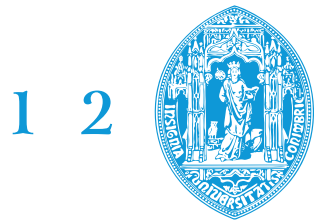
UNIVERSIDADE D
COIMBRA

Catarina Mendes Proença

EVOLUÇÃO DE UMA PLATAFORMA DE MONITORIZAÇÃO

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software orientada pelo Engenheiro João Almeida e pela Professora Maria José Marcelino e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2022



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Catarina Mendes Proença

Evolução de uma plataforma de monitorização

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software orientada pelo Engenheiro João Almeida e pela Professora Maria José Marcelino e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2022

Resumo

Atualmente, as instituições bancárias têm sido obrigadas a cumprir um número cada vez maior de requisitos para que possam responder às necessidades dos seus clientes e para que estes se sintam protegidos e seguros nas suas transações financeiras. Tendo isto em consideração, as infraestruturas existentes têm vindo a ser colocadas sobre a pressão de evoluir continuamente com o fim de fornecer mais segurança e confiabilidade aos seus clientes.

O presente documento descreve o estágio que ocorreu no seguimento de um projeto que se dedica à gestão de um sistema bancário que integra, numa aplicação central, todas as funcionalidades de uma instituição bancária internacional. Considerando o âmbito descrito, o estagiário esteve a trabalhar numa plataforma já existente, denominada Kamino. Esta é uma plataforma de monitorização preventiva desenvolvida no contexto do projeto que permite monitorizar as principais tarefas realizadas nas diferentes partes de um banco, recolhendo métricas relacionadas com os processos realizados internamente. Dentro destas, destacam-se essencialmente dois grandes grupos: métricas gerais de infraestrutura e métricas especificas de negócio. As métricas gerais de infraestrutura, comuns a qualquer outro *software*, referem-se a valores de CPU, RAM ou, por exemplo, a métricas relacionadas com a base de dados. Já as métricas *custom* do negócio são por exemplo referentes ao número de SMSs enviados, à distribuição de pedidos da agência bancária ou à duração dos pedidos efetuados ao servidor. A análise destes tipos de métricas permite detetar a operacionalidade do sistema e garantir que este se encontra a funcionar com qualidade. Assim sendo, esta plataforma apoia a equipa de suporte do projeto, assegurando uma maior qualidade no serviço prestado, uma vez que torna mais ágil o diagnóstico e a consequente resolução de potenciais problemas.

Como tal, e uma vez que a versão existente não oferecia resposta às necessidades dos *stakeholders*, o contributo do estagiário contemplou a evolução dessa plataforma para uma solução mais atual e adaptada às condicionantes existentes. Com esse fim, foi necessário proceder à reestruturação do modelo de propagação de métricas e à otimização do processo com vista à redução da latência na obtenção e da visualização das mesmas. Foi ainda necessário rever toda a alarmística existente e otimizá-la de forma a que não possua uma verbosidade excessiva e não perturbasse nem desfocasse a atenção dos utilizadores da plataforma.

Palavras-Chave

Sistema Bancário, Telemetria de Software, Software de Manutenção Preventiva

Abstract

Currently, banking institutions have been required to comply with an increasing number of requirements, so they can respond to its customers needs, namely protection and safety on their financial transactions. Taking this into account, the existing infrastructures have been put under pressure to evolve continuously in order to provide more security and reliability to its customers.

This document describes the internship that took place following a project dedicated to the management of a banking system that integrates, in a central application, all the functionalities of an international banking institution. Considering the scope described, the intern worked on the already existing platform called Kamino. This is a preventive monitoring platform developed in the context of the project that allows monitoring the main tasks carried out in the different parts of a bank, collecting metrics related to the processes carried out internally. Within these, two main groups stand out: general infrastructure metrics and business-specific metrics. General infrastructure metrics, common to any other software, refer to values of CPU, RAM or, for example, metrics related to the database. The custom metrics of the business are, for example, referring to the number of SMSs sent, the distribution of requests by banking agency or the duration of requests made to the server. The analysis of these types of metrics makes it possible to detect the system's availability and ensure that it is working with quality. Therefore, this platform supports the project's support team, ensuring a higher quality of the service provided, as it makes the diagnosis and consequent resolution of potential problems more agile.

Since the existing version did not respond to the stakeholders needs, the intern's contribution contemplated the evolution of this platform to a more up-to-date solution, adapted to existing constraints. To reach this end, it was needed to restructure the metrics propagation model and optimize the process that reduced latency on getting and viewing those values. It has still been necessary to review and optimize the existing alarm dashboard so it would not have an excessive verbosity and it would not disturb or distract the attention of the platform users.

Keywords

Banking System, Software Telemetry, Preventive Maintenance Software

Agradecimentos

Em primeiro lugar, gostaria de agradecer à Critical Software (CSW) pela oportunidade da realização do estágio que me deram, e a todas as pessoas que estiveram envolvidas ao longo dos últimos meses.

Em especial, ao meu orientador João Almeida, por toda a disponibilidade, apoio e pelo constante suporte no desenvolvimento deste projeto, tendo me ajudado a evoluir tanto a nível profissional como a nível pessoal. Agradecer ao Telmo Simões, por toda a atenção e preocupação demonstrada ao longo deste período. Agradecer também a todos os membros da equipa deste projeto, tanto pela ajuda incansável no processo de integração como em qualquer dúvida ou problema que me pudesse surgir.

À minha orientadora do Departamento de Engenharia Informática (DEI), professora Maria José Marcelino, por todo a disponibilidade e ajuda constante em questões do âmbito curricular.

Quero agradecer também a todos os meus amigos que, desde Videmonte a Coimbra, me apoiaram incondicionalmente e em todos os momentos do meu percurso académico.

Por último, quero agradecer aos meus pais e à minha irmã por terem estado presentes em todos os momentos da minha vida. Por toda a força que me transmitiram e por nunca me terem deixado cair nos momentos mais difíceis. Sem esta ajuda, a conclusão desta etapa nunca seria possível, e por isso, vou estar-lhes eternamente grata.

Índice

Acrónimos

AC *Acceptance Criteria*.

AIX *Advanced Interactive eXecutive*.

API *Application Programming Interface*.

APM *Application Performance Monitoring*.

AWS *Amazon Web Services*.

CSW Critical Software.

DBMS *Database Management System*.

DEI Departamento de Engenharia Informática.

EOL *End of line*.

FES *Frontend Scheduler*.

FTP *File Transfer Protocol*.

IaC *Infrastructure as Code*.

ID Identificador Único.

IT *Information technology*.

JMX *Java Management Extensions*.

LTS *Long Term Support*.

LXC *Linux Containers*.

NASA *National Aeronautics and Space Administration*.

RDBMS *Relational Database Management System*.

RTT *Round Trip Time*.

SaaS *Software as a Service*.

SFTP *SSH File Transfer Protocol*.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O objetivo deste documento é detalhar e reportar todas as ações decorrentes do estágio no âmbito do Mestrado em Engenharia Informática da Universidade de Coimbra que decorreu na Critical Software (CSW).

Este capítulo reflete o contexto do estágio e da entidade promotora envolvida, definindo também os objetivos, as várias fases do estágio e a estrutura do documento.

1.1 Contextualização

A entidade promotora deste estágio foi a CSW, uma empresa fundada em 1998 e que privilegia o desenvolvimento de sistemas críticos nas áreas de finanças, energia, comunicações, transportes, aeronáutica, espaço, saúde e estado. Esta empresa tem trabalhado com alguns parceiros bastantes conceituados tais como a *National Aeronautics and Space Administration* (NASA) ou as Forças Armadas do Reino Unido. Devido à necessidade de expansão, a empresa alargou-se a países como Inglaterra, Alemanha e Estados Unidos.

1.2 Projeto de estágio

O projeto em questão decorre no seguimento de um projeto entre a CSW e uma instituição bancária internacional. Esta entidade bancária foi fundada em 1993 e é uma das maiores instituições privadas do seu país. Atualmente, a satisfação dos seus clientes é uma das suas principais preocupações tendo por isso apostado na modernização dos sistemas financeiros e tecnológicos.

Com esse fim, esta entidade propôs-se a modernizar os sistemas financeiros tendo surgido o projeto alvo do estágio. Este projeto começou em 2011 e tinha como o principal objetivo a criação de um sistema de informação que uniformizasse os vários serviços disponibilizados nos balcões de *backoffice*, tendo a equipa o desafio de integrar num único sistema todos os outros sistemas utilizados pela instituição

bancária.

Simultaneamente, foram identificados outros objetivos tais como:

- Introdução da gestão documental de processos através de um sistema de gestão de documentos que suporte arquivos digitais de clientes;
- Controlo e mitigação de riscos operacionais através da implementação de processos de negócio;
- Garantir a correta execução dos processos, guiando o utilizador na sua execução e usando mecanismos para controlar, automatizar e monitorizar estes processos;
- Desenvolvimento de uma interface integrada que auxilie os utilizadores nas operações realizadas sobre o *core* bancário.

No contexto do projeto, foi desenvolvida uma solução de monitorização de desempenho de aplicações, denominada de Kamino. Esta plataforma é um sistema de *Near Real Time Applications Monitoring* que facilita a manutenção preventiva do projeto por parte da equipa de suporte do projeto. Para além de permitir a monitorização de todo o projeto e das suas métricas, esta plataforma fornece ainda tendências e perceções com base no histórico de dados que são recolhidos. A plataforma será descrita e detalhada com mais precisão na secção 2.2.

1.3 Objetivos do estágio

Este estágio está integrado no âmbito da plataforma de monitorização Kamino e teve como objetivo a evolução desta plataforma para uma solução mais atual e que esteja adaptada às condicionantes que existem atualmente, uma vez que a versão existente já não oferece resposta às necessidades atuais dos *stakeholders*.

Como resultado do trabalho efetuado durante o estágio, identificaram-se objetivos específicos e bem definidos para o mesmo, sendo eles os seguintes:

- Montar uma infraestrutura para desenvolvimento e testes da plataforma a melhorar;
- Reduzir a latência na transmissão de métricas;
- Rever e alterar, nos casos necessários, a frequência de obtenção de métricas;
- Rever e reconfigurar, nos casos necessários, a alarmística;
- Estabilizar e melhorar a robustez da plataforma;

Em suma, inicialmente foi fulcral criar as condições necessárias para que o desenvolvimento e a implementação de melhorias da plataforma não afetassem o que se encontra em produção.

Numa fase posterior do desenvolvimento foram revistos e otimizados aspetos referentes ao modelo de propagação de métricas e à latência que, neste momento, está inerente à obtenção e visualização das mesmas. Foi ainda revista e otimizada toda a questão referente à alarmística, com o objetivo de reduzir a verbosidade excessiva dos alarmes atuais, tendo sido necessário analisar o comportamento de cada alarme, a fim de perceber as condições da sua configuração.

Por fim, foram efetuadas melhorias relativamente à estabilização e robustez da plataforma para que fosse possível assegurar mais garantias quanto à disponibilidade da mesma.

1.4 Estrutura do documento

Este relatório encontra-se dividido em 4 capítulos:

- Capítulo 1 - Fornece o contexto necessário acerca do estágio bem como os objetivos definidos;
- Capítulo 2 - Corresponde ao estado da arte e consiste na análise de tecnologias alternativas às que já existem e na justificação da sua escolha;
- Capítulo 3 - Descreve os principais processos de engenharia adotados durante o estágio, incluindo a organização da equipa e o modelo de software definido. Apresenta também o plano de trabalho seguido realizado e expõe o processo de gestão de riscos efetuado;
- Capítulo 4 - Contém todas as informações referentes à análise funcional, correspondendo à primeira fase de desenvolvimento realizada;
- Capítulo 5 - Descreve os processos de garantia de qualidade realizados, destacando-se a análise de riscos e o plano de testes seguido;
- Capítulo 6 - Detalha o processo de implementação realizado, incluindo também a análise técnica e os testes elaborados em cada uma das tarefas.
- Capítulo 7 - Contém algumas considerações e conclusões finais acerca do estágio, incluindo ainda uma análise sobre o trabalho futuro a realizar na plataforma.

Capítulo 2

Estado da Arte

Este capítulo tem o objetivo de fornecer uma visão geral das várias plataformas e tecnologias atualmente relacionadas com a plataforma estudada e o de fornecer, simultaneamente, algumas alternativas à sua utilização existentes no mercado. Esta análise decorreu inicialmente sobre as componentes que compõem a plataforma em si (equivalente às partes) e posteriormente sobre a plataforma como uma solução completa (equivalente ao todo).

Durante esta análise é também apresentada uma comparação entre as diferentes alternativas à plataforma encontradas com objetivo de encontrar melhores soluções para o problema em questão. Consequentemente, é apresentada uma conclusão para a escolha, ou não, das mesmas tendo sempre em consideração as que mais se enquadram com os objetivos do projeto.

Esta decisão deve ser sempre realizada de forma ciente e consciente para que seja possível responder não só às necessidades e particularidades do projeto em questão, mas também às necessidades do cliente.

2.1 *Application Performance Monitoring*

Application Performance Monitoring (APM) refere-se ao processo de gestão do desempenho de aplicações de *software* que ajudam a garantir a qualidade de um determinado serviço através da análise de métricas e da configuração de toda a alarmística necessária.

Recorrendo a soluções APM, as empresas são capazes de monitorizar o desempenho dos seus produtos e de garantir que estes correspondem ao desempenho esperado. Estas soluções têm o objetivo de auxiliar na deteção e correção de problemas de desempenho fazendo com que estes erros não causem impacto no cliente final. As melhores soluções APM permitem ainda que as equipas (no caso, a equipa de suporte) consigam integrar não só o desempenho de aplicações como também de alguns resultados referentes a indicadores do negócio em si.

2.1.1 Tendências

No passado, o desenvolvimento de aplicações com vista à monitorização e à recolha de métricas não era um aspeto essencial, dando-se mais ênfase à implementação em si e à maneira como se lidava com os problemas.

Posteriormente, e considerando o aumento das exigências relativamente à disponibilidade das plataformas, foi necessário encontrar soluções que permitissem agir não só reativamente como também preventivamente.

Posto isto, o desenvolvimento orientado à monitorização de aplicações tornou-se um aspeto *must have* de qualquer plataforma, com o objetivo de entregar melhores resultados aos clientes. Assim sendo, foi necessário estabelecer um novo requisito que permita monitorizar completamente (de ponta a ponta) o desempenho de aplicações.

Atualmente, as equipas de *Information technology* (IT) que operam em ecossistemas cada vez mais orientados à *cloud* têm a necessidade de monitorizar constantemente todo o processo e todas suas etapas e componentes individuais. Esta monitorização em *real-time* deve ser automatizada e orientada a inteligência artificial, sendo esta a única maneira viável da equipa encontrar, resolver e prevenir problemas de desempenho da aplicação em tempo real.

A título de exemplo, a plataforma Dynatrace e um dos seus módulos (Davis ©) automatizam a análise de causas, recorrendo a inteligência artificial, dando resposta a arquiteturas *cloud* mais complexas. [24]

2.2 Kamino

No contexto do projeto onde este estágio se insere, foi desenvolvida a plataforma Kamino, que consiste numa solução APM que tem desde então ajudado a Critical Software (CSW) a entregar melhores resultados ao cliente deste projeto. Esta plataforma é um sistema de *Near Real Time Applications Monitoring* que facilita a manutenção preventiva por parte da equipa de suporte, fornecendo também tendências e perceções com base no histórico de dados.

O Kamino recorre a uma arquitetura baseada na Elastic Stack, uma ferramenta que vai ser detalhada na secção 2.2.1, tendo sido adaptada para corresponder às restrições que a entidade bancária em questão envolve.

Uma vez que a interface fornecida pela Elastic Stack, ou mais especificamente pela sua componente Kibana, não apresentava, na altura, as características e funcionalidades pretendidas, optou-se por juntar à Elastic Stack a solução do Grafana, permitindo assim uma visualização mais clara e completa das métricas, alertas e *dashboards* a monitorizar. A fim de atestar, no momento do desenvolvimento, a validade da decisão de recorrer ao Grafana em detrimento do Kibana, foi elaborada uma comparação na secção 2.2.4.

Posto isto, aquando do desenvolvimento inicial da plataforma, esquematizou-

se a arquitetura de forma detalhada na figura 2.1, que será alvo de debate no decorrer desta análise.

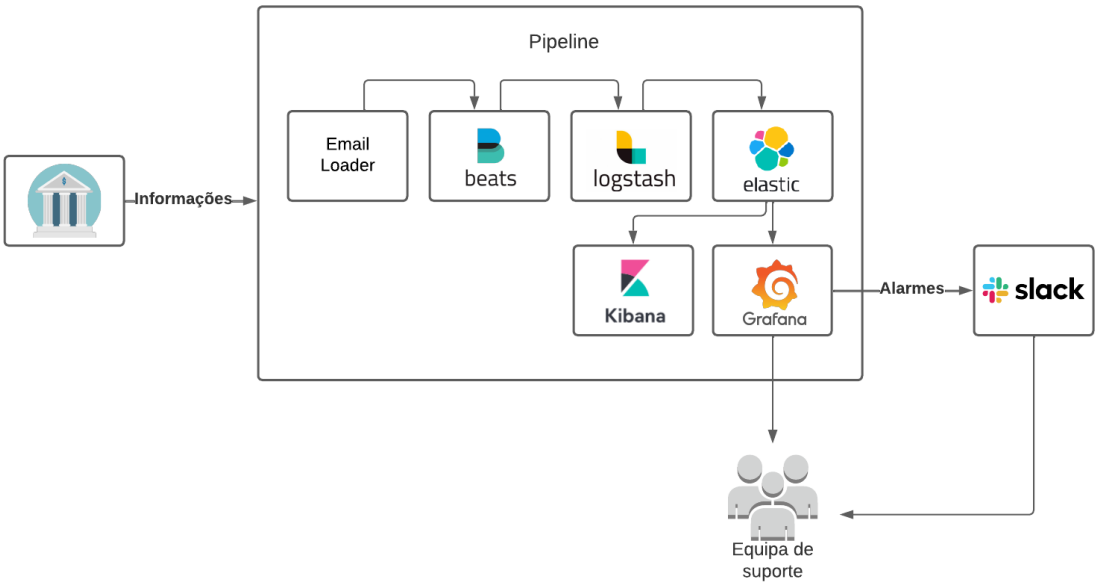


Figura 2.1: Arquitetura da plataforma Kamino

A plataforma Kamino tem ajudado a equipa de suporte do projeto da CSW nas suas tarefas diárias ao monitorizar o comportamento das aplicações (desempenho, disponibilidade, entre outros) e ao criar e receber alertas por limites de inatividade ou de desempenho. Para além disto, acompanha e compara o desempenho ao longo do tempo permitindo que se identifiquem as causas dos problemas e que os descubram antes que estes aconteçam.

Em suma, o Kamino reúne os seguintes aspetos em termos de arquitetura:

- Solução horizontalmente escalável;
- Solução robusta de pesquisa, de gestão de eventos e de análise de dados.

Já a nível da interface, apresentada na figura 2.2, permite ao utilizador pesquisar, visualizar, receber alertas e acompanhar as métricas presentes e identificadas para a plataforma. O utilizador pode explorar e partilhar *dashboards* com a sua equipa promovendo assim uma cultura orientada aos dados.

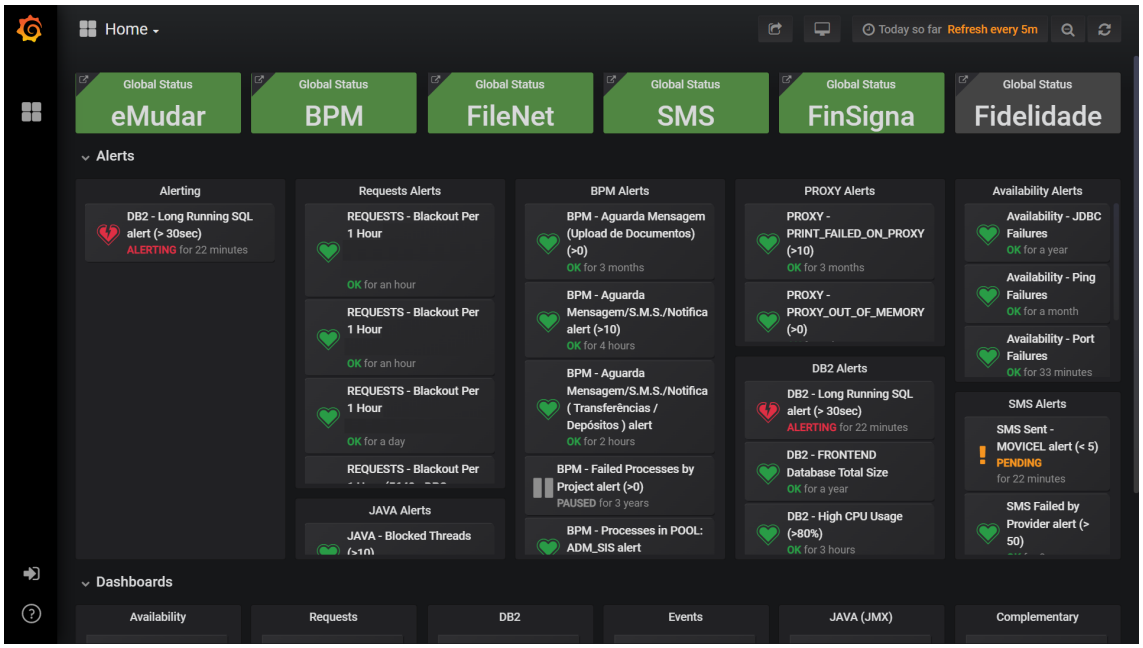


Figura 2.2: Interface do Kamino

2.2.1 Elastic Stack

A Elastic Stack [14] corresponde a um conjunto de produtos que permitem recolher e processar os dados de várias fontes e formatos, colocar esses dados num armazenamento de dados centralizado e ainda visualizar e analisar os dados de forma eficaz.

Esta plataforma subdivide-se em quatro produtos *open-source*, conforme indicado na figura 2.3: Beats, Logstash, Elasticsearch e Kibana.

Figura 2.3: Organização da Elastic Stack [20]

Beats

O Beats é um conjunto de agentes de dados (Filebeat, Metricbeat, Packetbeat, entre outros) com a finalidade de **enviar** dados de centenas ou milhares de sistemas para o Logstash ou para o Elasticsearch.

O Filebeat é utilizado no projeto em questão e inclui módulos que simplificam a recolha e processamento de métricas de sistema e de negócio.

Logstash

O Logstash é um acumulador de *logs* que **recolhe** dados de várias fontes de entrada, executa as diferentes transformações e envia os dados para vários destinos de saída. Esta ferramenta faz dinamicamente as ingestões, transformações e o envios de dados, independentemente do formato e da complexidade dos mesmos.

Apesar da Elasticsearch representar o destino de saída de dados mais utilizado, o Logstash permite reencaminhar estes dados para qualquer outro sítio, proporcionando uma maior flexibilidade na sua utilização.

Elasticsearch

A Elasticsearch é um mecanismo de **pesquisa e de análise** de texto, baseada na biblioteca de pesquisa Apache Lucene. Esta ferramenta permite armazenar, pesquisar e analisar grandes volumes de dados de forma rápida e em tempo real.

Representa uma base de dados NoSQL, ou não relacional, o que significa que armazena os dados de forma não estruturada. Normalmente recorre-se à Elastic Search quando são efetuadas grandes pesquisas de texto ou quando as bases de dados *Relational Database Management System* (RDBMS) não apresentam um bom desempenho. Esta ferramenta é ainda altamente personalizável e extensível através de *plugins*.

Kibana

O Kibana é uma ferramenta que constitui a camada de **visualização** da Elasticsearch e que fornece aos utilizadores a capacidade de analisar e visualizar os dados indexados pela Elasticsearch. Esta ferramenta facilita também o desenvolvimento sobre a Elastic Stack ao apresentar funcionalidades que auxiliam na gestão da toda a *stack*, como por exemplo a visualização de métricas acerca do estado das várias componentes da *suite* (Beats, Logstash e ElasticSearch).

A fim de facilitar a comparação com uma ferramenta semelhante mais adiante nesta análise, foram levantados alguns prós e contras desta ferramenta.

Prós:

- Interface eficiente e fácil de utilizar;
- Exportação de dados com suporte para os formatos `PDF` e `CSV`;
- Deteção complexa de erros e anomalias.

Contras:

- Limitado pela Elastic Stack;

2.2.2 Elastic Stack vs Prometheus

A Elastic Stack é, como já mencionado anteriormente, uma stack NoSQL utilizada para monitorização que recolhe dados de qualquer tipo de fonte e com qualquer formato, que os analisa e os mostra em tempo real.

Por outro lado, existe a alternativa do Prometheus que devemos considerar e analisar. Esta solução foi desenhada de forma semelhante à Elastic Stack, com o intuito de monitorizar e recolher métricas contendo também toda a questão de alerta de eventos.

De forma a podermos comparar as duas ferramentas com mais detalhe, foi elaborada a tabela 2.1.

Nota: Com a cor amarela foram assinaladas as características que não se enquadram nosobjetivos do projeto. Com a cor verde foram assinaladas as características que correspondem às necessidades atuais do projeto.

Características	**Elastic Stack** [14]	**Prometheus** [33]
Open source	Sim	Sim
Scope	Personalizável	Recolha de métricas
Integração de alertas	Email, Slack	Email, Slack
Base de dados	*Distributed Document Store* [8]	*Timeseries* DBMS [35]
Schema	Livre	Nome da métrica e pares chave-valor
Tipo de dados	Todo tipo de dados	Números

Tabela 2.1: Comparação entre Elastic Stack e Prometheus

Como podemos observar, e recorrendo também à identificação por cores, a Elastic Stack apresentou melhores resultados e justificou a escolha para a atual integração no projeto em questão. Dada a sua natureza e as suas necessidades, a possibilidade de poder ter liberdade quanto ao tipo de dados a monitorizar e quanto à sua origem torna a Elastic Stack mais indicada do que o Prometheus.

2.2.3 Grafana

O Grafana é uma solução *open-source* para análise de dados e métricas e para monitorização de aplicações com *dashboards* personalizáveis.

Este tipo de ferramenta ajuda a melhorar a produtividade das equipas e das organizações como um todo, podendo identificar-se os seguintes benefícios:

- Menos tempo de *debugging*;
- Resolução de problemas e de erros mais rápida;

- UI de visualização mais rica;
- Aumento da produtividade.

O facto de ser *open-source* torna-a expansível através de um sistema de *plugins*, permitindo que os utilizadores finais possam criar *dashboards* de monitorização mais complexos. Posto isto, foram levantados alguns prós e contras desta ferramenta.

Prós:

- Variedade de modelos e *plugins*;
- Suporte para várias fontes de dados (Prometheus, EleasticSeach, entre outros);
- *Dashboards* e alertas personalizáveis.

Contras:

- A recolha de dados e o armazenamento têm de ser configurados separadamente.

2.2.4 Grafana vs Kibana

Considerando que estas duas ferramentas são ambas soluções de monitorização e de análise de dados, considerou-se que seria interessante analisá-las e identificar as potenciais diferenças, tentando também justificar a escolha do Grafana para a integração principal do projeto em questão [42].

Características	**Kibana** [23]	**Grafana** [17]
Sistema Operativo	Linux, Mac e Windows	Linux, Mac e Windows
Integração de alertas	Sim	Sim
Open source	Sim	Sim
Integrações	Elastic Stack	Várias plataformas e bases de dados
Suporte de dados	Elasticsearch	InfluxDB, AWS, MySQL, PostgresSQL, entre outros
Scope	Logs	Métricas
Configuração	Deve ser configurado com a mesma versão da *stack*	Fácil uma vez que é independente
Visualização		

Tabela 2.2: Comparação entre Kibana e Grafana

Fundamentalmente, inferiu-se que uma das grandes diferenças é referente à sua finalidade e ao tipo de dados onde cada um atua. Por um lado, temos o Grafana que se foca essencialmente na análise e na visualização de alertas, métricas e *dashboards*. Por outro, temos o Kibana que, sendo parte integrante do Elasticsearch, se foca mais na análise de *logs* referentes a eventos de todo o processo.

Já a nível da visualização, tanto o Grafana como o Kibana oferecem várias opções de personalização que permitem que os utilizadores filtrem e visualizem os dados da forma muito livre. Em ambas as ferramentas é possível alterar as cores dos *dashboards*, de *labels*, dos eixos de gráficos ou até o tamanho dos *dashboards*. No entanto, o Grafana possui uma gama mais ampla de opções de personalização, facilitando também a alteração das configurações através de vários painéis de edição e de *dropdowns*. Devido à sua versatilidade e às poderosas operações que possuem, como a criação de *dashboards* para qualquer tipo de dados e de fontes, os *dashboards* do Grafana permitem uma visualização mais rica, tornando esta ferramenta preferível em relação às suas concorrentes.

Considerando o âmbito do projeto e após a análise dos prós e contras de cada ferramenta mencionados ao longo desta secção, conclui-se que, de facto, o Grafana continua a ser a ferramenta mais adequada a adotar, uma vez que apresenta as características mais apelativas e que o seu foco principal corresponde ao pretendido (alertas, métricas e *dashboards*). Apesar de não ter sido adotado como ferramenta de visualização, o Kibana é utilizado para apoio ao desenvolvimento do projeto e da plataforma, nunca sendo usado em detrimento do Grafana.

2.2.5 Docker

O Docker é uma plataforma *open-source* para desenvolver e executar aplicações, onde estas se encontram separadas e independentes da infraestrutura com o objetivo de poder ser possível desenvolver e fornecer um software com mais rapidez e eficácia [41]. Recorrendo às metodologias que esta *framework* fornece, é possível gerir a infraestrutura da mesma forma que se gerem as várias aplicações, reduzindo significativamente o atraso entre a escrita de código e sua execução em produção.

2.2.6 Docker vs Linux Containers

O Docker recorre ao *kernel* e aos recursos do Linux para isolar os processos em *containers*, podendo ser então executados de forma independente. O objetivo dos *containers* é criar essa independência mantendo a segurança que se teria caso fossem sistemas separados.

O *Linux Containers* (LXC) é um software *open-source* que representa uma solução criada para correr no sistema operacional Linux de forma a implementar os *containers* do sistema operativo.

Enquanto o LXC se foca na contentorização do sistema, o Docker ganha vantagem ao focar-se nas várias aplicações.

Analisando estas duas ferramentas, o Docker ganha vantagem nos seguintes aspetos [16]:

- **Portabilidade**: Agrupa numa só aplicação todas as configurações necessárias que necessita para correr sem problemas em qualquer máquina ou sistema operativo;
- **Flexibilidade**: Pode ser facilmente implementado e executado noutros ambientes, independentemente do sistema operativo ou das configurações;
- **Uso eficiente dos recursos** - Os *containers* não possuem núcleos de CPU e de memória próprios ou virtuais, facilitando a utilização de recursos diretamente do *host*.
- **Leve e rápido**: Não é necessário configurar imagens da máquina virtual tornando isto mais fácil e rápido de configurar, ocupando também menos espaço no disco.
- **Popularidade**: Tentou alinhar-se com o seu mercado-alvo e tendo conseguido encaixar-se em muitas organizações e empresas.

Considerando todas as características e vantagens mencionadas quanto ao Docker, foi possível inferir que este representa uma melhor solução. Esta plataforma integrava a implementação do projeto e, considerando a análise efetuada, entendeu-se que se devia manter.

2.3 Alternativas ao Kamino

Nesta secção é apresentada uma descrição para cada ferramenta identificada bem como os prós e contras de cada uma delas [18]. No final são consideradas todas as características e é elaborada uma conclusão final.

2.3.1 SigNoz

A SigNoz [30] é uma plataforma de monitorização *full-stack* e *open-source*, que promove uma interface unificada para a análise de métricas sem que haja necessidade de recorrer a outras ferramentas, como o Prometheus. Contém também tracking relativo a métricas de infraestrutura, assim como as médias de utilização do CPU ou o uso de memória do sistema. Esta plataforma é considerada a alternativa mais viável à plataforma de licença paga abordada na subsecção 2.3.3.

Prós:

- Fluxo de dados transparente;
- Instalação fácil em *single node*;

- Filtragem baseada em *tags*.

Contras:

- *Backend* precisa de ser instalado no mesmo *host* da aplicação;
- Documentação confusa;
- Custo associado à sua manutenção, assim como a memória e os seus *upgrades*.

2.3.2 Apache SkyWalking

A Apache SkyWalking [2] é uma plataforma APM e *open-source*, focada essencialmente na monitorização de sistemas distribuídos, o que inclui micro serviços, arquiteturas *cloud-native* e arquiteturas baseadas em *containers*, como o Docker.

Prós:

- Capacidade de manter o projeto leve e transparente;
- Fácil de integrar;
- Configurável através de *plugins*, permitindo uma visualização mais robusta;

Contras:

- Interface complexa no início;
- Custo variável consoante a instalação e manutenção da APM.

2.3.3 Dynatrace

A Dynatrace [36] é uma plataforma que recorre à inteligência artificial para fornecer monitorização de infraestruturas, de aplicações e de micros serviços. Esta plataforma fornece também o *Software Intelligence Hub* que permite efetuar integrações com várias tecnologias, incluindo com a *Amazon Web Services* (AWS), o Docker, Java ou Prometheus.

Prós:

- Boa solução para a monitorização de desempenho;
- Deteção e diagnóstico de problemas;
- Não requer intervenção humana;
- Gera relatórios mensais detalhados para a performance de aplicações.

Contras:

- Custo elevado;
- Documentação fraca;
- Gráficos das *dashboards* pouco descritivos;
- Pouca flexibilidade para os utilizadores personalizarem as *dashboards*.

2.3.4 Datadog

A Datadog [26] é uma plataforma de monitorização para aplicações *cloud-scale*, fornecendo *tracking* de servidores, de base de dados, de ferramentas e serviços através de uma solução baseada em *Software as a Service* (SaaS).

Contrariamente aos restantes produtos de monitorização, a Datadog foca-se essencialmente em apresentar uma visão integrada das ferramentas/serviços utilizadas pelas equipas de desenvolvimento e de operações. Além destes recursos, esta plataforma foi também desenvolvida para reunir dados de outras aplicações de *cloud* e de outras ferramentas de monitorização, abrangendo assim todo o ciclo de vida do projeto, desde a criação/alteração do código até à monitorização de alertas/métricas.

Prós:

- Indicada para monitorização de aplicações;
- Permite uma boa gestão de *logs*;
- Fácil de visualizar e analisar o histórico de métricas.

Contras:

- Custo elevado;
- Documentação fraca;
- Difícil de interagir para novos utilizadores.

2.3.5 AppDynamics

A AppDynamics [38] é uma solução de gestão de desempenho de aplicações pertencente à Cisco que aprimora o desempenho e a visibilidade das aplicações *multicloud*. Esta plataforma recorre à inteligência artificial para resolver problemas das aplicações e para prevenir que estes ocorram no futuro, bem como para realçar a visibilidade da arquitetura.

Prós:

- Visibilidade *end-to-end*;
- Monitorização de todas as bases de dados;
- Monitorização da infraestrutura;
- Documentação completa e estruturada de cada versão.

Contras:

- Custo elevado;
- Interface complexa e confusa;
- Falta de tutoriais e *insights*;

2.3.6 New Relic APM

A New Relic [27] é uma plataforma de monitorização de performance que fornece monitorização a aplicações web ou não-web, suportando aplicações de diversas linguagens de programação.

Os servidores da New Relic permitem obter informações em tempo real e retirar as tendências existentes nos dados de performance das aplicações. Esta plataforma permite ainda, e à semelhança da solução anterior, uma visualização *end-to-end* e também uma vasta variedade de gráficos e de relatórios.

Prós:

- Visibilidade *end-to-end*;
- Fácil de integrar com plataformas *cloud* como o Azure;
- Boa variedade de gráficos e relatórios;
- Boa integração com ferramentas como o Jira;

Contras:

- Custo elevado;
- Difícil de entender para novos utilizadores;
- Gestão de alertas complexa.

De forma a podermos comparar mais alguns aspetos das quatro ferramentas escolhidas, foi elaborada a tabela 2.3.

Nota: Com a cor amarela foram assinaladas as características que não se enquadram nos objetivos do projeto. Com a cor verde foram assinaladas as características que correspondem às necessidades atuais do projeto.

Ferramentas	**Open source**	**Licença** (desde)	**Sistemas operativos**	**Documentação**	**Interface**
Kamino (Elastic Stack + Grafana)	Sim	-	Linux, IBM AIX	Completa	Boa
SigNoz	Sim	-	Linux, MacOS [10]	Fraca	Boa
Apache SkyWalking	Sim	-	Linux, Win	Completa	Complexa no início
Dynatrace	Não	69$/mês [12]	Linux, IBM AIX, Win [13]	Fraca	Complexa no início
Datadog	Não	23$/mês [7]	Linux, IBM AIX, Win [6]	Fraca	Boa
AppDynamics	Não	90$/core [34]	Linux, Win [32]	Completa	Complexa
New Relic	Não	75$/mês [29]	Linux, Win, macOS [21]	Completa	Boa

Tabela 2.3: Comparação entre alternativas ao Kamino

A fim de realizar uma análise correta e bem estruturada, é necessário ter em conta todas as características do projeto. Uma vez que a plataforma Kamino foi desenvolvida no âmbito do projeto, era de esperar que se obtivesse um produto mais completo e que melhor respondesse às necessidades específicas dos *stakeholders*.

Uma das características necessárias é que a plataforma escolhida possa ser extensível tanto na criação de métricas e do tipo de dados que estas suportam como nos alertas disponíveis. Ao analisar esta questão, verificou-se que todas as alternativas eram extensíveis e flexíveis pelo que este ponto não teve impacto na decisão.

Já numa questão financeira, identificaram-se duas plataformas *open-source*. No entanto, estas plataformas não possuem o grau de maturidade exigido, não podendo ser optado como solução. Em ambas, a coleção de dados é efetuada através da ferramenta OpenTelemetry [31] e destina-se essencialmente à monitorização de arquiteturas baseadas em micro serviços, o que não corresponde às necessidades do projeto. Por outro lado, apesar do facto das restantes plataformas serem mais complexas, a questão da licença paga constitui, desde já, um fator importante na preferência pela escolha da ferramenta atual.

Quanto ao sistema operativo, é importante referir que a coleção de dados é feita

através do sistema *Advanced Interactive eXecutive* (AIX), uma distribuição da IBM baseada em UNIX. No entanto, o processamento referente à *stack* da Elastic e a sua visualização no Grafana assenta na arquitetura x86. Esta decisão implica que a recolha de dados seja uma *constraint* a esse nível mas garante simultaneamente uma maior flexibilidade a nível da visualização e do processamento dos dados. Assim sendo, e considerando a área de negócio em questão, a resistência à mudança e a burocracia envolvida nesse processo, torna a decisão de manter os mecanismos existentes numa realidade, com o objetivo de se poder agir mais rápido e com um menor custo possível. Posto isto, apenas três das soluções escolhidas poderiam ser destinas ao sistema atual da plataforma.

Por fim, o desenvolvimento da plataforma Kamino necessitou de uma grande quantidade de esforço humano por parte da CSW, tendo este esforço conduzido ao funcionamento atual da plataforma, que cumpre uma parte significativa dos requisitos necessários.

Considerando todos os pontos-chave mencionados anteriormente, conclui-se que a melhor decisão é manter a plataforma Kamino como solução APM para auxiliar as tarefas da equipa de suporte do projeto.

Capítulo 3

Gestão e planeamento do projeto

Este capítulo aborda os aspetos mais relevantes para a gestão do projeto, tratando assuntos como a organização da equipa, o ciclo de vida adotado, o planeamento e as reuniões de acompanhamento realizadas.

3.1 Organização da equipa

A equipa que trabalha atualmente neste projeto subdivide-se em 5 equipas mais pequenas, representadas na figura 3.1.

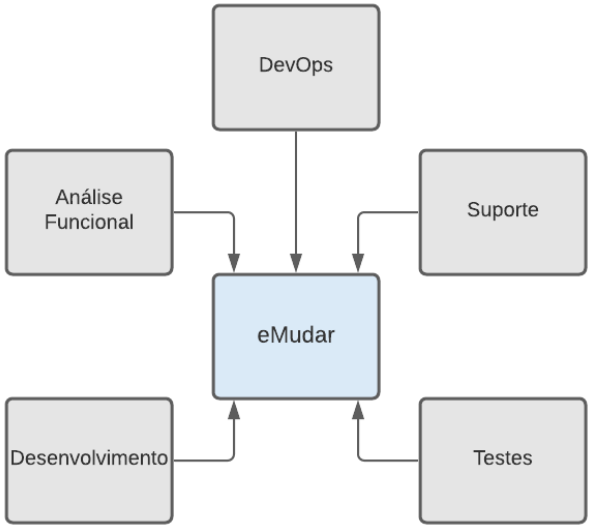


Figura 3.1: Estrutura da equipa do projeto

Análise Funcional - Esta equipa é responsável pela elicitação de requisitos com o cliente.

Desenvolvimento - Esta equipa é responsável pelo desenvolvimento de novas funcionalidades, devendo garantir sempre que estas estão de acordo com os re-

quisitos definidos pela equipa de análise funcional. É ainda responsável por garantir a manutenção das funcionalidades em produção.

Testes - Esta equipa é responsável pela realização de validações de *software*, sendo também responsável por identificar possíveis falhas preferencialmente antes do envio do software para produção.

Suporte - Esta equipa é responsável por fornecer suporte constante ao cliente e é a mais interessada no produto final resultante deste estágio, uma vez que os vai auxiliar nas suas tarefas e na deteção ativa de processos com erros.

DevOps - Esta é a equipa onde o estagiário se insere e é responsável pela manutenção das infraestruturas e pela gestão de falhas que possam ocorrer durante as *releases*, por exemplo. Ao automatizar os processos, é possível garantir que a plataforma permaneça funcional e sem falhas a reportar pelas outras equipas.

3.2 Ciclo de Vida

A escolha do ciclo de vida a adotar no projeto é uma decisão importante uma vez que deve refletir as necessidades e o contexto do mesmo.

Analisando o âmbito deste projeto e considerando também a vertente académica do estágio em questão, decidiu-se que nem todas as tarefas seriam refletidas pela metodologia escolhida, estando esquematizadas na figura 3.2

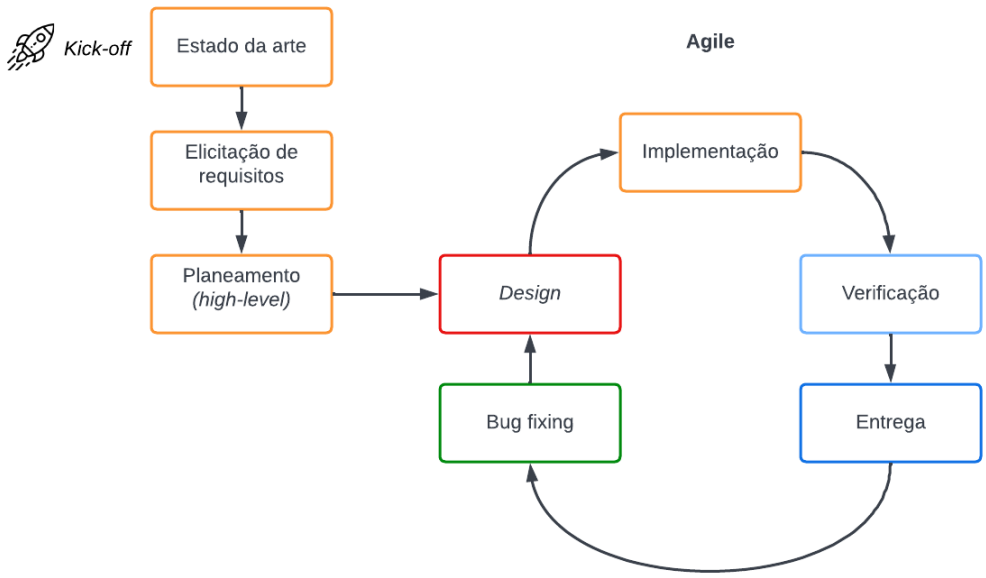


Figura 3.2: Metodologia adotada

Numa fase inicial, decorreu o *kick-off* do projeto e a consequente realização do estado da arte.

Posteriormente, procedeu-se à elicitação de requisitos junto dos *stakeholders*. Nesta

fase, identificaram-se os principais constrangimentos e melhorias para a plataforma, tendo a prioridade dos requisitos encontrados sido debatida com a equipa de suporte. Numa outra fase, analisou-se o âmbito de cada tarefa e enquadraram-se as que se consideraram viáveis tendo em conta a duração do estágio. Advindo destas tarefas, definiu-se o planeamento de alto nível para o segundo semestre. Estas tarefas não foram englobadas no seguimento metodologia escolhida, podendo considerar-se como apenas como um *pre planning* necessário ao início do ciclo de vida. Com esta decisão foi possível garantir que os prazos e objetivos propostos eram estabelecidos com uma maior precisão e que, simultaneamente, as *constraints* do estágio em questão eram também respeitadas.

Por sua vez, durante todo o desenvolvimento das funcionalidades propostas, optou-se por recorrer a uma metodologia *agile*, onde cada *feature* definida previamente no planeamento percorreu o ciclo de vida representado à direita na figura 3.2. Esta decisão tomou-se principalmente pelo facto de serem realizadas entregas graduais ao cliente, o que permitiu que houvesse um seguimento progressivo do estado do trabalho. Para além disso, favoreceu ainda a obtenção de *feedback* ao longo de todo o processo, o que facilitou a adaptação e correção do que tinha sido inicialmente previsto. Adicionalmente, e como consequência da realização de entregas curtas e com maior frequência, esta decisão permitiu reduzir consideravelmente a quantidade de esforço dedicado à correção de *bugs*, uma vez que qualquer correção era realizada imediatamente após o desenvolvimento. Por fim, possibilitou a possível alteração de prioridades das *features*, de forma a conseguir maximizar a entrega de valor da ferramenta.

Uma vez que a equipa em questão é apenas constituída por um elemento, o próprio estagiário, considerou-se que não seria correto atribuir uma designação concreta das principais metodologias *agile* (*Scrum*, *Lean* e *Kanban*), ficando denominado apenas de *agile*. No entanto, e considerando que o estagiário esteve inserido na equipa de DevOps, foram seguidas algumas técnicas da metodologia adotada pela equipa, o Kanban, assim como a entrega continua de valor ao cliente e a organização de tarefas num quadro de Kanban.

3.3 Planeamento

O planeamento do estágio consistiu essencialmente na realização da análise funcional, tendo incluído as seguintes tarefas:

- Análise das necessidades dos *stakeholders*;
- Elicitação de requisitos;
- Identificação e documentação de melhorias e alterações a serem aplicadas.

Quanto ao planeamento de cada tarefa, e a fim de ser estruturado de forma mais eficaz e de acordo com o ciclo de vida escolhido, seguiu um ciclo PDCA composto pelas seguintes fases: **P**lan, **D**o, **C**heck, **A**ct. Este ciclo é um método de gestão de

projetos e que tem como objetivo garantir a correta organização dos processos. Recorrendo a este método, é possível garantir que a análise técnica de cada *issue* é realizada continuamente e em *milestones* mais curtas e que as *lessons learned* são obtidas a tempo de as incluir na *milestone* seguinte. A junção deste ciclo à adoção de uma metodologia ágil facilita o cumprimento dos objetivos dentro do prazo e, considerando a entrega contínua de valor realizada em cada *milestone*, ajuda também no cumprimento dos requisitos conforme o planeamento anteriormente realizado [22].

Tendo isto em conta, o estágio foi caracterizado por três partes fulcrais de desenvolvimento: análise funcional, implementação e testes. Há ainda que considerar uma última parte que não foi visada abaixo que se destina à entrega do produto final e que corresponde à fase *Act*.

3.3.1 Análise técnica - *Plan*

Esta fase envolveu as seguintes tarefas:

- Familiarização com as ferramentas a utilizar;
- Recolha das informações necessárias para a tarefa em curso.

3.3.2 Implementação - *Do*

Esta fase compreendeu as seguintes tarefas:

- Implementação das alterações e de novas *features* na plataforma de monitorização;

3.3.3 Testes - *Check*

Esta fase foi composta pelas seguintes tarefas:

- Elaboração de um plano de testes;
- Execução dos testes;
- Documentação dos resultados obtidos.

3.3.4 Entrega - *Act*

Por fim, a última fase incluiu as seguintes tarefas:

- Promoção das alterações a produção;
- Comunicação das alterações efetuadas.

3.4 Reuniões

Reuniões diárias com a equipa de DevOps

Durante o segundo semestre, o estagiário esteve inserido em reuniões diárias, denominadas de *dailies*, com equipa de DevOps. Uma *daily* é uma reunião rápida e essencial para o funcionamento do modelo ágil, baseando-se mais particularmente em Scrum. Esta reunião é realizada no início de cada dia e visa revisitar as tarefas do dia anterior e planear as do dia em questão. Nesta reunião são ainda debatidos possíveis impedimentos, com o objetivo de os resolver e de desbloquear assim a execução de novas tarefas.

Reuniões a cada duas semanas com o *Technical Manager* e com o orientador da empresa

Durante o primeiro semestre foram realizadas reuniões a cada duas semanas com o *Technical Manager* do projeto e com o orientador da empresa com o objetivo de perceber o ponto de situação atual, onde se identificava o que tinha sido feito até então e quais as tarefas que deviam ser consideradas no futuro. Para além disso, foram realizadas várias reuniões informais com o orientador da empresa onde se debateram assuntos relevantes tanto do estágio em si como de outras dúvidas que pudessem eventualmente surgir.

Reuniões mensais com o orientador da empresa e da faculdade

Por fim, foram agendadas reuniões mensais com ambos os orientadores (entidade promotora e universidade) para permitir um acompanhamento mais próximo do estagiário e para o auxiliar não só na estrutura e revisão do planeamento definido como também no relatório de estágio.

3.5 Retrospetivas

A retrospetiva é uma parte essencial de uma metodologia *agile* e corresponde a uma reunião realizada no final de cada *milestone*, com o objetivo de discutir o que correu bem durante o ciclo anterior e o que deve ser melhorado na próxima *milestone*.

As retrospetivas foram suportadas pela ferramenta Miro [37], onde foram construídas essencialmente 3 áreas de foco:

- *What was good?*

Neste campo foram anotados todos aspetos que se entenderam ter ocorrido de maneira positiva.

Destas, destacam-se maioritariamente em todas as retrospetivas a conclusão das *milestones on time* e aplicação das *lessons learned* nas *milestones* seguintes.

Pode inferir-se também que a análise técnica efetuada foi claramente um ponto chave para identificar e justificar as alterações feitas, tendo por isso seguido sempre uma abordagem *data-driven*.

- *What was less good?*

Neste campo foram anotados todos aspetos que se consideraram ter ocorrido não tão positivamente .

Destas, destaca-se maioritariamente a inexperiência com as ferramentas utilizadas, causando por vezes alguma frustração. Aliado a isso, a dívida técnica existente dificultou também o desenvolvimento, tendo sido necessário um esforço extra. Esta dívida corresponde ao resultado de algumas decisões tomadas pelas anteriores equipas de desenvolvimento que, embora na altura sejam aparentemente mais simples e rápidas, apresentam um impacto bastante negativo a longo prazo.

- *Lessons Learned*

Neste campo foram feitas algumas reflexões sobre as lições retiradas da *milestone* que terminou.

Devido às responsabilidades do *team leader*, nem sempre era possível obter *feedback* de forma imediata, o que, em caso de problemas impeditivos, representava um problema. Como consequência, uma das mais importantes lições retiradas da 1ª *milestone* prendeu-se no aproveitamento do tempo e de eventuais pausas surgissem por falta de *feedback*. Estes tempos foram essencialmente aproveitados para a escrita do relatório, aliviando a alocação extra de tempo apenas para esse propósito.

3.6 Gestão de riscos

Considera-se um risco todo e qualquer fator que possa resultar em consequência negativa no futuro, não sendo uma certeza de ocorrência mas sim uma probabilidade. Em Engenharia de Software, é essencial que se proceda a uma gestão de riscos cuidada, de forma a minimizar o impacto da sua eventual ocorrência. Nesta secção é descrita a notação utilizada na avaliação dos riscos identificados para as principais componentes que constituíram este estágio, sendo expressos em função do seu impacto e da probabilidade de ocorrência.

Relativamente ao impacto, consideraram-se os seguintes riscos:

- **Catastrófico** - Compromete seriamente o sucesso do estágio e/ou a qualidade do produto entregue;
- **Crítico** - Caso não seja despendida alguma atenção na sua resolução, pode

comprometer também o sucesso do estágio e/ou a qualidade do produto entregue;

- **Marginal** - Não constitui, por si só, um entrave ao sucesso do estágio nem à qualidade do produto mas pode haver necessidade de efetuar alguns ajustes.

Relativamente à probabilidade de ocorrência, consideraram-se as seguintes categorias:

- **Alta** - Superior a 70%
- **Média** - Entre 40% a 70%
- **Baixa** - Inferior a 40%

Os riscos identificados e o plano de mitigação definido para cada um encontra-se detalhado na secção 5.1, no âmbito do capítulo referente à garantia de qualidade do produto.

3.7 Plano de Estágio

3.7.1 Primeiro Semestre

Durante o primeiro semestre era esperado um esforço de 40%, correspondendo a 16h semanais e, portanto, a dois dias por semana na empresa.

O apêndice 7.2 apresenta os diagramas de *Gantt* para o conjunto de tarefas elaboradas durante o primeiro semestre. O diagrama A.1 representa o planeamento inicial do primeiro semestre sendo que o diagrama A.2 representa o planeamento que foi efetivamente executado.

As diferenças entre os planeamentos ocorreram inicialmente na tarefa T06, onde se estendeu a duração da mesma para 6 semanas. Esta alteração decorreu na sequência de se ter incorporado também a escrita e documentação de requisitos na forma de *user stories*.

Posteriormente, com o adiamento da entrega intermédia de dia 17 de Janeiro para dia 24 de Janeiro, estendeu-se a realização do relatório uma semana com a finalidade de afinar alguns pormenores existentes.

3.7.2 Segundo Semestre

Durante o segundo semestre era esperado um esforço de 100%, correspondendo a 40h semanais e, portanto, a cinco dias por semana na empresa.

O apêndice 7.2 contém também os diagrama de *Gantt* para o conjunto de tarefas propostas para o segundo semestre. O diagrama A.3 representa o planeamento inicial estabelecido para o segundo semestre sendo que o diagrama A.4 representa o planeamento que foi efetivamente realizado.

A principal diferença entre os planeamentos ocorreu essencialmente na adição da tarefa T03 (`Corrigir corrupção de arquivos aquando da receção`) que, embora não planeada inicialmente, foi necessário incluir no âmbito do desenvolvimento. Esta tarefa, tratou-se de um *defect* já existente no *backlog* interno do projeto e que estava a causar alguns problemas no desenvolvimento das melhorias, encontrando-se devidamente documentado na secção 6.3.

Relativamente ao relatório de estágio, pressupôs-se que a sua elaboração fosse contínua ao longo do semestre, não tendo sido colocada nenhuma tarefa diretamente relativa ao seu desenvolvimento no diagrama. No entanto, foi colocada a tarefa T06 (`Finalização do relatório de estágio`) com o intuito de reservar duas semanas única e exclusivamente para a finalização do relatório e para a elaboração de algumas correções que pudessem vir a ser eventualmente necessárias.

Capítulo 4

Elicitação de requisitos

A elicitação de requisitos é a parte integrante da fase do projeto onde são extraídas e documentadas as informações sobre o que o cliente necessita e o que quer ver implementado. Nesta fase são identificadas as necessidades do sistema e as características que este deve possuir para um funcionamento conforme o esperado.

Numa primeira fase, registaram-se as necessidades funcionais da equipa de suporte tendo também identificado alguns problemas da plataforma que necessitam de correção.

4.1 Problemas encontrados

Os problemas encontrados dizem respeito à opinião do líder da equipa de suporte, dos membros da equipa de suporte e por fim do líder da equipa de DevOps, tendo sido documentados considerando esta divisão.

Líder da equipa de suporte

- É percetível que há um *delay* entre o que se encontra em produção e o que é realmente apresentado no Kamino.
- Quanto à alarmística relacionada com falhas no *upload* de ficheiros, é necessário verificar que apenas deve ser gerada uma notificação quando o tempo de erro ultrapassar um certo limite. Isto é, existem atualmente mecanismos que lidam com várias falhas transitórias, resolvendo-os através de re-execuções automáticas. Para casos como este, seria importante que a plataforma não notificasse o utilizador assim que encontrasse um erro, mas sim apenas quando não fosse possível recuperá-lo automaticamente, uma vez que pode constituir um falso positivo.
- O Kamino não inclui todas as ferramentas necessárias à equipa de suporte, havendo a tendência de recorrer, desde início, a ferramentas mais completas e com menos latência.

Membro da equipa de suporte

- É recorrente que a equipa de suporte inicie funções e que o Kamino não apresente qualquer tipo de dados.

  A solução estabelecida passa por afinar a hora de início do Kamino com a hora de início de funções da equipa de suporte.

Líder da equipa de DevOps

- É necessário garantir que as alterações que possam eventualmente ocorrer na topologia da infraestrutura do cliente sejam atempadamente refletidas na plataforma, de forma a não criar disrupção nas rotinas de trabalho da equipa de suporte.

Ponderou-se ainda que, numa fase mais avançada, poderia ser a equipa de suporte a criar as suas próprias *dashboards* e todo o conjunto de alertas necessários. No entanto, a equipa levantou algumas preocupações relativas à falta de tempo, disponibilidade e também à falta de conhecimento (ferramentas e processos) para a criação de novos *dashboards*, tendo a sugestão ficado em *standby*.

4.2 Tarefas

Depois de toda a análise da secção anterior, concluiu-se que as principais tarefas seriam:

- Redução da latência na transmissão de métricas;
- Revisão da frequência de obtenção de métricas;
- Revisão total da alarmística;
- Adição de uma *dashboard* para monitorizar os mecanismos de arquivo existentes no projeto;
- Adaptação da plataforma na sequência de alterações da infraestrutura de produção planeadas para um futuro próximo;
- Mudança do horário de arranque da plataforma;
- Criação de um *dashboard* de *timeouts*;
- Melhorar a robustez e a tolerância a falhas da plataforma;
- Revisão de métricas específicas;
- Ingestão de *logs*.

Após a identificação destas tarefas, procedeu-se à análise do *backlog* interno e já existente da plataforma, da qual resultaram 6 novas tarefas, que não tinham sido identificadas inicialmente pela atual equipa.

- Adicionar métrica relativa ao nº de tarefas concluídas no BPM;
- Adicionar alerta para processos de compensação de cheques;
- Adicionar alerta para processos de erro;
- Criar um *dashboard* realtivo a eventos;
- Colocar descrição das *Application Programming Interface* (API) no *dashboard* geral do sistema;
- Atualizar a *stack* tecnológica;

4.3 *User Stories*

Para documentar as tarefas encontradas recorreu-se a *user stories*, que correspondem a explicações informais e gerais de um requisito documentadas na perspetiva do cliente. Estas *stories* têm o objetivo de fornecer uma maneira simples e clara de definir um requisito de produto e também o de fornecer uma base sólida de comunicação, focando-se no que mais importa para o cliente.

Comparativamente a outros métodos de captura e de documentação de requisitos, as *users stories* apresentam as seguintes vantagens [3]:

- Definem com clareza o que se vai desenvolver, para quem e para quê mantendo sempre o foco no utilizador;
- Permitem a colaboração de outros membros da equipa com o fim de encontrar a melhor forma de atingir o objetivo definido;
- Encorajam o pensamento critico e criativo para a obtenção de um objetivo real;
- Geram um sentimento de satisfação a cada *story* concluída.

Para além da documentação dos requisitos, o estabelecimento de objetivos e metas é igualmente importante uma vez que permite estabelecer com clareza quais as prioridades e estratégias necessárias para atingir os objetivos.

Assim sendo, e com o objetivo de definir concretamente estas metas, optou-se pela utilização de objetivos SMART [5]. Assim sendo, estes devem ser e(s)pecíficos, (m)ensuráveis, (a)lcançáveis, (r)elevantes e por fim (t)emporais, tornando-os bastante interessantes na medida em que permitem saber o que necessita de ser realizado com o objetivo de atingir os resultados pretendidos.

Por sua vez, e no seguimento do objetivo acima mencionado, recorreu-se à definição de *Acceptance Criteria* (AC) para cada *user story*, garantindo assim que os objetivos estabelecidos são claros naquilo que se pretende atingir. Cada AC é referente a um conjunto de requisitos predefinidos, que devem ser cumpridos para marcar uma *user story* como terminada.

A seguinte lista apresenta as *user stories* que foram consideradas prioritárias, correspondendo aos requisitos *must have*, apresentando para cada uma delas o AC definido.

Nota: Assume-se que todos os AC abaixo definidos são válidos considerando condições de funcionamento normais.

Como membro da equipa de suporte:

1. **Quero que** se reduza o *delay* referente à transmissão de métricas **para** que estas sejam processadas tão rápido quanto possível;

   AC:

   (a) O sistema é determinístico para a transmissão de métricas;

   (b) O *worst case* do tempo relativo à transmissão de métricas não ultrapassa os 3 minutos.

2. **Quero que** a visualização dos dados seja a mais aproximada de *real time* possível **para** que o *tracking* e a descoberta de erros seja mais eficaz.

   AC:

   (a) O *worst case* do tempo relativo à visualização de métricas identificadas pela equipa de suporte não ultrapassa os 5 minutos.

3. **Quero que** a alarmística seja revista e reconfigurada **para** que se reduza a verbosidade excessiva da mesma que por vezes, desfoca a atenção de toda a equipa.

   AC:

   (a) O número de notificações enviadas para o canal de Slack sem necessidade não é superior a 1 por dia;

   (b) O tempo que um problema demora a ser notificado, desde que ocorre, não é superior a 5 minutos.

4. **Quero que** a plataforma seja mais estável e mais robusta **para** garantir que sempre que necessário apresente os dados que quero, falhando o menos possível.

   AC:

   (a) O número de falhas com origem em problemas na plataforma não é superior a 1 falha a cada 3 meses.

Como membro responsável pela evolução da plataforma:

1. **Quero que** sejam criadas infraestruturas que suportem o desenvolvimento e a realização de testes **para** que a equipa de suporte não seja afetada com possíveis modificações, comprometendo assim o desempenho da plataforma.

   AC:

   (a) O hardware está isolado da instância do Kamino de produção;

   (b) Os serviços estão replicados;

   (c) O ambiente de *staging* é alimentado por um *branch* diferente do de produção, a fim de garantir que o desenvolvimento e testes não afetam a instância principal até que seja feita a respetiva promoção a produção.

4.4 Prioritização

Para prioritizar os requisitos anteriormente identificados recorreu-se ao método de prioritização MoSCoW. Este acrónimo pode ser descomposto nas várias categorias de prioritização: **M**ust**o****S**hould**C**ould**o****W**illNot, podendo este último assumir o valor de **W**ish [28].

Com mais detalhe, cada categoria possui as seguintes características:

- ***Must have***: Necessidades não negociáveis e que são obrigatórias para a equipa;
- ***Should have***: Iniciativas que não são vitais mas que acrescentam um valor significante;
- ***Could have***: Iniciativas desejáveis mas que não têm grande impacto caso sejam descartadas;
- ***Will not have***: Iniciativas que não são prioritárias considerando o tempo de desenvolvimento definido.

A fim de poder categorizar cada requisito, realizou-se uma reunião com os *stakeholders* para perceber o que seria mais prioritário implementar na visão da equipa de suporte. Para que a comunicação fosse mais eficaz e a reunião mais produtiva, realizou-se anteriormente uma tabela semelhante à representada na tabela 4.1, onde todos os requisitos foram identificados com uma descrição, *scope*, *sub-scope* (quando aplicável) e ainda um campo a ser preenchido durante a prioritização.

No processo de prioritização, os requisitos *'must have'* foram identificados em primeiro lugar, os *'should have'* em segundo e por fim os *'could have'*, havendo ainda um outro requisito *'will not have'* que foi categorizado como tal numa fase anterior do projeto. Esta atribuição e consequentes detalhes de cada requisito foram registados na tabela 4.1, onde se encontram ordenados primeiramente por *scope* e posteriormente por *sub-scope*.

Em forma de suma, desta reunião resultaram:

- ***Must have***: 5 requisitos;
- ***Should have***: 2 requisitos;
- ***Could have***: 9 requisitos;
- ***Will not have***: 1 requisito.

Como mencionado anteriormente, os requisitos identificados resultaram não só das reuniões com os *stakeholders* mas também da análise do *backlog* interno já existente da plataforma, o que aumentou consideravelmente o número de requisitos a ponderar. A decisão de apenas realizar as 4 tarefas definidas como *must have*, num total de 17 tarefas possíveis, prendeu-se essencialmente na a elevada complexidade das mesmas, uma vez que exigiam um grande trabalho de desenvolvimento e de revisão da plataforma. Considerando a duração do estágio, não era possível adicionar outras tarefas ao planeamento, limitando o desenvolvimento àquelas que foram consideradas fulcrais pela equipa de suporte.

ID	Descrição	*Scope*	*Sub-Scope*	Prioridade
1	Reduzir latência na transmissão de métricas	Desenvolvimento	Métricas	Must have
2	Rever da frequência de obtenção de métricas	Desenvolvimento	Métricas	Must have
3	Adicionar métrica relativa ao nº de tarefas concluídas	Desenvolvimento	Métricas	Could have
4	Adicionar alerta para processos de cheques	Desenvolvimento	Métricas + alertas	Should have
5	Adicionar alerta para processos de erro	Desenvolvimento	Alertas	Should have
6	Rever e ajustar toda a alarmística	Desenvolvimento	Alertas	Must have
7	Criar *dashboard*de eventos	Desenvolvimento	*Dashboard*	Could have
8	Criar *dashboard* de arquivos	Desenvolvimento	*Dashboard*	Could have
9	Criar *dashboard* de *timeouts*	Desenvolvimento	*Dashboard*	Could have
10	Colocar descrição APIs	Desenvolvimento	*Dashboard*	Could have
11	Montar infraestruturas para *develop* e testes	Desenvolvimento	-	Must have
12	Atualizar a *stack* tecn.	Desenvolvimento	-	Could have
13	Adaptar a plataforma na sequência de alterações na infraestrutura	Desenvolvimento	-	Could have
14	Estabilizar e melhorar a robustez da plataforma	*Bug fixing*	-	Must have
15	Rever métricas específicas	*Bug fixing*	-	Could have
16	Mudar o horário de arranque da plataforma	Melhorias	-	Could have
17	Suportar a ingestão de *logs*	Desenvolvimento	-	Will not have

Tabela 4.1: Backlog e prioritização

Capítulo 5

Garantia de qualidade

Neste capítulo é abordado todo o trabalho desenvolvido no âmbito da garantia de qualidade do produto Kamino.

5.1 Análise de riscos

Esta secção tem como principal objetivo evidenciar os riscos identificados bem como apresentar a sua classificação através de uma matriz, considerando o seu impacto e probabilidade de ocorrência. Para além destes aspetos, são ainda abordados os planos de mitigação adotados tendo sempre em consideração que um plano adequado pesa os impactos de cada risco, prioritizando-os tendo também em conta sua probabilidade de ocorrência.

5.1.1 Identificação dos riscos

O *threshold* de sucesso deste projeto consistiu em satisfazer todos os requisitos *must have* mencionados na tabela 4.1 até ao final do estágio.

Assim sendo, identificaram-se os seguintes requisitos:

1. Curva de aprendizagem

A curva de aprendizagem relativa à familiarização com a plataforma poderá causar atrasos no planeamento atual e a inexperiência com a maior parte das ferramentas poderá também comprometer o desempenho do estágio.

2. Dívida técnica

A dívida técnica deixada pela equipa de desenvolvimento anterior e inerente à plataforma poderá requerer um esforço extra e originar um certo atraso no planeamento atual.

3. Comunicação com os *stakeholders*

A comunicação com os *stakeholders* poderá ser um entrave devido às responsabi-

lidades que cada possui no âmbito projeto. A equipa deve estar sempre alerta e a responder às necessidades do cliente, o que, em casos de indisponibilidade para outras questão, impactaria negativamente o planeamento inicialmente estabelecido.

4. Dependências no acesso a produção

Os acessos a produção, nomeadamente a obtenção de *logs* e a execução de *releases* em produção, não podem ser realizados diretamente pelo estagiário, necessitando sempre da intervenção de terceiros. Assim sendo, a indisponibilidade dos mesmos pode prejudicar o planeamento proposto, afetando igualmente o desempenho do estágio.

5. Equipa reduzida

Considerando que o projeto atual da plataforma Kamino tinha anteriormente atingido um estado de estabilidade e que portanto, não possuía uma equipa fixa nem consequentemente redundância de recursos, gera-se a questão da dependência de um único membro (líder da equipa de DevOps) para dar suporte na parte mais técnica da plataforma. A indisponibilidade do mesmo pode comprometer o alcance do *threshold* do sucesso, afetando seriamente o desempenho do estágio.

5.1.2 Matriz de Probabilidade-Impacto

O impacto de um risco pode ser considerado como o efeito de um determinado risco atingir o *threshold* de sucesso estabelecido, considerando a transformação de uma potencialidade para uma realidade. Por outro lado, a probabilidade do risco consiste na sua probabilidade da sua ocorrência, tendo a relação entre as duas componentes sido espelhada na matriz 5.1. [11]

X	Marginal	Crítico	Catastrófico
Alta		3	
Média		1,2,4	
Baixa			5

Tabela 5.1: Matriz de Probabilidade-Impacto

5.1.3 Mitigação dos riscos

Como estratégias de mitigação, identificaram-se as seguintes:

1. A experiência dos membros da equipa deve ser aproveitada para extrair conhecimentos acerca de questão relativas ao âmbito do projeto e também acerca de algumas das ferramentas utilizadas.

2. A mitigação da dívida técnica existente deve ser ponderada considerando o custo da escolha momentaneamente e o custo que essa dívida trará ao projeto

como um todo. Para além de ser necessário mitigar esta dívida, sempre que possível, é ainda preciso ter o cuidado de não a tornar ainda maior. É preciso ter sempre em mente que, por muito que os *stakeholders* desejem o desenvolvimento de novas funcionalidades para fazer face às necessidades, deve haver um equilíbrio entre esse aspeto e a manutenção da dívida técnica, procurando assim evitar problemas maiores no futuro.

3. As reuniões com a equipa de suporte devem ser agendadas com a noção de que a disponibilidade dos mesmos nem sempre é garantida. Embora o agendamento com antecedência assegure que ambas as partes estão ocorrentes da data prevista e do objetivo proposto para a reunião, o surgimento de problemas que necessitam do auxílio da a equipa é algo frequente, não havendo então a garantia que a reunião se irá, de facto, realizar. Assim sendo, deve ser garantida a flexibilidade na escolha de um outro horário para a sua realização.

4. Considerando a indisponibilidade de algum dos membros da equipa de DevOps, deve ser procurado outro membro com responsabilidade semelhantes, que possa auxiliar no acesso a produção. Caso nenhum deles esteja disponível, a tarefa em curso deve ser substituída pela escrita do relatório até que haja novamente hipótese de aceder ao pretendido.

5. No caso de indisponibilidade do membro da equipa, a etapa em curso deve ser preemptada pela tarefa de escrita do relatório, de forma a minimizar o impacto negativo no planeamento proposto.

5.2 Plano de testes

Tendo em conta o âmbito deste estágio e diversidade de tarefas realizadas, o plano de testes elaborado não seguiu um esquema predefinido.

Considerando a prática implementada de *continuous delivery*, o processo de testes foi gradual ao longo de cada *milestone* e assim sendo, a sua documentação foi feita em cada secção, aquando da finalização do desenvolvimento de cada tarefa.

O plano adotado foi documentado em cada secção, tendo sido o que se considerou mais adequado no âmbito dessa tarefa. Esse plano passou por vezes pela análise de *logs* dos *containers* utilizados ou pela pela criação de *scripts* em Python ou Bash que visavam a simulação das condições necessárias ao teste das funcionalidades implementadas. Em alguns dos casos foram realizados testes funcionais, tendo definido um conjunto dados de entrada e registado o resultado obtido em tabelas de testes.

Capítulo 6

Desenvolvimento

Este capítulo descreve as alterações implementadas na plataforma Kamino a fim de cumprir os requisitos anteriormente estabelecidos. A implementação detalhada consolidou essencialmente conhecimentos acerca de Python, Bash, Linux e de toda a *stack* da Elastic.

6.1 Configuração de uma máquina de *staging*

A criação de uma máquina de *staging*, semelhante ao Kamino, teve o principal objetivo de existir uma máquina disponível para ser utilizada no desenvolvimento e nos testes, sem que isto afetasse a máquina Kamino de produção e sem que, consequentemente, causasse impacto à equipa de suporte.

6.1.1 Análise técnica

Ao fazer o levantamento das máquinas existentes e da configuração de cada uma, constatou-se que existia já, na altura, uma máquina de *staging* criada a partir do Kamino. No entanto, a versão do sistema operativo Ubuntu configurada na máquina era a 16 e, tendo em conta que esta versão já se encontra sem suporte [15], houve a necessidade de a eliminar e de criar uma nova máquina com a versão do sistema operativo Ubuntu 20.04. Esta é uma versão *Long Term Support* (LTS) e assim sendo, permite o suporte gratuito durante um longo período tempo, até 2030 no caso [39].

Inicialmente foi necessário ganhar algum contexto sobre as ferramentas Ansible e Terraform, que não só são essenciais para a criação da máquina em si como também para a configuração de todo o seu contexto (discos, rede, segurança e software).

6.1.2 Implementação

Esta máquina foi criada no Azure com auxílio das ferramentas mencionadas na secção 6.1.1. Com o intuito de criar facilmente uma máquina com as características que desejamos, foram criados os *playbooks* e os *roles* em Ansible necessários para que, recorrendo unicamente a um linha executada na linha de comandos, seja criada uma máquina com todos os requisitos pretendidos, englobando desde características do hardware, do sistema operativo, do software de base instalado e também da sua configuração.

Um *playbook* é um conjunto de tarefas, habitualmente complexas, que são executadas com pouco ou nenhum envolvimento humano com o intuito de automatizar ações como a criação de artefactos ou de máquinas, tendo neste caso auxiliado na criação de infraestruturas de IT. Os *playbooks* são executados em grupo e por *host*, estando os detalhes como dispositivos de rede ou os servidores, inseridos num inventário Ansible.

Relativamente ao Terraform, esta é uma ferramenta de *Infrastructure as Code* (IaC) que permite gerir a infraestrutura através de ficheiros de configuração legíveis por humanos, seguindo uma abordagem declarativa em detrimento da configuração manual numa interface gráfica. Assim sendo é possível criar, modificar e configurar toda a infraestrutura de forma segura e consistente, tornando a infraestrutura desenvolvida facilmente modificável e extensível. Algumas das vantagens da abordagem IaC adotada prendem-se essencialmente na possibilidade de dispor de uma infraestrutura determinística e consequentemente reproduzível. Estas características garantem que, ao recriar toda esta infraestrutura, o resultado final é apresentado com as configurações desejadas, requerendo um esforço consideravelmente reduzido.

6.1.3 Testes

Com o objetivo de verificar a correta implementação do *playbook* criado, foi feito um teste que confirma se, de facto, o comportamento era o correto.

Os *containers* criados são *stateless* a nível dos dados, uma vez que estes dados se encontram num volume do *host*, podendo então cada um deles ser descartável e recriado sem qualquer perda de informação.

Por conseguinte, a recriação da máquina como um todo e a consequente eliminação de todos os seus *containers* conduziria à perda de informação. No entanto, e tendo em conta que os dados da máquina representavam dados de teste, e eram portanto irrelevantes, a máquina passou assim a ser igualmente descartável, podendo ser recriada a qualquer momento sem que haja qualquer constrangimento.

Como forma de testar este aspeto, a máquina foi totalmente apagada e posteriormente recriada, tendo-se verificado que esta foi recriada com todas as características pretendidas, o que era o comportamento esperado.

6.1.4 Resultados e entrega

A fim de ser possível validar e marcar a *user story* anteriormente definida como completa, analisaram-se os *Acceptance Criteria* (AC) estabelecidos na secção 4.3:

AC	Resultado
(a) O hardware está isolado da instância do Kamino de produção	✓
(b) Os serviços estão replicados	✓
(c) O ambiente de *staging* é alimentado por um *branch* diferente do de produção	✓

Tabela 6.1: Validação dos AC definidos no âmbito da criação de uma máquina de *staging*

Relativamente aos pontos (a) e (b), os serviços necessários foram corretamente replicados e o hardware é igualmente independente da máquina de produção. Quanto ao ponto (c), foi garantido que a máquina de *staging* era alimentada por um *branch* diferente de produção, para que fosse possível desenvolver sem causar impacto na plataforma de produção e consequentemente na equipa. O processo de desenvolvimento passou pela utilização de *feature branches*, o que permitia que as alterações fossem testadas antes da integração, através de *pull requests*, no *master*.

6.2 Redução da latência na transmissão de métricas

6.2.1 Análise técnica

Investigando a motivação inicial para a criação da plataforma e o respetivo enquadramento ao longo do tempo, a possibilidade de monitorizar relatórios era inicialmente o foco principal da plataforma, auxiliando na análise das tendências dos dados obtidos. No entanto, decidiu-se posteriormente que a questão da alarmística era algo que deveria ser também considerado, tendo sido acrescentada à plataforma.

Assim sendo, e analisando o contexto histórico do desenvolvimento da plataforma, a receção dos dados tão rápido quanto possível não era, numa primeira fase, uma prioridade. De igual forma, e devido ao facto de não haver, no momento, uma prova de conceito efetuada, a escalabilidade do servidor de email era também desconhecida, tentando-se assim minimizar o número de *emails* enviados.

No entanto, a experiência mostrou que o atraso inerente a esta abordagem não era aceitável, tendo em conta os novos de requisitos de monitorização e alarmística em *real-time*, originando assim as melhorias implementadas neste capítulo.

A fim de conseguirmos ter uma visão mais clara de como e onde se pode reduzir a latência na transmissão de métricas, foi essencial analisar o ciclo de vida de uma métrica. Isto é, saber o tempo que demora em cada uma das fases que o processamento de uma métrica possui para ser possível calcular posteriormente o tempo total e para consequentemente conseguir traçar um plano geral.

Posto isto, recorreu-se à definição de casos de uso para melhor explicar o ciclo de vida de uma métrica especifica.

Caso de uso: *Threads* bloqueadas

Esta é uma das várias métricas recolhidas no projeto e, sendo a rapidez de obtenção algo que a equipa de suporte valoriza, foi a escolhida para servir de exemplo ao ciclo de vida de uma métrica, que passa pela sua obtenção, pelo processamento e finalmente pela notificação através dos canais de comunicação da equipa.

Assim sendo, detalharam-se as várias etapas do processo conjuntamente com os vários tempos de obtenção de dados de cada uma delas.

1. Recolha da métrica

A recolha de métricas tinha inicialmente configurado um tempo de obtenção de 5 minutos, estando o tempo da sua recolha restrita ao intervalo de [0,5] minutos.

2. Envio da métrica

O envio de métricas pressupõe semelhantemente um tempo de *loading* de 5 minutos, estando o tempo do seu envio restrito ao intervalo de [0,5] minutos. A justificação da anterior decisão deste intervalo é idêntica à descrita no ponto acima.

3. Mailloader

Nesta fase do processo era necessário medir o tempo que decorre desde que um *email* é enviado até que este é recebido e processado pela plataforma. Assim sendo, com o objetivo de obter o tempo mais próximo possível da realidade, seria importante recolher uma amostra significativamente ampla.

Para gerar estes dados foi criado um *script* em Python que retirava do email recebido os *timestamps* de envio e chegada, calculando posteriormente a diferença entre esses tempos. Os dados gerados e cada resultado obtido foram escritos diretamente numa folha de Excel com o objetivo de facilitar o cálculo de médias e de desvios padrão, necessários para o intervalo de confiança. Considerando apenas relevante o horário de expediente da equipa de suporte, foram desprezados alguns dos *outliers* encontrados.

Terminada a recolha de todos os dados, procedeu-se ao cálculo do intervalo de confiança e da sua posterior análise e interpretação.

Nota: Considerando a granularidade ao minuto que os tempos anteriores detinham, ponderou-se que limitar os resultados obtidos ao campo das décimas de segundo seria irrelevante e a sua variância poderia ser portanto desprezável. No entanto, e tendo em conta que os dados dos intervalos de confiança são re-

sultantes de cálculos intermédios, considerou-se que, por questões de precisão, as décimas de segundo deveriam ser consideradas sempre que possível. Assim sendo, o resultado final da *timeline* de transmissão de métricas foi definido com a granularidade do segundo.

Média da amostra = 31.9 segundos
Desvio padrão da população (δ) = 33.4
Tamanho da amostra (N) = 140

Com o valor crítico (α) de 0.05, o que corresponde a 95% de confiança), e com zc de 1.96, obteve-se um intervalo de confiança de (28.6, 35.2) segundos. Considera-se zc como o valor apropriado da distribuição normal padrão para o nível de confiança pretendido, neste caso de 95%.

Conclusão:

Com base nos dados fornecidos, o intervalo de confiança 95% para a média da população é $28.6 < \mu < 35.2$, o que indica que estamos 95% confiantes de que a verdadeira média da população μ está contida no intervalo (28.6, 35.2) segundos.

4. Filebeat

Nesta fase do processo era necessário medir o tempo que decorre desde que o *email* foi recebido até que foi lido e finalmente enviado para o Logstash, onde será pré-processado.

Assim sendo, o registo dos tempos de processamento foi efetuado através da análise de *logs*.

Média da amostra = 35 segundos
Desvio padrão da população (δ) = 2.9
Tamanho da amostra (N) = 6

Com o valor crítico (α) de 0.05, o que corresponde a 95% de confiança), e com zc de 1.96, obteve-se um intervalo de confiança de (32.7, 37.3) segundos.

Conclusão:

Com base nos dados fornecidos, o intervalo de confiança 95% para a média da população é $32.7 < \mu < 37.3$, o que indica que estamos 95% confiantes de que a verdadeira média da população μ está contida no intervalo (32.7, 37.3) segundos.

5. Logstash

A análise do tempo de processamento no *logstash* baseou-se essencialmente na análise de *logs* do próprio *container*. Assim sendo, e na impossibilidade de visualizar o pretendido, foram analisados os vários níveis de *debug* apresentados na consola. Consultando a documentação oficial do *logstash* [25], inferiu-se que o nível de *logs* configurado (`info`) listava processos que demorassem mais de 1 segundo a executar. Assim sendo, concluiu-se que, uma vez que nenhum evento de chegada de pedidos era visualizado, estes demoravam menos de 1 segundo a ocorrer. Para obter um valor mais preciso seria necessário proceder à alteração

do nível de *logs* e, analisando a pouca relevância desse resultado, considerou-se o tempo de processamento desta etapa como desprezável.

6. Elastic Search - Notificação

Esta etapa engloba todo o processo que decorre desde o final do processamento de dados pelo *Filebeat* até à notificação dos membros da equipa de suporte via Slack, etapa que conclui o ciclo de vida da métrica.

Uma vez que o Grafana apenas disparava e consequentemente alertava em casos em que um certo *threshold* fosse atingido, foi necessário reproduzir essas condições.

Como anteriormente indicado no caso de uso, recorreu-se ao caso das *threads* bloqueadas para simular essas condições.

Para conseguir injetar os dados e registar como resultado os tempos num Excel, foi novamente desenvolvido um *script* em Python que tratava da injeção de ficheiros com métricas especificamente preparadas para disparar um alerta, configurando os *timestamps* de cada um e gerando um Identificador Único (ID) e aleatório para cada um dos ficheiros. A modificação em cada iteração deste ID era fulcral uma vez que um ficheiro com um ID já existente não era considerado e não produzia qualquer efeito na visualização dos dados.

Os ficheiros foram injetados de 1 em 1 minuto, sendo que a *query* configurada que verificava se o *threshold* era atingido, foi avaliada de 1 em 1 segundo. Esta última configuração permitiu diminuir o *overhead* do envio da notificação para o Slack, garantindo assim que a notificação era enviada o mais rápido possível.

Média da amostra = 18 segundos
Desvio padrão da população (δ) = 6.35
Tamanho da amostra (N) = 20

Com o valor crítico (α) de 0.05, o que corresponde a 95% de confiança), e com zc de 1.96, obteve-se um intervalo de confiança de (15.2, 20.8) segundos.

Conclusão:

Com base nos dados fornecidos, o intervalo de confiança 95% para a média da população é $15.2 < \mu < 20.8$, o que indica que estamos 95% confiantes de que a verdadeira média da população μ está contida no intervalo (15.2, 20.8) segundos.

Resultados:

Como resultado desta análise obteve-se a *timeline* da figura 6.1:

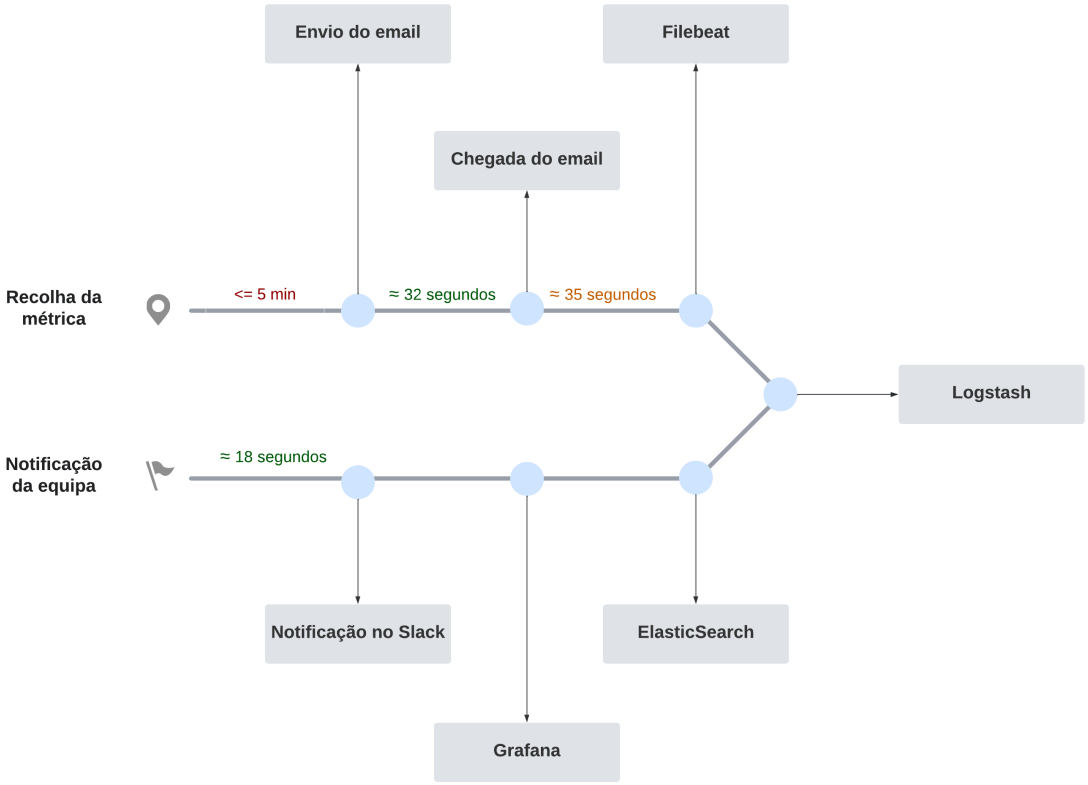


Figura 6.1: *Timeline* desde a recolha até à notificação de uma métrica

Ao analisar cuidadosamente todos os resultados obtidos, podemos inferir que há claramente uma fase do processo que ocupa a maioria do tempo, correspondendo ao tempo de recolha de uma métrica, analisado na secção 6.2.2. Há ainda uma outra fase que, embora a diferença seja pouco significativa, demora um pouco mais que as restantes, correspondendo ao tempo que decorre desde a chegada do email até ao envio para processamento do Filebeat, analisado na secção 6.2.2).

No mesmo seguimento, pôde igualmente concluir-se que as ferramentas da Elastic Stack apresentavam um grande desempenho a nível de tempos de processamento uma vez que, no conjunto das quatro ferramentas, foram as que melhores resultados obtiveram.

Por fim, e com base nesta análise e nos intervalos de confiança obtidos, calcularam-se os tempos mínimos e máximos que decorrem desde o momento em que a recolha da métrica é realizada até que se é notificado acerca da mesma.

Tempo **máximo** = 300 + 34 + 37 + 21 = 392 segundos $\approx$ 7 minutos Tempo **mínimo** = 1 + 23 + 32 + 15 = 72 segundos $\approx$ 1 minuto

6.2.2 Implementação

Envio de emails

Cada instrução de recolha e de envio de métricas está agendada a partir de uma expressão Cron. Este formato é utilizado para agendar tarefas repetitivas, executadas periodicamente em horários fixos, datas ou intervalos. [1] Por exemplo, a expressão `0 2/5 * ? * MON-FRI *`, que corresponde a uma das famílias de métricas, significa que a função agendada executa de 5 em 5 minutos, a começar no minuto 2, a todas as horas, de segunda a sexta, todos os meses.

Assim sendo, cada métrica tem uma expressão a si associada fazendo com que sejam executadas periodicamente, mediante o definido na expressão.

Anteriormente, a configuração da recolha e do envio de emails recorria a um processamento semelhante ao processamento por *batchs*.

O processamento por *batchs* corresponde ao processamento de várias transações através de um grupo ou de um lote de informações. Uma das vantagens deste tipo de processamento consiste na redução da sobrecarga do sistema para grandes quantidades de dados, uma vez que o processo é executado um menor número de vezes. Por sua vez, o processamento em *real time* trata de imediato os dados recebidos, sem necessidade de aguardar pela finalização de outros processos. Assim sendo, a grande vantagem desta abordagem, no contexto da plataforma, prende-se essencialmente na rapidez da obtenção das informações. Como anteriormente explicado, uma das necessidades da equipa de suporte era a de obter as informações o mais rápido possível para que a deteção de problema e consequente resolução fossem igualmente rápidos. Seguindo este tipo de processamento, as informações são enviadas a cada obtenção, evitando assim atrasos no envio de cada família de métricas e promovendo uma visualização mais rápida dos dados no Grafana.

Aplicando este método à plataforma Kamino, identificou que existia um *job* alvo de agendamento, denominado de `KAMINO_PACKAGE_AND_SEND`, que era responsável por fazer o empacotamento da informação e de posteriormente enviar essas métricas. Este agendamento era executado a cada 5 minutos, a todas horas e todos os dias. Enquanto isso, os restantes *jobs* eram unicamente responsáveis por recolher as várias métricas, tornando-as disponíveis para o posterior empacotamento e envio.

O processo referente a este *job* encontra-se exemplificado na figura 6.2.

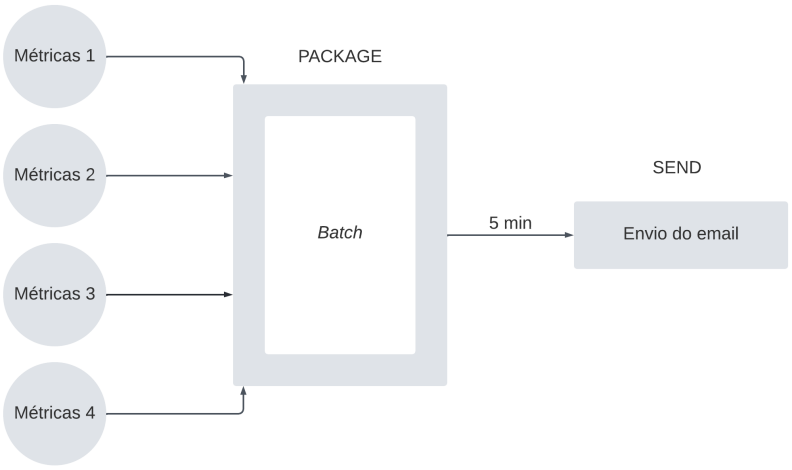


Figura 6.2: Implementação atual do processamento de métricas

Uma vez que o processamento por *batchs* atrasava, no máximo em 5 minutos, a transmissão das métricas,conclui-se que a utilização deste tipo de processamento não era ideal, pelo que se decidiu que um dos objetivos seria eliminar este *job* e tentar implementar algo semelhante a um processamento *real-time*. Isto é, o objetivo passaria por, assim que a métrica fosse recolhida, proceder de imediato ao seu envio eliminado o facto de aguardar pela recolha de outras métricas. A solução proposta encontra-se igualmente esquematizada na figura 6.3

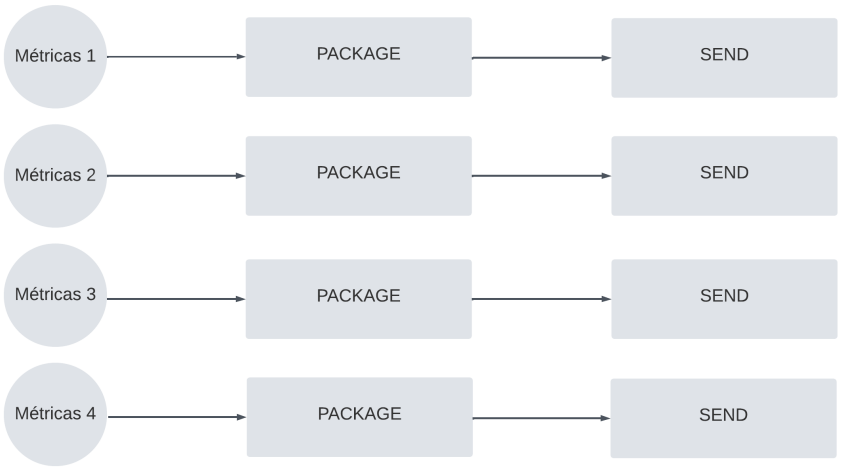


Figura 6.3: Solução proposta com a remoção da utilização de *batches*

Para ser possível implementar esta solução foi necessário garantir que o *refactor* efetuado no coletor de dados, no âmbito do novo tipo de processamento, foi realizado com o objetivo de permitir que as várias fases pudessem passar a ser executadas de forma atómica e independente, evitando assim o acesso e modificação simultânea por processos diferentes, o que poderia gerar vários conflitos.

Analisando a implementação inicial, observou-se que o armazenamento dos ficheiros era feito em pastas globais e indiferenciadas, normal considerando o pro-

cessamento anterior, mas que, considerando a solução proposta, traria alguns impedimentos.

Posto isto, optou-se por identificar cada pasta pelo nome da família de métricas a que a métrica em questão pertencia, adicionando-se também *timestamp* atual (data e hora).

Mailloader

Como identificado na análise da figura 6.1, havia ainda alguma latência associada ao processamento das informações por parte do Mailloader.

Depois de analisar os procedimentos inerentes a este *container*, inferiu-se que a chegada de novos emails era verificada a cada 30 segundos. Este agendamento era conseguido novamente através de um agendamento Cron, responsável por executar um *script* em Bash de minuto a minuto. Uma vez que este tipo de agendamento não possui a granularidade do segundo, tipicamente não seria possível agendar processos de 30 em 30 segundos. No entanto, para contornar esta situação, recorreu-se ao uso de *sleeps*, simulando a execução do *script*, neste caso, 2x por minuto, como demonstrado na figura 6.4.

Posto isto, com o objetivo de reduzir a latência já identificada, reduziu-se o tempo de processamento para 20 segundos, passando a chegada de novos emails a ser verificada 3x por minuto.

```
* * * * * /kamino-apps/mailloader/software/mailloader.sh >> /kamino-apps/mailloader/logs/mailloader.log 2>&1
* * * * * sleep 20; /kamino-apps/mailloader/software/mailloader.sh >> /kamino-apps/mailloader/logs/mailloader.log 2>&1
* * * * * sleep 40; /kamino-apps/mailloader/software/mailloader.sh >> /kamino-apps/mailloader/logs/mailloader.log 2>&1
```

Figura 6.4: Excerto de código que contém os agendamentos Cron relativos ao Mailloader

6.2.3 Testes

De forma a promover as alteração implementadas a produção, sem que isto causasse impacto à funcionalidade e operação da plataforma, foi necessário configurar o coletor de dados numa máquina existente do projeto. Aquando da configuração de uma máquina de *staging*, na *milestone* detalhada no capitulo 6.1, procedeu-se à replicação da base de dados, de toda a *stack* da Elastic, Mailloader e também do Grafana. Porém, a configuração do coletor de dados não foi incluída nessa replicação. No entanto, com as alterações efetuadas no âmbito desta *milestone*, percebeu-se que a configuração do coletor seria fulcral para a fase de desenvolvimento e também para a fase final de testes, de forma a garantir a qualidade das alterações propostas. Tendo em conta que o coletor de dados necessita de estar associado a um ambiente do *software* bancário, considerou-se que não era viável criar um ambiente deste tipo única e exclusivamente para o desenvolvimento e teste do Kamino. Assim sendo, e uma vez que a equipa de DevOps detinha já alguns ambientes dedicados à realização testes, recorreu-se à

configuração e posterior utilização de um deles. Embora tenha sido necessário reestruturar algumas questões, próprias de cada ambiente, esta solução mostrou-se menos complexa e dispendiosa do que anterior solução de criação e configuração de um novo ambiente.

Numa fase inicial, cada métrica foi testada isoladamente para garantir o seu correto funcionamento, tendo os resultados sido registado na tabela 6.2. Esta tabela relaciona cada família de métricas com o número de execuções realizadas, registando, por fim, o resultado obtido.

Posteriormente, e a fim de verificar se a paralelização de execuções era garantida, recorreu-se a um *script* em Bash que, fazendo uso do operador `&`, simulava várias chamadas paralelas. A tabela 6.17 apresenta os resultados obtidos, estabelecendo a ligação entre o número de famílias de métricas distintas utilizadas em cada teste com o número de execuções realizadas. Considera-se por 1 execução paralela com 2 famílias de métricas distintas, a chamada única de cada família de métricas envolvidas, perfazendo o total de 2 chamadas por *script* de teste. Seguindo o mesmo raciocínio, 2 execuções paralelas de 2 famílias de métricas distintas correspondem a um total de 4 chamadas por *script* de teste, representado no teste nº1.

Nota: A cor verde indica que o teste em questão cumpriu com os requisitos e que portanto, apresentou o resultado esperado, correspondendo neste caso à correta execução de cada *script* de teste.

Teste	Família de métricas	Nº de execuções	Resultado
1	BPM Performance	1	
2		10	
3	AS400 Performance	1	
4		10	
5	DB2 Performance	1	
6		10	
7	Frontend Performance	1	
8		10	
9	Monitor Systems	1	
10		10	
11	SMS Performance	1	
12		10	
13	WAS Performance	1	
14		10	

Tabela 6.2: Testes individuais

Teste	Nº famílias de métricas	Nº de execuções paralelas	Resultado
1	2	2	
2	2	4	
3	3	3	
4	3	6	
5	4	4	
6	4	8	
7	5	5	
8	5	10	
9	6	6	
10	6	12	
11	7	7	
12	7	14	

Tabela 6.3: Testes de paralelização

6.2.4 Resultados e entrega

Como já mencionado, depois de todas as alterações terem sido testadas internamente foram promovidas a produção e comunicadas à equipa. No entanto, devido a outros constrangimentos da máquina do Kamino de produção, detetaram-se alguns conflitos que foram de imediato identificados e resolvidos.

Como resultado das alterações efetuadas, obteve-se a *timeline* da figura 6.5.

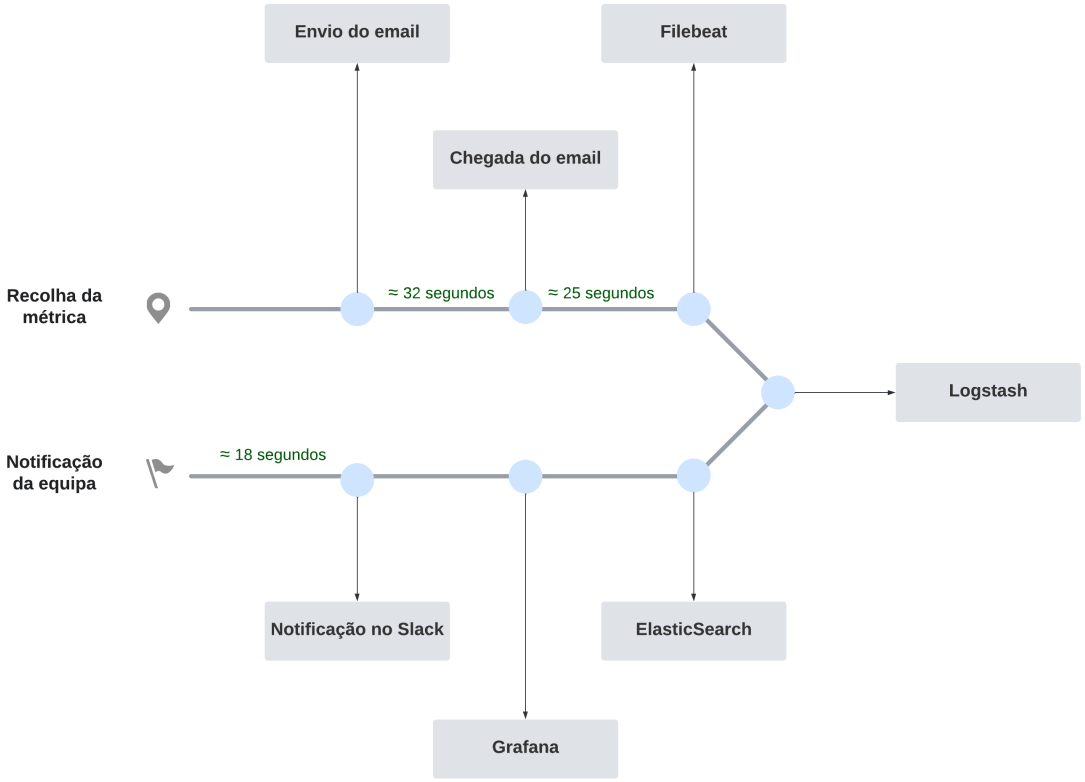


Figura 6.5: *Timeline* desde a recolha até à notificação de uma métrica

Refazendo a análise de tempos anterior, obtiveram-se os seguintes resultados:

Tempo **mínimo** = 23 + 22 + 15 = 60 segundos ≈ 1 minuto
Tempo **máximo** = 34 + 27 + 21 = 82 segundos ≈ 1.40 minutos

Para melhor perceber o impacto das alterações implementadas, efetuou-se uma comparação dos tempos de *best* e de *worst case*, descrita na tabela 6.4.

Best case		*Worst case*	
Antes	Depois	Antes	Depois
1m13s	1m	6m53s	1m22s

Tabela 6.4: Comparação de tempos

Os resultados obtidos permitem concluir que houve uma melhoria bastante significativa nos tempos de processamento de métricas, o que em termos práticos se traduz num *speedup* de 504% relativamente ao *worst case*.

Por fim, e para que ser possível validar e marcar a *user story* relativa a esta *milestone* como completa, analisaram-se os AC estabelecidos na secção 4.3:

AC	Resultado
(a) O sistema é determinístico para a transmissão de métricas	✓
(b) O *worst case* não ultrapassa os 3 minutos	✓

Tabela 6.5: Validação dos AC definidos no âmbito da redução da latência na transmissão de métricas

Relativamente ao ponto (a), ao optar pela processamento em *real time* em detrimento do processamento por *batches*, é garantido que o processamento é determinístico e que os tempos de processamento são iguais para cada família de métricas. Já no ponto (b), e analisando os tempos registados na tabela 6.4, conclui-se que o tempo relativo ao *worst case* é inferior a 3 minutos, o que permite assinalar esta *user story* como finalizada.

6.3 Impedir corrupção de arquivos aquando da receção

No passado, para que fosse possível proceder ao envio de informação reunida num ficheiro zip, foram realizadas algumas modificações relativamente ao formato do ficheiro que se pretendia enviar. Uma vez que a extensão de ficheiros `.zip` fazia parte das extensões bloqueadas pelo servidor de *email*, para ser possível enviar este ficheiro foi necessário renomeá-lo para passar a incluir uma extensão que não fosse bloqueada pelo Outlook, tendo-se optado pelo formato permitido de texto `.txt`. [4]

Ao aumentar o fluxo de envio e receção de emails, tornou-se visível e mais recorrente um problema já existente e relativo à corrupção de alguns dos dados. Este problema era visível ao nível dos *logs* do Mailloader, sendo endereçadas questões como a obtenção de novos *emails* ou a separação do *email* em informações válidas para processamento.

Assim sendo, aquando da identificação do problema, inferiu-se que este era refletido na fase de separação de *emails*. Esta componente encontrava-se implementada em Talend [19], o que, não tendo o código fonte dessa componente, tornava difícil a deteção da origem do problema.

6.3.1 Análise técnica

Posto isto, para localizar o problema esta questão e tendo em conta que o *debug* através do Talend não era possível, acedeu-se por *SSH File Transfer Protocol* (SFTP) à maquina e retirou-se um dos ficheiros aparentemente corrompidos. Posto isto, refez-se manualmente o processo de descompactação da informação e confirmou-se que, de facto, o ficheiro estava corrompido.

Para localizar a origem do problema seguiu-se todo o fluxo do processamento

o que levou a concluir que o email era enviado corretamente e que o problema residiria então aquando da receção. Depois de alguma investigação, percebeu-se que estes ficheiros corrompidos possuíam um *End of line* (EOL) diferente do esperado, o que impedia a correta descompressão.

6.3.2 Implementação

Como referido, a antiga implementação visava o processamento de ficheiros de texto `.txt`, tornando isso mais propício ao *parsing* de texto, quando na verdade se trata de um ficheiro binário. A fim de impedir essa corrupção, optou-se por alterar a extensão do ficheiro para a de um ficheiro binário `.bin`, sendo uma extensão permitida pelo servidor de email. [4] Esta alteração permitiu manter o ficheiro legítimo, impedindo assim o processamento de texto e a consequente incorreta conversão de dados.

Ponderou-se ainda na reconfiguração do processo de codificação para Base64. No entanto, essa reestruturação iria requerer algum esforço extra o que não seria exequível, tendo em conta que a solução atual era já viável e funcional.

6.3.3 Testes

A fim de validar as correções efetuadas, foram injetados ficheiros contendo a nova extensão com formato binário na máquina de *staging* do Kamino. No final dessa operação, analisaram-se os *logs* do *Mailloader* e confirmou-se que, de facto, o processamento de emails tinha prosseguido. Numa outra vertente, tinha já sido anteriormente configurado um canal de alerta no Slack destinado a todos os alertas da máquina de *staging*, com o objetivo de não perturbar os canais principais de comunicação da equipa. Com o desimpedimento da *pipeline* do Mailloader, era suposto que a informação fluísse através dos *containers* e que estes chegassem à camada de visualização do Grafana. Assim sendo, a chegada da notificação ao canal de Slack alertando novamente para a presença de dados foi a confirmação de que a solução escolhida tinha, de facto, sido implementada com sucesso.

6.4 Rever e alterar a frequência de obtenção de métricas

No seguimento do problema identificado pela equipa de suporte na secção anterior, surgiu também a necessidade de rever as frequências de obtenção de métricas e de as ajustar caso seja necessário. Assim é possível garantir que, conjuntamente com a otimização da latência, a informação é obtida e consequentemente visualizada o mais rápido possível.

6.4.1 Análise técnica

Com o objetivo de melhor perceber a distribuição de métricas ao longo dos agendamentos, procedeu-se à realização de um levantamento técnico, que consistiu essencialmente na recolha, por meio de tabelas da base de dados, dos agendamentos configurados para cada uma.

Posteriormente, para perceber qual o tempo de execução de cada família de métricas, recorreu-se à análise de *logs* de cada uma. Com base nesta análise retiraram-se cerca de 10 dados, utilizando-os no cálculo posteriormente da média do tempo de execução.

Os resultados obtidos foram registados na tabela 6.6.

ID	**Família de métricas**	**Tempo de execução** (s)	**Agendamento**
1	BPM Performance	11	0 2/5 * ? * MON-FRI *
2	AS400 Performance	7	0 3/5 * ? * MON-FRI *
3	DB2 Performance	77	0 0/5 * ? * MON-FRI *
4	DB2 Performance Archive	35	0 0 8 ? * MON-FRI *
5	DB2 Performance Catalog	2h	0 15 8 ? * MON-FRI *
6	Frontend Performance	20	0 2/5 * ? * MON-FRI *
7	Monitor Systems	44	0 4/5 * ? * MON-FRI *
8	Package and send	9	0 5/5 * ? * MON-FRI *
9	SMS Performance	4	0 1/5 * ? * MON-FRI *
10	WAS Performance	220	0 0/5 * ? * MON-FRI *

Tabela 6.6: Análise técnica da configuração de métricas

Analisando os agendamentos em `CRON` configurados, é possível inferir que todas as métricas, à exceção da métrica com ID4 e ID5, ocorrem num intervalo de 5 minutos, estando cada uma delas configurada para disparar em momentos diferentes dos 5 minutos. O ID4 e o ID5 não foram considerados na análise elaborada tendo em conta que apresentam uma frequência de obtenção relativamente baixa, de uma vez por dia. Estas métricas são relativamente constantes ao longo dia, tendo-se considerado que a sua obtenção diária seria suficiente. Por outro lado, e como resultado da secção 6.2, o agendamento do ID8 foi descontinuado, e portanto também não vai ser considerado neste capítulo.

Para melhor visualizar os agendamentos, elaborou-se um gráfico que exibe o momento de disparo de cada métrica, identificada pelo ID.

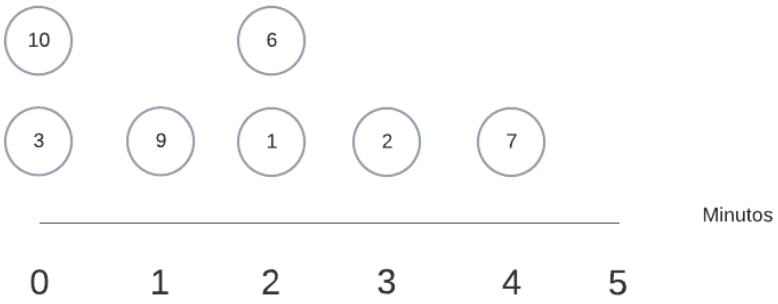


Figura 6.6: Posicionamento atual dos agendamentos

Como exceção, temos as métricas ID4 e ID5, que possuem o seguinte agendamento:

ID4 - Às 8h, todos os dias, todos os meses;
ID5 - Às 8:15h, todos os dias, todos os meses.

Considerando a implementação atual de envio em tempo real, foi necessário ter em consideração que a tomada de decisões não causava sobrecarga de agendamentos, com o intuito de não criar *overhead* no processo de processamento e envio de métricas.

6.4.2 Implementação

Separar métricas de performance

Como mencionado pela equipa de suporte, a obtenção de dados relativos a *threads* era uma questão extremamente importante a analisar e posteriormente a otimizar.

A função responsável pelo processamento de métricas relativas a performance (CPU, memória ou *garbage collector*), obtidas por *Java Management Extensions* (JMX) era anteriormente também responsável pelo processamento de métricas relativas a *threads*. No entanto, essa acoplação gerava a espera *inline* por todos os processamentos necessários, o que tornava este método bastante pesado.

Como solução optou-se então por separar, ao nível do coletor de dados, as métricas relativas a *threads* das métricas relativas a performance e obtidas por JMX. Esta alteração permitiu não só otimizar a recolha deste tipo de métricas, uma vez que as operações eram mais leves, como evitou também a possível sobrecarga no agendamento configurado na função original.

Criar novo agendamento para a recolha de métricas de *threads*

Como resultado do secção anterior, surgiu a necessidade de criar e de configurar um novo agendamento para o processamento de métricas relativas a *threads*.

Com esse intuito, foi necessário analisar qual a posição mais conveniente para colocá-lo e qual a frequência adequada. Era importante que esta decisão respeitasse algumas condicionantes, sendo que um dos pontos chave seria a distribuição equilibrada de carga ao longo dos minutos.

Figura 6.7: Alterações nos agendamentos

Correspondendo o ID 10.1 à função original (ID 10) e o ID 10.2 à função relativa a *threads*, considerou-se que seria útil reajustar a frequência desta última para 2 minutos. Relativamente ao posicionamento, o minuto 1 e o minuto 3 eram dos minutos que menos carga tinham, tendo-se então optado por colocar nesses pontos.

Otimizar o processamento de métricas relativas a *threads*

Considerando a análise técnica refletida na tabela 6.4, concluiu-se que as métricas relativas à performance do WAS, obtidas por JMX - CPU, memória ou *garbage collector* (ID10 - WAS Performance) são aquelas que, atualmente, consomem mais tempo, desde o início da sua obtenção até ao envio dos dados.

Após algum estudo e decomposição dessa função nas várias fases de processamento, determinou-se que a maior parte do tempo era despendido em **esperas ativas**, assim como o detalhado na tabela 6.7.

Método	**Tempo de execução** (s)	**Nº de execuções**	**Tempo**
Sleep	10	4	0m:40s
Sleep	30	4	2m:00s
Recolha de métricas	2	8	0m:16s
Empacotamento e envio	9	1	0m:09s

Tabela 6.7: Decomposição temporal do processamento de métricas relativas a *threads*

A análise do código sugeria que os *sleeps* eram referentes à espera pela escrita de um ficheiro `JSON`. No entanto, a forma como a escrita deste ficheiro estava

construída garantia por si só que a próxima operação apenas era efetuada quando esta escrita terminasse, pelo que o uso destas esperas ativas era desnecessária. A fim de garantir que a sua remoção não causava quaisquer constrangimentos, foram efetuados alguns testes, que se encontram documentados na secção 6.4.3.

Alterar o agendamento de eventos e métricas de *request performance*

Uma vez mais, como resultado da reunião com o *team leader* da equipa de suporte, identificou-se a necessidade de rever as métricas relativas a eventos e a métricas de *request performance*, de forma a que fossem recolhidas com mais frequência.

Ao analisar o método correspondente ao processamento e recolha destas métricas, inferiu-se que, das seis *queries* que são executadas, apenas duas não faziam parte do grupo que se pretendia otimizar (eventos e *request performance*).

Recorrendo novamente à análise técnica da tabela 6.6, retirou-se que o conjunto das seis *queries* demoravam no total cerca de 20 segundos. No entanto, e considerando a solução implementada de remoção do *batching*, é necessário retirar a esse tempo, o tempo de empacotamento e de envio, o que resulta em cerca de 2 segundos por *query*.

Posto isto, e tendo em conta a rápida execução de cada uma das *queries*, concluiu-se que não seria necessário dividir a função nas métricas pretendidas. Uma vez que o foco se concentrava em 4 das 6 métricas, a possível separação de funções traria, no máximo, uma otimização de 4 segundos, o que não justificava a criação de um novo agendamento e o consequente envio de mais emails.

Reunindo todas as conclusões retidas, a solução adotada passou assim por obter as métricas com mais frequência, tendo sido apenas necessário modificar o agendamento Cron. Este agendamento estava anteriormente configurado para ocorrer a cada 5 minutos e depois de aplicada esta alteração, passou a ocorrer a cada 3 minutos.

Separar métricas de performance de base de dados

À semelhança da secção anterior, surgiu também a necessidade de rever métricas de *long running queries* e de *locks*. A fim de perceber onde se poderia otimizar estes aspetos, foi efetuada uma análise técnica que resultou na tabela 6.8.

Método	**Tempo de execução** (s)	**Nº de execuções**	**Tempo**
Execução de *queries*	12	6	1m:12s
Empacotamento e envio	9	1	0m:09s

Tabela 6.8: Decomposição temporal do processamento de métricas relativas à performance de base de dados

Como se pode inferir pela decomposição efetuada, a função responsável pela re-

colha destas métricas é também responsável pela recolha de outras 5 métricas. Considerando que cada conjunto de métricas relativas à base de dados demorava cerca de 12s a ser obtido, a anterior acoplação tornava-se pouco ineficiente para as métrica que pretendíamos. Assim sendo, a forma de otimizar este aspeto foi separá-la das restantes.

Criar novo agendamento para a recolha de métricas de *long running queries* e de *locks*

Como resultado da secção anterior, surgiu a necessidade de configurar um novo agendamento para o processamento de métricas relativas a *long running queries* e a *locks*. Para isso, foi necessário analisar qual a posição mais adequada para o colocar e qual a frequência que devia apresentar, tendo sempre em conta as condições dos agendamentos gerais.

Correspondendo o ID 3 à função original e o ID 3.1 à função relativa a *long running queries* e a *locks*, considerou-se que seria adequado reajustar a frequência para 3 minutos. Relativamente ao posicionamento, o minuto 4 era dos que apresentava menos carga. No entanto, ao colocar a frequência e o minuto indicado, o minuto 0 ficava sobrecarregado. Para contornar esta questão, optou-se por fazer *shift* de 1 minuto do agendamento do ID3, tendo o agendamento final ficado com o aspeto da figura 6.8.

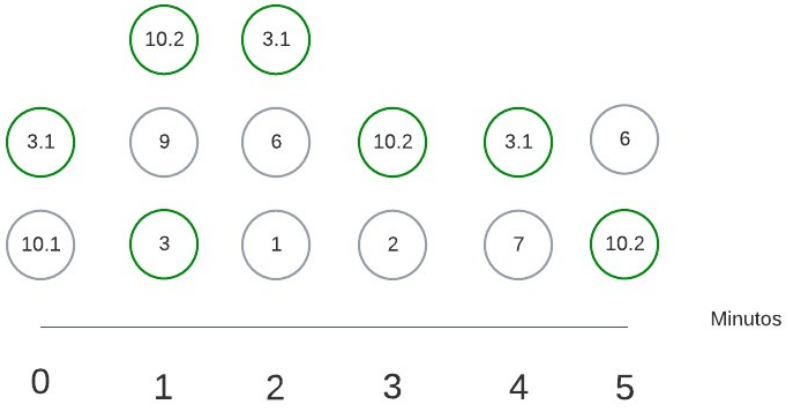


Figura 6.8: Alterações nos agendamentos

6.4.3 Testes

A fim de validar a questão relativa à otimização de *threads*, foram efetuados alguns testes assim como a simulação de execuções paralelas. Com a eliminação das esperas ativas, era importante garantir que mesmo em execuções sucessivas, a função tem o comportamento desejado.

Nos testes efetuados variou-se não só o tempo de espera de cada *sleep* como também o número de execuções paralelas. Para simular e verificar esta última questão, recorreu-se a um *script* em Bash que simulava a situação pretendida.

Teste	**Sleep**	**Exec. paralelas** (s)	**Resultado**
1	30s	4	
2	30s	2	
3	15s	2	
4	10s	2	
5	-	2	
6	-	3	
7	-	5	
8	-	10	
9	-	20	

Tabela 6.9: Teste de execuções paralelas

Para garantir com mais confiança de que a alteração proposta não traria quaisquer constrangimentos, elaborou-se um caso de teste com condições extremas e bastante improvável de acontecer, a fim de perceber se o comportamento era de facto o esperado. No teste 9, ao validar um número de execuções paralelas relativamente elevado, consegue garantir-se que, mesmo em casos de extrema frequência de pedidos, o comportamento se mantém como o desejado.

6.4.4 Resultados e entrega

Como resultado desta secção, as modificações efetuadas aos agendamentos resultaram no seguinte esquema:

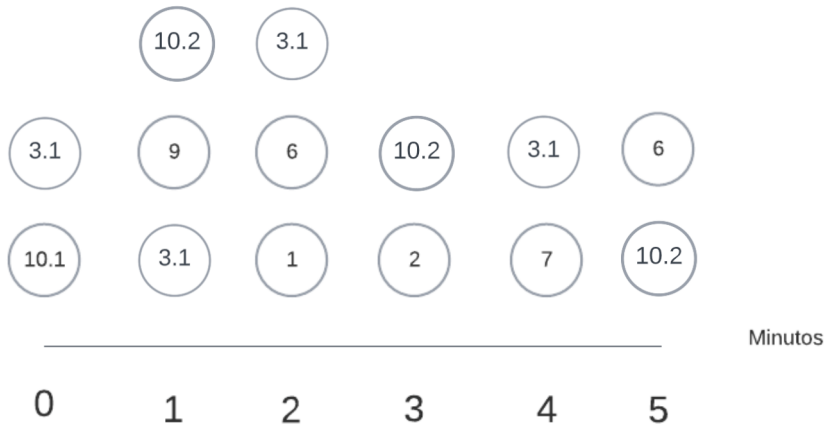


Figura 6.9: Resultado final do posicionamento de agendamentos

Legenda:

1 - BPM Performance
2 - AS400 Performance
3 - DB2 Performance
3.1 - DB2 Locks and Long Running Queries

6 - Frontend Performance
7 - Monitor Systems
9 - SMS Performance
10.1 - WAS Performance
10.2 - WAS Performance Threads

Todas as modificações efetuadas, incluindo a alteração de frequências ou a separação de funções, garantiram a otimização da obtenção de métricas, que era o objetivo pretendido.

Para melhor se visualizarem os efeitos das alterações, analisaram-se os casos práticos que a equipa de suporte considerou mais relevantes, comparando-se, para cada caso, o tempo necessário antes e depois das alterações.

Caso prático	*Best case*		*Worst case*	
	Antes	Depois	Antes	Depois
Threads bloqueadas	2m40s	18s	7m40s	2m18s
Long running queries	1m20s	37s	6m20s	2m37s
Request performance	28s	28s	5m28s	3m28s

Tabela 6.10: Comparação de tempos de obtenção de métricas

As alterações propostas foram realizadas ao nível do coletor de dados o que implicou que, para proceder à entrega de valor do produto, fosse necessário efetuar uma nova *release*.

Por fim, e com o objetivo de ser possível validar e marcar a *user story* referente a esta *milestone* como completa, analisou-se o AC estabelecido na secção 4.3.

AC	**Resultado**
(a) O *worst* case do tempo relativo à transmissão e visualização de métricas identificadas pela equipa de suporte não ultrapassa os 5 minutos	✓

Tabela 6.11: Validação dos AC definidos no âmbito da revisão da frequência de obtenção de métricas

Analisando a tabela 6.10, é possível retirar que o tempo mais elevado relativo ao *worst case* é de cerca de 3 minutos. Assim sendo, é possível concluir que o AC definido se encontra implementado e que consequentemente, esta *user story* pode ser marcada como concluída.

6.5 Rever e reconfigurar totalmente a alarmística

A configuração de alertas permite detetar problemas no sistema alguns momentos após estes terem acontecido. Ou seja, quanto mais robustos os alertas forem, mais úteis serão na identificação e resolução rápida de problemas, minimizando a interrupção dos serviços.

Assim sendo, esta secção tem o objetivo de detalhar as mudanças efetuadas no âmbito da revisão e da reconfiguração da alarmística, abordando questões como a revisão da responsividade de alertas.

6.5.1 Análise técnica

Analisando o modo de configuração dos alertas existentes, verificou-se que esta era feita através da interface gráfica do Grafana. Nesta é possível configurar as *queries*, colocá-las num *dashboard* e definir também as janelas temporais que esta analisa em cada momento.

Assim sendo, esta interface encontra-se refletida na figura 6.10 e permite configurar mais detalhadamente os seguintes aspetos:

- **Condição** - indica a *query* que vai acionar a regra do alerta, incluindo o *threshold* de disparo;
- ***Evaluate every*** - especifica a frequência de avaliação da *query*;
- ***Evaluate for*** - especifica a duração pela qual a condição deve ser verdadeira antes que o alerta seja acionado.

No caso de ser adicionado algum valor na componente `For`, o alerta aguarda o tempo configurado antes de emitir algum alerta, passando a `Pending` durante essa espera. Posteriormente, e caso a condição do alerta se mantiver verdadeira, o alerta passará de `Pending` para `Alerting` e enviará notificações de alerta. Ou seja, em termos práticos este tempo corresponde, no caso de ter sido acionada uma regra de alerta, ao tempo que decorre desde o estado `OK` ao estado de `Alerting`.

Já o `Evaluate every` corresponde à janela temporal que a *query* vai analisar, garantindo a passagem do estado `Alerting` ao estado de `OK`.

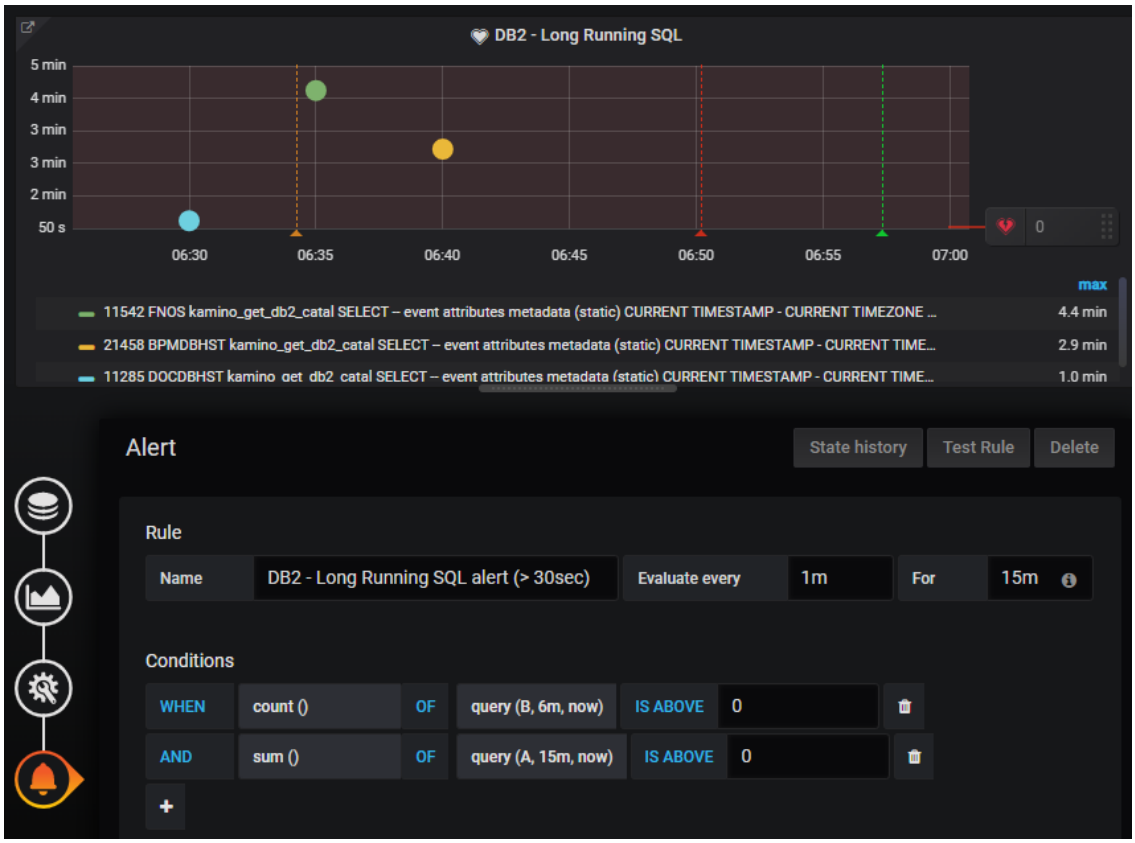


Figura 6.10: Interface de configuração de alertas no Grafana

6.5.2 Implementação

Workaround do duplo disparo do *Frontend Scheduler* (FES)

Ao analisar os *logs* de cada família de métricas, detetou-se que ocorriam alguns erros ocasionais e com frequência diária. Esta situação já tinha sido anteriormente identificada sendo que se atribuiu a causa do problema um possível *double fire* por parte do servidor.

Este *double fire* dizia respeito a um disparo por nó, constituindo esta questão uma eventual falha ou limitação no mecanismo de agendamentos. Como consequência, a nível do coletor de dados, os dados eram obtidos 2 vezes mas nenhum deles não era enviado, o que constituía por si só um grande problema para a plataforma. Assim sendo, e visto que os ficheiros originados pelo duplo disparo apresentavam *timestamps* e nome de *scope* iguais, esta situação conduzia à colisão de ficheiros e à consequente perda de dados.

Este problema era já conhecido pela equipa e encontrava-se fora do *scope* do estágio. Precisamente por isto e por ser um assunto de maior complexidade e impacto, considerou-se que não seria viável seguir uma abordagem que levasse a uma solução mais permanente e transversal.

Assim sendo, a solução que se tomou para contornar este problema passou por acrescentar um número aleatório, entre 0 e 99999, à *path* de cada ficheiro para

impedir a colisão e a consequente perda dos dados.

Corrigir falha de alguns alertas no arranque

Embora a recolha de métricas seja realizada durante 24h, de segunda a sexta-feira, o processamento destas métricas e a alarmística inerente apenas são efetuados no horário de 6h-18h. Assim sendo, aquando do arranque da máquina pela manhã, existe sempre um *backlog* de 6h que é necessário processar.

Posto isto, identificou-se que no arranque do Kamino de produção existiam alertas que disparavam sempre com a indicação de falta de dados (`NO DATA`) e que, decorridos alguns minutos, voltavam ao estado esperado (`OK`). Sendo este um problema recuperável e apenas motivado pelo atraso no processamento destas métricas no início do dia, esta questão necessitava de ser revista considerando também que trazia ainda mais interferências nos canais de Slack da equipa de suporte.

Em termos práticos, todos os dias, por volta das 6h da manhã, os seguintes alarmes disparavam num dos canais de Slack da equipa:

- JAVA - Blocked Threads;
- READ_TIMEOUT (2004) Event;
- CONNECTION_POOL_EXAUSTED (2023) Event;
- AS400_CONNECTION_DROP (2092) Event;

Uma vez que o problema era determinístico, depois de alguma análise concluiu-se que o tempo de espera configurado anteriormente para o arranque da interface do Grafana não era o suficiente para indexar toda a informação. Este tempo era estático (15 minutos) e ao arrancar o Grafana antes de toda a informação estar processada e indexa na Elastic Search, os alarmes disparavam, sem haver necessidade para tal.

Para isto foi construído um *script* de Python que recorre à biblioteca do DockerSDK para Python [9]. Este *script* foi feito com o objetivo de reduzir as probabilidades de alertas com `NO DATA`, correspondendo a falso-positivos.

Adotando uma linguagem de alto nível, o *script* ligava-se ao *container* do Filebeat, através da biblioteca já referida, e analisava todos os *logs* desse *container* desse dia. Posteriormente, as condições verificavam se o *timestamp* em processamento no Filebeat era próximo do *timestamp* atual. Caso fosse, significava que a fila de processamento era pequena e que a execução podia então prosseguir. Numa outra fase, esperava-se até que os dados chegassem ao Filebeat, vindos do Mailloader, para impedir a ocorrência de falsos positivos relacionados com a ausência de dados. Isto é, sem esta questão, o sistema indicaria a ocorrência de anormalidades quando na realidade, nessa iteração, os dados ainda não tinham chegado.

Por fim, ao executar todo o *script* corretamente, era dada a instrução de arranque do Grafana.

Analisar e rever *threshold* de alertas

Numa fase inicial foi feita toda a análise técnica referente aos alertas, englobando desde informações referentes aos *thresholds* até à família de métricas a que cada alerta pertence.

Posto isto, e no seguimento da reunião com o *team leader* da equipa de suporte, surgiu a necessidade de alterar alguns *thresholds* que estavam desajustados com as necessidades atuais, e ainda a de apagar alguns dos alertas, que não sendo já necessários, acabavam por constituir ruído nos canais de alerta.

As alterações foram as seguintes:

- ***Threads* bloqueadas**

Este alerta diz respeito ao número de *threads* bloqueadas. Tendo em conta o *feedback* de suporte e considerando o *threshold* atual, o alerta era disparado quando atingia certo valor, mas rapidamente recuperava e ficava `OK`. Posto isto, surgiu a necessidade de aumentar o *threshold* para diminuir o número de "interferências"e para dar ênfase a problemas de *threads* bloqueadas que efetivamente não recuperem e que precisem de intervenção humana.

O anterior *threshold* de 20 *threads* passou para 50 *threads*.

- ***Transation log***

Este alerta era referente a uma métrica especifica da base de dados, sendo que o anterior *threshold* de 10% passou agora para 20%.

- **Eventos de *read timeout***

Este alerta era referente à ocorrência de eventos *read timeout*. No entanto. chegou-se à conclusão que este tipo alerta já não se enquadrava atualmente nas necessidades da equipa, pelo que foi removido.

- **Pedidos DJU**

Este alerta diz respeitos a pedidos efetuados à direção jurídica do banco e à semelhança do alerta anterior, já não se enquadra nas necessidades da equipa, pelo que foi também removido.

Em suma, foram feitas as seguintes alterações:

Alerta	*Threshold*	
	Antes	Depois
Threads bloqueadas	20	50
Transation log	10%	20%
Remoção da alarmística relativa a: Eventos de *Read timeout*		
Remoção da alarmística relativa a: Pedidos DJU		

Tabela 6.12: Alterações efetuadas

Rever responsividade de alertas

Uma questão bastante importante na alarmística é a sua responsividade, correspondendo isto à capacidade de dar uma resposta rápida e adequada a cada situação.

Com a utilização regular da plataforma, claramente se percebe que alguns dos alertas não possuíam uma configuração que permitisse atingir uma boa responsividade.

Como podemos ver na figura 6.11, o alerta disparou quando o problema já tinha sido ultrapassado. Ou seja, num cenário hipotético, a equipa recebeu o alerta no canal de Slack e reuniu esforços para localizar e resolver o problema. No entanto, no momento em que receberam a notificação, o problema já se encontrava resolvido, o que tornou este esforço desnecessário.

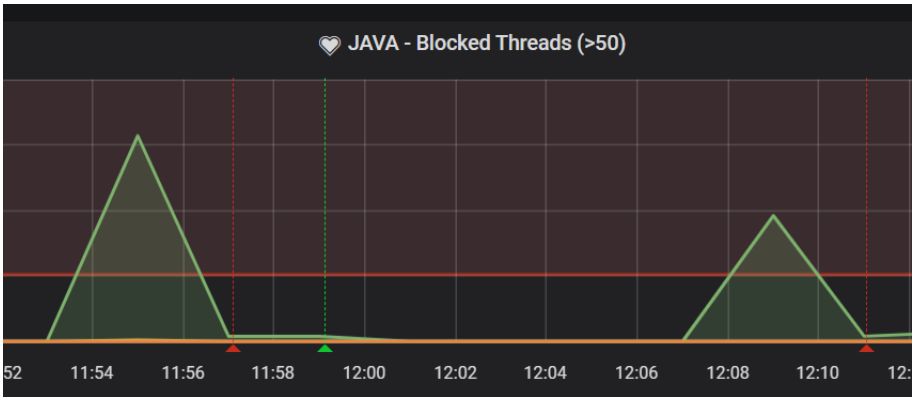


Figura 6.11: *Dashboard* de *threads* bloqueadas

Para minimizar os impactos que este problema provocava, foram efetuadas alterações a nível da responsividade, sendo elas as seguintes:

- **SMSs enviados**

Este alerta diz respeito ao número de SMSs que são enviados para uma operadora do país em causa. Uma vez que esta operadora tem cada vez menos utilizadores, a janela temporal de análise encontrava-se desajustada com a realidade, pelo que foi necessário aumentá-la.

A janela temporal anterior era de 30 minutos sendo que passou a ser de 2 horas.

• **Transaction log**

Identificou-se que, após ocorrer um erro e consequente disparo de um alerta relativo à base de dados, *transaction log*, apenas passado 1h é que era disparado um novo alerta com a indicação de que estava tudo bem. Para a equipa de suporte esta questão não era aceitável, devido à latência associada à deteção do erro. Assim sendo, a janela temporal, que anteriormente era de 1 hora, foi revista e passou a ser de 15 minutos.

De forma mais concisa, as alterações efetuadas foram sintetizadas na tabela 6.13.

Alerta	Janela temporal de análise da query	
	Antes	Depois
Sent - Movicel	30m	2h
Transation log	1h	15m

Tabela 6.13: Alterações efetuadas à janela temporal de análise das *queries*

A fim de continuar a realizar melhorarias a nível da responsividade dos alertas, foi necessário refazer a análise técnica do tempo de processamento de cada família, correspondente ao *Round Trip Time* (RTT), considerando as modificações implementadas nas secções anteriores.

O RTT é calculado com base na soma do tempo de obtenção, da periodicidade (atualizados na secção 6.4) e por fim do *overhead* base. Este *overhead* base corresponde ao tempo que decorre desde a obtenção dos dados até à sua visualização no Grafana, espelhado na figura 6.5

RTT = Tempo de obtenção + Periodicidade + *Overhead* base

A fim de garantir que a alteração dos tempos não causava nenhum constrangimento na visualização dos dados e dos alertas, considerou-se que era importante acrescentar algum tempo extra ao valor do RTT. O tamanho da folga definida correspondeu a **60%** do valor do RTT, tendo a decisão final do tempo sido baseada igualmente nesse valor.

Decisão final = RTT + Folga

Para melhor identificar as alterações efetuadas, elaborou-se a tabela 6.14 que reúne as alterações de todos os alertas afetados.

Alerta	RTT	Folga	Decisão	Antes
Locks longer than 5s	34s + 2m + 1m15s = 3m49s	2m17s	6m	1h
Long Running SQL				
JDBC Failures	32s + 5m + 1m15s = 6m47s	4m04s	10m	1h
Ping Failures				
Port Failures				
WSDL Failures				
Aguarda Notificação	16s + 5m + 1m15s = 6m31s	3m:55s	10m	30m
Aguarda Notificação (Transferências/ Depósitos)				25m
Pool: ADM_SIS				15m
Serviços Centrais				15m
Blocked Threads	15s + 2m + 1m15s = 3m30s	2m06s	6m	15m
Connection Pool Exausted	22s + 3m + 1m15s = 4m37s	166s	7m	15m
AS400 Connection Drop				

Tabela 6.14: Alterações efetuadas na alarmística

6.5.3 Testes

Considerando a questão do *workaround* do duplo disparo do FES, foram efetuados alguns teste antes das alterações terem sido promovidas a produção. Uma vez que, no âmbito da tarefa da secção 6.2, o coletor de dados do Kamino tinha sido já configurado num ambiente interno, os testes foram realizados sobre esse ambiente. Os testes passaram novamente pela injeção de ficheiros com o novo formato e na consequente análise de *logs*, a fim de perceber o comportamento das métricas com esta alteração. Os *logs* não apresentaram qualquer anomalia, tendo-se concluído que a implementação estava corretamente desenvolvida.

Analisando agora a questão relativa aos alertas que disparavam aquando do arranque matinal, efetuaram-se alguns testes maioritariamente relativos à replicação de sucessivos arranques da máquina do Kamino Staging. Esta simulação permitia tentar reproduzir o motivo que anteriormente conduzia ao erro e verificar que efetivamente, este já não se mantinha.

Como se pode observar na figura 6.12, o arranque da máquina provocou um erro por volta das 16:31h.

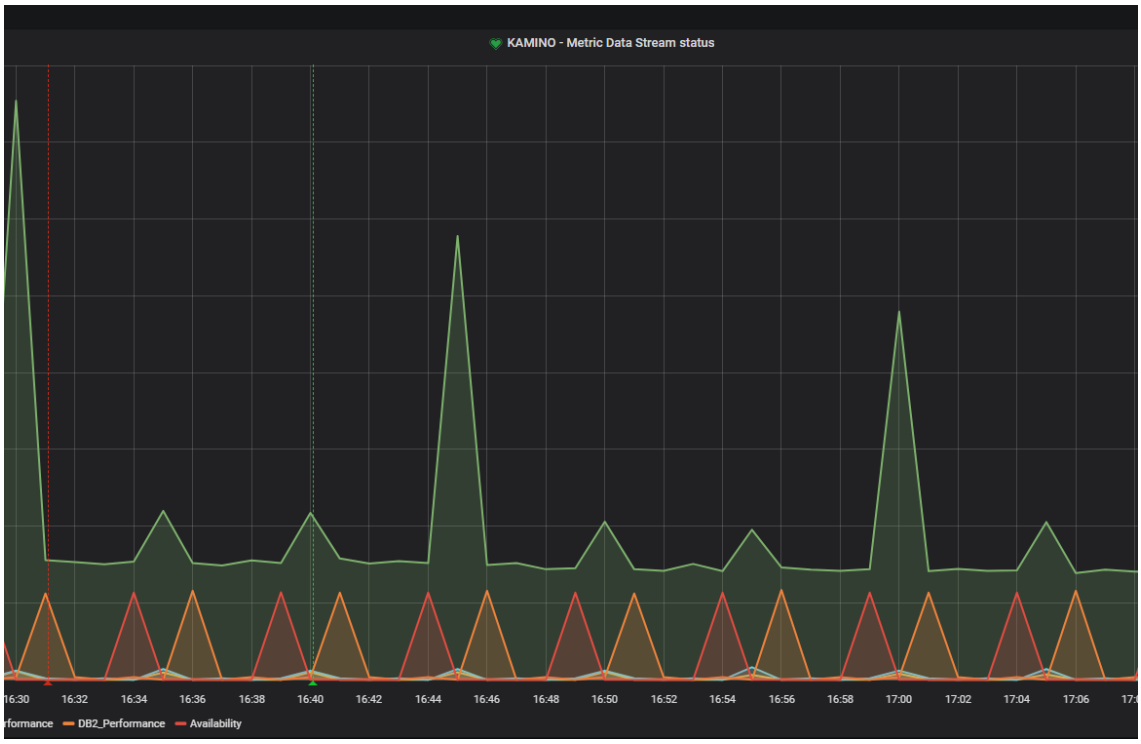


Figura 6.12: *Dashboard* que informa a existência de dados no Kamino

No canal do Slack, o alarme disparou por volta da mesma hora, como esperado.

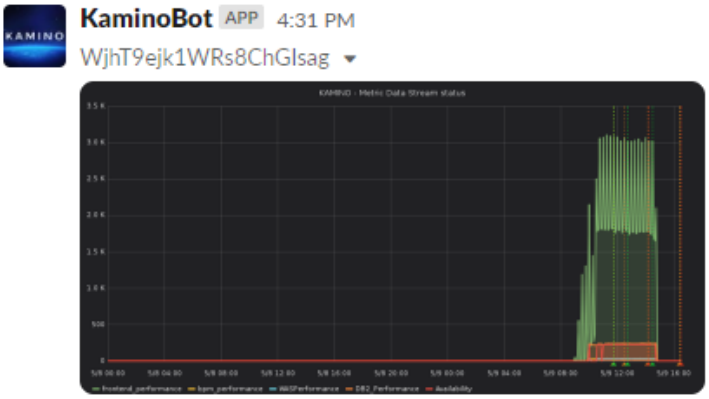


Figura 6.13: Alertas no canal de testes do Slack

Mais tarde, ao acrescentar a chamada do *script* criado no âmbito deste *issue* e ao arrancar novamente a máquina, obteve-se o seguinte resultado:

```
*** Running Mail Loader ***
creating /home/ops/run_mailloader.sh
a487fef8c7d0168f631a4c1d11226b6507c66133935139cb5cde5c8e142ccf0f
mailloader container provisioned and started.

*** Waiting to start Grafana ***
1st start of the day - checking queue to have enough time to index the pending data before starting Grafana
Calculating remaining work...
Without data, waiting to arrive...
Indexing date 2022-05-09 15:54:00

Last timestamp processed was 2022-05-09 15:54:00 and the actual timestamp is 2022-05-09 17:00:26
Internally decided to start Grafana!

*** Running Grafana ***
creating /home/ops/run_grafana.sh
832452d64df8774c10d79b6ae6db310d943396858cffed9f242c74e4491924d3
grafana container provisioned and started.
*** PROVISIONING COMPLETE ***
```

Figura 6.14: *Output* da consola

Como se pode observar, o execução aguarda até receber dados e ao recebê-los, compara o *timestamp* em processamento com o *timestamp* atual, decidindo posteriormente que deve arrancar o Grafana. Analisando a figura 6.12, é possível verificar que por volta das 17:00h não foi registado nenhum constrangimento, o que permitiu concluir que o problema em questão foi corretamente resolvido.

Relativamente às restante tarefas, de alteração de *thresholds* e de tempos de visualização, o processo de testes passou por aplicar as alterações pretendidas ao Kamino de *staging* e verificar visualmente no Grafana que os valores tinham sido alterados corretamente.

6.5.4 Resultado e entrega

Através da figura 6.15, e recorrendo novamente ao caso de uso de *threads* bloqueadas, é possível observar que a resposta dada mediante o aparecimento de um problema foi bastante mais eficaz. Ao ser detetado um pico acima do *threshold* definido por volta das 10h:27m, a plataforma levou apenas 1 minuto a enviar o alerta acerca deste problema, tendo a notificação sido enviada por volta 10h:28m. Posteriormente, e depois do problema resolvido por volta das 10h:37m, foi enviado o alerta de `OK`, por volta das 10h:39m, demorando apenas 2 minutos a notificar novamente a equipa. Embora, neste caso, a melhoria relativa a *threads* (15m $\rightarrow$ 6m) não tenha sido tão significativa com no caso dos *locks* (1h $\rightarrow$ 6m), as melhorias são igualmente visíveis, o que prova que as alterações efetuadas contribuíram bastante para as melhorias de rapidez registadas no processo de resposta da plataforma.

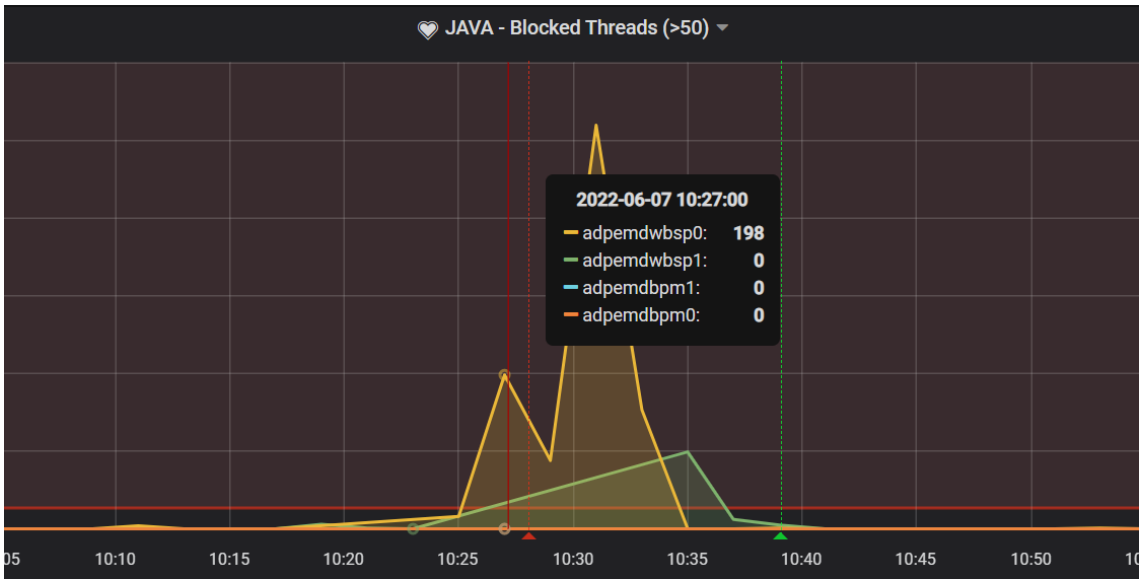


Figura 6.15: *Dashboard* relativo a *threads* bloqueadas depois das modificações

Por fim, e com o objetivo de ser possível validar e marcar a *user story* referente a esta *milestone* como completa, analisaram-se os AC estabelecidos na secção 4.3:

AC	Resultado
(a) O número de notificações enviadas para o canal de Slack sem necessidade não é superior a 1 por dia	✓
(b) O tempo que um problema demora a ser notificado, desde que ocorre, não é superior a 10 minutos	✓

Tabela 6.15: Validação dos AC definidos no âmbito da revisão da alarmística

Relativamente ao ponto (a), foi feito um levantamento no canal de Slack para comprovar que este AC tinha sido cumprido. Já no ponto (b), a análise espelhada na tabela 6.14 permite concluir que este AC foi igualmente efetivado, permitindo assim a conclusão desta *user story*.

6.6 Estabilizar e melhorar a robustez da plataforma

Entende-se por robustez a capacidade do sistema funcionar mesmo em condições anormais. Assim sendo, esta secção aborda as alterações realizadas na plataforma, como o suporte de formatos de informação inválidos, com o objetivo de a tornar mais robusta.

6.6.1 Análise técnica

Esporadicamente, a equipa de suporte indicava nos canais de comunicação que o Kamino não apresentava dados, constituindo isto um entrave ao seu trabalho.

Posto isto, e depois de consultada a análise realizada e anteriormente documentada pela equipa, chegou-se à conclusão de que o problema residia essencialmente no Logstash, que se encontrava inoperacional a cada vez que o Kamino estava indisponível. No passado, para se resolver esta situação, o *container* era reiniciado múltiplas vezes, regressando posteriormente ao normal. No entanto, isto não era viável, pelo que se decidiu endereçar este problema no âmbito das melhorias de robustez.

Outro problema que afetava a robustez residia na chegada do campo `type` com formatação incorreta. Por algum motivo, o *parse* de informação era mal feito e os tipos chegavam, por vezes, mal formatados. Ao prosseguir para a indexação no Elastic Search, este campo não correspondia a nenhum dos esperados e então a informação não era atribuída a nenhum dos nós. Um das questões a resolver prendia-se na identificação e consequente resolução deste problema.

Com o objetivo de melhor visualizar e analisar a indexação dos dados, recorreu-se à extensão Elasticsearch head [40], correspondendo ao *front end web* que permite interagir com um *cluster* da Elastic Search. Como se pode observar na figura 6.16, a informação não foi atribuída ao nó devido e por sua vez, os nomes dos índices encontram-se igualmente mal formatados.

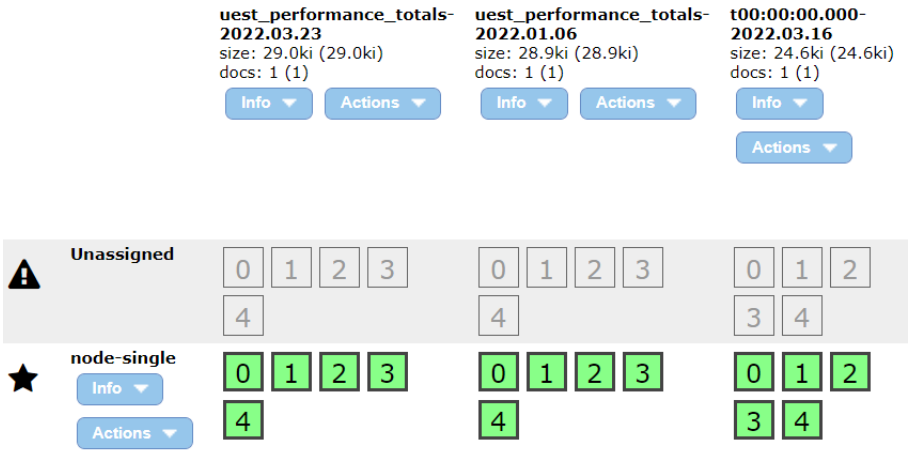


Figura 6.16: Problemas na atribuição de nós da Elastic Search

6.6.2 Implementação

Lidar com *timestamps* inválidos

Para conseguir encontrar o problema e o motivo que o estava a causar, foi necessário replicá-lo. Para isto, o objetivo passou por injetar consecutivamente ficheiros de erro e aguardar que o *Logstash* crashe, interrompendo assim a pipeline de processamento de dados e causando um *outage* na plataforma.

Para proceder à injeção da falha, recorreu-se a um *script* em Python que gerava documentos `JSON` válidos com campos inválidos e que colocava na pasta de entrada do Filebeat. Ao analisar o *backlog* existente, inferiu-se que os principais

problemas residiam na conversão de *timestamps*, uma vez que, por vezes, estes campos apresentam o formato de `Long` ou `NULL`, e não o formato de data esperado para um valor de *timestamp*. Estes problemas originavam, como seria de esperar, exceções que paravam a execução do Logstash.

Como mencionada na subsecção 2.2.1, o Logstash permite definir uma *pipeline* de processamento de dados, recebidos através do Filebeat e enviados depois de aplicadas as transformações definidas para a Elastic Search. Assim sendo, a pipeline é essencialmente constituída por 3 fases: Input, Filter e Output.

Para resolver este problema, foi necessário adicionar no ficheiro referente ao *filter* algumas condições que verificavam o formato do *timestamp* antes de lhe ser aplicada qualquer tipo de conversão. Estas condições garantem que, caso este campo seja do tipo `Long` ou `NULL`, é adicionada uma *tag* ao evento em processamento, com a indicação de erro no formato do *timestamp*. Já no output, acrescentou-se uma configuração que, ao verificar a presença da *tag* indicativa de erro, não só não processa esse evento como passou também a notificar via email acerca desse erro. Este email contém informações fulcrais para a deteção posterior da causa do problema, contendo campos como a *path* do ficheiro que o originou, o ambiente em que o problema ocorreu, o *timestamp* e o conteúdo do ficheiro.

Como auxílio, foram analisados os *logs* dos *containers*, sobretudo do Logstash, para perceber quais as dados a serem processados em cada instante e qual o comportamento do *container*. Para além dos *logs*, o Kibana auxiliou na visualização do fluxo de dados pelos vários *containers*.

Lidar com campo mal formatado

Para resolver este problema, a lógica foi bastante semelhante à do problema da subsecção anterior. Como injetor de falhas, recorreu-se ao mesmo ficheiro, que colocava, à medida do tempo, ficheiros com o campo *type* incorreto, a fim de simular uma situação próxima da realidade. A solução passou novamente pela inclusão de *tags*. Na fase do *input*, e antes de entrar no *filter*, o evento contém *à priori* uma lista de tags. A estas *tags*, foram acrescentados todos os tipos que permitidos e esperados pela *elastic search*, *kamino-wasperformance*, por exemplo. Posteriormente, foi novamente acrescentada uma verificação no *filter* que detetava se o tipo recebido era válido, através da análise de *tags*. Caso seja um tipo inválido, foi adicionada uma *tag* ao evento em processamento, com a indicação de erro na formatação do campo tipo. Já no output, acrescentou-se uma configuração que, ao verificar a presença da *tag* indicativa de erro, notificava via email acerca desse erro. O não envio destes ficheiros para a Elastic Search preveniu eventos mal indexados, como os que podem ser observados na figura 6.16.

Lidar com emails danificados

Durante o desenvolvimento desta *milestone* surgiu um problema que originava a indisponibilidade da plataforma, e como tal, considerando o elevado impacto para equipa de suporte, se incluiu no âmbito das melhorias de robustez. Após ter

decorrido uma *release* do sistema bancário, verificou-se que o último email enviado pelo coletor de dados chegou com a informação incompleta, contendo apenas o *header*. Ao tentar processar este email, ocorreu uma exceção no *Message Separator* do Mailoader, tendo sido assim responsável pelo *outage* no processamento de emails.

Para resolver este problema, foi criado um *script* em Python que, ao detetar que algo não correu bem ao nível do *Message Separator*, é iniciado. Este *script* procura pelo ficheiro incompleto, que contém apenas o *header*, e move-o da pasta atual de processamento para um pasta que contém todos os ficheiros mal formatados. Ao apagar este ficheiro da pasta, o processamento de emails começa a processar todo o *backlog* existente até ao momento, não causando assim a perda de dados dos emails seguintes.

Para detetar e permitir o *follow up* de outros problemas, acrescentou-se o envio de emails para qualquer outra exceção que possa ocorrer durante a execução do Logstash.

Apesar das soluções adotadas conduzirem à perda de dados dos emails danificados, garantem uma maior confiança a nível de robustez, o que era um dos grandes objetivos desta *milestone*. A abordagem do envio de notificações, na forma de *email*, com informações pertinentes acerca do problema, irá auxiliar a equipa a corrigir o problema na sua origem, sendo este um ponto bastante importante para o desenvolvimento.

6.6.3 Testes

Relativamente ao problema relacionado com os *timestamps* inválidos, foram feitos testes relativamente aos campos do `timestamp ocurred` e do `timestamp collected`. Já no caso do campo `type`, escolheu-se um dos 6 tipos definidos e efetuaram-se variações no formato do mesmo, de forma a conseguir replicar os vários cenários possíveis.

Nota: A cor verde indica que o teste em questão cumpriu com os requisitos e que portanto, foi realizado com sucesso. No entanto, definição difere entre os vários testes efetuados. No caso dos testes 1, 4 e 7, a definição de sucesso do teste corresponde ao correto processamento do *email* em causa. Já nos restantes testes, a definição de sucesso corresponde ao descarte do email em causa e ao posterior envio da notificação, em forma de *email*.

Teste	Campo	Formato	Resultado
1	`timestamp ocurred`	TIMESTAMP	
2	`timestamp ocurred`	Long	
3	`timestamp ocurred`	None	
4	`timestamp collected`	TIMESTAMP	
5	`timestamp collected`	Long	
6	`timestamp collected`	None	
7	`type`	"kamino-performance"	
8	`type`	"ino-performance"	
9	`type`	-performance"	
10	`type`	"ance"	

Tabela 6.16: Testes realizados relativamente às melhorias de robustez

No que diz respeito à questão da chegada de emails danificados, os testes efetuados passaram maioritariamente pela injeção de emails propositadamente danificados. O *script* em Python foi alvo de *debug* e de posteriores testes, tendo sido colocado na máquina de *staging* por SFTP. Assim sendo, foi possível garantir que, antes da realização do processo de testes como um todo, o acesso a pastas era o correto ou que a versão de Python desenvolvida estava de acordo com a da máquina Kamino, em Python3.

Posto isto, e conjuntamente com as conclusões retiradas da tabela 6.16, todos os testes foram realizados com sucesso, o que nos permite garantir que as alterações foram corretamente implementadas.

6.6.4 Resultado e entrega

A fim de ser possível validar e marcar a *user story* referente a esta *milestone* como completa, analisou-se o AC estabelecido na secção 4.3:

AC	**Resultado**
(a) O número de falhas com origem em problemas na plataforma não é superior a 1 falha a cada 3 meses	✓

Tabela 6.17: Validação dos AC definidos no âmbito das melhorias de robustez

Neste caso, e tendo em conta que esta foi a última melhoria realizada no âmbito deste estágio, a janela temporal decorrida desde as alterações não permite garantir com certeza que este AC tenha sido alcançado. No entanto, nesta janela temporal, registou-se o envio de um *email* com a notificação acerca da existência de um problema, o que permite concluir que as melhorias realizadas foram capazes de auxiliar na prevenção de um possível erro.

Capítulo 7

Considerações finais

Este capítulo tem o objetivo de apresentar as considerações finais retiradas acerca do estágio que decorreu na Critical Software (CSW).

7.1 Conclusão

Durante a primeira fase do estágio, foi realizado todo um trabalho de integração não só na empresa em si como também na equipa do projeto. Esta integração permitiu desenvolver um sentido crítico relativamente às necessidades do mercado de trabalho e das suas condicionantes, que contrastam com as do meio académico. Tendo isto em conta, é necessário salientar que houve sempre a necessidade de criar um equilíbrio entre as necessidades da empresa e as do meio académico, nomeadamente para a escrita do relatório de estágio. Simultaneamente, essa primeira fase do estágio permitiu que o estagiário contactasse com o mundo real, aplicando alguns dos conhecimentos adquiridos durante o seu percurso académico. Durante o primeiro semestre foram aplicadas essencialmente técnicas de análise funcional e de planeamento de tarefas, adquiridas em Engenharia de Requisitos.

Apesar destes conhecimentos constituírem uma boa base de trabalho, a integração numa empresa real tornou-se um desafio devido à falta de experiência profissional do estagiário.

A segunda fase do estágio focou-se na implementação de melhorias na plataforma, tirando partido de algum conhecimento adquirido durante o curso, como Sistemas Operativos ou Introdução à Programação e Resolução de Problemas. Estas unidades curriculares garantiram os conhecimentos base de Linux, Python e Bash necessários em vários segmentos do projeto, havendo no entanto outras cadeiras que, embora em componente mais reduzidas, contribuíram de igual forma para o desenvolvimento de algumas das tarefas. No entanto, os conhecimentos necessários ao desenvolvimento do projeto iam bastante além dos adquiridos até ao momento, pelo que foi necessário algum esforço extra para conciliar todos estes aspetos e conceitos.

Assim, uma das grandes dificuldades sentidas foi a introdução de tecnologias totalmente desconhecidas em cada tarefa, o que, devido à diversidade de *scopes* de cada uma, levava à ocorrência de uma curva de aprendizagem bastante acentuada. Por outro lado, numa fase inicial, a questão do contacto com uma nova realidade aliada, novamente, à diversidade de tecnologias inseridas, dificultaram a adaptação do estagiário.

No entanto, é importante salientar que tanto os orientadores, da CSW e do Departamento de Engenharia Informática (DEI), como toda a equipa do projeto suavizaram as dificuldades encontradas, bem como todo processo de integração que esta mudança inclui. É de destacar ainda que a disponibilidade e sentimento de entre-ajuda da equipa contribuíram para uma comunicação e interação eficaz durante todo projeto pelo que se pode considerar ter sido uma mais-valia durante todo o trabalho desenvolvido ao longo do estágio.

O cumprimento de todos os requisitos *must have* assim como o correto seguimento do planeamento de estágio elaborado, permitem concluir que os principais objetivos deste estágio foram alcançados com sucesso.

7.2 Trabalho futuro

Como trabalho futuro, identificaram-se algumas ações que poderão ser posteriormente implementadas, com o objetivo de melhorar a experiência com a plataforma por parte da equipa de suporte da plataforma.

Relativamente ao método de comunicação e de envio de informação atualmente implementados, seria interessante analisar novas soluções. Uma das tecnologias a estudar poderia ser uma *File Transfer Protocol* (FTP) *bridge*. Este protocolo permite aceder a ficheiros que se encontrem no servidor, podendo realizar *downloads*, *uploads* e transferências entre dois ambientes. Este tipo de comunicação é já usada na equipa de DevOps, para proceder à realização de *releases*, pelo que se trata de uma tecnologia bastante poderosa. O atual envio através emails requer um *overhead* fixo associado ao envio de cada email (30s). Comparando os resultados de cada abordagem, optando pela FTP *bridge*, a latência passaria a ser entre 7s e 15s, sendo este valor 3x menor que o atual. No entanto, esta solução acarreta algumas contrapartidas, como a persistência de dados. Ao contrário do que acontece com o *email*, o FTP possui mecanismos de limpeza automáticos, realizados fora do nosso controlo, tornando estes dados voláteis. Para contornar esta situação, seria necessário recorrer a uma máquina que recolhesse constantemente os dados recebidos e que os armazenasse, o que tornaria este processo bastante mais complexo e com um maior custo associado. Assim sendo, a análise relativa a esta abordagem deve ser realizada com bastante cuidado, registando os impactos que a sua implementação poderá eventualmente produzir.

Num outro aspeto, e no âmbito da alterações efetuadas relativamente à robustez da plataforma, o descarte de *emails* danificados foi implementado como solução possível para a resolução dos problemas de robustez identificado. No entanto, esta solução não é a ideal, fazendo com que se possa identificar como melhoria

futura o seguimento desses emails. Assim sendo, este aspeto tornaria possível a identificação do motivo que originou o problema e a consequentemente resolução na sua origem.

Por fim, e considerando o *backlog* resultante da elicitação de requisitos, devem ser visados todos os requisitos que não foram desenvolvidos no âmbito deste estágio. Deve ser dado ênfase á adaptação da *stack* tecnológica assim como à adaptação da plataforma na sequência de alterações de infraestrutura.

Referências

[1] *A Cron expressions* [2022], https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm.

[2] *Apache skywalking* [2022], https://skywalking.apache.org/.

[3] Atlassian [2021], 'User stories: Examples and template', https://www.atlassian.com/agile/project-management/user-stories. Acesso em 15 de Dezembro de 2021.

[4] *Blocked attachments in Outlook* [2022], https://support.microsoft.com/en-us/office/blocked-attachments-in-outlook-434752e1-02d3-4e90-9124-8b81e49a Acesso em 2 de Julho de 2022.

[5] Boogaard, K. [2022], 'How to write smart goals', https://www.atlassian.com/blog/productivity/how-to-write-smart-goals.

[6] *Datadog Integrations* [2021], https://docs.datadoghq.com/integrations/. Acesso em 15 de Dezembro de 2021.

[7] *Datadog Pricing* [2021], https://www.datadoghq.com/pricing/. Acesso em 15 de Dezembro de 2021.

[8] *Data in: Documents and indices: Elasticsearch Guide* [2021], https://www.elastic.co/guide/en/elasticsearch/reference/master/documents-indices.html. Acesso em 18 de Janeiro de 2022.

[9] *Docker SDK for Python* [2022], https://docker-py.readthedocs.io/en/stable/index.html#. Acesso em 9 de Maio de 2022.

[10] *Docker standalone: Signoz* [n.d.], https://signoz.io/docs/install/docker. Acesso em 28 de Junho de 2022.

[11] Dorofee, A. J., Alberts, C. J., Higuera, R. P., Murphy, R. L., Walker, J. A. and Williams, R. C. [1997], *Continuous Risk Management Guidebook*, Carnegie Mellon University, 44.

[12] *Dynatrace pricing* [2021], https://www.dynatrace.com/pricing/. Acesso em 15 de Dezembro de 2021.

[13] *Dynatrace technology support* [2021], https://www.dynatrace.com/support/help/technology-support. Acesso em 15 de Dezembro de 2021.

[14] *Elastic Stack* [2021], https://www.elastic.co/elastic-stack/. Acesso em 15 de Dezembro de 2021.

[15] Feliciano, R. N. [2022], 'Deprecating ubuntu 14.04 and 16.04 images: Stay secure with modern ubuntu', https://circleci.com/blog/ubuntu-14-16-image-deprecation/. Acesso em 22 de Junho de 2022.

[16] Gikonyo, J. [2021], 'Linux containers vs docker - what is the difference and why docker is better', https://www.section.io/engineering-education/lxc-vs-docker-what-is-the-difference-and-why-docker-is-better/. Acesso em 15 de Dezembro de 2021.

[17] *Grafana: The Open Observability Platform* [2021], https://grafana.com/, journal=Grafana Labs. Acesso em 24 de Janeiro de 2022.

[18] Guchin, A. [2018], 'Apm tools comparison: Which one should you choose?', https://dzone.com/articles/apm-tools-comparison-which-one-should-you-choose. Acesso em 15 de Dezembro de 2021.

[19] *Healthy Data, healthy business - modern cloud ETL* [2022], https://www.talend.com/.

[20] Horovits, D. [2020], 'The complete guide to the elk stack', https://logz.io/learn/complete-guide-elk-stack/. Acesso em 24 de Novembro de 2021.

[21] *Introduction to infrastructure integrations* [2021], https://docs.newrelic.com/docs/infrastructure/infrastructure-integrations/get-started/introduction-infrastructure-integrations/. Acesso em 15 de Dezembro de 2021.

[22] K.A.Chandrakanth, M. [2022], 'Paper on plan do check act (pdca) - improving quality through agile accountability', https://agilealliance.org/wp-content/uploads/2016/01/PDCA.pdf. Acesso em 17 de Junho de 2022.

[23] *Kibana: Explore, visualize, Discover Data* [2021], https://www.elastic.co/kibana/. Acesso em 24 de Janeiro de 2022.

[24] Kurejko, J. [2021], 'Current trends in the field of apm (application performance monitoring)', https://adastra.digital/en/resources/blog/current-trends-field-apm-application-performance-monitoring/. Acesso em 28 de Abril de 2022.

[25] *Logging: Logstash reference [8.2]* [2022], https://www.elastic.co/guide/en/logstash/current/logging.html. Acesso em 22 de Junho de 2022.

[26] *Modern monitoring & security* [2021], https://www.datadoghq.com/. Acesso em 21 de Janeiro de 2022.

[27] *Monitor, debug, and improve your entire stack* [2021], https://newrelic.com/. Acesso em 21 de Janeiro de 2022.

[28] *Moscow prioritization* [2021], https://www.productplan.com/glossary/moscow-prioritization/.

[29] *New Relic Pricing* [2021], https://newrelic.com/pricing. Acesso em 15 de Dezembro de 2021.

[30] *Open source APM: Signoz* [2022], https://signoz.io/.

[31] *OpenTelemetry* [2022], https://opentelemetry.io/.

[32] *Platform Requirements* [2021], https://docs.appdynamics.com/21.2/en/application-performance-monitoring-platform/planning-your-deployment/platform-requirements. Acesso em 15 de Dezembro de 2021.

[33] *Prometheus* [2021], https://prometheus.io/. Acesso em 15 de Dezembro de 2021.

[34] *Simple pricing for enterprise APM and Observability* [2021], https://www.appdynamics.com/pricing. Acesso em 15 de Dezembro de 2021.

[35] *Storage: Prometheus* [2021], https://prometheus.io/docs/prometheus/latest/storage/. Acesso em 15 de Dezembro de 2021.

[36] *The Leader in Automatic and Intelligent Observability* [2021], https://www.dynatrace.com/. Acesso em 21 de Janeiro de 2022.

[37] *The visual collaboration platform for every team: Miro* [2022], https://miro.com/.

[38] *The world's #1 APM solution* [2021], https://www.appdynamics.com/. Acesso em 21 de Janeiro de 2022.

[39] *Ubuntu release cycle* [2022], https://ubuntu.com/about/release-cycle. Acesso em 2 de Julho de 2022.

[40] Vorapoap [2022], 'Elasticsearch head extension - chrome extension containing the elasticsearch head application', https://github.com/vorapoap/elasticsearch-head-chrome. Acesso em 2 de Junho de 2022.

[41] *What is Docker?* [2018], https://www.redhat.com/en/topics/containers/what-is-docker. Acesso em 15 de Dezembro de 2021.

[42] Yigal, A. [2020], 'Grafana vs. kibana: The key differences to know', https://logz.io/blog/grafana-vs-kibana/. Acesso em 15 de Dezembro de 2021.

Apêndices

Apêndice A

Diagramas de Gantt

			Setembro			Outubro				Novembro					Dezembro				Janeiro		
ID da tarefa	Nome da tarefa	Duração (em semanas)	12 a 18	19 a 25	26 a 02	03 a 09	10 a 16	17 a 23	24 a 30	31 a 06	07 a 13	14 a 20	21 a 27	28 a 04	05 a 11	12 a 18	19 a 25	26 a 01	02 a 08	09 a 15	17
T01	Receçao na empresa	1																			
T02	*Ramp Up*	2																			
T03	Familiarização com as ferramentas de trabalho	1																			
T04	Apresentação e exploração da plataforma	1																			
T05	Estado da arte	10																			
T06	Elicitação de requisitos	6																			
T07	Relatório intermédio	12																			Entrega

Tabela A.1: Diagrama de Gantt inicial - 1º Semestre

			Setembro			Outubro				Novembro					Dezembro				Janeiro			
ID da tarefa	Nome da tarefa	Duração (em semanas)	12 a 18	19 a 25	26 a 02	03 a 09	10 a 16	17 a 23	24 a 30	31 a 06	07 a 13	14 a 20	21 a 27	28 a 04	05 a 11	12 a 18	19 a 25	26 a 01	02 a 08	09 a 15	16 a 22	24
T01	Receçao na empresa	1																				
T02	*Ramp Up*	2																				
T03	Familiarização com as ferramentas de trabalho	1																				
T04	Apresentação e exploração da plataforma	1																				
T05	Estado da arte	10																				
T06	Elicitação de requisitos	6																				
T07	Relatório intermédio	12																				Entrega

Tabela A.2: Diagrama de Gantt final - 1º Semestre

					Fevereiro			Março				Abril				Maio					Junho				Julho
ID da tarefa	Nome da tarefa	Início	Fim	Duração (em dias uteis)	13 a 19	20 a 26	27 a 05	06 a 12	13 a 19	20 a 26	27 a 02	03 a 09	10 a 16	17 a 23	24 a 30	01 a 07	08 a 14	15 a 21	22 a 28	29 a 04	05 a 11	12 a 18	19 a 25	26 a 02	04
T01	Montar infraestrutura para desenvolvimento e testes do Kamino	14/fev	01/mar	12																				
T02	Reduzir a latência na transmissão de métricas	02/mar	01/abr	23																				
T03	Rever e alterar a frequência de obtenção de métricas	04/abr	29/abr	18																				
T04	Rever e reconfigurar totalmente a alarmística	02/mai	25/mai	18																				
T05	Estabilizar e melhorar a robustez da plataforma	26/mai	17/jun	15																				
T06	Finalizar o relatório de estágio	20/jun	02/jul	10																				Entrega

Tabela A.3: Diagrama de Gantt inicial - 2º Semestre

					Fevereiro			Março				Abril				Maio					Junho				Julho
ID da tarefa	Nome da tarefa	Início	Fim	Duração (em dias uteis)	13 a 19	20 a 26	27 a 05	06 a 12	13 a 19	20 a 26	27 a 02	03 a 09	10 a 16	17 a 23	24 a 30	01 a 07	08 a 14	15 a 21	22 a 28	29 a 04	05 a 11	12 a 18	19 a 25	26 a 02	04
T01	Montar infraestrutura para desenvolvimento e testes do Kamino	14/fev	01/mar	12																				
T02	Reduzir a latência na transmissão de métricas	02/mar	01/abr	23																				
T03	Corrigir corrupção de arquivos aquando da receção	04/abr	08/abr	5																				
T04	Rever e alterar a frequência de obtenção de métricas	11/abr	29/abr	13																				
T05	Rever e reconfigurar totalmente a alarmística	02/mai	25/mai	18																				
T06	Estabilizar e melhorar a robustez da plataforma	26/mai	17/jun	15																				
T07	Finalizar o relatório de estágio	20/jun	02/jul	10																				Entrega

Tabela A.4: Diagrama de Gantt final - 2º Semestre