obtém a imagem mais recente do Kibana 8.2.2 (até à data de publicação deste documento), de seguida copia-se o diretório local de configuração para o diretório de configuração do *container* do Kibana e finalmente expõe-se a porta 5601 que é a porta utilizada internamente pelo Kibana por omissão.

O ficheiro local de configuração do Kibana pode ser analisado no excerto de código 5.23.

```
elasticsearch.username: kibana_system
elasticsearch.password: ${KIBANA_PASSWORD}
elasticsearch.hosts: ${ELASTIC_HOSTS}
server.name: Online-Boutique-Logs-Dashboard
server.port: ${KIBANA_PORT}
monitoring.ui.container.elasticsearch.enabled: true
```

Excerto de código 5.23: Configuração do componente Kibana

Esta configuração define atributos tais como as credenciais a utilizar pelo Kibana sempre que este comunica com o ElasticSearch, o nome e porta do servidor do Kibana e finalmente um atributo que indica ao Kibana para disponibilizar a interface gráfica de monitorização dos containers com instâncias do ElasticSearch em execução.

De modo grosseiro, estas são as configurações necessárias para se levantar um *container* capaz de executar o componente Kibana, sem a necessidade de comunicar com outros componentes. Apresentam-se de seguida ilustrações da interface gráfica em funcionamento de modo a comprovar a implementação dos requisitos inicialmente citados nesta subsecção.

Autenticação

A figura 5.5 ilustra a interface oferecida pelo Kibana de modo a um colaborador efetuar autenticação.

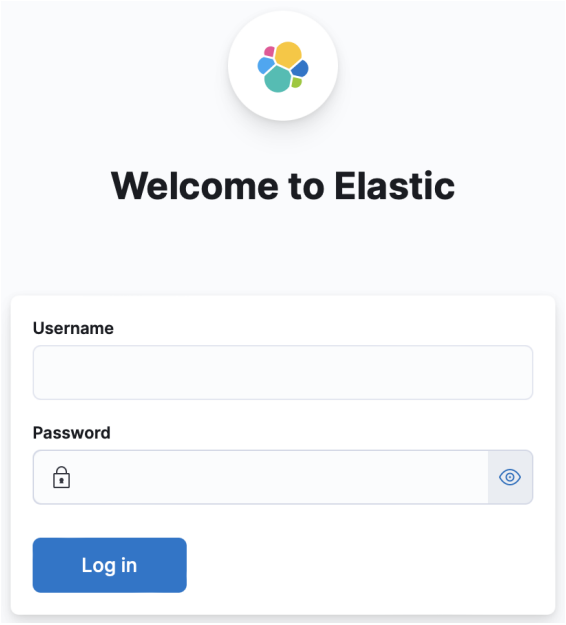

Figura 5.5: Interface gráfica de autenticação do Kibana

Trata-se de uma interface relativamente simples, onde são disponibilizadas duas caixas de texto onde o colaborador deve introduzir as suas credenciais e posteriormente pressionar o botão "Log in".

Configuração de um Novo Utilizador

A figura 5.6 representa a interface gráfica de configuração de um novo utilizador no sistema.

Esta interface quando acedida pelo administrador, disponibiliza caixas de texto que devem ser preenchidas pelo próprio e que são referentes ao *username*, nome completo, endereço de email, password e cargo do utilizador que se pretende adicionar ao sistema.

Os cargos/privilégios em causa permitem configurar permissões de leitura, escrita e execução relativamente às interfaces gráficas do Kibana e aos dados que estas facultam. Desta forma consegue-se restringir as funcionalidades que o novo utilizador acede.

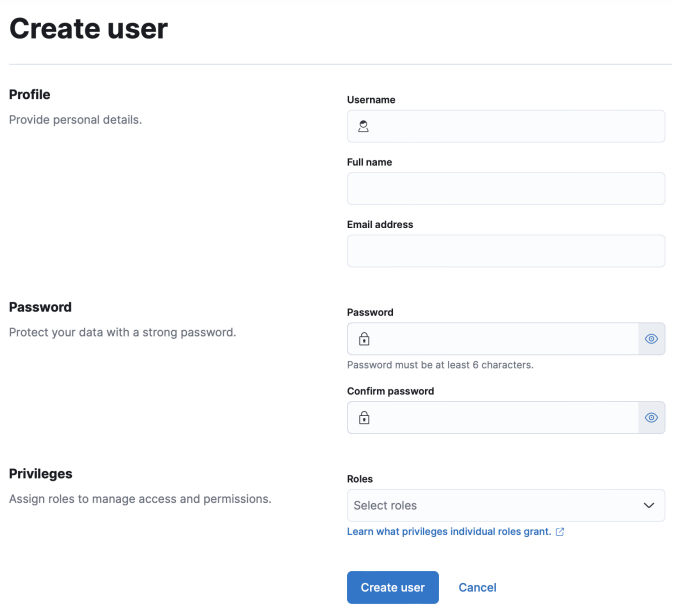

Figura 5.6: Interface gráfica de configuração de um utilizador do sistema

Configuração de um Novo Alerta

No que diz respeito ao processo de configuração de alertas, este já é mais complexo e como tal requer uma interface gráfica mais sofisticada conforme apresentado pelas figuras 5.7 e 5.8.

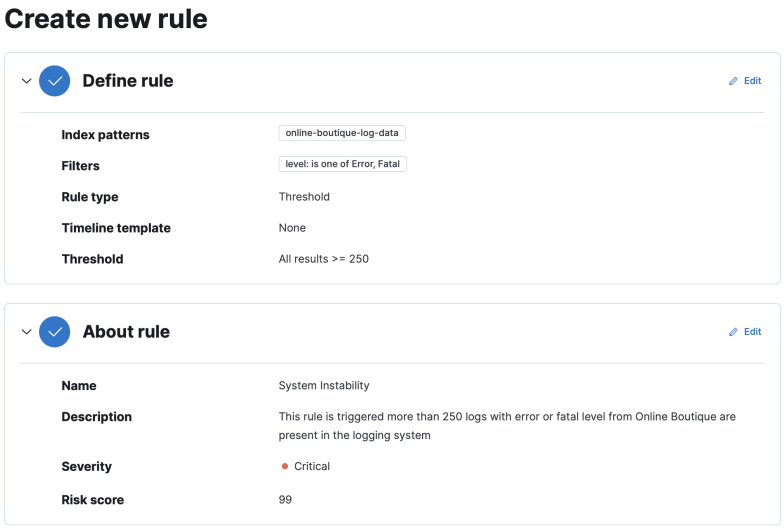

Figura 5.7: Interface gráfica de configuração de alerta do sistema

Os primeiros passos da configuração de um novo alerta, requerem que o administrador indique quais os índices que vão ser alvo de monitorização periódica, qual a regra a se aplicar e qual o patamar máximo que quando atingido considera a regra em causa como infringida.

A próxima etapa solicita ao administrador que introduza dados relativamente ao nome, descrição, severidade e pontuação de risco do novo alerta. A severidade e a pontuação do

alerta são aspetos relevantes, pois permitem facilmente detetar a gravidade do alerta em causa quando este é constatado pelos colaboradores.

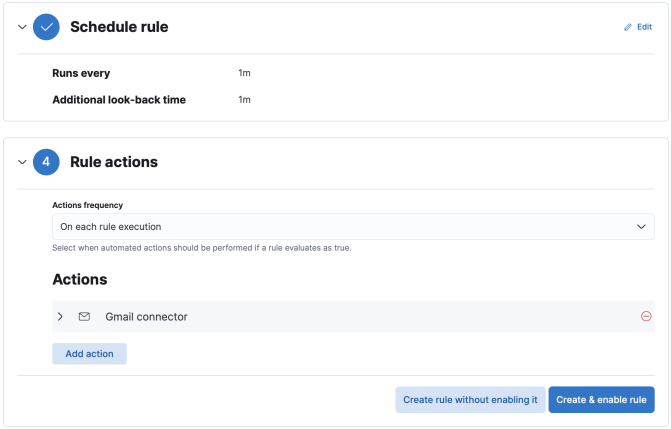

Figura 5.8: Interface gráfica de configuração de alerta do sistema

A parte final desta configuração exige a indicação da periodicidade de execução da regra e a seleção da ação a desencadear no caso de se infringir a regra em mão.

Consulta de Logs

Visualizar e filtrar a informação dos *logs* são operações que podem ser realizadas através do uso da interface gráfica exposta na figura 5.9.

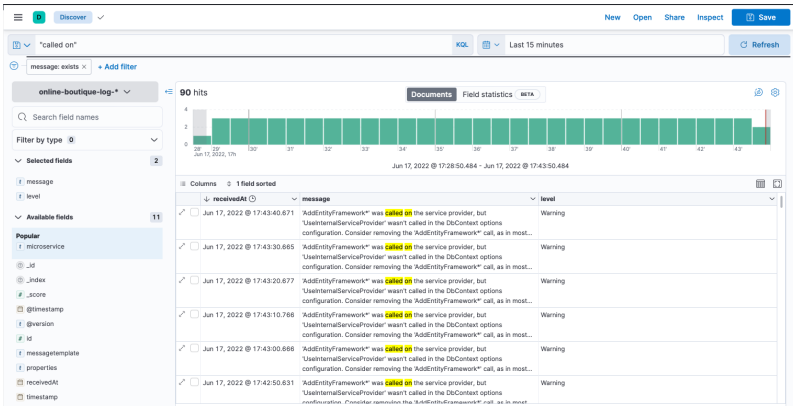

Figura 5.9: Interface gráfica de consulta de informação dos *logs*

Com base na imagem referida, é possível observar que os resultados obtidos sofrem uma filtragem textual pois têm de conter o texto "called on". Além deste parâmetro, devem também de conter o campo "message"através do filtro "message: exists"e a informação que aparece na tabela deve fazer referência aos campos "message"e "level". A tabela em causa permite ordenar de forma individual as colunas que lhe pertencem. Por último é possível constatar a existência de um gráfico de barras que expõe o número de documentos que satisfazem os filtros selecionados para um determinado período de tempo.

De forma complementar, o sistema é capaz de apresentar os resultados em formatos gráficos diversificados. Estes formatos podem ser visualizados na figura 5.10.

Para o exemplo ilustrado na figura em causa, utiliza-se um gráfico circular para representar a percentagem de *logs* que advêm dos diferentes microsserviços. Adicionalmente pode-se utilizar outro tipo de gráfico conforme indicam as sugestões presentes na figura 5.10.

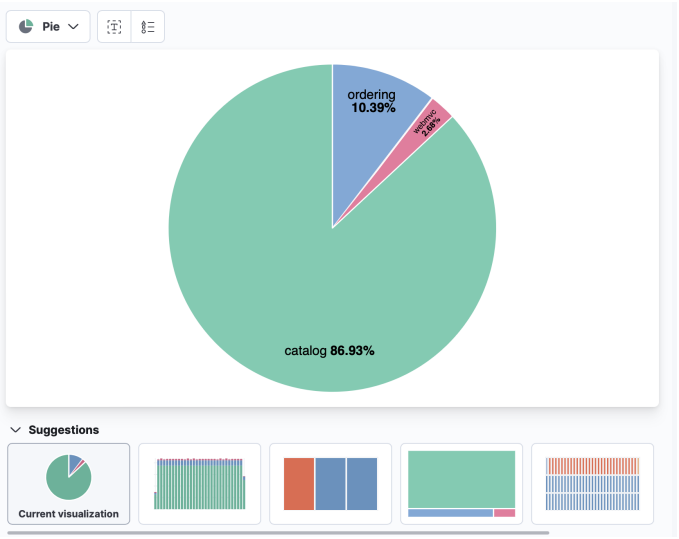

Figura 5.10: Visualização gráfica dos resultados dos *logs*

Configuração da Rotação de Logs

O Kibana também permite alterar a periodicidade de rotação de *logs* (**R.14**) que é aplicada por omissão aos índices criados pelo ElasticSearch. Esta operação pode ser realizada na interface gráfica ilustrada na figura 5.11.

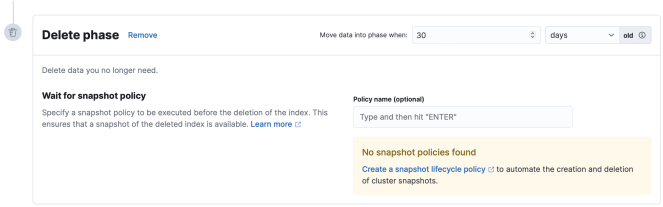

Figura 5.11: Configuração da rotação de *logs*

Com base nesta interface gráfica, consegue-se configurar a remoção da informação dos *logs* para intervalos arbitrários de tempo. Estes intervalos incluem grandezas temporais na ordem de nano-segundos até dias.

5.3.3 Segurança do Sistema

Conforme previsto nos requisitos **R.10** e **R.11**, o sistema deve garantir a autenticidade e integridade dos *logs* quando estes são transportados. Para este fim, decidiu-se utilizar o protocolo de segurança SSL (*Secure Sockets Layer*) nas comunicações entre os diversos componentes, quer estes sejam componentes internos ou externos.

ElasticSearch

O ElasticSearch disponibiliza serviços para o exterior, como tal é necessário assegurar que quem consome os ditos serviços fá-lo de forma segura. Os pares chave-valor que foram adicionados ao ficheiro de configuração **elasticsearch.yml** podem ser visualizadas no excerto de código 5.24.

```
1  xpack.security.enabled: true
2  xpack.security.http.ssl.enabled: true
3  xpack.security.http.ssl.key: ${HTTP_SSL_KEY_PATH}
4  xpack.security.http.ssl.certificate: ${HTTP_SSL_CERTIFICATE_PATH}
5  xpack.security.http.ssl.certificate_authorities:
   ↪ ${HTTP_SSL_CERTIFICATE_AUTHORITIES_PATH}
6  xpack.security.http.ssl.verification_mode: certificate
7  xpack.security.transport.ssl.enabled: true
8  xpack.security.transport.ssl.key: ${TRANSPORT_SSL_KEY_PATH}
9  xpack.security.transport.ssl.certificate: ${TRANSPORT_SSL_CERTIFICATE_PATH}
10 xpack.security.transport.ssl.certificate_authorities:
   ↪ ${TRANSPORT_SSL_CERTIFICATE_AUTHORITIES_PATH}
11 xpack.security.transport.ssl.verification_mode: certificate
```

Excerto de código 5.24: Configuração de parâmetros de segurança do ficheiro elasticsearch.yml

Tanto as comunicações entre os nós do ElasticSearch (chaves que contêm `transport.ssl`) como as comunicações com o exterior (`http.ssl`) estão configuradas para operar sobre SSL e para utilizar certificados e chaves SSL. Por último, relembra-se que as chaves cujo valor encontra-se representado entre **${}** obtêm o valor que lhes corresponde a partir de variáveis de ambiente.

Kibana

Este componente além de necessitar comunicar com o ElasticSearch precisa também de expor serviços para serem consumidos por clientes HTTP, como por exemplo um navegador web. Deste modo é necessário garantir que tanto as comunicações do browser para o Kibana e do Kibana para o ElasticSearch são fiáveis.

O excerto de código 5.25 retrata as chaves adicionadas ao ficheiro de configuração **kibana.yml**.

```
1 server.ssl.enabled: true
2 server.ssl.key: ${SSL_KEY_PATH}
3 server.ssl.certificate: ${SSL_CERTIFICATES_PATH}
4 elasticsearch.ssl.certificateAuthorities: ${SSL_CERTIFICATES_AUTHORITIES_PATH}
```

Excerto de código 5.25: Configuração de parâmetros de segurança do ficheiro kibana.yml

De maneira ao Kibana confiar nas ligações do ElasticSearch, é necessário adicionar a chave **elasticsearch.ssl.certificateAuthorities**. Esta deve apontar para o caminho do certificado da entidade emissora do mesmo. Relativamente à comunicação com o browser, o Kibana deve de atuar sobre SSL e definir a respetiva chave e certificado SSL.

É importante realçar que os navegadores web ao tentarem interagir com o Kibana, vão lançar um aviso de certificado inválido. Esta situação surge pois o certificado de autoridade é auto-assinado e consequentemente o browser não consegue verificar a legitimidade dele

(ver figura 5.12) Se eventualmente fosse utilizado um certificado legítimo (e.g entidade emissora genuína), este aviso pela parte do browser já não ocorria.

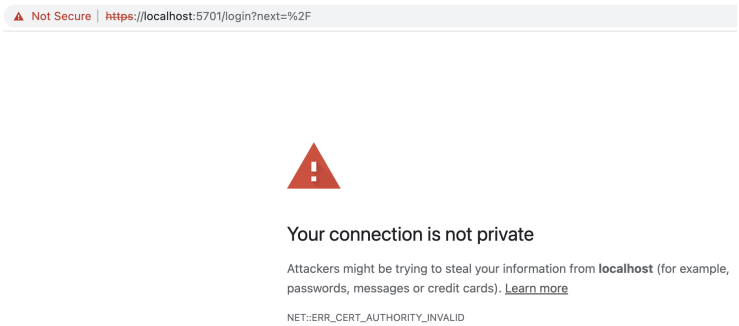

Figura 5.12: Aviso lançado pelo browser ao tentar aceder o Kibana

Logstash

De uma forma análoga ao Kibana, o Logstash necessita de configurar a segurança de dois cenários. Um deles remete para a comunicação com o ElasticSearch, o outro diz respeito à interação com o Filebeat.

As configurações de segurança para ambos os cenários foram efetuadas aquando da definição das *pipelines* que fazem parte do Logstash segundo indica o excerto de código 5.26:

```
input {
    beats {
        #omitted other keys
        ssl => true
        ssl_certificate_authorities => "${CERTIFICATE_AUTHORITIES_PATH}"
        ssl_certificate => "${CERTIFICATE_PATH}"
        ssl_key => "${KEY_PATH}"
        ssl_verify_mode => "force_peer"
    }
}

#omitted other steps

output {
    elasticsearch {
        #omitted other keys
        cacert => "${CERTIFICATE_AUTHORITIES_PATH}"
        ssl => true
    }
}
```

Excerto de código 5.26: Configuração de parâmetros de segurança das *pipelines* do Logstash

A ligação com o ElasticSearch mais uma vez necessita apenas do certificado de autoridade, por outro lado a comunicação com o Filebeat requer que o Logstash ative o modo SSL e que defina a localização do certificado de autoridade, e dos parâmetros SSL (chave, certificado e método de verificação).

Filebeat

Por último, o componente Filebeat visto que comunica unicamente com o Logstash apenas precisa das alterações apresentadas no excerto de código 5.27.

```
1 output.logstash:
2  ssl.certificate_authorities: ${CERTIFICATE_AUTHORITIES_PATH}
3  ssl.certificate: ${CERTIFICATE_PATH}
4  ssl.key: ${KEY_PATH}
```

Excerto de código 5.27: Configuração de parâmetros de segurança do ficheiro filebeat.conf

As três chaves referidas no excerto de código supracitado dizem respeito ao certificado de autoridade, ao certificado SSL e à chave SSL respetivamente. Sem estas chaves adequadamente definidas qualquer tentativa de envio de dados para o Logstash é imediatamente rejeitada.

5.3.4 Integração dos Componentes do Sistema

Até ao momento analisou-se e abordou-se o processo de configuração das imagens Docker de cada um dos componentes do sistema. Agora é necessário pormenorizar a forma como estas imagens são incorporadas em *containers* que executam e partilham informação em tempo real e que dão origem ao correto funcionamento do sistema preconizado nesta dissertação.

Para este efeito, optou-se por utilizar a plataforma Docker Compose pois trata-se de uma tecnologia especializada em orquestrar e configurar as interações e dependências que existem entre os diversos *containers* que constituem o sistema.

Antes de se realizar uma análise detalhada da configuração dos *containers* que executam os componentes do sistema, elaborou-se um grafo representado na figura 5.13 que denota as dependências que existem entre os diferentes *containers* do sistema.

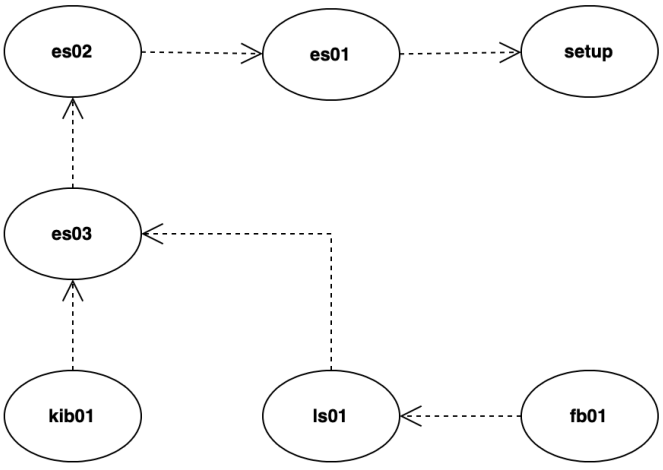

Figura 5.13: Grafo de dependências entre os *containers* do sistema

De seguida sintetiza-se as funções dos *containers* presentes na figura supracitada:

- **setup**: representa um *container* que executa a imagem oficial do ElasticSearch e é responsável por configurar os certificados de segurança SSL e por automatizar tarefas tais como a criação de utilizadores e não apresenta nenhuma dependência;
- **es01**: representa um *container* que executa a imagem do componente ElasticSearch previamente configurada. Depende do sucesso das operações realizadas pelo *container* **setup**;
- **es02**: Mesma função do *container* **es01**, só que agora depende do sucesso do container **es01**;
- **es03**: Mesma função do *container* **es01**, só que agora depende do sucesso do container **es02**;
- **kib01**: representa um *container* que executa a imagem do componente Kibana previamente configurada. Depende do sucesso das operações realizadas pelo *container* **es03**;
- **ls01**: representa um *container* que executa a imagem do componente Logstash previamente configurada. Depende do sucesso das operações realizadas pelo *container* **es03**;
- **fb01**: representa um *container* que executa a imagem do componente Filebeat previamente configurada. Depende do sucesso das operações realizadas pelo *container* **ls01**.

Com estes conceitos definidos, passa-se a explicar a configuração dos *containers* anteriores presentes no ficheiro **docker-compose.yml** da solução.

Note-se que os *containers* definidos no mesmo ficheiro docker-compose.yml partilham a mesma rede interna, isto é uma característica por omissão da tecnologia Docker Compose que permite aos *containers* comunicar entre si.

Setup

A definição e respetiva configuração do *container setup* pode ser constatada no excerto de código 5.28.

A imagem utilizada para este *container* é a imagem oficial do ElasticSearch fornecida pelos próprios desenvolvedores da ELK *stack*. Configura-se o volume local "certs"de maneira a persistir os certificados gerados pelo *container* mesmo quando este termina de executar.

Logo após, define-se o comando a executar pelo *container* mal este seja iniciado, que resumidamente gera e persiste os certificados de autoridade e as chaves privadas de cada um dos *containers* do sistema. Também automatiza tarefas tais como a definição de credenciais de acesso do Logstash e do Kibana. Por último, define-se o código a executar pelo *container* de movo a averiguar se este encontra-se num estado saudável. Este código verifica se os certificados foram criados e extraídos para os corretos diretórios.

Es01, Es02, Es03

Como a definição destes três *containers* é muito semelhante decidiu-se apresentar a configuração do *container es01* pois abrange as configurações dos restantes *containers* que

```
setup:
  image: docker.elastic.co/elasticsearch/elasticsearch:8.2.2
  volumes:
    - certs:/usr/share/elasticsearch/config/certs
  command: >
    bash -c '
      bin/elasticsearch-certutil ca --silent --pem -out config/certs/ca.zip;
      unzip config/certs/ca.zip -d config/certs;
      bin/elasticsearch-certutil cert --silent --pem -out config/certs/certs.zip --in
      ↪ config/certs/instances.yml --ca-cert config/certs/ca/ca.crt --ca-key config/certs/ca/ca.key;
      unzip config/certs/certs.zip -d config/certs;
      curl -s -X POST --cacert config/certs/ca/ca.crt -u elastic:${ELASTIC_PASSWORD} -H
      ↪ "Content-Type: application/json" https://es01:9200/_security/user/logstash_internal -d "{
      ↪ \"password\": \"${LOGSTASH_INTERNAL_PASSWORD}\", \"roles\":
      ↪ [\"logstash_writer\"], \"full_name\": \"Logstash internal user\" }";'
  healthcheck:
    test: ["CMD-SHELL", "[ -f config/certs/es01/es01.crt ]"]
    interval: 1s
    retries: 120
```

Excerto de código 5.28: Configuração parcial do *container setup*

executam a imagem do componente ElasticSearch (es02 e es03). O excerto de código 5.29 apresenta a configuração em causa.

```
es01:
  depends_on:
    setup:
      condition: service_healthy
  image: ${DOCKER_HUB_USER}/online-boutique-elasticsearch:${TAG_NAME}
  volumes:
    - certs:/usr/share/elasticsearch/config/certs
    - esdata01:/usr/share/elasticsearch/data
  ports:
    - ${ES_PORT}:9200
  env_file:
    - env/elastic-1.env
  healthcheck:
    test:
      [
        "CMD-SHELL",
        "curl -s --cacert config/certs/ca/ca.crt https://localhost:9200 | grep -q 'missing authentication
        ↪ credentials'",
      ]
    interval: 10s
    retries: 120
```

Excerto de código 5.29: Configuração do *container es01*

Inicialmente começa-se por definir uma dependência com o *container* de *setup*, no caso dos *containers* es02 e es03 as suas dependências são respetivamente o *container* es01 e es02. A imagem usada pelo *container* é a do componente ElasticSearch e os volumes utilizados são o "certs"e o "esdata01". O primeiro permite obter os dados dos certificados enquanto que o último permite persistir toda a informação dos *logs* mesmo quando o *container* é parado ou removido.

De seguida, expõe-se a porta 9200 para o exterior de maneira a possibilitar o acesso à API do *ElasticSearch* por aplicações fora do ambiente de execução Docker. Além disso, fornece-se

um ficheiro com a extensão .env que define várias de ambiente utilizadas pelo *container* e que são definidas à priori na fase de Deploy das *pipelines*. A etapa de verificação do estado atual destes *containers* consiste em efetuar um pedido GET à API e confirmar que o servidor responde com uma mensagem indicativa da inexistência de credenciais de acesso.

Kib01

A definição do *container* kib01 pode ser analisada no excerto de código 5.30.

```
kib01:
  depends_on:
    es03:
      condition: service_healthy
  image: ${DOCKER_HUB_USER}/online-boutique-kibana:${TAG_NAME}
  volumes:
    - certs:/usr/share/kibana/config/certs
    - kibanadata:/usr/share/kibana/data
  ports:
    - ${KIBANA_PORT}:5601
  env_file:
    - env/kibana.env
  healthcheck:
    test:
      [
        "CMD-SHELL",
        "curl -s -I --cacert config/certs/ca/ca.crt https://localhost:5601 | grep -q 'HTTP/1.1 302
        ↪  Found'",
      ]
    interval: 10s
    retries: 120
```

Excerto de código 5.30: Configuração do *container kib01*

Esta configuração começa por indicar que o *container* kib01 depende diretamente do *container* es03. Utiliza-se a imagem do componente Kibana e define-se o volume que contém os certificados de segurança e o volume responsável por armazenar a configuração e todos os dados pertencentes ao *container* Kibana. Depois, disponibiliza-se a porta 5601 para o exterior e indica-se qual o ficheiro que contém as variáveis de ambiente para o *container* em causa.

Relativamente ao estado do *container*, realiza-se um pedido à API do Kibana e verifica-se se a resposta retorna um código de estado 302. Se for este o caso então considera-se o estado do *container* como saudável.

Ls01

O *container* que executa o componente Logstash tem a sua definição descrita no excerto 5.31.

Com base no excerto acima representado, é possível afirmar que este *container* depende do *container* es03 e utiliza a imagem do componente Logstash. Também define volumes para aceder aos certificados de segurança e para persistir a informação gerada pelo próprio. Está configurado para receber as variáveis de ambiente via ficheiro de texto e para utilizar duas redes. A rede "default"corresponde à criada cria por omissão quando se executa o Docker Compose, a rede "online-boutique-system-network"consiste na rede interna do

```
ls01:
  depends_on:
    es03:
      condition: service_healthy
  image: ${DOCKER_HUB_USER}/online-boutique-logstash:${TAG_NAME}
  volumes:
    - certs:/usr/share/logstash/config/certs
    - logstashdata:/usr/share/logstash/data
  env_file:
    - env/logstash.env
  networks:
    - online-boutique-system-network
    - default
  healthcheck:
    test:
      [
        "CMD-SHELL",
        "curl -I -XGET 'localhost:9600/?pretty' | grep -q 'HTTP/1.1 200 OK'",
      ]
    interval: 10s
    retries: 120
```

Excerto de código 5.31: Configuração do *container ls01*

sistema Online Boutique na qual o Logstash necessita de aceder para obter a informação das diversas bases de dados.

A última etapa traduz-se na efetuação de um pedido à API do Logstash de maneira a verificar que a resposta retornada pelo servidor apresenta um código de estado 200.

Fb01

O sétimo e último *container* fb01 encontra-se definido no excerto de código 5.32.

```
fb01:
  depends_on:
    ls01:
      condition: service_healthy
  image: ${DOCKER_HUB_USER}/online-boutique-filebeat:${TAG_NAME}
  volumes:
    - certs:/usr/share/filebeat/certs
    - filebeatdata:/usr/share/filebeat/data
    - online-boutique-system_paymentlogs:/home/payment
    - online-boutique-system_webmvclogs:/home/webmvc
  env_file:
    - env/filebeat.env
  healthcheck:
    test: ./filebeat test config
    interval: 30s
    retries: 5
```

Excerto de código 5.32: Configuração do *container fb01*

Este *container* depende da estabilidade do *container* ls01 e usa a imagem do componente Filebeat. Além de definir os volumes que contêm os certificados e que persistem dados do Filebeat, necessita também de acesso aos sistema de ficheiros da máquina que executa o sistema Online Boutique. Para este fim, definiu-se e partilhou-se dois volumes adicionais (

`online-boutique-system_paymentlogs:/home/payment` e `online-boutique-system_webmvclogs:/home/webmvc`) o que viabiliza a leitura dos ficheiros de *logs* por parte do Filebeat.

Por último e de igual modo aos *containers* anteriormente analisados, indica-se qual o ficheiro que possui as variáveis de ambiente e qual o código a executar para verificar o estado do *container* Filebeat. O código em questão verifica se a configuração do Filebeat encontra-se corretamente definida.

Capítulo 6

Avaliação da Solução

Por conta da implementação e posterior implantação da solução, existe agora a necessidade de a avaliar. Este processo de avaliação deve averiguar se os requisitos previamente estabelecidos foram cumpridos na sua totalidade e deve também de verificar a qualidade da solução proposta. Para atingir este propósito, são formuladas hipóteses de investigação ligadas ao problema em causa, referidos os indicadores e fontes de informação. Este capítulo inclui também a especificação da metodologia de avaliação e é finalizado com uma análise dos resultados obtidos nos diferentes métodos de avaliação empregues.

6.1 Hipóteses de Investigação

Uma hipótese é uma possível resposta a uma pergunta de pesquisa. Trata-se de um palpite ou presunção com base no qual um estudo deve ser conduzido. Esta hipótese é testada para uma potencial aprovação ou rejeição. No caso desta ser aceite, demonstra-se que a presunção estava correta [114].

Posto isto, as hipóteses de investigação que de seguida são formuladas, devem ir ao encontro dos objetivos mencionados na secção 1.3. Desta forma, as hipóteses são as que se seguem:

- **H.1**: A solução proposta é capaz de colmatar as lacunas das plataformas de centralização de *logs* atualmente no mercado;
- **H.2**: Os requisitos funcionais e não funcionais referentes à solução proposta, são cumpridos na íntegra e apresentam um alto grau de usabilidade;
- **H.3**: A arquitetura aplicada à solução proposta, apresenta altos níveis de escalabilidade e desempenho.

A finalidade das hipóteses referidas é demonstrar que a solução proposta não apresenta lacunas no que toca à sua qualidade, não inclui erros ou falhas e acima de tudo garantir que a solução comporta-se conforme esperado, face aos objetivos inicialmente delineados.

Frisa-se que os termos "alto grau", "altos níveis", "na íntegra"presentes nas hipóteses anteriores, são avaliados mediante várias metodologias (**QEF** e **SUS**) que são aprofundadas numa fase posterior do presente capítulo.

Também é importante realçar que as hipóteses acima descritas estão intimamente ligadas às perguntas de pesquisa inicialmente definidas, nomeadamente a hipótese **H.1** visa responder à pergunta **RQ.1**, **H.2** a **RQ.2** e finalmente **H.3** a **RQ.3**.

Por último a pergunta de pesquisa **RQ.4**, que questiona os fatores a considerar no momento de avaliação da solução vai ser respondida neste capítulo, mais especificamente aquando da definição e utilização da metodologia de avaliação.

6.2 Indicadores e Fontes de Informação

Por intermédio da secção anterior, é possível constatar duas características passíveis de serem avaliadas. Uma delas encontra-se relacionada com a **usabilidade** do produto final cujo processo de avaliação consiste na elaboração de um **questionário**. Este questionário que está assente no método de avaliação **System Usability Scale** (SUS), deve ser aplicado a indivíduos que atuem na área de manutibilidade de sistemas informáticos. Com base nas respostas do questionário torna-se possível avaliar de que forma os requisitos de usabilidade se encontram completos.

A outra características que deve também ser alvo de avaliação diz respeito ao grau de **qualidade** da solução final. Esta característica é avaliada pelo próprio autor com recurso ao modelo de avaliação **Quantitative Evaluation Framework** (QEF).

Escolheram-se os indicadores em causa propositadamente pelo facto destes relacionarem-se diretamente com as hipóteses de investigação previamente formuladas.

O indicador de **qualidade** como avalia os aspetos funcionais e não funcionais da solução, visa apurar as afirmações efetuadas pelas hipóteses **H.1** e **H.3**, uma vez que estas focam-se em aspetos de qualidade da solução como por exemplo a escalabilidade, o desempenho e as funcionalidades prestadas pela mesma. De forma adicional, o indicador de **qualidade** também confere até certo ponto a hipótese **H.2**, pois este é responsável por verificar a taxa de cumprimento quer dos requisitos funcionais como dos não funcionais.

Por outro lado o indicador de **usabilidade** como lida com preocupações de navegabilidade e acessibilidade é mais adequado para atender a hipótese de investigação **H.2**, dado que esta centra-se no grau de usabilidade da componente gráfica do sistema.

6.3 Metodologia de Avaliação

Esta secção apresenta as ferramentas e processos utilizados no propósito de avaliar as métricas de usabilidade, qualidade e desempenho. Neste sentido, os processos de avaliação impostos à solução final são contextualizados e pormenorizados.

6.3.1 Testes de Software

Como mencionado nos capítulos anteriores, todos os componentes que fazem parte da solução são desenvolvidos por terceiros. Contudo, isto não é um fator impeditivo no que toca à realização de testes ao comportamento do sistema.

Os testes que de seguida são apresentados foram escritos antes de se implementar as funcionalidades da solução, isto significa que numa fase preliminar do projeto os testes embora pudessem ser executados nunca finalizavam com sucesso. Esta técnica de escrever testes antes de se obter um produto funcional, está assente nos princípios do TDD (*Test-Driven-Development*) [115].

Testes Unitários

O autor considerou desnecessário efetuar testes unitários aos componentes da ELK *stack* pois apesar de disponibilizarem o código fonte, a responsabilidade de fazer os ditos testes recai nas pessoas que desenvolvem esses mesmos componentes.

Testes de Integração

Em oposição aos testes unitários, os testes de integração já são relevantes para o projeto em mão, uma vez que a interação entre componentes é completamente da responsabilidade do autor. Os requisitos que se encaixam nesta variante de testes são o **R.4**, **R.9** e **R.14**. Os testes em causa foram implementados com recurso à tecnologia NodeJS. Esta tecnologia proporciona um ambiente capaz de executar código JavaScript fora do contexto de um *browser*.

De seguida apresenta-se o excerto de código 6.1 que denota a implementação de um teste de integração aplicado ao projeto.

```
1 test('Ensure database logs are properly processed and sent to Elastic', (t) => {
2   sendToOnlineBoutiqueDb('OrderingDb', process.env.DB_USER, process.env.DB_PASWORD,
     ↪  DUMMY_LOG_DATA);
3   setTimeout(() => {
4     const document = getDocumentByIdFromElastic(DUMMY_LOG_ID,
       ↪  process.env.ELASTIC_USER, process.env.ELASTIC_PASSWORD);
5     assert(document != null);
6     assert.deepEqual(DUMMY_LOG_DATA, document);
7   }, 10000);
8 });
```

Excerto de código 6.1: Código Javascript que efetua um teste de integração ao sistema

O teste representado no excerto acima consiste em inserir um *log* diretamente na base de dados do Online Boutique e verificar se o dito *log* além de existir na camada de persistência, apresenta o conteúdo esperado. Como a solução desenvolvida utiliza um mecanismo de *polling* para ler os *logs* das base de dados, é necessário garantir que quando este é inserido aguarda-se 10 segundos antes de se proceder ao pedido de obtenção de documento.

Posto isto, a tabela 6.1 exibe os testes de integração efetuados aos componentes do sistema.

Tabela 6.1: Testes integração efetuados ao sistema

Cenário de teste	Resultado
Inserir logs na base de dados e verificar se são processados e armazenados	✔
Criar um alerta e infringir a sua regra e verificar se são enviadas notificações	✔
Verificar se a rotação de logs remove apenas os logs necessários	✔
Inserir logs nos ficheiros de texto e verificar se são processados e armazenados	✔

Tendo como fundamento a análise efetuada aos testes de integração, considera-se que os requisitos inicialmente referidos nesta subsecção são cumpridos.

Testes de Carga

Para verificar se os comportamentos definidos pelos requisito **R.3**, **R.5** e **R.19** estão de facto a ser cumpridos, é imperativa a realização de testes de carga. Este tipo de testes tem como objetivo sobrecarregar e congestionar o sistema, através de um volume alargado de pedidos num curto período de tempo, de maneira a avaliar o comportamento do sistema nas circunstâncias referidas.

A tecnologia selecionada para efetuar testes de carga denomina-se Locust e permite especificar e escrever os ditos testes através da linguagem de programação Python.

O excerto de código 6.2 expõe o teste de carga efetuado ao sistema.

```
1  class User(HttpUser):
2      wait_time = constant(1)
3
4      @task
5      def hello_world(self):
6          self.client.get("/")
7
8  @events.test_start.add_listener
9  def on_test_start(environment, **kwargs):
10     global initial_documents_count
11     initial_documents_count = get_documents_count(environment)
12
13 @events.test_stop.add_listener
14 def on_test_stop(environment, **kwargs):
15     final_documents_count = get_documents_count(environment)
16     documents_processed_count = final_documents_count - initial_documents_count
17     if(documents_processed_count < MINIMUM_DOCS_REQUIRED):
18         sys.stderr(
19             f'Only {documents_processed_count} documents have been processed')
20         sys.exit(1)
21     print("Documents processed: ", documents_processed_count)
```

Excerto de código 6.2: Código Python que efetua testes de carga ao sistema

Adicionalmente transmite-se informação sobre o modo de execução dos testes por intermédio de *flags* do comando: `locust -u 20 --run-time 10s`

Antes de se dar início aos testes, confere-se o número de documentos existentes e guarda-se o resultado numa variável global. De seguida, simulam-se pedidos à página inicial do sistema Online Boutique (linha 6 do excerto), estes pedidos são efetuados a cada segundo e de forma paralela por 20 utilizadores (`-u 20`) durante 10 segundos (`--run-time 10s`) o que contabiliza um total de 200 pedidos[1].

No final dos testes de carga, conta-se novamente o número de documentos e se a diferença entre a contagem atual e a contagem inicial for inferior ao número mínimo configurado (1000 no contexto deste projeto), então considera-se que o sistema não foi capaz de processar em média 100 *logs* por segundo.

O teste de carga previamente analisado afeta três componentes do sistema (Filebeat, Logstash e ElasticSearch) pois a ação de visitar a página inicial do Online Boutique implica agrupar, tratar e armazenar os *logs* produzidos pelo mesmo.

[1] Um pedido à página web do Online Boutique gera em média 40 *logs* no sistema

Finalmente, durante a implementação do projeto e inclusive na solução final, o sistema nunca falhou este teste. Em média o número de documentos processados em 10 segundos rondou a casa das dezenas de milhares. Em função destes resultados o autor concluiu que os requisitos **R.3**, **R.5** e **R.19 foram cumpridos com sucesso**.

Testes de Segurança

Não menos importantes, os testes de segurança visam certificar que os requisitos **R.10** e **R.11** estão devidamente implementados, o que significa que se forem fornecidas credenciais inválidas ou efetuados acessos através do protocolo HTTP, o sistema deve rejeitar a conexão.

A tabela 6.2 especifica os diferentes cenários que foram alvo de testes.

Tabela 6.2: Testes de segurança efetuados ao sistema

Cenário de teste	Resultado
Sistema retorna o código **401** quando fornecidas credenciais inválidas	✔
Sistema retorna o código **200** quando fornecidas credenciais válidas	✔
Sistema retorna uma resposta **vazia** quando utilizado o protocolo HTTP	✔
Sistema retorna o código **403** quando utilizada uma funcionalidade restrita	✔

Para a implementação dos testes supramencionados utilizou-se a linguagem de programação Python conforme pode ser observado no excerto de código 6.3.

```
1 def ensure_401_error_when_using_no_credencials(host, port):
2     response = make_request(
3         f'https://{host}:{port}/online-boutique-log-data/_search')
4     status = response.status_code
5     assert status == 401, "Response status code should be 401"
```

Excerto de código 6.3: Código Python que efetua um teste de segurança ao sistema

O código acima retratado tenta obter informação do índice *online-boutique-log-data* e verifica se a resposta por parte do servidor (neste caso o ElasticSearch) tem o código 401.

Por último e segundo os testes em causa e com base no facto do sistema utilizar o protocolo TLS, é seguro afirmar que os requisitos de segurança do sistema são cumpridos na íntegra.

Testes E2E

De modo a averiguar o comportamento dos requisitos que envolvem a utilização de todos os componentes do sistema, optou-se por escrever testes E2E (*end-to-end*) que de grosso modo simulam as ações dos utilizadores finais do sistema diretamente no browser. Os requisitos que se enquadram neste tipo de teste são o **R.24**, **R.23**, **R.22**, **R.21**, **R.16**, **R.15**, **R.8**, **R.17**.

O excerto de código 6.4 denota o processo de implementação de um teste E2E.

Este tipo de testes quando comparados aos anteriores é o que apresenta uma maior complexidade de desenvolvimento. Por esta razão, o autor decidiu utilizar a tecnologia Cypress que por intermédio da linguagem JavaScript permite expressar os testes E2E.

```
it("Ensure system denies authentication when given wrong credentials", () => {
    cy.visit(`https://${host}:${port}/online-boutique-log-data/_search`);
    cy.get('input[name="username"]').type("random-account-" + uuid());
    cy.get('input[name="password"]').type("random-account-" + uuid());
    cy.get('button[type="submit"]').click();
    cy.get('span[class="euiCallOutHeader__title"]').first().should(($span) => {
        const text = $span.text();
        expect(text).to.equal("Username or password is incorrect. Please try again.");
    });
});
```

Excerto de código 6.4: Código Javascript que efetua um teste E2E ao sistema

Relativamente ao excerto em causa, o cenário nele retratado é o da tentativa de autenticação com credenciais inválidas por parte do utilizador não autenticado. Além deste exemplo e de forma adicional elaboraram-se testes E2E que abrangem os restantes casos de uso conforme evidenciado pela tabela 6.3.

Tabela 6.3: Testes E2E efetuados ao sistema

Cenário de teste	Resultado
Autenticação com credenciais válidas	✔
Visualização da informação dos logs	✔
Pesquisa da informação dos logs por filtros	✔
Ordenação dos resultados dos logs	✔
Filtragem da informação por texto	✔
Adição de novo colaborador no sistema	✔
Adição de regra de notificação	✔

Uma vez realizados estes testes e segundo os resultados de cada um deles é plausível afirmar que os requisitos que estão subjacentes às ações efetuadas nos testes são cumpridos na totalidade.

6.3.2 Quantitative Evaluation Framework

Como mencionado anteriormente, uma das metodologias de avaliação empregues a este projeto remete para a ferramenta QEF. Este modelo de avaliação, desenvolvido por Paula Escudeiro e José Birrada [116], compara o **sistema real** com um **sistema ideal**. Entende-se como sistema real o estado atual da solução e considera-se como sistema ideal o estado ideal que deve ser evidenciado pela solução final.

Esta comparação consiste em colocar ambas as soluções num espaço composto por três dimensões (ver figura 6.1), a fim de se calcular a distância euclidiana entre elas.

O modelo em causa assinala que a qualidade do sistema é inversamente proporcional à distância entre o sistema ideal e o sistema real [116], isto significa que quanto maior for a distância calculada mais longe a solução real situa-se da solução ideal.

O *QEF* está assente em três conceitos essenciais: **Dimensão**, **Fator** e **Requisito**. Estes conceitos seguem a seguinte hierarquia: dimensão é composta por fatores que por sua vez são compostos por requisitos.

As dimensões interligam-se com a qualidade final da solução, pois a qualidade desta relaciona-se diretamente com o grau de desempenho de cada uma das dimensões.

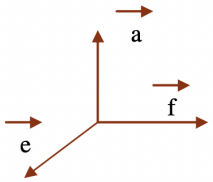

Figura 6.1: Espaço tridimensional

Fonte: [116]

Os fatores são caracterizados por:

- **Importância**: Diz respeito à divisão da soma dos graus de relevância dos requisitos a si associados pela soma de graus de relevância de todos os requisitos da dimensão onde o fator se encontra inserido;
- **Cumprimento**: Diz respeito à divisão da soma da avaliação de cumprimento dos requisitos pelo valor total de cumprimento de requisitos que fazem parte do fator.

Finalmente, os requisitos são caracterizados por:

- **Relevância**: Grau de relevância do requisito e pode ser classificado numa escala de 0, 2, 4, 6, 8 e 10, sendo 10 a classificação que corresponde ao maior nível de relevância;
- **Avaliação**: Grau de avaliação que corresponde à taxa de cumprimento de um requisito e pode ser classificada numa escala percentual de 0, 25, 50, 75 e 100, sendo 100 a taxa que corresponde ao maior nível de cumprimento.

No âmbito deste projeto definiram-se as dimensões de **Suportabilidade**, **Eficiência** e **Funcionalidade**.

A figura 6.2 ilustra os fatores e respetivos requisitos que estão associados à dimensão Suportabilidade.

Dimensão	Wij (Factor Weight j in Dim i) [0,1]	Fator	rwjk (Peso Requisito no Fator) {2, 4, 6, 8, 10}	Requisito
Suportabilidade	0.43	Adaptabilidade	8	SA01 - A plataforma está disponível em diferentes browsers
			10	SA02 - O sistema permite a inclusão de novos formatos de logs de forma rápida e fácil
	0.57	Manutibilidade	10	SM01 - O sistema permite escalar de um modo vertical e horizontal
			6	SM02 - O sistema deve estar acompanhado de documentação completa e detalhada
			8	SM03 - O sistema não afetou as aplicações já existentes

Figura 6.2: Dimensão Suportabilidade

Conclui-se então que esta dimensão é composta por dois fatores e cinco requisitos. Sendo o factor Manutibilidade o que apresenta uma maior importância. Esta característica vai ao encontro da temática do projeto, pois é necessário reduzir o tempo gasto a efetuar tarefas de manutenção. Por fim, a escala de avaliação associada a estes requisitos pode ser encontrada no anexo B.1.1.

De uma forma idêntica à dimensão anterior, a dimensão de Eficiência, representada na figura 6.3, é também composta por dois fatores e cinco requisitos. A escala de avaliação dos requisitos encontra-se no anexo B.1.2.

Dimensão	Wij (Factor Weight j in Dim i) [0,1]	Fator	rwjk (Peso Requisito no Fator) {2, 4, 6, 8, 10}	Requisito
Eficiência	0.42	Desempenho	10	ED01 - O sistema é capaz de processar a informação rapidamente
			10	ED02 - O sistema apresentam os dados em tempo real
	0.58	Navegação	10	EN01 - O sistema apresenta uma boa estrutura e permite ao colaboradores aceder à informação de forma intuitiva
			10	EN02 - Os erros apresentados pelo sistema encontram-se tratados
			8	EN03 - As ações dos utilizadores fornecem um feedback visual

Figura 6.3: Dimensão Eficiência

A última dimensão, Funcionalidade, encontra-se expressa na figura 6.4, os respetivos requisitos podem ser visualizados em maior detalhe a partir do anexo B.1.3.

Esta dimensão, ao contrário das anteriores, é composta por seis fatores e 19 requisitos. Os fatores que apresentam um maior nível de importância são respetivamente a Visualização Textual e o Tratamento Da Informação (este último é igualmente importante à Qualidade de Conteúdo). Estes fatores são os que, na opinião do autor, devem de apresentar um maior grau de importância tendo em conta a finalidade da solução, uma vez que esta se deve focar essencialmente em tratar **logs** e fornecer uma plataforma de visualização dos mesmos.

Dimensão	Wij (Factor Weight j in Dim i) [0,1]	Fator	rwjk (Peso Requisito no Fator) {2, 4, 6, 8, 10}	Requisito
Funcionalidade	0.18	Qualidade do Conteúdo	10	FQ01 - Todo o conteúdo apresentado é relevante para o utilizador
			10	FQ02 - O conteúdo textual encontra-se bem escrito e as frases bem construídas
			10	FQ03 - A informação encontra-se organizada por área funcional
	0.12	Alertas	10	FA01 - O sistema envia alertas aos utilizadores aquando da existência de anomalias
			10	FA02 - O sistema permite configurar o threshold no qual os alertas se devem basear
	0.14	Visualização Gráfica	8	FG01 - O sistema apresenta a informação em diversos formatos gráficos
			10	FG02 - O sistema permite filtrar a informação dos gráficos
			6	FG03 - Apresenta valores estatísticos relativos à informação dos logs, como a média e o desvio padrão
	0.22	Visualização Textual	10	FT01 - O sistema apresenta a informação dos logs em formato textual
			10	FT02 - O sistema permite pesquisar por informação através de texto completo
			8	FT03 - Fornece sugestões de correção do texto introduzido pelo utilizador
			10	FT04 - O sistema permite ordenar e agrupar os resultados obtidos
	0.18	Tratamento da Informação	10	FI01 - O sistema é capaz de tratar logs em diversos formatos
			10	FI02 - O sistema remove os logs a partir de uma determinada data
			10	FI03 - O sistema garante a autenticidade e integridade da informação que é tratada
	0.16	Utilizador	8	FU01 - O sistema permite a autenticação do utlizador
			4	FU02 - O sistema permite recuperar a palavra passe
			8	FU03 - O sistema restringe as funcionalidades consonte o cargo do utilizador
			6	FU04 - O sistema permite a configuração de utilizadores

Figura 6.4: Dimensão Funcionalidade

6.3.3 System Usability Scale

No sentido de medir o nível de usabilidade da solução optou-se por utilizar o método de avaliação **System Usability Scale**. Este método desenvolvido por Brooke em 1986 [117], consiste num questionário que permite aos detentores de um determinado produto ou serviço avaliá-lo de forma rápida e fácil.

As razões que levaram o autor à escolha deste método advêm do facto do questionário ser adequado a interfaces de computador (e.g web sites), ser relativamente rápido e fácil de utilizar e finalmente porque proporciona uma pontuação única numa escala que é facilmente compreendida [117].

No que concerne a escala, esta é composta por cinco classificações:

1 - Discordo Totalmente

2 - Discordo

3 - Indiferente

4 - Concordo

5 - Concordo Totalmente

O questionário em si é composto por 10 questões, que podem ser observadas na tabela 6.4.

Como referido na secção 6.2 deste capítulo a audiência alvo deste questionário é precisamente indivíduos que realizem atividades de manutenção de sistemas informáticos, em específico aqueles que se focam no diagnóstico de aplicações baseadas em microsserviços.

Cada participante do questionário deve responder às perguntas com um dos valores da escala previamente mencionada. As perguntas que não se encontram respondidas devem ser atribuídas o valor três da escala aquando da interpretação dos resultados [118].

Tabela 6.4: Questões do questionário SUS [117]

#	**Afirmação**
1	Gostaria de utilizar este sistema frequentemente
2	Achei o sistema desnecessariamente complexo
3	Achei o sistema fácil de utilizar
4	Achei que necessitaria de apoio de uma pessoa técnica para usar este sistema
5	Achei que as várias funções deste sistema encontram-se bem integradas
6	Achei que havia demasiada incoerência neste sistema
7	Imaginaria que grande parte das pessoas aprenderia a utilizar este sistema muito rapidamente
8	Achei o sistema muito complicado de se usar
9	Senti-me bastante confiante ao utilizar o sistema
10	Precisei de aprender muitas coisas antes de poder avançar com este sistema

Uma vez respondido o questionário, o próximo passo reflete-se na interpretação dos resultados. Como tal, para as perguntas com conotação positiva (1, 3, 5, 7 e 9) deve-se subtrair 1 ao seu respetivo valor. Em contrapartida, para se obter o valor das perguntas com conotação negativa (2, 4, 6 e 8) é necessário efetuar o cálculo de 5 subtraído como valor da respetiva resposta [118].

De seguida e com base nas indicações supracitadas, efetua-se a soma de todos os valores e multiplica-se o resultado por 2.5 obtendo-se assim a pontuação geral do SUS. É a partir desta pontuação que se pode obter conclusões relativamente à usabilidade do sistema [118].

A figura 6.5 indica qual a avaliação a se atribuir consoante a pontuação obtida.

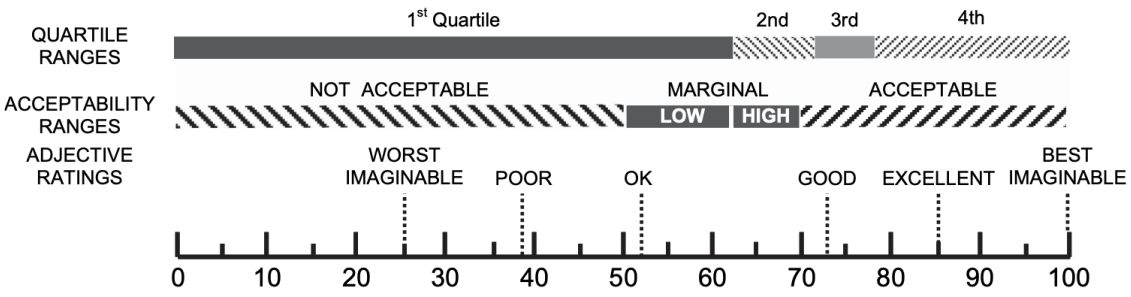

Figura 6.5: Pontuação SUS

Fonte: [117]

Com base nesta figura é possível inferir que uma pontuação inferior a 50 indica que o sistema apresenta graves problemas no que toca à sua usabilidade. O melhor cenário possível inclui uma pontuação muito próxima do valor 100, no entanto este é um cenário pouco realista. Na perspetiva do autor um valor mais razoável rondaria na casa dos 75-85 pontos.

Por último, apresenta-se uma enumeração dos passos a efetuar pelos participantes antes de procederem à realização do questionário em causa:

1. Navegar até à página principal da solução final;
2. Autenticar no sistema com as credenciais fornecidas;
3. Ordenar *logs* por hora de receção dos mais antigos para os mais recentes;
4. Introduzir texto de maneira a obter os *logs* referentes ao microsserviço "webmvc";
5. Criar um filtro que retorne os *logs* correspondentes a avisos do sistema;
6. Visualizar a mensagem e as propriedades em causa para um determinado *log*;
7. Efetuar operação de *logout* no sistema.

O cenário constituído pelas etapas anteriores foca-se sobretudo nos atos de consulta e filtragem da informação existente nos *logs* do sistema Online Boutique. Trata-se de uma característica intencional por parte do autor, pois a operação de consultar *logs* é a que provavelmente mais vezes será realizada.

6.4 Análise de Resultados

Esta secção pretende averiguar se os métodos aplicados à metodologia de avaliação previamente abordada produziram resultados capazes de validar os objetivos inicialmente traçados.

Os testes de software avaliam de uma forma superficial os requisitos estabelecidos no início do projeto. A avaliação de qualidade (método QEF) examina se a solução atende às necessidades dos utilizadores finais. Por fim, a avaliação da usabilidade (método SUS) avalia a facilidade com que um utilizador final interage com a solução desenvolvida.

6.4.1 Resultados dos Testes de Software

Dos testes realizados à solução final, quer sejam testes de integração, carga, segurança ou E2E, quase todos apenas validam cenários de sucesso. Isto significa que existem cenários com fluxos alternativos que não foram alvo de testes, o que leva à conclusão que os testes efetuados apesar de validarem o *"happy path"* não são tão exaustivos como inicialmente pretendido pelo autor.

Ainda assim considera-se que os resultados obtidos durante a fase de testes foram positivos, pois asseguram que o sistema comporta-se conforme esperado na grande maioria das circunstâncias.

6.4.2 Resultado da Avaliação de Qualidade

A avaliação de qualidade da solução, tal como referido na subsecção 6.3.2 está assente nos conceitos da metodologia QEF. A figura 6.6 evidencia os resultados obtidos com base na dita metodologia.

Todos os 29 requisitos são avaliados de forma individual e regem-se pelas diretrizes de taxa percentual de cumprimento definidas à priori (ver anexo B.1).

Ao analisar a figura em causa, é possível observar as percentagens de cumprimento das três dimensões (suportabilidade, eficiência e funcionalidade). Começando pela dimensão de **funcionalidade**, que das três é a que apresenta a menor percentagem de requisitos cumpridos (87.75%), rapidamente apercebe-se que o fator **visualização gráfica** ficou ligeiramente aquém das expectativas obtendo uma percentagem de 72.92 de cumprimento. Esta situação surge pois a visualização gráfica dos *logs* contém poucos detalhes estatísticos (e.g é desprovida do desvio padrão, da variância, etc.).

q	D	Qi	Dimensão	Qj	Wij (Factor Weight j in Dim i) [0,1]	Fator	rwjk (Peso Requisito no Fator) {2, 4, 6, 8, 10}	Requisito	% Cumprimento
93%	0.17	92.9	Suportabilidade	100	0.43	Adaptabilidade	8	SA01 - A plataforma está disponível em diferentes browsers	100
							10	SA02 - O sistema permite a inclusão de novos formatos de logs de forma rápida e fácil	100
				87.5	0.57	Manutibilidade	10	SM01 - O sistema permite escalar de um modo vertical e horizontal	100
							6	SM02 - O sistema deve estar acompanhado de documentação completa e detalhada	50
							8	SM03 - O sistema não afetou as aplicações já existentes	100
		89.64	Eficiência	100	0.42	Desempenho	10	ED01 - O sistema é capaz de processar a informação rapidamente	100
							10	ED02 - O sistema apresentam os dados em tempo real	100
				82.14	0.58	Navegação	10	EN01 - O sistema apresenta uma boa estrutura e permite ao colaboradores aceder à informação de forma intuitiva	100
							10	EN02 - Os erros apresentados pelo sistema encontram-se tratados	50
							8	EN03 - As ações dos utilizadores fornecem um feedback visual	100
		87.75	Funcionalidade	83.33	0.18	Qualidade do Conteúdo	10	FQ01 - Todo o conteúdo apresentado é relevante para o utilizador	50
							10	FQ02 - O conteúdo textual encontra-se bem escrito e as frases bem construídas	100
							10	FQ03 - A informação encontra-se organizada por área funcional	100
				75	0.12	Alertas	10	FA01 - O sistema envia alertas aos utilizadores aquando da existência de anomalias	50
							10	FA02 - O sistema permite configurar o threshold no qual os alertas se devem basear	100
				72.92	0.14	Visualização Gráfica	8	FG01 - O sistema apresenta a informação em diversos formatos gráficos	100
							10	FG02 - O sistema permite filtrar a informação dos gráficos	100
							6	FG03 - Apresenta valores estatísticos relativos à informação dos logs, como a média e o desvio padrão	25
				100	0.22	Visualização Textual	10	FT01 - O sistema apresenta a informação dos logs em formato textual	100
							10	FT02 - O sistema permite pesquisar por informação através de texto completo	100
							8	FT03 - Fornece sugestões de correção do texto introduzido pelo utilizador	100
							10	FT04 - O sistema permite ordenar e agrupar os resultados obtidos	100
				100	0.18	Tratamento da Informação	10	FI01 - O sistema é capaz de tratar logs em diversos formatos	100
							10	FI02 - O sistema remove os logs a partir de uma determinada data	100
							10	FI03 - O sistema garante a autenticidade e integridade da informação que é tratada	100
				84.61	0.16	Utilizador	8	FU01 - O sistema permite a autenticação do utlizador	100
							4	FU02 - O sistema permite recuperar a palavra passe	0
							8	FU03 - O sistema restringe as funcionalidades consonte o cargo do utilizador	100
							6	FU04 - O sistema permite a configuração de utilizadores	100

Figura 6.6: Resultados da avaliação QEF aplicada à solução final

De seguida, a dimensão **eficiência** obteve uma classificação intermédia quando comparadas às restantes (89.64%) e o fator que impactou esta avaliação foi a **navegação**, mais especificamente os erros que o sistema apresenta para os utilizadores finais, que em alguns dos casos expõem detalhes de implementação (e.g erros em formato JSON) e por este motivo considerou-se uma taxa de cumprimento de 50% em relação ao requisito em causa.

A terceira e última dimensão, **suportabilidade**, é a que das três apresenta a maior taxa de cumprimento (92.9%). O fator que impediu esta dimensão de obter a classificação máxima foi o de **manutibilidade** uma vez que a documentação desenvolvida não é completamente extensa e detalhada ao ponto de obter uma taxa de 100%. A documentação em causa consiste de ficheiros changelog.md (gerados automaticamente pela pipeline), *issues* do gitlab e o readme.md do projeto (gerado automaticamente pelo gitlab), todavia na perspetiva do autor estes artefactos não satisfazem uma documentação devidamente completa e detalhada.

Segue-se um breve esclarecimento dos requisitos que não apresentam uma taxa de 100% e que até agora não foram abordados:

- **FQ01** - Não se considerou este requisito como 100% cumprido pois existem funcionalidades disponíveis na interface gráfica que não se enquadram no escopo deste projeto, isto acontece porque o componente Kibana já as oferece de forma nativa;

- **FA01** - Para este requisito ser cumprido na totalidade o sistema deve suportar enviar notificações quer via email quer via SMS. Como o autor optou somente por implementar o mecanismo de envio de emails, é apenas justo prejudicar a classificação deste requisito;
- **FU02** - Como a única forma de recuperar a palavra passe é por intermédio de um administrador ou de um *superuser*, este precisa de configurar uma nova palavra-passe para o utilizador desejado, considerou-se este requisito como não implementado o que por consequência levou à atribuição da classificação 0 ao requisito em causa.

Os requisitos que restam e que nesta subsecção não foram aludidos, apresentam uma taxa de cumprimento de 100% que pode ser corroborada quer pela subsecção dos testes de software quer pelas ilustrações e excertos de código presentes no capítulo de implementação.

A partir da taxa de cumprimento obtida para cada uma das três dimensões - **Suportabilidade (92,9%)**, **Eficiência (89.64%)** e **Funcionalidade (87.75%)** - é possível calcular a percentagem que representa a qualidade do sistema real com base na equação 6.1.

$$q = \frac{log(\frac{1+92.90}{100}) + log(\frac{1+89.64}{100}) + log(\frac{1+87.75}{100})}{3 * log(2)} = 93\% \tag{6.1}$$

Adicionalmente calculou-se a distância euclidiana recorrendo à equação 6.2:

$$D = \sqrt{(1 - \frac{92.90}{100})^2 + (1 - \frac{89.64}{100})^2 + (1 - \frac{87.75}{100})^2} = 0.17 \tag{6.2}$$

Segundo a análise elaborada nesta subsecção, é possível afirmar que a solução real encontra-se a uma **distância euclidiana de 0.17** da solução ideal e representa um sistema com **93% das capacidades** esperadas de um sistema ideal.

6.4.3 Resultados da Avaliação de Usabilidade

A avaliação da usabilidade da solução, tal como mencionado na subsecção 6.3.3 é realizada através das respostas obtidas no questionário de usabilidade do sistema.

O questionário em causa contou com a participação de 10 indivíduos que trabalham maioritariamente na área de desenvolvimento e manutenção de software. As classificações obtidas para cada uma das questões podem ser observadas no gráfico de barras ilustrado na figura 6.7.

Salienta-se que no que toca às questões ímpares, um melhor resultado implica uma classificação mais próxima possível de 5, em contrapartida e relativamente às questões pares, um melhor resultado envolve uma classificação igual ou próxima de 1.

Com isto em mente e de um modo geral o autor considerou que face às respostas obtidas, é possível concluir que estas representam resultados positivos no que concerne a usabilidade do sistema. No entanto e de modo a finalizar a avaliação SUS é necessário calcular a pontuação geral do sistema. O cálculo em causa pode ser realizado mediante a equação 6.3.

Seja

$$A = (4.3 - 1) + (4.2 - 1) + (4.7 - 1) + (4.3 - 1) + (4.1 - 1)$$
$$B = (5 - 1.8) + (5 - 2.2) + (5 - 1.1) + (5 - 1.7) + (5 - 1.6)$$

Então

$$P = \frac{A + B}{0.4} = 83 \tag{6.3}$$

Como o pontuação obtida corresponde ao **valor 83** e conforme assinala a figura 6.5, é possível afirmar que a usabilidade do sistema encontra-se bastante próxima do patamar excelente. Como consequência deste resultado, considera-se que requisitos como a usabilidade e navegabilidade do sistema encontram-se devidamente implementados.

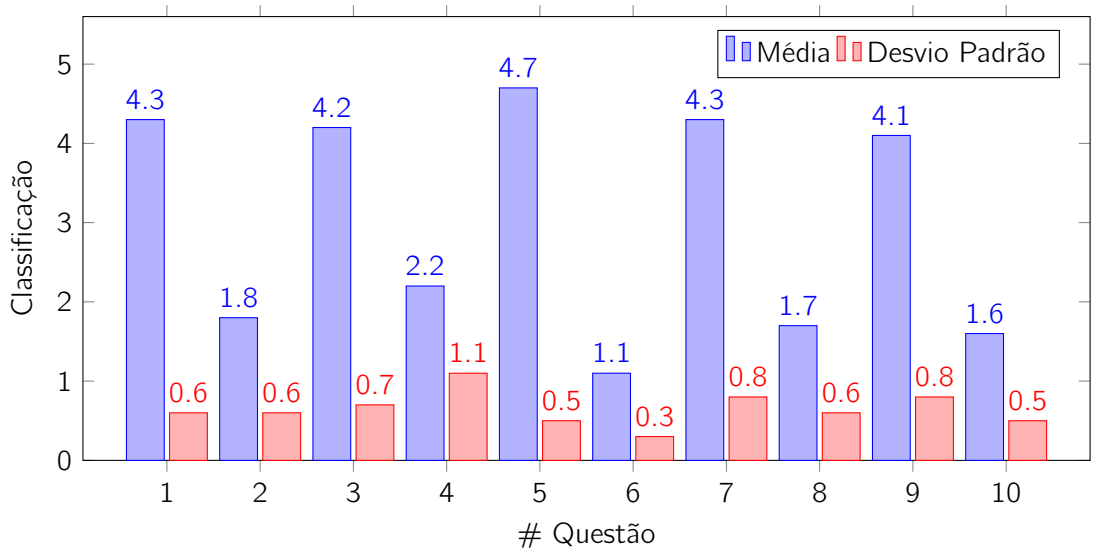

Figura 6.7: Média e desvio padrão das respostas obtidas no questionário

Contudo reconhece-se que o número de amostras utilizadas na avaliação desta componente é relativamente reduzido. Existe a possibilidade da pontuação final do sistema ser superior ou inferior à obtida, no caso de se utilizar uma quantidade mais alargada de amostras, que no âmbito da avaliação SUS corresponde a um maior número de participantes no questionário.

6.4.4 Resultados das Hipóteses de Investigação

No início deste capítulo, em particular na secção 6.1 formularam-se hipóteses de investigação no intento de demonstrar que a solução final comporta-se conforme esperado perante os objetivos inicialmente delineados. Deste modo e através dos indicadores e fontes de informação expressos na secção 6.2, estabeleceu-se uma relação entre as hipóteses de investigação e os ditos indicadores.

Em função da análise de resultados até ao momento efetuada e uma vez que esta envolve os indicadores **qualidade** e **usabilidade**, resta agora conferir os resultados obtidos relativamente às hipóteses de investigação. Este resultado encontra-se explícito na seguinte listagem:

- **H.1**: - Como a solução final denota um conjunto de funcionalidades similares a uma plataforma atual de centralização de *logs* e ao mesmo tempo apresenta uma maior

flexibilidade e um menor custo monetário, confirma-se a afirmação proposta pela hipótese em questão;

- **H.2**: - Como os valores obtidos na classificação dos indicadores de qualidade e usabilidade foram satisfatórios e refletem classificações semelhantes a sistemas ideais, é possível confirmar a afirmação sugerida pela hipótese em causa;
- **H.3** - Como a arquitetura da solução final permite escalar facilmente de modo horizontal (como exemplo tem-se as três instâncias do componente ElasticSearch) e conforme indicam os resultados dos testes de carga, verifica-se a hipótese em mão.

Por último e estando assente nos resultados obtidos na presente subsecção conclui-se que os cenários expostos pelas hipóteses de investigação encontram-se efetivados na solução final.

Capítulo 7

Conclusão

Dado o projeto como concluído, é pertinente olhar para o trabalho desenvolvido e refletir sobre os resultados obtidos em cada uma das fases do projeto. Este capítulo é iniciado por uma enumeração dos objetivos alcançados a fim de apurar o êxito do projeto. A seguir, mencionam-se as limitações que a solução final apresenta e as potenciais melhorias que lhe podem ser aplicadas num trabalho futuro. Finalmente e numa vertente mais pessoal, o autor oferece uma apreciação global de todo o processo de desenvolvimento do trabalho efetivado nesta dissertação.

7.1 Objetivos Concretizados

Na fase primordial do projeto foram estabelecidos vários objetivos. É relevante nesta fase do documento averiguar se os objetivos em causa foram cumpridos com sucesso pois a taxa de cumprimento dos ditos objetivos coloca em causa o sucesso deste projeto.

Assim sendo, relembra-se que a finalidade deste projeto remete para o desenvolvimento de uma solução capaz de **extrair, processar, armazenar e apresentar** *logs* provenientes de diferentes microsserviços. Para este efeito foram definidos objetivos na secção 1.3 que encontram-se abaixo sintetizados:

- **Análise de soluções existentes no mercado** - Análise do funcionamento de soluções de monitorização de *logs* de modo a identificar-se lacunas ou pontos de melhoria;
- **Literatura do estado da indústria para recolha de requisitos** - Análise de projetos e relatórios técnicos que retratem o tópico de monitorização de *logs* a fim de auxiliar o processo de levantamento de requisitos;
- **Desenho de uma arquitetura robusta e escalável**: Desenhar uma arquitetura aplicando boas práticas de desenvolvimento de software, que deve ser concretizada na implementação da solução;
- **Fatores a considerar na avaliação da solução** - Definição de uma metodologia concreta de avaliação que deve ser aplicada à solução final.

O sucesso do **primeiro objetivo**, centrado na análise de soluções de monitorização de *logs* do mercado, é visível no decorrer do capítulo 2 mais propriamente na secção 2.3. Esta secção evidenciou que grande parte das soluções analisadas contêm as suas funcionalidades restritas a um ou vários serviços de subscrição. Além disso, também foi possível averiguar que estas soluções não são flexíveis ao ponto de satisfazer os requisitos deste projeto. Como exemplo tem-se o processo de recolha de informação que ficaria da responsabilidade do

sistema monitorizado, algo que no âmbito deste projeto pretende-se evitar o tanto quanto possível.

Relativamente ao **segundo objetivo**, que alude ao processo de levantamento de requisitos, considera-se que este da mesma forma que o anterior também foi bem sucedido. A recolha de requisitos efetuada pelo autor envolveu não só um estudo das soluções existentes no mercado bem como uma análise detalhada de relatórios e dissertações académicas a fim de encontrar valores tangíveis de atributos de qualidade. Um exemplo concreto desta abordagem refere-se ao requisito **R.3** que indica que o sistema deve tratar 100 *logs* por segundo sem ocorrer perda de informação.

O **terceiro objetivo** que toca ao desenho de uma arquitetura robusta e escalável também foi considerado como bem sucedido. Esta afirmação está assente na análise realizada ao longo do capítulo 4. Esta análise incluiu a discussão de diferentes alternativas de diagramas de componentes, implantação e sequência, o que resultou na implementação de um sistema composto por componentes da ELK *stack* que apresenta um grau elevado de automatização, modularidade e configurabilidade.

O sucesso do último e **quarto objetivo**, referente aos fatores a considerar no momento da avaliação da solução final, encontra-se explícito no capítulo 6, em particular na definição da metodologia de avaliação empregue à solução final. Esta metodologia implicou a utilização de três métodos que avaliam consoante diretrizes bem definidas a qualidade e a usabilidade do sistema. Os primeiros dois métodos (testes de software e QEF) visam garantir a qualidade da solução enquanto o último método (SUS) destina-se a classificar a usabilidade da solução.

Em suma e em virtude da concretização do objetivos previamente mencionados, é seguro afirmar que o presente projeto foi elaborado com sucesso.

7.2 Limitações e Trabalho Futuro

Mesmo que de um modo geral os objetivos referidos na secção anterior tenham sido alcançados, é necessário olhar com um espírito crítico para todo o trabalho elaborado ao longo da presente dissertação.

Primeiramente e a começar pelo requisitos não funcionais, o autor reconhece que podem ser efetuadas melhorias no que concerne preocupações como a escalabilidade, em particular os *containers* Docker são capazes de serem orquestrados por tecnologias mais robustas que o Docker-Compose, nomeadamente a tecnologia Kubernetes ou Docker Swarm. Além disso, reconhece-se que no que diz respeito à implantação da solução, esta poderia tirar proveito de mais do que uma máquina física, isto que não acontece devido a restrições monetárias e de hardware do autor.

A pipeline de integração e entrega contínua implementada nos repositórios de cada projeto apenas lida com cenários felizes, também denominados de *happy-path*. Seria interessante para um trabalho futuro implementar um mecanismo capaz de reverter as alterações colocadas na nuvem, no casos dos testes realizados pós-implantação falharem.

Também é importante destacar que no que diz respeito a questões de internacionalização e localização, o Kibana não suporta mais do que um idioma de interface gráfica ao mesmo tempo. Isto implica que se for necessário suportar vários idiomas em simultâneo é obrigatório executar pelo menos mais do que uma instância do Kibana.

Uma das formas de notificar os colaboradores da organização implementadas neste projeto é através do uso de emails. No entanto pode ser mais vantajoso notificar esses mesmo colaboradores via SMS ou notificações *push*, porque desta maneira é mais provável que os colaboradores se apercebam mais rapidamente de problemas ou anomalias no sistema.

Um tópico que até ao momento não foi abordado mas que merece algum destaque, relaciona-se com a personalização da solução. Devido à natureza dos componentes utilizados na solução, o sistema (em particular a interface gráfica) contém elementos gráficos como títulos, caixas de texto ou ilustrações que fazem referência à organização desenvolvedora da ELK *stack*. Pode-se considerar um ponto de melhoria remover os ditos elementos ou então, no contexto de uma organização real, adaptá-los às necessidades da mesma.

7.3 Apreciação final

Na experiência académica e profissional do autor quando se desenvolvem aplicações baseadas em microsserviços, questões como monitorização, auditoria e análise de *logs* são muitas das vezes colocadas em segundo plano ou no pior dos casos menosprezadas. Foi esta uma das razões que levaram à escolha do tema abordado nesta dissertação. Outra razão, relaciona-se com o facto de que no percurso académico do autor, estes tópicos raramente eram aprofundados.

De um ponto de vista teórico o autor sente-se bastante satisfeito com os conceitos abordados nesta dissertação, até porque à priori deste projeto alguns deles eram completamente ignorados ou esquecidos pelo próprio.

Já numa perspetiva mais prática, o autor chegou à conclusão que relativamente a questões de monitorização e centralização de *logs* não há necessidade de "reinventar a roda", isto acontece porque existem inúmeras soluções e tecnologias *open-source* que quando conjugadas entre si, conseguem satisfazer as necessidades da maior parte do sistemas, no que toca a preocupações como monitorização.

Sobre as tecnologias empregues na solução, nomeadamente a ELK *stack* o autor não detinha nenhuma experiência ou conhecimento anterior delas, apenas possuía uma experiência bastante superficial com a tecnologia Docker. Isto foi um fator crucial no desenvolvimento deste projeto pois serviu como um fator de motivação e encorajamento para o autor.

Não menos importante, o autor também ficou agradado por aprender e aplicar novas metodologias de avaliação como o QEF e o SUS. Acima de tudo, o autor considera um mais valia o trabalho realizado nesta dissertação pois implica um avanço no percurso académico do mesmo. Ainda mais importante do que isso, foi todo o conhecimento adquirido que certamente será colocado em prática no futuro.

No final de contas o projeto responde às necessidades inicialmente estabelecidas, isto acontece pois a solução final é provida de características que melhoram o processo de deteção de anomalias em arquiteturas baseadas em microsserviços. Estas características incluem aspetos como uma interface gráfica centralizada de visualização de *logs* e mecanismos automáticos de notificações de irregularidades no sistema. Na perspetiva do autor, todos estes fatores vêm reduzir a taxa de erro humano e o tempo despendido no diagnóstico e resolução de falhas no sistema.

Por último e em todas as vertentes incluídas na realização desta dissertação, o autor considera este projeto um sucesso.

Referências

[1] Paolo Di Francesco, Ivano Malavolta e Patricia Lago. «Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption». Em: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 21–30. doi: 10.1109/ICSA.2017.24.

[2] Prathap Irudayaraj e Saravanan P. «Adoption Advantages Of Micro-Service Architecture In Software Industries». Em: *International Journal of Scientific Technology Research* 8 (set. de 2019), pp. 183–186.

[3] Jeanderson Cândido, Maurício Aniche e Arie Deursen. «Log-based software monitoring: a systematic mapping study». Em: *PeerJ Computer Science* 7 (mai. de 2021), e489. doi: 10.7717/peerj-cs.489.

[4] Chris Smith. *The 7 main types of server monitoring*. Mar. de 2019. url: https://knowtechie.com/the-7-main-types-of-server-monitoring/ (acedido em 06/11/2021).

[5] Xiang Zhou et al. «Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study». Em: *IEEE Transactions on Software Engineering* 47.2 (2021), pp. 243–260. doi: 10.1109/TSE.2018.2887384.

[6] Ivan Beschastnikh et al. «Debugging Distributed Systems». Em: *Commun. ACM* 59.8 (jul. de 2016), pp. 32–37. issn: 0001-0782. doi: 10.1145/2909480. url: https://doi.org/10.1145/2909480.

[7] Shona McCombes. *Developing strong research questions*. Mar. de 2021. url: https://www.scribbr.com/research-process/research-questions/ (acedido em 08/11/2021).

[8] Hayden Donelley. *Why do you need a methodology?* url: http://www.rangahau.co.nz/rangahau/55/ (acedido em 10/11/2021).

[9] Anna-Maria Teperi, Nadezhda Gotcheva e Kirsi Aaltonen. «16 - Design thinking perspective for developing safety management practices in nuclear industry». Em: *Human Factors in the Nuclear Industry*. Ed. por Anna-Maria Teperi e Nadezhda Gotcheva. Woodhead Publishing Series in Energy. Woodhead Publishing, 2021, pp. 309–326. isbn: 978-0-08-102845-2. doi: https://doi.org/10.1016/B978-0-08-102845-2.00016-8. url: https://www.sciencedirect.com/science/article/pii/B9780081028452000168.

[10] Jan vom Brocke, Alan Hevner e Alexander Maedche. «Introduction to Design Science Research». Em: set. de 2020, pp. 1–13. isbn: 978-3-030-46780-7. doi: 10.1007/978-3-030-46781-4_1.

[11] Sharon Christa et al. «Software Maintenance: From the Perspective of Effort and Cost Requirement». Em: vol. 469. Jan. de 2017, pp. 759–768. isbn: 978-981-10-1677-6. doi: 10.1007/978-981-10-1678-3_73.

[12] Jamie Morgan. *What is Centralized Log Management (CLM)?* url: https://www.missioncloud.com/blog/what-is-centralized-log-management-clm (acedido em 12/11/2021).

[13] DNSstuff. *Centralized Logging Solutions: Best Practices and Management*. Mar. de 2021. url: https://www.dnsstuff.com/centralized-log-management-solutions (acedido em 12/11/2021).

[14] Lilanga Gamage. *Why logging matters for software engineers*. Mar. de 2018. url: https://medium.com/bitsfactory/why-logging-matters-for-software-engineers-8f6eadf86329 (acedido em 13/11/2021).

[15] Nist e Emmanuel Aroms. *NIST 800-92 Guide to Computer Security Log Management*. Scotts Valley, CA: CreateSpace, 2012. isbn: 1470100401.

[16] Joaquim Abreu. «Development of a Centralized Log Management System». Tese de mestrado. Dissertação (Mestrado) - UMA, 2020, p. 2.

[17] Monika Dávideková e Michal Greguš. «Software Application Logging: Aspects to Consider by Implementing Knowledge Management». Em: ago. de 2016, pp. 102–107. doi: 10.1109/OBD.2016.22.

[18] March 7 e Chrissy Kidd. *Tracing vs Logging vs Monitoring: What's the Difference?* Mar. de 2019. url: https://www.bmc.com/blogs/monitoring-logging-tracing/ (acedido em 15/11/2021).

[19] Mar. de 2019. url: http://sangsoonam.github.io/2019/03/10/android-analyze-stack-trace-from-plain-text-log.html (acedido em 15/11/2021).

[20] Tek-Tools. *What is logging as a service? how it helps in log management*. Abr. de 2021. url: https://www.tek-tools.com/apm/what-is-laas (acedido em 05/01/2022).

[21] Patrick Hubbard. *Logging as a service helps you focus on the big picture*. Jan. de 2019. url: https://www.networkcomputing.com/cloud-infrastructure/logging-service-helps-you-focus-big-picture (acedido em 05/01/2022).

[22] SolarWinds. *Logging as a service: Benefits of Centralizing Log Data*. Nov. de 2020. url: https://logicalread.com/logging-as-a-service-benefits-of-centralizing-log-data/#.YemGOS-l35g (acedido em 05/01/2022).

[23] SentinelOne. *Logging as a service: Why the cloud is so attractive: Scalyr*. Out. de 2021. url: https://www.sentinelone.com/blog/logging-as-a-service-cloud/ (acedido em 05/01/2022).

[24] Splunk. *Why splunk*. url: https://www.splunk.com/en_us/about-us/why-splunk.html (acedido em 06/01/2022).

[25] Loggly. *Use cases - how loggly can help: Loggly*. Set. de 2020. url: https://www.loggly.com/loggly-use-cases/ (acedido em 06/01/2022).

[26] LogDNA. *About logdna*. url: https://www.logdna.com/about (acedido em 06/01/2022).

[27] SumoLogic. *Who we are and what we do*. url: https://www.sumologic.com/company/about-us/ (acedido em 06/01/2022).

[28] Logz.io. *Log analysis software that combines Elk + machine learning*. Jan. de 2022. url: https://logz.io/about-us/ (acedido em 06/01/2022).

[29] Seq Datalust. *Centralized structured logs for .NET, Java, node.js*. url: https://datalust.co/seq (acedido em 06/01/2022).

[30] Radu Gheorghe. *20+ best log management tools amp; monitoring software [2022]*. Jan. de 2022. url: https://sematext.com/blog/best-log-management-tools/ (acedido em 30/01/2022).

[31] Cloud. *Cloud server logging strategies*. Out. de 2019. url: https://realscale.cloud66.com/cloud-server-logging-strategies/ (acedido em 10/02/2022).

[32] Amruta Vatare e Pratibha Adkar. *Review Paper on Centralized and Distributed Version Control System*. Vol. 2. Mai. de 2019.

[33] Nazatul Nurlisa Zolkipli, Amir Ngah e Aziz Deraman. «Version Control System: A Review». Em: *Procedia Computer Science* 135 (jan. de 2018), pp. 408–415. doi: 10.1016/j.procs.2018.08.191.

[34] *What is Git?* url: https://www.educative.io/edpresso/what-is-git (acedido em 30/11/2021).

[35] Craig Smith. *20 Amazing GitHub Statistics*. Nov. de 2021. url: https://expandedra%20mblings.com/index.php/github-statistics/ (acedido em 30/11/2021).

[36] GitLab. *About GitLab*. url: https://about.gitlab.com/company/ (acedido em 30/11/2021).

[37] Abi Tyas Tunggal. *Bitbucket vs GitHub (Updated for 2022): UpGuard*. Jan. de 2022. url: https://www.upguard.com/blog/bitbucket-vs-github (acedido em 30/11/2021).

[38] Roshan e Sameer Mohammad. «Replacing Virtual Machine with Docker to Build or Deploy Application». Em: *SSRN Electronic Journal* 7 (jun. de 2020), pp. 54–58.

[39] Amit M Potdar et al. «Performance Evaluation of Docker Container and Virtual Machine». Em: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet'19), pp. 1419–1428. issn: 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.04.152. url: https://www.sciencedirect.com/science/article/pii/S1877050920311315.

[40] Ling-Hong Hung et al. «GUIdock: Using Docker Containers with a Common Graphics User Interface to Address the Reproducibility of Research». Em: *PloS one* 11 (abr. de 2016), e0152686. doi: 10.1371/journal.pone.0152686.

[41] Zhen Hua Liu et al. «Closing the Functional and Performance Gap between SQL and NoSQL». Em: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 227–238. isbn: 9781450335317. doi: 10.1145/2882903.2903731. url: https://doi.org/10.1145/2882903.2903731.

[42] Hazael Phiri e Douglas Kunda. «A Comparative Study of NoSQL and Relational Database». Em: *Zambia ICT Journal* 1 (dez. de 2017), pp. 1–4. doi: 10.33260/zictjournal.v1i1.8.

[43] Jatin e Shalini Batra. «MONGODB Versus SQL: A Case Study on Electricity Data». Em: jan. de 2016, pp. 297–308. isbn: 978-981-10-0286-1. doi: 10.1007/978-981-10-0287-8_28.

[44] DB-Engines. *Search Engines - DB-Engines Encyclopedia*. url: https://db-engines.com/en/article/Search%20Engines (acedido em 04/12/2021).

[45] Vinicius dos Santos. *Como reduzir uma palavra ao seu radical em python (Stemming)*. Dez. de 2021. url: https://www.computersciencemaster.com.br/como-reduzir-uma-palavra-ao-seu-radical-em-python-stemming/ (acedido em 20/12/2021).

[46] DB-Engines. *Engines Ranking*. url: https://db-engines.com/en/ranking.

[47] Anjali Chauhan. «A Review on Various Aspects of MongoDb Databases». Em: *International journal of engineering research and technology* 8 (2019).

[48] Shannon Bradshaw, Kristina Chodorow e Eoin Brazil. *MongoDB: the definitive guide: powerful and scalable data storage*. OReilly, 2020.

[49] Thais Alencar. *MongoDB: como relacionar dados*. Jul. de 2019. url: https://medium.com/@thaisdalencar/mongodb-como-relacionar-dados-3e6e8f136590 (acedido em 07/12/2021).

[50] Andrew C. Oliver. *What is Apache Solr*. Jul. de 2017. url: https://www.infoworld.com/article/3209685/why-you-should-use-apache-solr.html (acedido em 09/12/2021).
[51] CBR. *What is Apache Lucene? Search Library Has Heritage, Remains Robust*. Jan. de 2020. url: https://techmonitor.ai/techonology/hardware/apache-lucene (acedido em 09/12/2021).
[52] Cay S. Horstmann. *Core java*. Pearson Education Inc, 2019.
[53] Asaf Yigal. *Solr vs. Elasticsearch: Whos The Leading Open Source Search Engine?* Nov. de 2021. url: https://logz.io/blog/solr-vs-elasticsearch/ (acedido em 10/12/2021).
[54] Apache Solr. *JSON Query DSL*. Out. de 2017. url: https://solr.apache.org/guide/7_1/json-query-dsl.html (acedido em 10/12/2021).
[55] Gedalyah Reback. *What is Elasticsearch: Tutorial for Beginners*. Nov. de 2021. url: https://logz.io/blog/elasticsearch-tutorial/ (acedido em 11/12/2021).
[56] Clinton Gormley e Zachary Tong. *Elasticsearch: the definitive guide:* OReilly, 2015.
[57] Rafal Kuć. *Solr vs Elasticsearch: Performance Differences More [2022]*. Jan. de 2022. url: https://sematext.com/blog/solr-vs-elasticsearch-differences/ (acedido em 20/01/2022).
[58] Mustafa AKCA, Tuncay Aydoğan e Muhammer Ilkuçar. «An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and Elasticsearch». Em: *International Journal of Intelligent Systems and Applications in Engineering* 4 (dez. de 2016), pp. 8–8. doi: 10.18201/ijisae.271328.
[59] Shivang. *What is Grafana? Why Use It? Everything You Should Know About It*. Jan. de 2022. url: https://www.scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/ (acedido em 20/01/2022).
[60] Nicholas Samuel. *Kibana Alternatives: Grafana, Splunk, and Knowi*. Out. de 2021. url: https://medium.com/geekculture/kibana-alternatives-c697aef14e00#aea0 (acedido em 22/01/2022).
[61] Asaf Yigal. *Grafana vs. Kibana: The Key Differences to Know*. Nov. de 2021. url: https://logz.io/blog/grafana-vs-kibana/ (acedido em 22/01/2022).
[62] Akash Sureka. *What is Kibana Used for? –10 Important Features to Know*. url: https://www.clariontech.com/platform-blog/what-is-kibana-used-for-10-important-features-to-know (acedido em 22/01/2022).
[63] Elastic. *Metricbeat quick start: installation and configurationedit*. url: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-installation-configuration.html (acedido em 23/01/2022).
[64] Elastic. *Kibana queries and filters: Packetbeat Reference [7.16]*. url: https://www.elastic.co/guide/en/beats/packetbeat/current/kibana-queries-filters.html (acedido em 23/01/2022).
[65] Daniel Berman. *Filebeat vs. Logstash - The Evolution of a Log Shipper*. Fev. de 2020. url: https://logz.io/blog/filebeat-vs-logstash/ (acedido em 02/02/2022).
[66] Sachin Manpathak. *Kubernetes Logging and Monitoring - Part1: Fluentd architecture*. Jun. de 2021. url: https://platform9.com/blog/kubernetes-logging-and-monitoring-the-elasticsearch-fluentd-and-kibana-efk-stack-part-1-fluentd-architecture-and-configuration/ (acedido em 26/01/2022).
[67] Sadequl Hussain. *Hybrid Cloud Challenges: How to Optimize Monitoring And Analytics For ITOPS?* Jul. de 2020. url: https://www.xplg.com/what-is-logstash/#portx (acedido em 27/01/2022).

[68] Dusty Lefevre. *Logstash vs. Fluentd, a drag race*. Out. de 2020. url: https://medium.com/flowfactor/logstash-vs-fluentd-a-drag-race-d4df351ef3f6 (acedido em 28/01/2022).

[69] James Hamilton et al. «SCADA statistics monitoring using the Elastic Stack (Elasticsearch, Logstash, Kibana)». Em: (2018), TUPHA034. 5 p. doi: 10.18429/JACoW-ICALEPCS2017-TUPHA034. url: https://cds.cern.ch/record/2305659.

[70] Priya Pedamkar. *Filebeat vs Logstash: Top 5 essential differences of filebeat vs logstash*. Mar. de 2021. url: https://www.educba.com/filebeat-vs-logstash/ (acedido em 05/02/2022).

[71] Mohammad Imranur Rahman, Sebastiano Panichella e Davide Taibi. «A curated Dataset of Microservices-Based Systems». Em: *Joint Proceedings of the Summer School on Software Maintenance and Evolution*. Tampere, Finland: CEUR-WS, set. de 2019.

[72] Microsoft. *Cross-platform .NET sample microservices and container based application that runs on Linux Windows and macOS.* Jun. de 2022. url: https://github.com/dotnet-architecture/eShopOnContainers (acedido em 16/06/2022).

[73] Joseph Albahari e Ben Albahari. *C 8.0 Pocket reference: Instant help for C 8.0 programmers*. O'Reilly Media, Inc., 2020.

[74] Walter Shields. *SQL Quickstart Guide: The simplified beginner's guide to managing, analyzing, and manipulating data with SQL*. ClydeBank Technology, 2020.

[75] Da Silva Maxwell Dayvson. *Redis Essentials: Harness the power of Redis to integrate and manage your projects efficiently*. Packt Publishing, 2015.

[76] Keng Siau. «An Analysis of Unified Modeling Language (UML) Graphical Constructs Based on BWW Ontology». Em: *Journal of Database Management* 21 (jan. de 2010).

[77] Richard Knaster. *Advanced topic - domain modeling*. Out. de 2021. url: https://www.scaledagileframework.com/domain-modeling/ (acedido em 02/02/2022).

[78] Dinesh Thakur. *What is software requirement? types of requirements.* Nov. de 2013. url: https://ecomputernotes.com/software-engineering/softwarerequire%20ment (acedido em 03/02/2022).

[79] Marcus Hanikat. «Centralized log management for complex computer networks». Independent thesis Basic level (professional degree). KTH Royal Institute of Technology, 2018, p. 92.

[80] Ondřej Lomič. «Log management in distributed environment». Bachelor thesis. Masaryk University Faculty of Informatics, 2017, p. 39.

[81] Antti Vainio. «Implementation of Centralized Logging and Log Analysis in Cloud Transition». Tese de mestrado. Aalto University School of Science, 2018, p. 81.

[82] Lawrence Chung. *Non-functional requirements in software engineering*. Springer, 2012.

[83] Fabrício Ferrari Campos. *FURPS+*. Fev. de 2009. url: https://qualidadebr.wordpress.com/2008/07/10/furps/ (acedido em 04/02/2022).

[84] Coepd. *What is FURPS+?* Ago. de 2014. url: https://businessanalysttraining%20hyderabad.wordpress.com/2014/08/05/what-is-furps/ (acedido em 04/02/2022).

[85] Mubarak Sami Mohammad. *A formal component-based software engineering approach for developing trustworthy systems*. 2009.

[86] Atilla Elçi. *Handbook of Research on Contemporary Perspectives on web-based systems*. E-CONTENT GENERIC VENDOR, 2018.

[87] G R Kermode, S Sivaloganathan e T M M Shahin. «Value analysis—the technique: State of the art and future directions». Em: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 214.4 (2000), pp. 301–312. doi: 10.1243/0954405001517694. eprint: https://doi.org/10.1243/0954405001517694. url: https://doi.org/10.1243/0954405001517694.

[88] Michael Snell. *Cost-benefit analysis (2nd edition)*. Institution of Civil Engineers, 2010.

[89] Jongbae Kim e David Wilemon. «Strategic Issues in Managing Innovation's Fuzzy Front-End». Em: *European Journal of Innovation Management* 5 (mar. de 2002), pp. 27–39. doi: 10.1108/14601060210415153.

[90] Peter A. Koen et al. «1 Fuzzy Front End : Effective Methods , Tools , and Techniques». Em: 2002.

[91] Peter Koen et al. «Providing Clarity and A Common Language to the "Fuzzy Front End"». Em: *Research-Technology Management* 44.2 (2001), pp. 46–55. doi: 10.1080/08956308.2001.11671418. eprint: https://doi.org/10.1080/08956308.2001.11671418. url: https://doi.org/10.1080/08956308.2001.11671418.

[92] Jongbae Kim e David Wilemon. «Focusing the Fuzzy Front-End in New Product Development». Em: *Ramp D Management* 32 (mai. de 2003). doi: 10.1111/1467-9310.00259.

[93] Dac Teoli, Terrence Sanvictores e Jason An. *SWOT Analysis*. StatPearls Publishing, Treasure Island (FL), 2021. url: http://europepmc.org/books/NBK537302.

[94] Alan Sarsby. *SWOT ANALYSIS: A GUIDE TO SWOT for business studies students*. Spectaris Ltd, 2016.

[95] Kamal M.Al-Subhi Al-Harbi. «Application of the AHP in project management». Em: *International Journal of Project Management* 19.1 (2001), pp. 19–27. issn: 0263-7863. doi: https://doi.org/10.1016/S0263-7863(99)00038-1. url: https://www.sciencedirect.com/science/article/pii/S0263786399000381.

[96] Thomas L. Saaty. «What is the Analytic Hierarchy Process?» Em: *Mathematical Models for Decision Support*. Ed. por Gautam Mitra et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 109–121. isbn: 978-3-642-83555-1.

[97] Halil Shevket Neap e Tahir Celik. «Value of a Product: A Definition». Em: *International Journal of Value-Based Management* 12 (1999), pp. 181–191.

[98] Philip Kotler e Kevin Lane Keller. *Marketing management: Analysis, planning, implementation and control*. Prentice Hall, 2005.

[99] Alexander Osterwalder. «The business model ontology a proposition in a design science approach». Em: 2004.

[100] Alexander Osterwalder et al. *Value proposition design: How to create products and services customers want*. Wiley, 2015.

[101] Vinicius Almeida. *Cadeia de Valor: O que É e por que montar uma para a sua empresa*. Dez. de 2021. url: https://www.euax.com.br/2019/10/cadeia-de-valor/ (acedido em 12/02/2022).

[102] Ana Serafim. *O modelo de cadeia de valor de Michael Porter*. Mar. de 2013. url: https://www.portal-gestao.com/blog/699100-o-modelo-de-cadeia-de-valor-de-michael-porter.html (acedido em 12/02/2022).

[103] Joachim Karlsson. «Managing software requirements using quality function deployment». Em: *Software Quality Journal* 6 (2004), pp. 311–326.

[104] QualityOne. *QFD: Quality Function Deployment: Quality-one*. Abr. de 2021. url: https://quality-one.com/qfd/ (acedido em 15/02/2022).

[105] M. Fowler. «Design - Who needs an architect?» Em: *IEEE Software* 20.5 (2003), pp. 11–13. doi: 10.1109/MS.2003.1231144.

[106] Philippe Kruchten. «The 4+1 View Model of Architecture». Em: *IEEE Software* 12 (nov. de 1995), pp. 45–50. doi: 10.1109/52.469759.

[107] TechTerms. *Backend*. Abr. de 2020. url: https://techterms.com/definition/backend (acedido em 05/03/2022).

[108] Microsoft. *Asynchronous messaging primer*. Ago. de 2015. url: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/dn589781(v=pandp.10)?redirectedfrom=MSDN (acedido em 15/03/2022).

[109] Harshit Jain. *Polling and streaming - concept amp; scenarios*. Ago. de 2020. url: https://www.geeksforgeeks.org/polling-and-streaming-concept-scenarios/ (acedido em 17/03/2022).

[110] David C. Young. *Software Development Methodologies*. Rel. téc. Alabama Supercomputer Center, 2013.

[111] Justin Baker. *Feature branching using feature flags - dzone devops*. Mar. de 2021. url: https://dzone.com/articles/feature-branching-using-feature-flags-1 (acedido em 12/05/2022).

[112] Mojtaba Shahin, Muhammad Ali Babar e Liming Zhu. «Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices». Em: *IEEE Access* 5 (2017), pp. 3909–3943. doi: 10.1109/ACCESS.2017.2685629.

[113] Alicia Noors Mark.B. *YAML: Quick start including data processing and generation with Python and PHP*. Independently published, 2019.

[114] ReadingCraze. *Hypothesis formulation in research*. Fev. de 2015. url: http://readingcraze.com/index.php/hypothesis-formulation-research/ (acedido em 14/02/2022).

[115] Ayse Tosun et al. «On the Effectiveness of Unit Tests in Test-Driven Development». Em: *Proceedings of the 2018 International Conference on Software and System Process*. ICSSP '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 113–122. isbn: 9781450364591. doi: 10.1145/3202710.3203153. url: https://doi.org/10.1145/3202710.3203153.

[116] Paula Escudeiro e José Bidarra. «Quantitative Evaluation Framework (QEF)». Em: *Conselho Editorial/Consejo Editorial* (jan. de 2008), p. 16.

[117] Bangor et al. «The System Usability Scale (SUS): an Empirical evaluation». Em: *International Journal of Human-Computer Interaction* 24 (ago. de 2008), pp. 574–. doi: 10.1080/10447310802205776.

[118] James Lewis e Jeff Sauro. «The Factor Structure of the System Usability Scale». Em: vol. 5619. Jul. de 2009, pp. 94–103. isbn: 978-3-642-02805-2. doi: 10.1007/978-3-642-02806-9_12.

Apêndice A

Implementação

A.1 Metodologia de Desenvolvimento

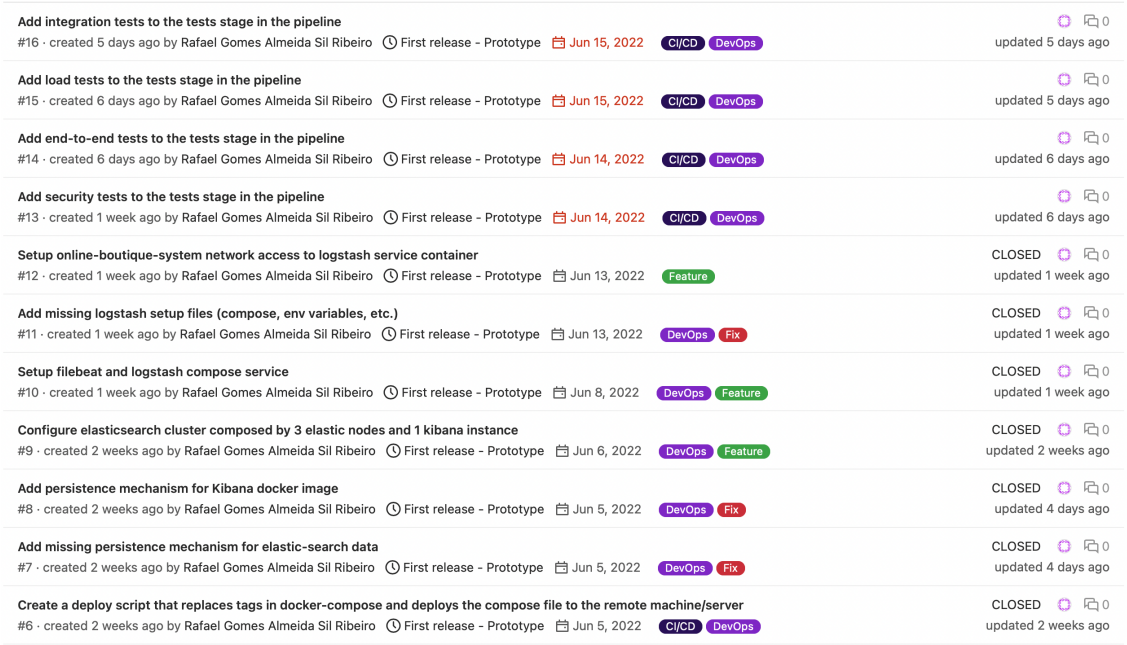

Figura A.1: *Issues* utilizados no desenvolvimento do projeto

A.2 Integração e Entrega Contínua

```
1  #!/usr/bin/env bash
2
3  TAG_VALUE=$1
4  DOCKER_HUB_USER_VALUE=$2
5  HOST_KIBANA_PORT_VALUE=$3
6  HOST_ELASTIC_PORT_VALUE=$4
7  KIBANA_PASSWORD_VALUE=$5
8  ELASTIC_PASSWORD_VALUE=$6
9  LOGSTASH_INTERNAL_PASSWORD_VALUE=$7
10 REMOTE_USER=$8
11 REMOTE_HOST=$9
12
13 sed "s/#TAG_NAME_VALUE/$TAG_VALUE/g;
   ↪  s/#DOCKER_HUB_USER_VALUE/$DOCKER_HUB_USER_VALUE/g;
14 s/#KIBANA_PORT_VALUE/$HOST_KIBANA_PORT_VALUE/g;
   ↪  s/#ES_PORT_VALUE/$HOST_ELASTIC_PORT_VALUE/g
15 s/#KIBANA_PASSWORD_VALUE/$KIBANA_PASSWORD_VALUE/g;
   ↪  s/#ELASTIC_PASSWORD_VALUE/$ELASTIC_PASSWORD_VALUE/g;
   ↪  s/#LOGSTASH_INTERNAL_PASSWORD_VALUE/
   ↪  $LOGSTASH_INTERNAL_PASSWORD_VALUE/g;" .env > .env.tmp && mv .env.tmp .env
16
17
18 sed "s/#ELASTIC_PASSWORD_VALUE/$ELASTIC_PASSWORD_VALUE/g;" env/elastic-1.env
   ↪  > env/elastic-1.env.tmp && mv env/elastic-1.env.tmp env/elastic-1.env
19 sed "s/#KIBANA_PASSWORD_VALUE/$KIBANA_PASSWORD_VALUE/g;" env/kibana.env >
   ↪  env/kibana.env.tmp && mv env/kibana.env.tmp env/kibana.env
20 sed "s/#LOGSTASH_INTERNAL_PASSWORD_VALUE/
   ↪  $LOGSTASH_INTERNAL_PASSWORD_VALUE/g;" env/logstash.env > env/logstash.env.tmp
   ↪  && mv env/logstash.env.tmp env/logstash.env
21
22 ssh $REMOTE_USER@$REMOTE_HOST "source ~/.zshrc && cd
   ↪  /Users/$REMOTE_USER/OnlineBoutiqueLogSystem && docker-compose down
   ↪  --remove-orphans && docker-compose rm -f && docker-compose pull"
23 scp -r docker-compose.yml .env env/
   ↪  $REMOTE_USER@$REMOTE_HOST:/Users/$REMOTE_USER/OnlineBoutiqueLogSystem
24 ssh $REMOTE_USER@$REMOTE_HOST "source ~/.zshrc && cd
   ↪  /Users/$REMOTE_USER/OnlineBoutiqueLogSystem && docker-compose up --build -d"
```

Excerto de código A.1: Conteúdo do ficheiro deploy.sh

Apêndice B

Quantitative Evaluation Framework

B.1 Escala de Avaliação dos Requisitos

Esta secção expõe os requisitos, e as respetivas escalas de avaliação, que estão interligados aos fatores e dimensões representados na secção 6.3.2.

B.1.1 Dimensão #1: Suportabilidade

Adaptabilidade	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
SA01 - A plataforma está disponível em diferentes browsers	Apenas acessível em 1 browser	-	Apenas acessível em 3 browsers	-	Acessível em 4 ou mais browsers
SA02 - O sistema permite a inclusão de novos formatos de logs de forma rápida e fácil	Apenas acessível em 1 browser	-	Apenas acessível em 3 browsers	-	Acessível em 4 ou mais browsers

Figura B.1: Escala de Avaliação dos Requisitos do Fator Adaptabilidade

Manutibilidade	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
SM01 - O sistema permite escalar de um modo vertical e horizontal	Não suporta qualquer tipo de escabilidade	-	Apenas suporta escalabilidade vertical	-	Suporta escalabilidade vertical e horizontal
SM02 - O sistema deve estar acompanhado de documentação completa e detalhada	Nenhuma documentação fornecida	-	Documentação incompleta	-	Documentação completa e detalhada
SM03 - O sistema não afetou as aplicações já existentes	Foi necessário alterar substancialmente os sistemas já existentes	-	Foram necessárias algumas alterações às aplicações já existentes	-	Nenhumas ou poucas alterações foram necessárias

Figura B.2: Escala de Avaliação dos Requisitos do Fator Manutibilidade

B.1.2 Dimensão #2: Eficiência

Desempenho	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
ED01 - O sistema é capaz de processar a informação rapidamente	O sistema demora mais do que 1 minuto a processar 100 logs	-	O sistema demora 20 segundos a processar 100 logs	-	O sistema demora menos de 1 segundo a processar 100 logs
ED02 - O sistema apresentam os dados em tempo real	Os sistema tem uma latência superior a 5 minutos	-	O sistema tem uma latência entre 1 a 5 minutos	-	O sistema tem uma latência inferior a 10 segundos

Figura B.3: Escala de Avaliação dos Requisitos do Fator Desempenho

Navegação	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
EN01 - O sistema apresenta uma boa estrutura e permite ao colaboradores aceder à informação de forma intuitiva	O sistema encontra-se mal estruturado e apresenta má navegabilidade	-	O sistema apresenta algumas falhas de estruturação e de navegabilidade	-	O sistema está bem estruturado e apresenta boa navegabilidade
EN02 - Os erros apresentados pelo sistema encontram-se tratados	Os erros apresentados não são tratados, expondo assim detalhes de implementação	-	Alguns erros não são tratados	-	Todos os erros apresentados são tratados pelo sistema
EN03 - As ações dos utilizadores fornecem um feedback visual	Nenhum feedback é fornecido ao utilizador	-	Apenas algumas acções fornecem feedback visual	-	Todas as acções efetuadas pelo utilizador oferecem feedback visual

Figura B.4: Escala de Avaliação dos Requisitos do Fator Navegação

B.1.3 Dimensão #3: Funcionalidade

Qualidade do Conteúdo	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
FQ01 - Todo o conteúdo apresentado é relevante para o utilizador	O sistema presenta muito conteúdo que não tem relevância para o utilizador	-	O sistema apresenta algum conteúdo irrelevante ou redundante	-	O sistema não apresenta conteudo redudante ou irrelevante
FQ02 - O contéudo textual encontra-se bem escrito e as frases bem construídas	O conteúdo apresenta falhas graves no que concerne a ortografia e sintaxe	-	Alguns erros ortográficos e de sintaxe existem	-	Não existem a presença deste tipo de erros
FQ03 - A informação encontra-se organizada por área funcional	A informação encontra-se espalhada e não é dividida por área funcional	-	Alguma informação não se encontra dividida por área funcional	-	Toda a informação encontra-se dividida por área funcional

Figura B.5: Escala de Avaliação dos Requisitos do Fator Qualidade do Conteúdo

Alertas	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
FA01 - O sistema envia alertas aos utilizadores aquando da existência de anomalias	O sistema não é capaz de mandar alertas	-	O sistema manda alertas apenas num determinado formato	-	O sistema é capaz de enviar alertas em mais do que um formato
FA02 - O sistema permite configurar o threshold no qual os alertas se devem basear	Não configurável	-	Configurável apenas via código fonte	-	Configurável a partir de um ficheiro de configurações

Figura B.6: Escala de Avaliação dos Requisitos do Fator Alertas

Visualização Gráfica	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
FG01 - O sistema apresenta a informação em diversos formatos gráficos	O sistema não é capaz de apresentar a informação graficamente	O sistema apenas suporta 1 tipo de gráfico	O sistema apenas suporta 2 tipos de gráficos	-	O sistema suporta 3 ou mais tipos de gráficos
FG02 - O sistema permite filtrar a informação dos gráficos	Não permite a capacidade de filtrar a informação dos g´raficos	-	Os filtros são poucos e limitados a pequenos intervalos de tempo	-	Os filtros são vastos e permitem o utilizador especificar um dado intervalo de tempo
FG03 - Apresenta valores estatísticos relativos à informação dos logs, como a média e o desvio padrão	Não apresenta valores estatísticos	Apenas apresenta a média	Apresenta a média e o desvio padrão	-	Apresenta diversos valores, como a média, desvio padrão, mediana

Figura B.7: Escala de Avaliação dos Requisitos do Visualização Gráfica

Visualização Textual	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
FT01 - O sistema apresenta a informação dos logs em formato textual	O sistema não é capaz de apresentar a informação em forma textual	-	Não apresenta toda a informação relevante de forma textual	-	Apresenta a informação relevante de forma textual
FT02 - O sistema permite pesquisar por informação através de texto completo	Não oferece pesquisa por texto	Permite pesquisar mas até um determinado número de caracteres	Permite pesquisar mas os resultados têm informação não desejada	-	Permite pesquisar e os resultados vão de encontro com o texto introduzido
FT03 - Fornece sugestões de correção do texto introduzido pelo utilizador	Não oferece sugestões	-	As sugestões nem sempre vão de encontro com o pretendido do utilizador	-	As sugestões são relevantes e vão de encontro com as necessidades do utilizador
FT04 - O sistema permite ordenar e agrupar os resultados obtidos	Não oferece ordenação nem agrupamento	Apenas oferece ordenação	Apenas oferece agrupamento	-	Oferece ordenação e agrupamento dos resultados

Figura B.8: Escala de Avaliação dos Requisitos do Fator Visualização Textual

Tratamento da Informação	Wfk - Fullfilment (%)				
Requisito	**0**	**25**	**50**	**75**	**100**
FI01 - O sistema é capaz de tratar logs em diversos formatos	O sistema não é capaz de tratar logs	O sistema apenas é capaz de tratar logs em um formato	O sistema apenas é capaz de tratar logs em dois formatos	O sistema é capaz de tratar logs em todos com ocasionais perdas de informação	O sistema é capaz de tratar logs em todos os formatos sem perdas de informação
FI02 - O sistema remove os logs a partir de uma determinada data	O sistema não é capaz de remover os logs	-	O sistema remove os logs mas fâ-lo de uma forma demorada	-	O sistema é capaz de remover os logs de forma rápida e para uma determinada data
FI03 - O sistema garante a autenticidade e integridade da informação que é tratada	O sistema não garante quer a autenticidade ou integridade da informação		O sistema apenas garante umas delas	-	O sistema garante autenticidade e integridade da informação

Figura B.9: Escala de Avaliação dos Requisitos do Fator Tratamento da Informação

Utilizador	**Wfk - Fullfilment (%)**				
Requisito	**0**	**25**	**50**	**75**	**100**
FU01 - O sistema permite a autenticação do utlizador	O sistema não oferece mecanismos de autenticação	-	O sistema apresenta falhas no sistema de autenticação	-	O sistema é capaz de autenticar o utilizador
FU02 - O sistema permite recuperar a palavra passe	O sistema não oferece mecanismos de recuperação da palavra passe	-	O sistema nem sempre envia a palavra passe para o e-mail do utilizador	-	O sistema permite recuperar a senha enviando-a sempre para o e-mail do utilizador
FU03 - O sistema restinge as funcionalidades consonte o cargo do utilizador	O sistema não restringe as funcionalidades	-	O sistema apenas restringe alguma das funcionalidades supostas	-	O sistema restringe todas as funcionalidades supostas
FU04 - O sistema permite a configuração de utilizadores	O sistema não permite configurar utilizadores	O sistema apenas permite adicionar utilizadores	O sistema permite adicionar e remover utilizadores	-	O sistema permite adicionar, remover e atualizar utilizadores

Figura B.10: Escala de Avaliação dos Requisitos do Fator Utilizador